ProjetVALMEM

Sujet: « Vérification des circuits mémoires »

- Document 2 -

Traduction Automatique de Descriptions VHDL en automates temporisés

Structure du programme VDHL2TA

Rédigé par Abdelrezzak BARA
Encadré par Mme. Emmanuelle ENCRENAZ

Laboratoire d'accueil : Laboratoire d'Informatique de PARIS 6 – LIP6
Equipe: Architecture des Systèmes intégrés et Micro électronique (ASIM)
Unité Mixte de Recherche - UMR 7606 (CNRS - UPMC)

Dans ce document, nous allons présenter le programme qui permet de générer le réseau d'automates temporisés décrits en hytech ou en uppaal à partir d'une description spécifiée dans le langage de description de VHDL, ainsi que les principales structures de données employées. Tout d'abord, nous allons commencer par décrire l'analyseur qui permet de donner une représentation structurelle de la description en VHDL d'entrée. Puis, nous allons présenter la fonction de génération de la description hytech ou uppaal associée à la description VHDL à partir du format intermédiaire produit par l'analyseur. Cette fonction implémente les algorithmes de traduction des affectations des signaux et des processus du programme VHDL en automates temporisés. Nous terminerons ensuite par présenter les résultats de tests obtenus sur l'un des exemples des modèles passés sur le programme. L'architecture du programme est décrite dans le schéma de la figure mentionnée ci-dessous.

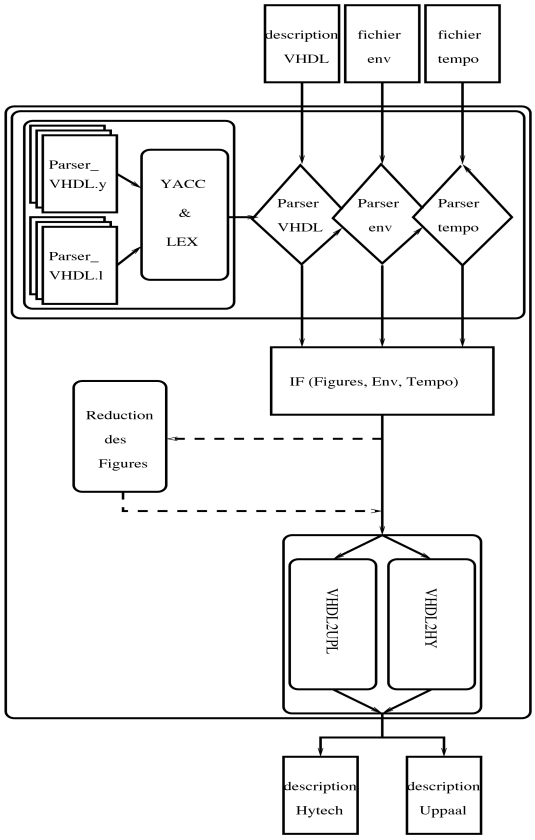


Figure 1: Le programme de génération des descriptions en Hytech ou Uppaal.

1.1. L'analyseur des descriptions VHDL:

Comme nous le montre le schéma de la figure 1 mentionné ci-dessus, ce programme intègre un analyseur des descriptions en VHDL qui est établi, en utilisant le générateur des analyseurs Yacc & Lex, à partir de la grammaire spécifiée par Pirouz (qu'on a étendu par des règles de génération des processus). L'analyseur a le rôle de traduire les descriptions en VHDL dans un format concret, structures de figures (le format intermédiaire que le programme va employer par la suite pour générer les descriptions en hytech ou en uppal associées à ces descriptions en VHDL) après avoir vérifié que ces descriptions d'entrée sont bien conformes au formalisme VHDL. Evidemment, l'analyseur traite juste un fragment donné du langage VHDL, décrit par la grammaire présentée dans l'annexe A. les descriptions accéptés ne contiennent que des affectations simples et des processus de type If-elseif . . . -elseif {else} qui affectent une valeur à un seul signal donné. Leur format est mentionné ci-dessous.

```
1)      s <= f (e_1, . . . , e_n).
2)      Process_Name Process (e_1, . . . , e_n)
```

```
begin
.       if guard_1
.               then s <= f_1 () ;
.       else-if guard_2
.               then s <= f_2 () ;
.       .
.       .
.       else-if guard_n
.               then s <= f_n () ;
end
```

Voici un exemple d'une description qu'on peut passer sur l'analyseur.

```
ENTITY M_1 IS
  PORT (
         s   : out   BIT;
         e_1 : in    BIT;
         e_2 : in    BIT;
         e_3 : in    BIT;
         e_4 : in    BIT;
            e_5 : in    BIT
  );
END M_1;

ARCHITECTURE RTL OF M_1 IS
  SIGNAL v_1 : BIT;
BEGIN
    REG10: PROCESS (e_1, e_2, e_3,
e_4)
   BEGIN
     IF e_1 = '1' THEN
       v_1 <= e_3;
     ELSIF e_2 = '1' THEN
       v_1 <= e_4;
     END IF;
   END PROCESS;

   s <= (not (v_1) or not (e_5));

END
```

Comme on a cité ci-dessous, les figures générées après l'analyse d'une description VHDL donnée sont décrites formellement dans notre programme par la structure suivante :

- Représentation des figures comportementales.

```
figure_st
{
   char *name;              /* nom de la figure */
   ablin_st *in;            /* pointeur vers la liste des entrees */
   ablsig_st *aux;          /* pointeur vers les signaux auxiliaires */
   ablsig_st *out;          /* pointeur vers la liste des sorties */
   figure_st *next;         /* pointeur sur la figure suivante */
}
figure_t;
```

- Représentation des signaux d'entree.

```
 ablin_st
{
   struct ablin_st *next;            /* pointeur sur l'entree suivante */
   char *name;                       /* nom du signal */
}
ablin_t;
```

- Representation des signaux auxilieres et des signaux de sortie.

```
ablsig_st
{
   char *name;              /* nom du signal */
   int statement_type;      /* affectation d'un signal, ou un process */
   abl_st *abl;             /* definition du comportement. Soit une fonction ou  */
                            /* un processus */
   reduced_signals_t reduced_signals; /* definit la liste des signaux supprimés
                            lors la phase de la réduction. Initialement, elle est vide. */
   int after;               /* temps de calcul de l'expressions en ns */
   ablsig_st *next;         /*pointeur sur la definition du signal suivant */
}
ablsig_t;
```

- Representation des processus :

```
process_st
{
   char * name;             /* nom du processus */
   strings_t * in_signals;/* la liste des noms des signaux d'entrée */
   char * out_signal;       /* le nom du signal de sortie */
   abls_t * guards;         /* la liste des guardes figurant dans le processus */
   struct abls_t * functions;       /* la liste des fonctions dont la valeur
                                       est affectée au signal de sortie */
}
process_t;
```

- Representation des abls :

```
abl_st
{
   abl_t * next;      /* pointeur sur l'abl suivante */
   abl_t *data;       /* soit une constante booléenne (0,1), soit un signal ou une abl.*/
}
bl_t;
```

Les abls sont des structures de données inductives introduites par Pirouz afin de représenter les fonctions

logiques et les expressions booléennes. En utilisant la structure de base abl_t, on peut définir les fonctions inductivement comme suit:

- Les constantes booléennes 0 et 1 sont réprésentées par les abls abl (0, ∅) et abl (1, ∅) respectivement.
- chaque signal s est répresenté par l'abl abl (s, ∅).
- Les opérateurs logiques NOT, AND, OR et XOR sont réprésentés par les abls suivantes abl(NOT, ∅), abl (AND, ∅), abl (OR, ∅) et abl (XOR, ∅).
- Si t_1 et t_2 sont deux abls qui représentent les deux fonctions booléennes f_1 et f_2, et op_1 et op_2 sont les abls qui représentent respectivement les opérateurs logiques unaires et binaires, alors abl(op_1, abl(t_1, ∅)) et abl(op_2, abl(t_1, abl (t_2, ∅))) représentent respectivement les fonctions *op_1 f_1* et *f_1 op_2 f_2*.

En utilisant ces structures, le format intermédiaire obtenu après l'analyse de la description M_1 est donné dans la figure 2.

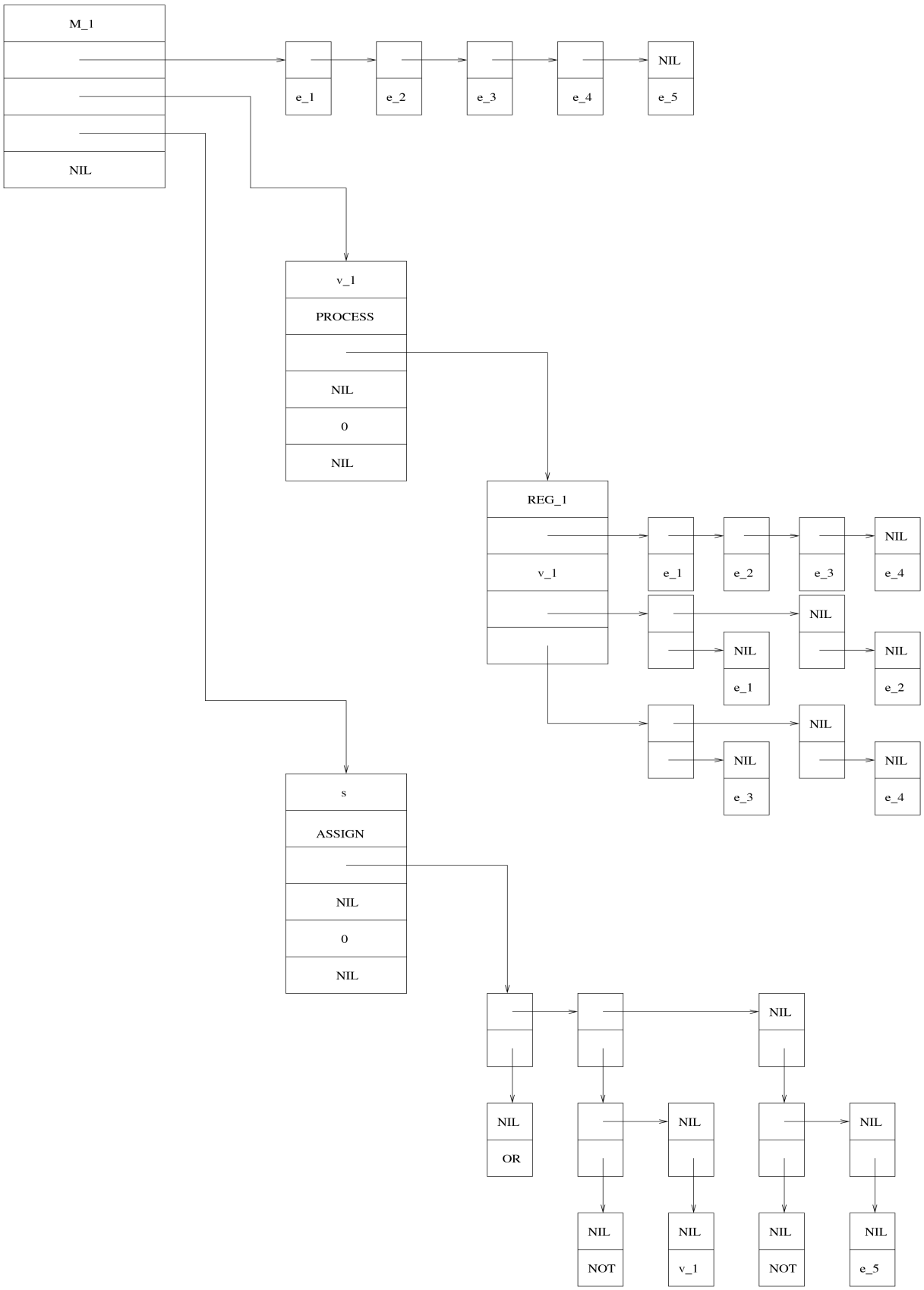


Figure 2: La figure associée au modèle VHDL M_1.

1.2. Réduction des figures :

On peut simplifier les affectations des signaux des figures générées par l'analyseur en utilisant la stratégie de réduction décrite ci-dessous. Cette dernière est basée sur l'ensemble de règles de réécriture suivantes (règles de transitivité) .

- $s'' <= NOT(s') \wedge s' <= NOT(s) \Rightarrow s'' <= s$ *(01)*
- $s'' <= s' \wedge s' <= s \Rightarrow s'' <= s$ *(02)*
- $s'' <= s' \wedge s' <= NOT(s) \Rightarrow s'' <= NOT(s)$ *(03)*
- $s'' <= NOT(s') \wedge s' <= s \Rightarrow s'' <= NOT(s)$ *(04)*

telque delta (s'') = delta (s') + delta (s).

Remarque : On voit facilement que le système de réécriture constitué de ces règles de réduction est confluent.

La stratégie de réduction est décrite comme suit:

- Appliquer les règles de ce système d'une manière itérative sur les affectations des signaux de la figure qu'on veut réduire, jusqu'à ce qu'aucune règle de réduction ne soit applicable.

La fonction REDUCTION-FIGURE mentionnée ci-dessous implémente bien cette stratégie de réduction. La variable *appliquer* donne une indication sur le fait qu'il reste encore des règles de réduction qui sont applicables.

```
REDUCTION-FIGURE (figure)
01        appliquer <- 1
01        Tanque appliquer
02        faire       appliquer <- 0
03        Figure = Reduction-Figure (Figure, R, appliquer)
05        retourne Figure
```

On note que la fonction *Reduction-Figure* a le rôle de réduire les affectations des signaux de la figure, en appliquant les régles de réduction du système de réecriture *R*.

1.3. Génération des descriptions en Hytech et Uppaal:

A partir de la structure de figure associée à la description en VHDL d'entrée, notre programme intègre deux fonctions nommées Figure2Hytech et Figure2Uppaal qui ont le rôle de générer, en utilisant la fonction de base nommée *GenererDescriptionRAT*, respectivement la description en hytech et la description en uppaal du réseau d'automates temporisés associés aux affectations des signaux auxiliaires et sortie de la figure. Cette dernière fonction est basée sur l'algorithme de traduction des processus et des affectations des signaux en automates temporisés qu'on a présenté dans le premier document. Ci-dessous, on se contente de donner la description formelle de la fonction de génération des descriptions Hytech. Celles de génération de description d'uppaal est définie presque de la même façon que la précédente.

Figure2Hytech (figure, description_hytech, . . .)
01 GenererDescriptionRAT (figure, description_hytech, . . .) .
02 GenererSectionAnalyseSysteme (figure, description_hytech, . . .).

GenererDescriptionRAT (figure, description_hytech, . . .)
01 GenererSectionVariablesDescriptionRAT (figure, description_hytech, . . .).
02 GenerateSectionCompositionAutomatesDescriptionRAT (figure, description_hytech, . . .).

Comme elle est décrite ci-dessus, la première fonction *GenererSectionVariablesDescriptionRAT* permet de générer la déclaration des horloges, des variables discrètes et des paramètres du réseau d'automates temporisés associés à la figure, tandis que la deuxième fonction permet de générer ce réseau d'automates. Cette dernière fait des appels aux deux fonctions *Process2hytech* et *Assignement2hytech* qui implémentent l'algorithme décrit dans le document 1.

GenererSectionVariablesDescriptionRAT (figure, hytech_file, . . .)
01 GenererSectionVariablesClockDescriptionRAT (figure, description_hytech, . . .).
02 GenerateSectionVariablesDiscreteDescriptionRAT (ablfig, description_hytech, . . .).
03 GenerateSectionVariablesParameterDescriptionRAT (ablfig, description_hytech, . . .).

GenererSectionCompositionAutomatesDescriptionRAT (figure, hytech_file, . . .)
01 pour chaque signal de {out_signal (figure) U aux_signals (figure)}
02 do si type_statement(signal) = process
03 alors Process2hytech (signal, Process(signal), description_hytech).
04 sinon Assignement2hytech (signal, ABL(signal), fighytech_file).

Comme on voit, la fonction *GenererDescriptionRAT* ne produit pas explicitement des structures d'automates temporisés. En effet, elle génère directement leur description.

Ci-dessous, on trouve en format graphique les automates associés à la description en hytech ou uppaal générée par le programme associé au modèle VHDL M_1. Pour des raisons de simplicité on a omis la description de l'étiquetage des états et des transitions. Il est déjà décrit dans le document 1.

– *A propos de l'instruction d'affectation du signal de sortie s:*

Comme on a vu dans le document 1, les instructions d'affectation des signaux sont décrites en un seul automate qui est similaire à l'automate principal des processus. Dans ces automates, les locations 4 n'existent pas. L'automate associée à l'instruction s <= not (v_1) or not (e_5) est donné ci-dessous.

On a renommé les locations l_f , l_x0 et l_x1 par 3_ass, 2_ass et 1_ass respectivement dans l'automate P'.

On rappelle que :

- 3_ass représente l'état du signal de sortie dans lequel est stable. Les deux états 1_ass et 2_ass représentent respectivement les états de calcul de sortie *s_up* et *s_down*.
- Les étiquetages 1-2 et 2-1 représentent des disjonctions de transitions dont l'application, qui est causée par un changement de l'un des signaux d'entrée, provoque une modification sur la valeur du signal de sortie.
- Les étiquetages 1-1, 2-2 et 3-3 représentent des disjonctions de transitions dont l'application, qui est causée par un changement de l'un des signaux d'entrée, ne provoque aucun changement sur la valeur du signal de sortie.
- Les étiquetages 1-3 et 2-3 représentent des disjonctions de transitions dont l'application provoque une valeur stable pour le signal de sortie.
- Les étiquetages 3-1 et 3-2 représentent des disjonctions de transitions dont l'application, qui est causée par un changement des signaux d'entrée, provoque une modification sur la valeur stable du signal de sortie.

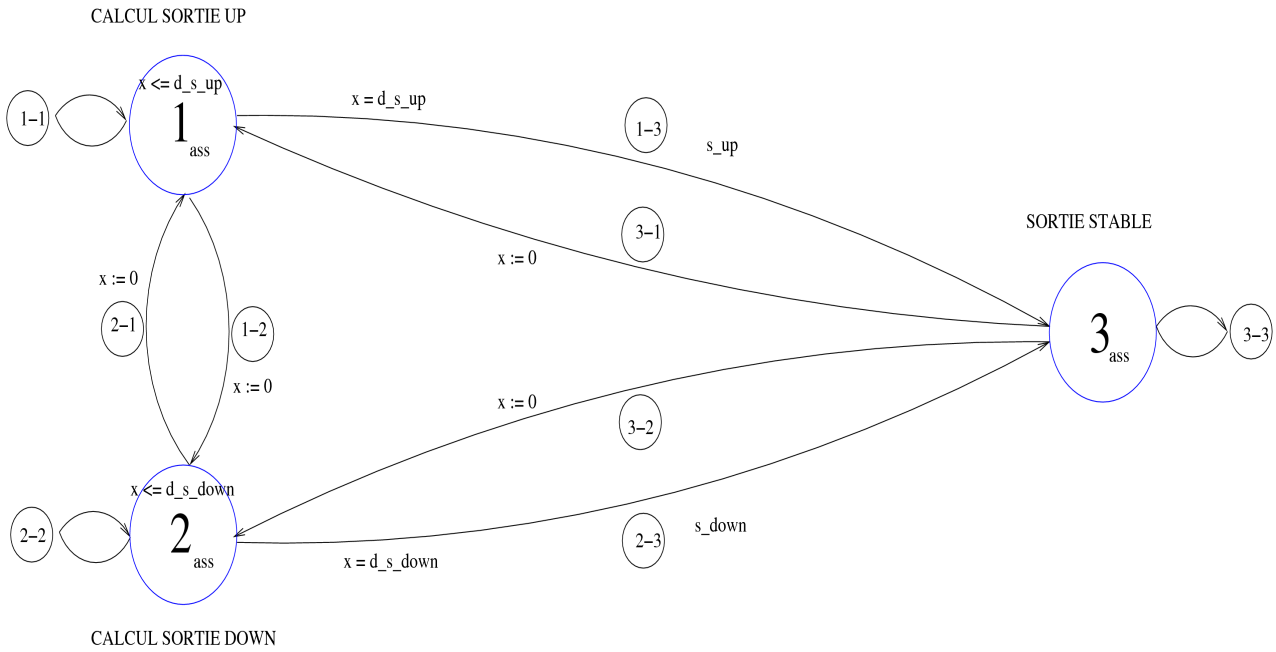


Figure 3 : L'automate associé à l'affectation du signal s.

A propos du processus REG_1:

Les automates temporisés dont la composition modélise ce processus sont donnés comme suit:

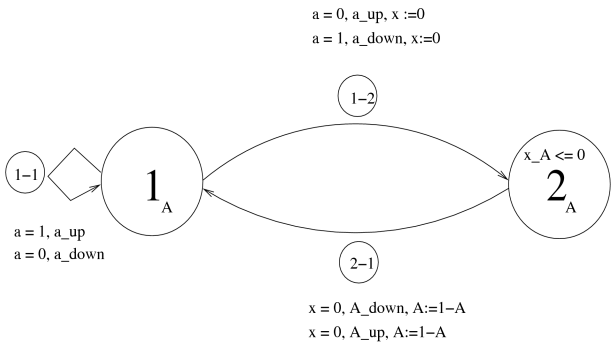


Figure 4.a : L'automate A de la condition A = e_1.

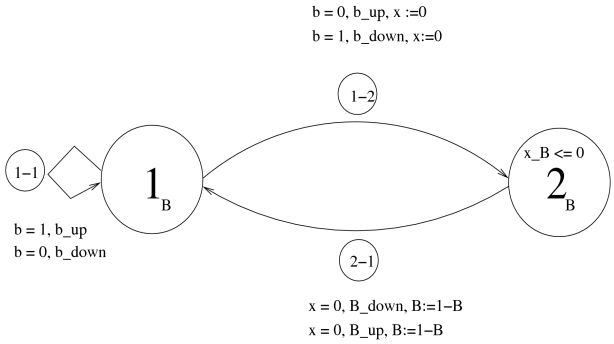


Figure 4.b : L'automate B de la condition B = e_2.

Comme on voit, on a renommé les locations l_x1, l_x0, l_f et l_s par 1p', 2p', 3p' et 4p' respectivement dans l'automate P'.

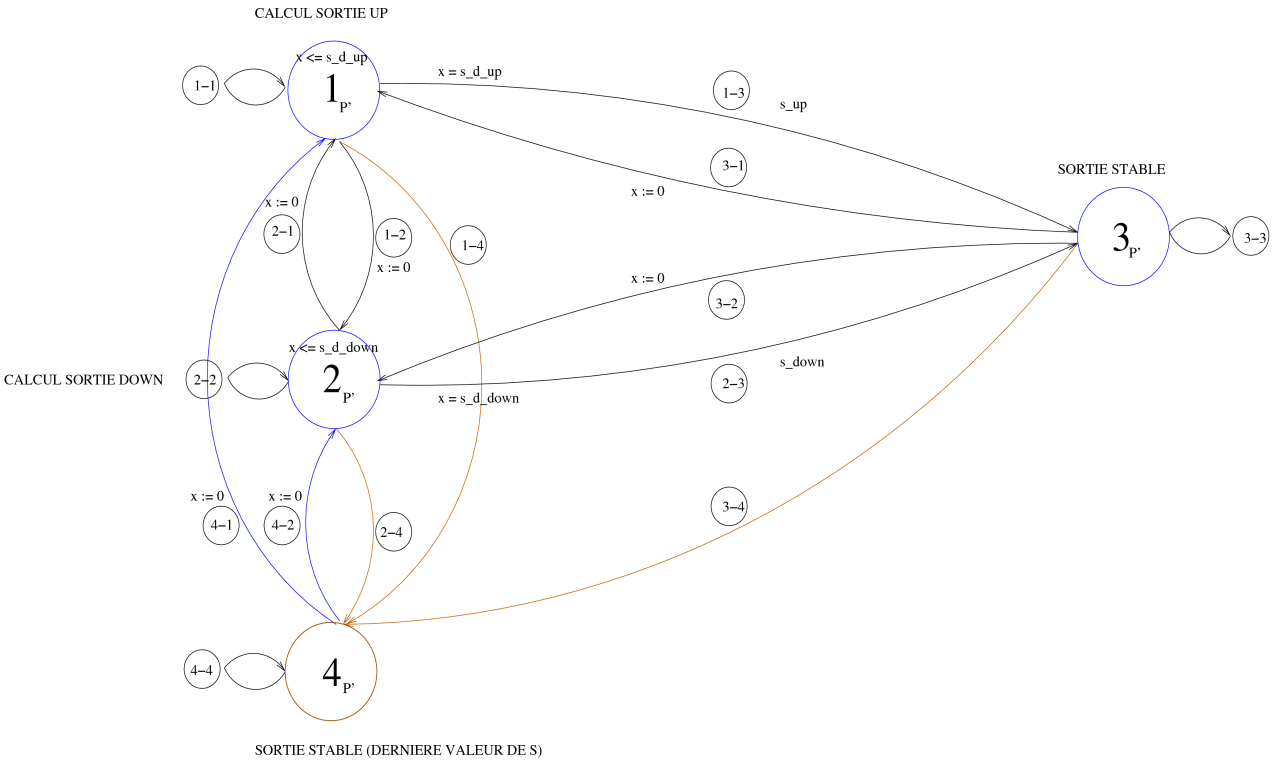


Figure 4.c : L'automate principale P'.

On rappele que :

a). Pour les automates A & B:

- 1_A (1_B) représente l'état dont la valeur de la garde A (B) est stable, et l'état 2_A (2_B) représente l'état de délai nulle où on change la valeur du signal intermédiaire correspondant à la valeur de la garde A (B) par une transition sortante vers l'état stable 1_A (1_B).

- 1-1 représent une disjonction de transitions qu'on peut appliquer lorsque il aura des changements des signaux de la garde A (B) sans provoquer aucune modification sur la valeur de cette dernière. Inversement, 1-2 représente une disjonction des transitions dont l'application causée par un changement d'un signal de la garde A (B) implique le changement de la valeur de cette garde.
- La transition sortante de l'état de délai nulle 2_A (2_B) est associée à la modification de la variable intermédiaire correspondante à la garde A (B).

b). Pour l'automate P':

- L'état 3_P' représente l'état du signal de sortie lorsqu'il est stable. Dans cet état le signal de sortie prend la valeur de l'une des affectations de processus dépendemment des valeurs des gardes. Les deux états 1_P' et 2_P' représentent respectivement les états de calcul de de sortie *up* et *down*. L'état 4_P' représente l'état dans lequel on garde l'ancienne valeur pour le signal de sortie si aucune garde de processus n'est satisfaite.
- L'étiquetage des transitions est défini presque de la même manière que celui des transitions de l'automate d'affectation du signal s. Ici, l'application des transitions est causée par le changement des variables intemédiaires associées aux gardes et aussi par le changement des signaux apparus dans les affectations de processus. Donc , les transitions sont sensibles aux signaux de sensibilité du processus.

- Automatiser La vérification :

Afin établir les temps de *setup* minimum pour les signaux d'entrée, on a défini une fonction qui permet, à partir d'un environnement initial définie par l'utilisateur (qu'on suppose satisfaisant toutes les propriétés de la spécification à vérifier), d'effectuer la tâche de la vérification du modèle généré sur la propriété à vérifier d'une manière itérative sur les valeurs décrémentées des temps setup des signaux d'entrée jusqu'à que la propriété devienne fausse. La description formelle de la fonction est donnée i-dessous :

ParametricVerification (figure, env)
fin_dec [0, ... , nb(inputs_signals(figure))] = 0.

Figure2upl (figure, uppaal_description, . . .)
res_verifyta = verifyta (uppaal_description, query_filename);
si res_verifyta
alors tant que (exist un signal s ∈ inputs_signals(figure) telque fin_dec [s] = 0.)
faire decrementer le temps de setup dun signal d'entrée donné.
Figure2upl (figure, uppaal_description, . . .)
res_verifyta = verifyta (uppaal_description, query_filename);
si ! res_verifyta

```
                    alors   incrémenter le temps setup du signal  s.
                            fin_dec [s] = 1.
                    finsi
                fin_tantque
                afficher l'environnement env pour lequel la specification est satifaite.
        sinon  afficher la specification n'est satifaite pour l'environnement d'entrée.
        fin_si
end
```

On note que la fonction verifyta joue la rôle de la commande de vérification du système uppaal.

2. Tests sur des modèles VHDL:

Dans cette section, on va présenter les résultats de tests déroulés sur quelques modèles en VHDL qu'on a passé sur notre programme. Parmi ceux-ci, on trouve la description LSV1.vhd, présenté dans l'annexe C, associée à l'architecture SPSMALL qui a été déjà analysé dans [CEFX 06a] et [CEFX 06b]. Dans l'annexe B, on présente la description exp1.vhd qui contient quelques instructions d'affectation des signaux et deux processus avec une seul garde. Toutes les autres descriptions testées, jusqu'au présent, sont décrites dans le document 3.

2.2. La description LSV1 :

Le but de cette partie est de comparer les résultats de tests obtenues sur les descriptions générées par le programme par rapport à celles qui sont décrites dans [CEFX.06a & CEFX.06c]. Tout d'abord, on rappelle ci-dessous la représentation structurelle de la description en VHDL LSV1, sous forme d'un graphe fonctionnel et temporel abstrait, associée à une implémentation de l'architecture SPSMALL.

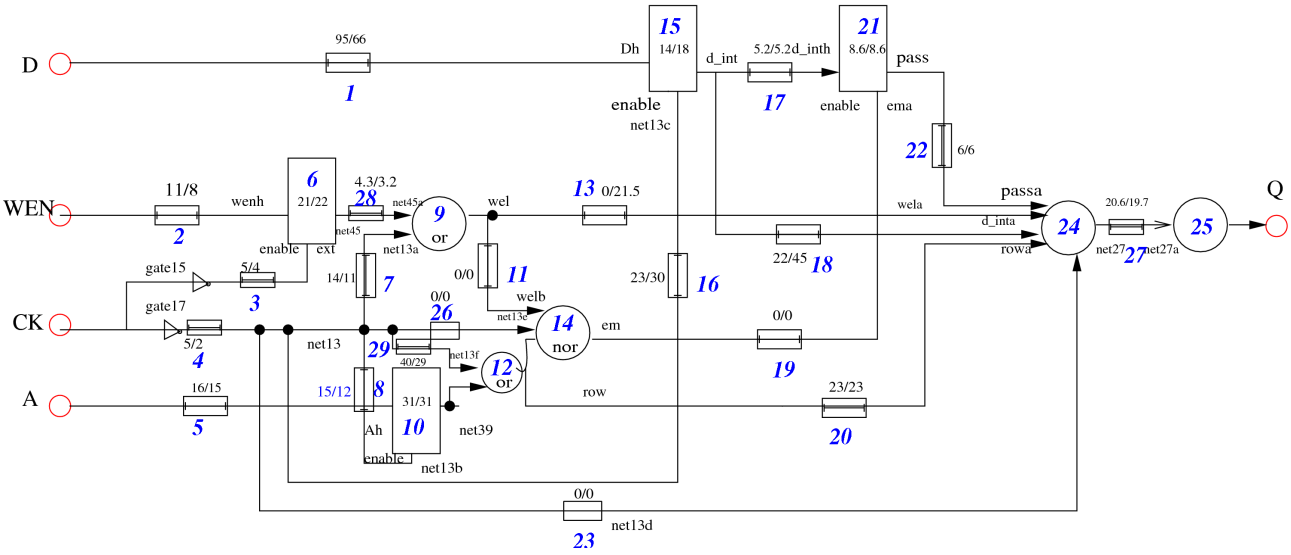


Figure 5: AFTG de SPSMALL (SP1) ([CEFX 06a]).

Ici, comme a été déjà fait dans [CEFX.06a & CEFX.06c], on va faire l'analyse des deux implémentations SP1 et SP2 de l'architecture SPSMALL. On note que ces deux dernières sont identiques au niveau de leur représentation structurelle (leur fonction est identique). En revanche, les délais de propagation de leurs composants peuvent être différents. Ils sont représentés par des graphes fonctionnels et temporels abstraits (AFTG). Comme conséquence, les descriptions en uppaal ou hytech associées à ces deux implémentations sont presque les mêmes. Elles ne diffèrent que dans les valeurs à affecter aux paramètres associées aux délais des signaux internes. La figure mentionnée ci-dessus représente l'implémentation SP1.

Les descriptions en hytech ou en uppaal, associées à ces implémentations SP1 et SP2, générées par la programme contiennent :

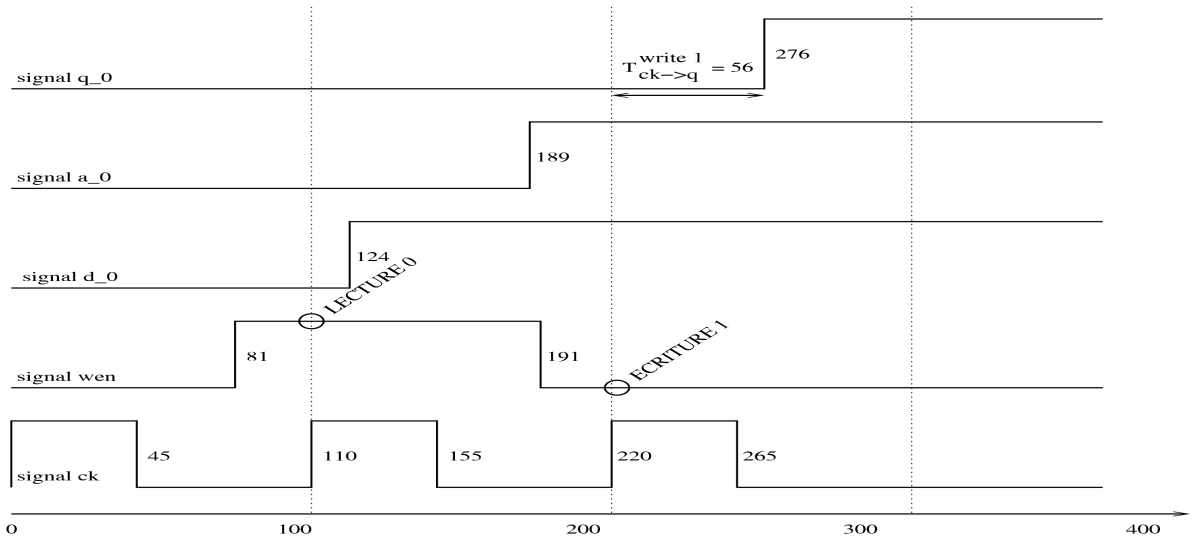
- 1553 lignes dans la description en hytech et 1291 lignes dans la description en uppaal.
- 27 automates (sans compter l'automate d'environnement).
- 28 horloges.
- 32 variables discrètes.
- 62 paramètres.

Evidemment, les descriptions en hytech générées qui contiennent beaucoup d'automates ne passent pas sur l'outil Hytech. En revanche, l'outil Uppaal supporte l'analyse des descriptions de cette taille (Ceci est dû au fait determinisme des automates). En effet, on les avait testé avec l'outil uppaal jusqu'à présent sur trois environnements différents avec lesquels on peut générer des écritures des valeurs 0 et 1 sur la sortie de l'architecture q_0.

Les résultats d'analyse sur ces trois environnements de ces classes sont décrits brièvement sur les chronogrammes suivants. Comme on voit la dessous, on a représenté que les signaux d'entrée ck, wen, d_0, a_0 et le signal de sortie q_0, pour des raisons de lisibilité.

Environnement 1:

- Avec les délais de SP1:



On note que le graphe d'atteignabilité, soit GA, associée à l'automate obtenu par la composition du modèle associé au programme vhdl et l'automate d'environnement satisfait la propriété suivante :

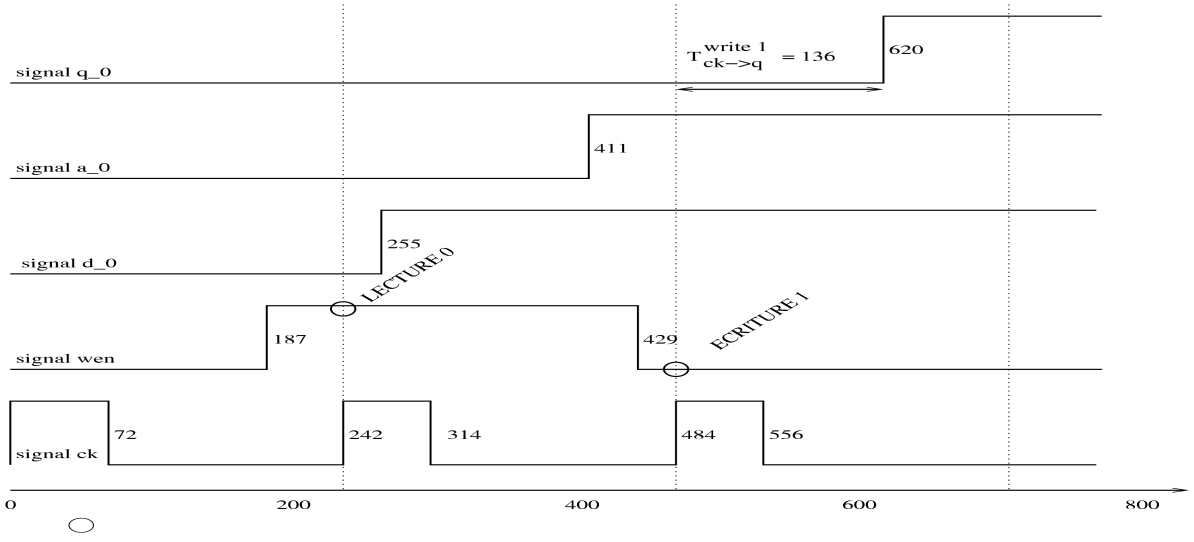
GA |= A[] ((t>156 and t<276) imply q_0 == 0) and (t>276 imply q_0 == 1))

Bien qu'il ne satisfait pas la propriété suivante :

A[] ((t>156 and t<277) imply q_0 == 0) and (t>277 imply q_0 == 1))

Donc, ça correspond bien à ce qui est décrit sur le diagramme.

- Avec les délais de SP2:



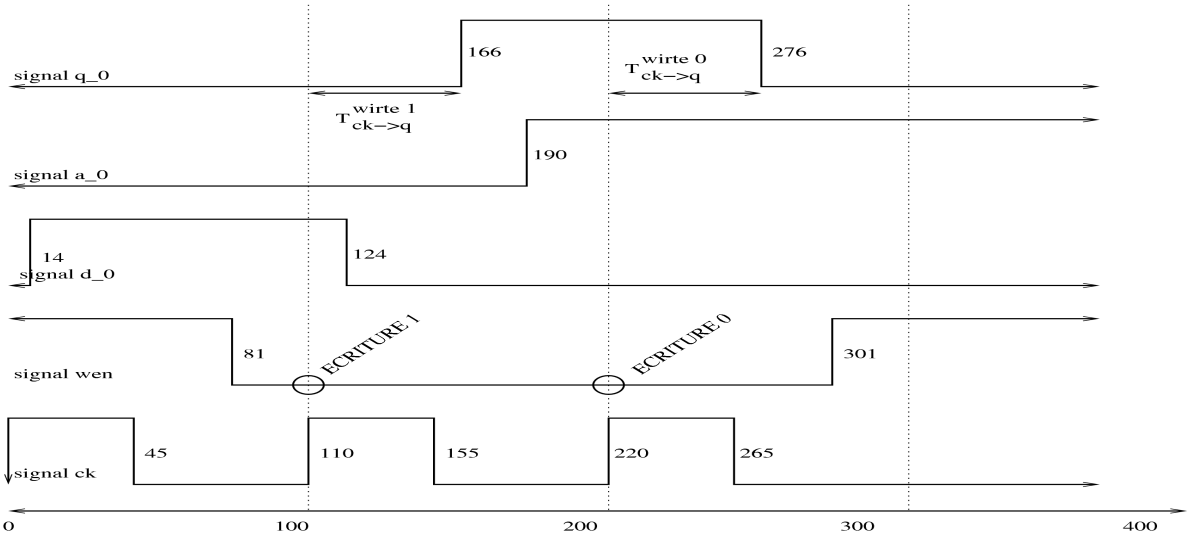
Ici, le graphe d'atteignabilité GA vérifie que :

GA |= A[] ((t>383 and t<620) imply q_0 == 0) and (t>620 imply q_0 == 1))

GA |/= A[] ((t>383 and t<621) imply q_0 == 0) and (t>621 imply q_0 == 1))

Environnement 2:

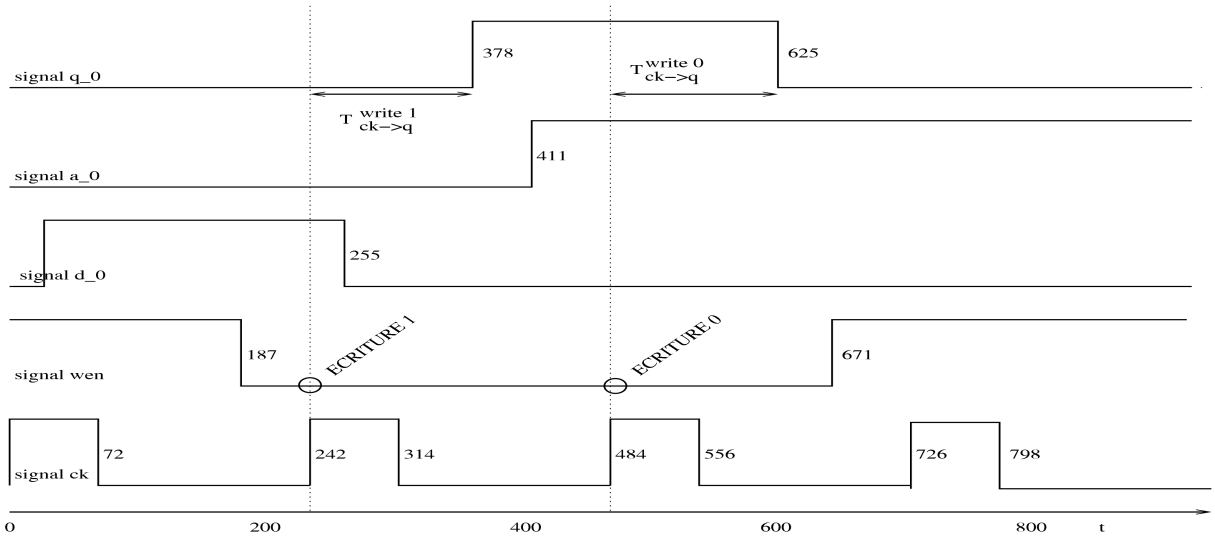
- Avec les délais de SP1:



avec le graphe d'atteignabilité obtenu GA, on a :

GA |= A[] (((t > 275) imply q_0 == 0) and ((t > 166 and t < 275) imply q_0 == 1))
GA |/= A[] (((t > 276) imply q_0 == 0) and ((t > 166 and t < 276) imply q_0 == 1))

- Avec les délais de SP2:

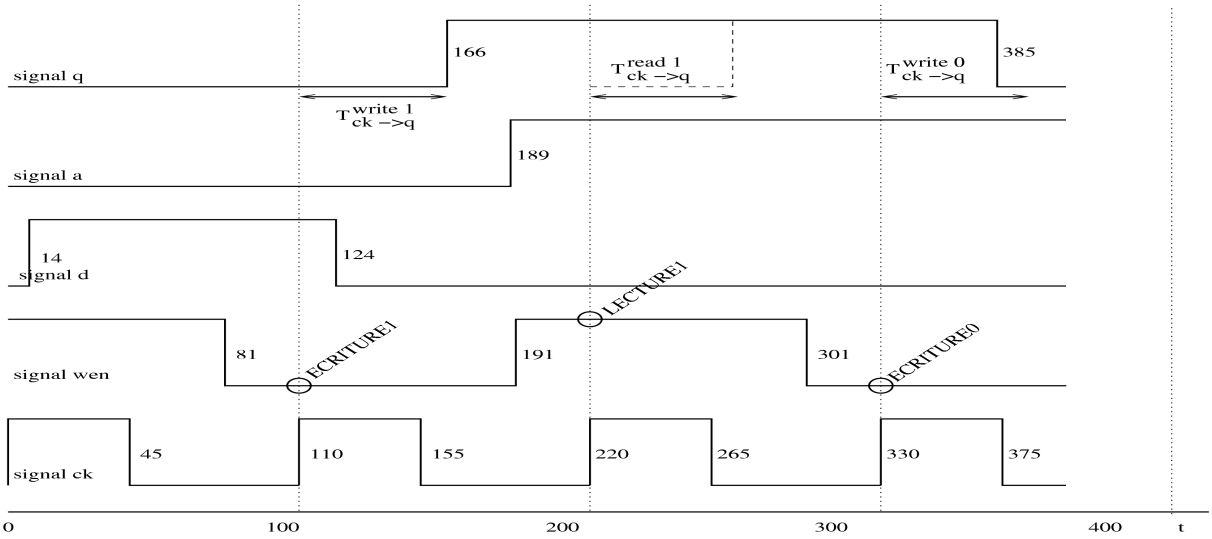


Le graphe d'atteignabilité obtenu GA vérifie ces propriétés:

GA |= A[] (((t > 625) imply q_0 == 0) and ((t > 378 and t < 625) imply q_0 == 1))
GA |/= A[] (((t > 626) imply q_0 == 0) and ((t > 378 and t < 626) imply q_0 == 1))

Environnement 3:

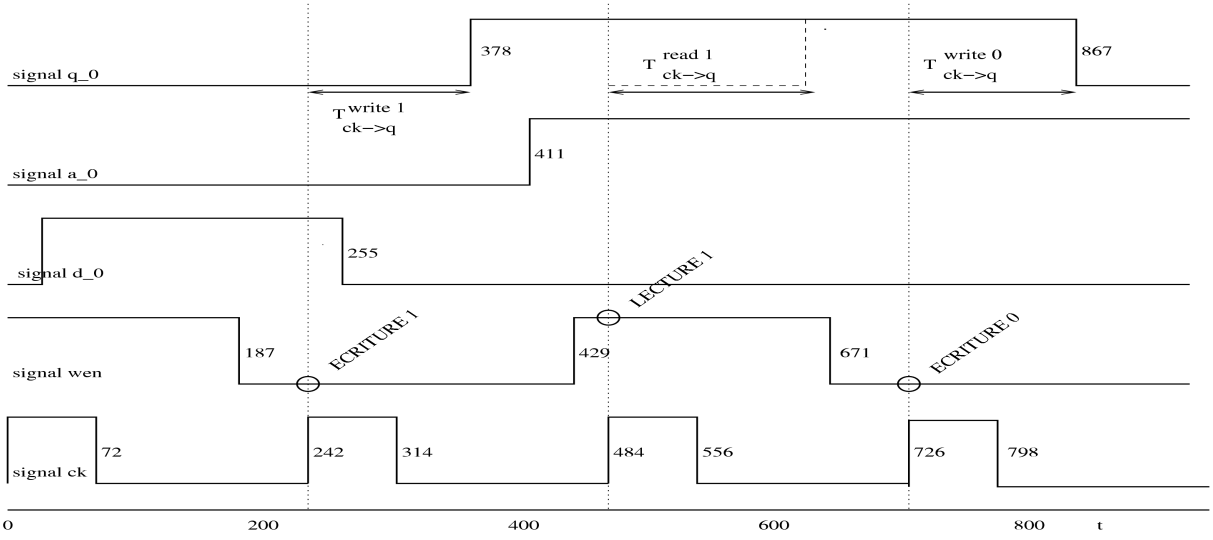
- Avec les délais de SP1:



Le graphe d'atteignabilité obtenu GA vérifie que :

GA |= A[] (((t > 385) imply q_0 == 0) and ((t > 166 and t < 385) imply q_0 == 1))
GA |/= A[] (((t > 386) imply q_0 == 0) and ((t > 166 and t < 386) imply q_0 == 1))

- Avec les délais de SP2:



graphe d'atteignabilité obtenu GA vérifie la première contrainte suivante :

A[] (((t > 867) imply q_0 == 0) and ((t > 378 and t < 867) imply q_0 == 1))

et il ne vérifie pas la contrainte :

A[] (((t > 868) imply q_0 == 0) and ((t > 378 and t < 868) imply q_0 == 1))

Comme on voit les résultats de tests sont bien conformes à ce qu'on a attendu. En ce qui concerne l'implémentation SP1 (l'implémentation SP2), tous les temps de réponse de l'opération d'écriture des valeurs 0 et 1, notée t_write(0) et t_write(1), sont égaux respectivement aux valeurs 56 (136) et 55 (141) qui vérifient la propriété du temps de réponse t_write ≤ t_write_max = 56 (141), où t_write_max dénote le temps maximum nécessaire pour l'opération d'écriture sur la sortie q. Comme on voit sur les chronogrammes, les contraintes ci-dessus sont vérifiées pour les valeurs optimales des temps setup t_setupd = 96 (229), t_setupwen = 29 (55) , t_setupa = 31 (73), associées aux signaux d'entrée D, WEN et A.

Les résultats obtenus et leur comparaison avec celles cités dans [CEFX 06a], sont récapitulés dans les tables mentionnées ci-dessous. On note que l'unité de temps est de 10 ps.

Pour la l'implémentation SP1:

computed response time	value of the datasheet
$t^{read}_{CK\rightarrow Q} = 74$	$t^{read}_{max} == 77$
$t^{write}_{CK\rightarrow Q} = 56$	$t^{write}_{max} == 56$

Table 1.a : Temps de réponse pour SP1, obtenus dans [CEFX 06a].

computed response time
$t^{write}_{CK\rightarrow Q}(0) = 55$
$t^{write}_{CK\rightarrow Q}(1) = 56$

Table 1.b : Temps de réponse pour SP1, obtenus par l'outil.

setup parameter	optimal value obtained by computation	optimal value obtained by simulation	value of the datasheet
t^{D}_{setup}	95	95	108
t^{WEN}_{setup}	29	36	48
t^{A}_{setup}	31	30	58

Table 1.c : Temps de setup optimal pour SP1, obtenus dans [CEFX 06a].

setup parameter	optimal value obtained by the programme
t^{D}_{sctup}	96
t^{WEN}_{setup}	29
t^{A}_{setup}	31

Table 1.d: Temps de réponse pour SP1, obtenus par l'outil.

Pour la l'implémentation SP2:

computed response time	value of the datasheet
$t^{read}_{CK\rightarrow Q} = 169$	$t^{read}_{max} == 169$
$t^{write}_{CK\rightarrow Q} = 142$	$t^{write}_{max} == 142$

Table 2.a : Temps de réponse pour SP2, obtenus dans [CEFX 06a].

computed response time
$t^{write}_{CK\rightarrow Q}(0) = 141$
$t^{write}_{CK\rightarrow Q}(1) = 136$

Table 2.b : Temps de réponse pour SP2, obtenus par l'outil.

setup parameter	optimal value obtained by computation	optimal value obtained by simulation	value of the datasheet
t_{setup}^{D}	229	229	241
t_{setup}^{WEN}	55	55	109
t_{setup}^{A}	73	74	110

Table 2.c : Temps de setup optimal pour SP2, obtenus dans [CEFX 06a].

setup parameter	optimal value obtained by the programme
t_{setup}^{D}	229
t_{setup}^{WEN}	55
t_{setup}^{A}	73

Table 2.d: Temps de réponse pour SP2, obtenus par l'outil.

Il reste encore à tester d'autres environnements qui provoquent des lectures de 0 et 1 sur les points mémoires avec d'autres modèles vhdl, car le modèle lsv.vhd associée à l'architecture SPSMALL ne nous permet de voir bien ces lectures comme il n'a qu'un seul point mémoire sur lequel on effectue des lectures et écritures. Pour cela, on reprend l'architecture présenté dans [CEFX06c] qui est une extension de l'architecture précédente avec deux points mémoires. On nomme le modèle VHDL associé à cette dernière architecture LSV2. Le AFTG associée à l'implémentation SP1 de cette architecture est donnée sur la figure 6.

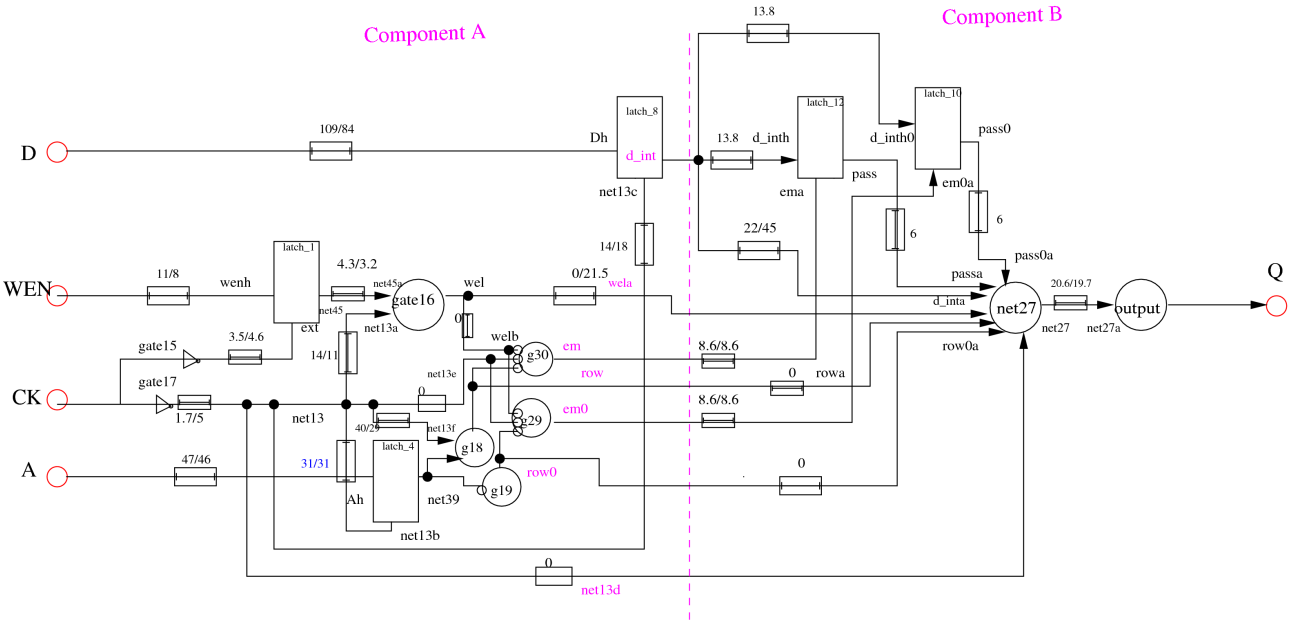


Figure 6: AFTG de l'extenssion de l'architecture SPSMALL avec deux points mémoires ([CEFX 06c]).

2.2. La description LSV2 :

Les descriptions en hytech ou en uppaal, associées à ces implémentations SP1 et SP2, générées par la programme contiennent :

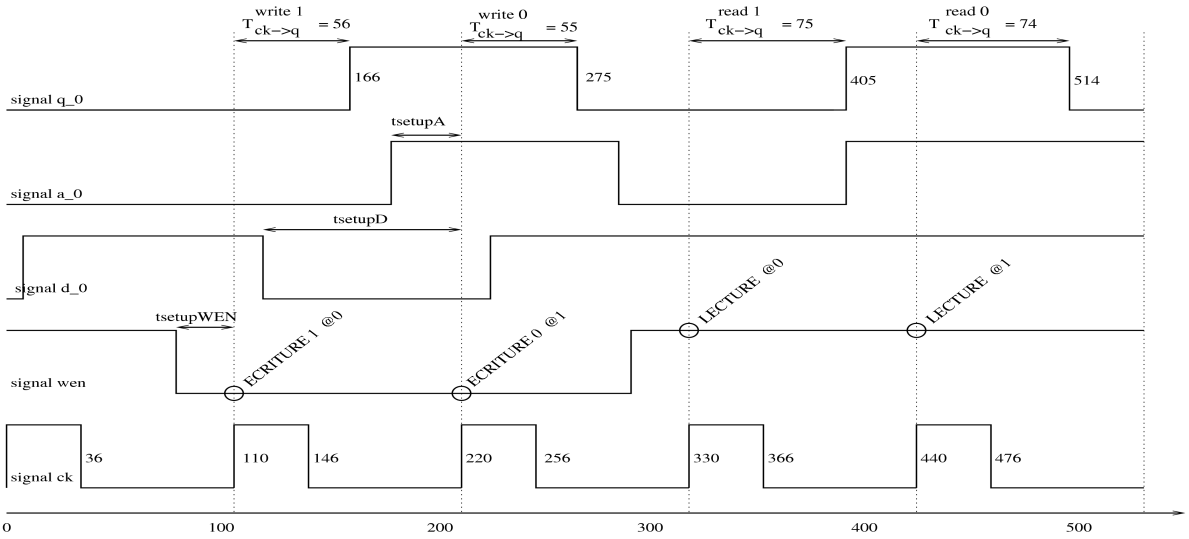
- 2026 lignes dans la description en hytech et 1723 lignes dans la description en uppaal.
- 34 automates (sans compter l'automate d'environnement).
- 35 horloges.

- 35+4 variables discrètes.
- 78 Paramètres.

A partir de la description uppaal générée, on peut tester des environnements d'écriture et lecture des valeurs 0 et 1 sur et à partir des deux points mémoires du modèle VHDL. Jusqu'à présent, on avait testé un seul environnement qui induit une écriture de 1 sur le premier point mémoire et 0 sur le deuxième point, suivies d'une lecture de chaque contenue de ces points, tout en respectant l'ordre de l'écriture, afin de voir bien les valeurs lues sur le sortie q_0. On va prendre en compte les deux implémentations précédents SP1 et SP2.

Environnement 1:

- Implementation SP1:



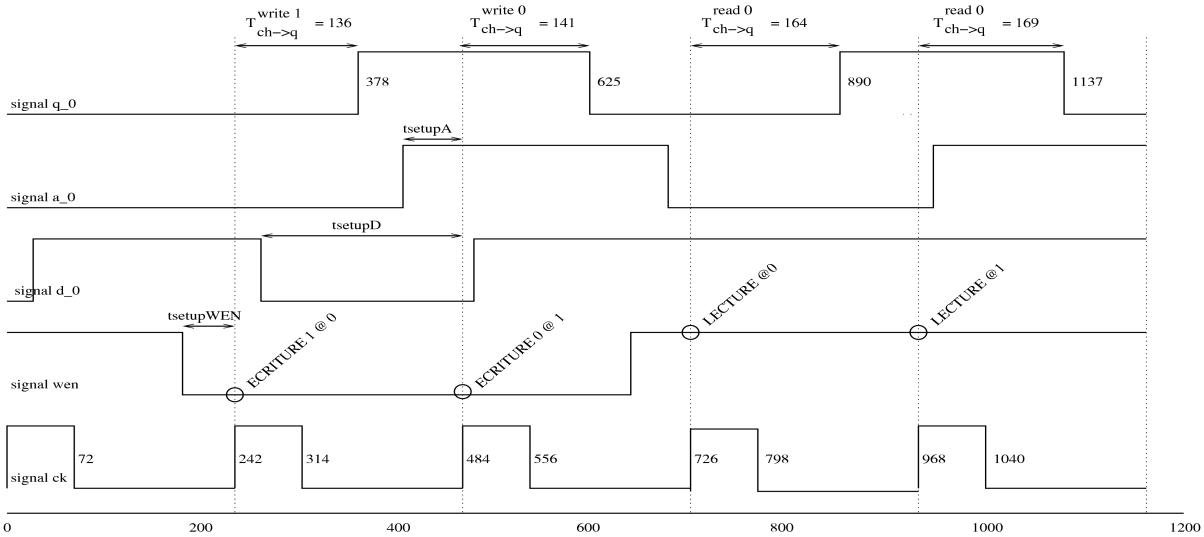
Le graphe d'atteignabilité obtenu GA vérifie la propriété.

```
A[] (       ( ((t > 275 and t < 405) or (t > 514)) imply q_0 == 0)  and

            ( ((t > 166 and t < 275) or (t > 405 and t < 514)) imply q_0 == 1)
)
```

Bien qu'il ne vérifie pas :

```
A[] (       ( ((t > 275 and t < 405) or (t > 515)) imply q_0 == 0)  and
            ( ((t > 166 and t < 275) or (t > 405 and t < 515)) imply q_0 == 1)
)
```

- Implementation SP2:



Le graphe d'atteignabilité obtenu GA vérifie la propriété.

A[] ((((t > 625 and t < 890) or (t > 1137)) imply q_0 == 0) and
(((t > 378 and t < 625) or (t > 890 and t < 1137)) imply q_0 == 1)
)

Bien qu'il ne vérifie pas :

A[] ((((t > 625 and t < 890) or (t > 1138)) imply q_0 == 0) and
(((t > 378 and t < 625) or (t > 890 and t < 1138)) imply q_0 == 1)
)

On peut encore remarquer facilement que les temps de lecture et d'ecritue sur la sortie q_0 correspondent parfaitement à ce qu'on avu dans [CEFX.06a & CEFX.06c]. En considérant l'implémentation SP1 (SP2), on a pour les valeurs du temps de setup : t_setupA ∈ {58, . . . , 34} ({110, . . . , 73}), t_setupD ∈ {108, . . . , 96} ({241, . . . , 229}), t_setupWEN ∈ {48, . . . , 29} ({109, . . . , 55}), les temps de réponses pour les commandes de lecture de 0 t_read0 = 74, de lecture de 1 t_read1 = 75, d'écriture de 0 t_write0 = 55 et d'écriture de 1 t_write1 = 56.

Par contre pour les valeurs des temps t_setupA = 33 (72) ou t_setupWEN = 28 (54), la propriété présenté ci-dessus n'est pas vérifiée sur tous les chemins. Ceci est dû à l'apparition simultané des signaux d'entrée *en_latchA* et *a_h* du latch *latch_A* (*en_latchWEN* et *WEN_h* du *latch_WEN*) qui peut induit deux valeurs différents sur la sortie du latch (soit la sortie prend la nouvelle valeur, ou soit-même). En conséquence, l'apparition des chemins qui ne satisfait la propriété peuvent être apparues. Donc, on doit ajouter toujours une unité de temps dans les sous expressions de l'expression, qui contient des délais des latech sur lequel propage le signal d'entrée.

Les résultats obtenus et leur comparaison avec celles cités dans [CEFX 06a], sont aussi récapitulés dans les tables mentionnées ci-dessous. On rappelle que que l'unité de temps est de 10 ps.

Pour la l'implémentation SP1:

computed response time	value of the datasheet
$t^{read}_{CK\rightarrow Q} = 74$	$t^{read}_{max} == 77$
$t^{write}_{CK\rightarrow Q} = 56$	$t^{write}_{max} == 56$

Table 3.a : Temps de réponse pour SP1, obtenus dans [CEFX 06a].

computed response time	
$t^{write}_{CK\rightarrow Q}(0) = 55$	$t^{read}_{CK\rightarrow Q}(0) = 74$
$t^{write}_{CK\rightarrow Q}(1) = 56$	$t^{read}_{CK\rightarrow Q}(1) = 75$

Table 3.b : Temps de réponse pour SP1, obtenus par l'outil.

setup parameter	optimal value obtained by computation	optimal value obtained by simulation	value of the datasheet
t^{D}_{setup}	95	95	108
t^{WEN}_{setup}	29	36	48
t^{A}_{setup}	31	30	58

Table 3.c : Temps de setup optimal pour SP1, obtenus dans [CEFX 06a].

setup parameter	optimal value obtained by the programme
t^{D}_{setup}	96
t^{WEN}_{setup}	29
t^{A}_{setup}	34

Table 3.d: Temps de réponse pour SP1, obtenus par l'outil.

Pour la l'implémentation SP2:

computed response time	value of the datasheet
$t^{read}_{CK\rightarrow Q} = 169$	$t^{read}_{max} == 169$
$t^{write}_{CK\rightarrow Q} = 142$	$t^{write}_{max} == 142$

Table 4.a : Temps de réponse pour SP2, obtenus dans [CEFX 06a] & [Xu 06].

computed response time	
$t^{write}_{CK\rightarrow Q}(0) = 141$	$t^{read}_{CK\rightarrow Q}(0) = 169$
$t^{write}_{CK\rightarrow Q}(1) = 136$	$t^{read}_{CK\rightarrow Q}(1) = 164$

Table 4.b : Temps de réponse pour SP2, obtenus par l'outil.

setup parameter	optimal value obtained by computation	optimal value obtained by simulation	value of the datasheet
t^{D}_{setup}	229	229	241
t^{WEN}_{setup}	55	55	109
t^{A}_{setup}	73	74	110

Table 4.c : Temps de setup optimal pour SP2, obtenus dans [CEFX 06a] & [Xu 06].

setup parameter	optimal value obtained by the programme
t^{D}_{setup}	229
t^{WEN}_{setup}	55
t^{A}_{setup}	73

Table 4.d: Temps de réponse pour SP2, obtenus par l'outil.

Bibliographie

[CEFX 06a]. R. Chevallier, E. Encrenaz, L. Fribourg, W. Xu, *Timing Analysis of an Embedded Memory: SPSMALL*, WSEAS 10th international conference on circuits, july 2006, Greece.

[CEFX 06b]. R. Chevallier, E. Encrenaz, L. Fribourg, W. Xu*Timed Verification of a generic architecture of a memory circuit using parametric timed automata*, Formal Methods in System Design, vol 34, no 1, pages 59-81.

[CEFX 06c]. R. Chevallier, E. Encrenaz, L. Fribourg, W. Xu, *Dynamic Timing Analysis of an Embeded Memory: SPSMALL*, Rapport interne du projet Blueberries, Juin 2006.

[Xu 06]. W. Xu, *Timing Analysis of SPSMALL*, internal report, june 06

Annexe A

1. La grammaire :

Règles syntaxiques :

```
file            : entity architecture
                ;

entity          : ENTITY VHDLID IS PORT '(' ports ')' ';' END VHDLID ';'
                ;

ports           : port | ports ';'  port
                ;
port            : VHDLID ':' mode BIT
                ;
mode            : IN | OUT
                ;

architecture    : ARCHITECTURE VHDLID OF VHDLID IS def_signals TBEGIN statements END ';'
                ;

def_signals     : def_signal
                | def_signals def_signal
                ;
def_signal      : SIGNAL VHDLID ':' BIT ';'
                ;

statements      : Ɛ | statements signal_assignment_statement | statements process_statement
                ;

signal_assignment_statement   : VHDLID AFFECT expr1
                ;
expr1           : expr |  expr op2 expr
```

```
                ;

expr        : BITVALUE | VHDLID | '(' expr op2 expr ')' | NOT expr  | '(' expr ')'
                ;

expr            : VHDLID EQ BITVALUE
                ;

process_statement :         process_label ':' PROCESS '(' signals_names_list ')'
                            TBEGIN
                                        process_statement_part
                            END PROCESS ';'
                ;

process_statement_part :  if_statement ;

if_statement :              IF if_statement_ END IF ';'
                ;

if_statement_ :             condition THEN signal_assignment_statement
                            ELSIF if_statement_
                |           condition THEN signal_assignment_statement
                |           condition THEN signal_assignment_statement
                            ELSE signal_assignment_statement
                ;

condition     : guard_expr1  ;
guard_expr1 : guard_expr |  guard_expr op2 guard_expr
guard_expr        :  VHDLID EQ BITVALUE
                    |  '(' guard_expr op2 guard_expr ')'
                    |  NOT guard_expr
                    |  '(' guard_expr ')'
                    ;

signals_names_list :        signal_name | signals_names_list ',' signal_name
                ;

process_label :             VHDLID ;
signal_name :               VHDLID ;

op2 :  AND | OR | XOR.
```

Annexe B

2.1. La description exp1:

On décrit tout d'abord ci-dessous la description exp1.vhd.

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

-- Entity Declaration

ENTITY Exp1 IS
  PORT (
        Q_0 : out  BIT;
        CK : in    BIT;
        CSN : in   BIT;
        D_0 : in    BIT
--        vdd : in BIT;
--        gnd : in BIT
  );
END exp1;

-- Architecture Declaration

ARCHITECTURE RTL OF Exp1 IS
  SIGNAL v_18_E_net81 : BIT;
  SIGNAL v_18_E_net85 : BIT;
  SIGNAL v_18_E_net83 : BIT;
  SIGNAL v_18_E_data_delay_H : BIT;
  SIGNAL v_17_12_10_net13 : BIT;
  SIGNAL CLK_H : BIT;
  SIGNAL v_18_E_clk_local_L : BIT;
  --SIGNAL v_18_E_clk_local_H : BIT;
  SIGNAL v_17_12_10_net96 : BIT;
  SIGNAL v_17_12_10_ext_cs_H : BIT;
  SIGNAL v_17_12_10_ext_cs_N : BIT;
  SIGNAL v_18_E_data_delay_H_inv : BIT;
  SIGNAL v_17_12_clk_sig_H : BIT;
  SIGNAL v_17_12_10_net41 : BIT;

BEGIN
  v_18_E_net81 <= not (D_0);
  v_18_E_net85 <= not (v_18_E_net81);
  v_18_E_net83 <= not (v_18_E_net85);
  v_18_E_data_delay_H <= not (v_18_E_net83);

  v_17_12_10_net13 <= ( not (CK) or
                        not (v_17_12_10_ext_cs_N));

  v_17_12_clk_sig_H <= not (v_17_12_10_net13);
  v_17_12_10_net41 <= not (v_17_12_clk_sig_H);
  CLK_H <= not (v_17_12_10_net41);
  v_18_E_clk_local_L <= not (CLK_H);
  --v_18_E_clk_local_H <= not (v_18_E_clk_local_L);

  v_17_12_10_net96 <= not (CSN);
  v_17_12_10_ext_cs_H <= not (v_17_12_10_net96);

  Q_0 <= v_18_E_data_delay_H_inv;

REG10: PROCESS (CK, v_17_12_10_ext_cs_H)
BEGIN
  IF CK = '0' THEN
    v_17_12_10_ext_cs_N <= not (v_17_12_10_ext_cs_H);
  END IF;
END PROCESS;
REG12: PROCESS (v_18_E_clk_local_L,
                v_18_E_data_delay_H)
BEGIN
  IF v_18_E_clk_local_L = '1' THEN
    v_18_E_data_delay_H_inv <= not (v_18_E_data_delay_H);
  END IF;
END PROCESS;
END;
```

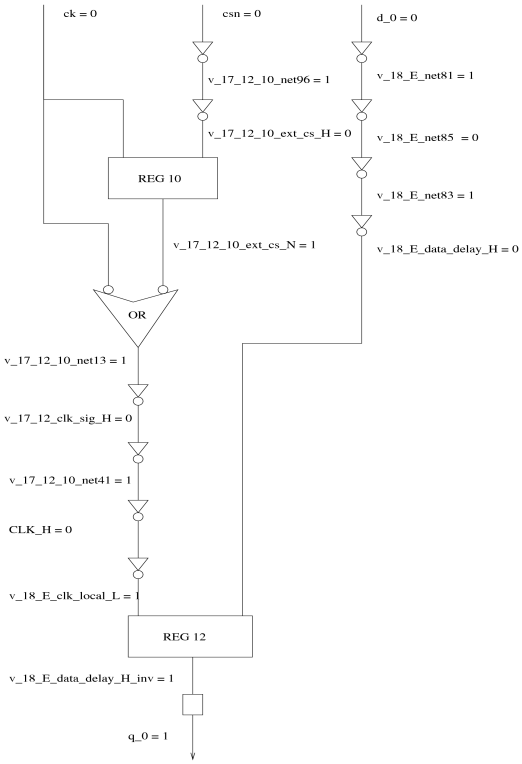
La description en hytech et la description en uppaal, associées à cette description, générées par la programme avec les options d'optimisation contient :

- 447 lignes dans la description en hytech et XX lignes dans la description en uppaal.
- 7 automates (sans compter l'automate d'environnement).
- 8 horloges.
- 10 variables discrètes.
- 18 Paramètres.

Elles sont générées après avoir optimiser la figure générée par l'analyseur. La description en VHDL associée à la figure obtenue après la phase d'optimisation est présentée ci-dessous.

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

-- Entity Declaration

ENTITY Exp1 IS
  PORT (
       Q_0 : out   BIT;
        CK : in    BIT;
       CSN : in    BIT;
       D_0 : in    BIT
         );
END Exp1;

-- Architecture Declaration

ARCHITECTURE RTL OF Exp1 IS
  SIGNAL v_18_E_data_delay_H : BIT;
  SIGNAL v_17_12_10_net13 : BIT;

  SIGNAL v_18_E_clk_local_L : BIT;
  --SIGNAL v_18_E_clk_local_H : BIT;

  SIGNAL v_17_12_10_ext_cs_H : BIT;
  SIGNAL v_17_12_10_ext_cs_N : BIT;

  SIGNAL v_18_E_data_delay_H_inv : BIT;

BEGIN
  v_18_E_data_delay_H <= D_0;

  v_17_12_10_net13 <= ( not (CK) or
                        not (v_17_12_10_ext_cs_N));

  v_18_E_clk_local_L <= v_17_12_10_net13;
  --v_18_E_clk_local_H <= not (v_18_E_clk_local_L);

  v_17_12_10_ext_cs_H <= CSN;

  Q_0 <= v_18_E_data_delay_H_inv;

  REG10: PROCESS (CK, v_17_12_10_ext_cs_H)
  BEGIN
   IF CK = '0' THEN
     v_17_12_10_ext_cs_N <= not (v_17_12_10_ext_cs_H);
   END IF;
  END PROCESS;
  REG12: PROCESS (v_18_E_clk_local_L,
                  v_18_E_data_delay_H)
  BEGIN
   IF v_18_E_clk_local_L = '1' THEN
     v_18_E_data_delay_H_inv <= not (v_18_E_data_delay_H);
   END IF;
  END PROCESS;
  END;
```

Les circuits associés à ces deux descriptions sont décrits ci-dessous.



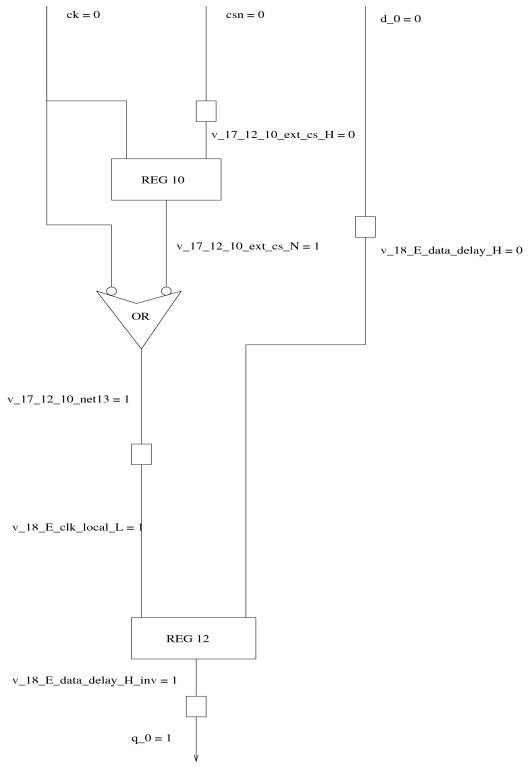


Figure 5.a : le circuit exp1 ***Figure 5.b : l'optimisation du circuit exp1***

On avait testé les deux descriptions obtenues sur 5 environnements différents. Tous les délais des signaux des quatre premiers environnements sont égaux à 1. En revanche, les délais des signaux du dernier environnement sont donnés comme suit:

delta0(reg12) = delta1(reg12) = 18 ,

delta0(reg10) = delta1(reg10) = 10 ,

delta0(v_17_12_10_ext_cs_h) = delta1(v_17_12_10_ext_cs_h) = 7 ,
delta0(v_17_12_10_net96) = delta1(v_17_12_10_net96),

delta0(v_18_e_clk_local_l) = delta1(v_18_e_clk_local_l) = 7 ,
delta0(clk_h) = delta1(clk_h) = 6 ,
delta0(v_17_12_10_net41) = delta1(v_17_12_10_net41),
delta0(v_17_12_clk_sig_h) = delta1(v_17_12_clk_sig_h),

delta0(v_17_12_10_net13) = delta1(v_17_12_10_net13),

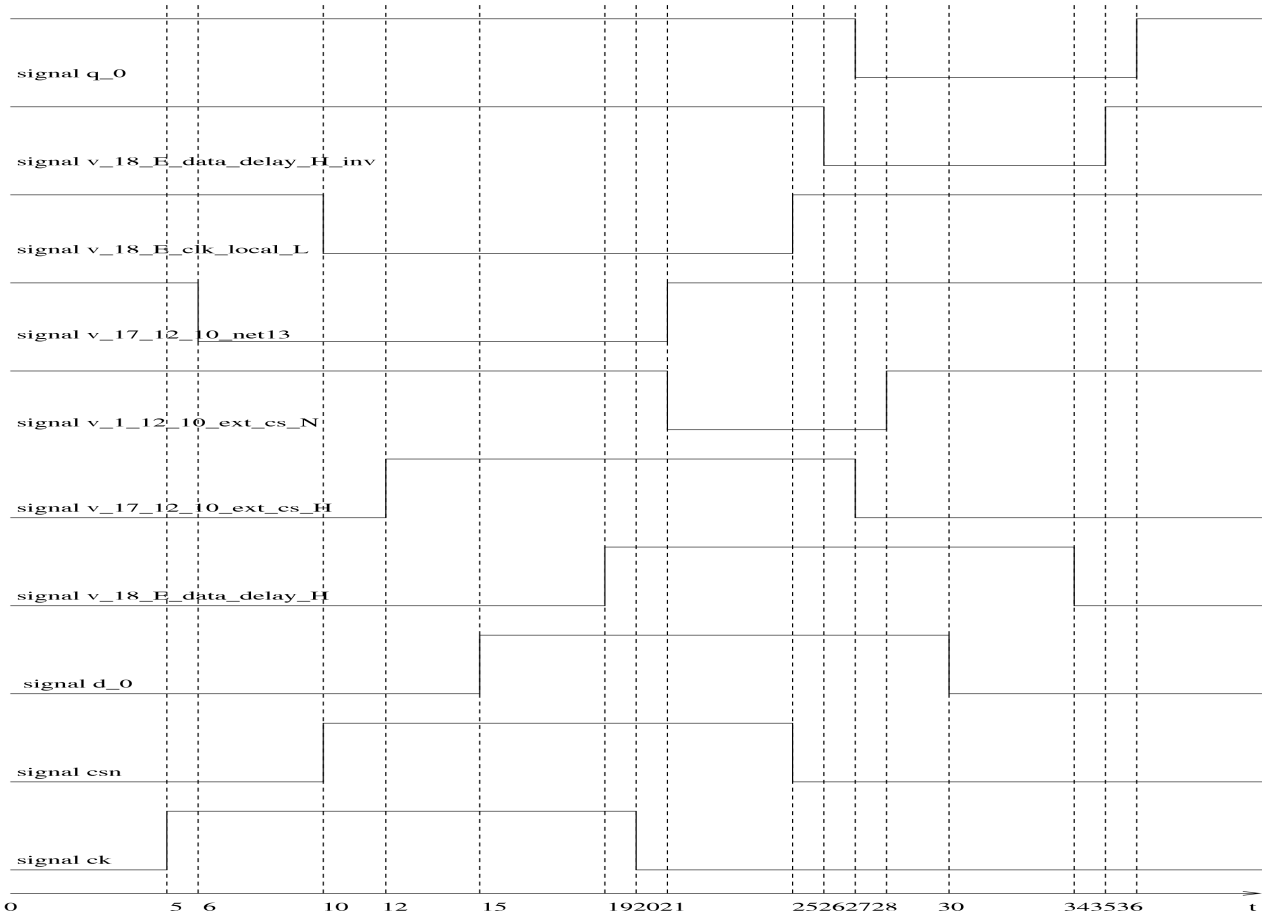
delta0(v_18_e_data_delay_h) = delta1(v_18_e_data_delay_h) = 28 ,
delta0(v_18_e_net83) = delta1(v_18_e_net83),
delta0(v_18_e_net85) = delta1(v_18_e_net85) = 25 ,
delta0(v_18_e_net81) = delta1(v_18_e_net81) = 25 ,

delta(q_0) = delta1(q_0) = 12;

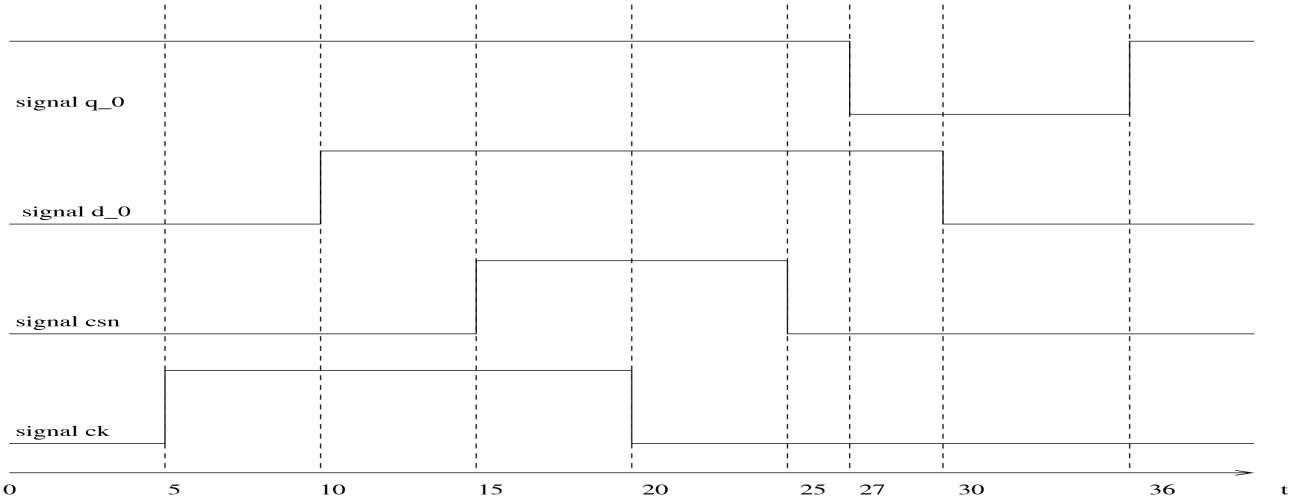
Les résultats de d'analyse sont donnés par les diagrammes présentés ci-dessous dans lesquels on a omis l'apparition des signaux auxiliaires dans les quatre derniers diagrammes. Comme on voit, les valeurs du signal de sortie q_0 correspondent parfaitement à ce qu'on a attendu. On l'avait prouvé lors la phase de la vérification de avec hytech et uppaal.

On note aussi que la propriété de stabilité des valeurs des signaux de sortie et intermédiaires du circuit, est vérifiée sur le graphe d'atteignabilité dans les environnements. On rappelle que les détails de tests sont décrits dans la document 3.

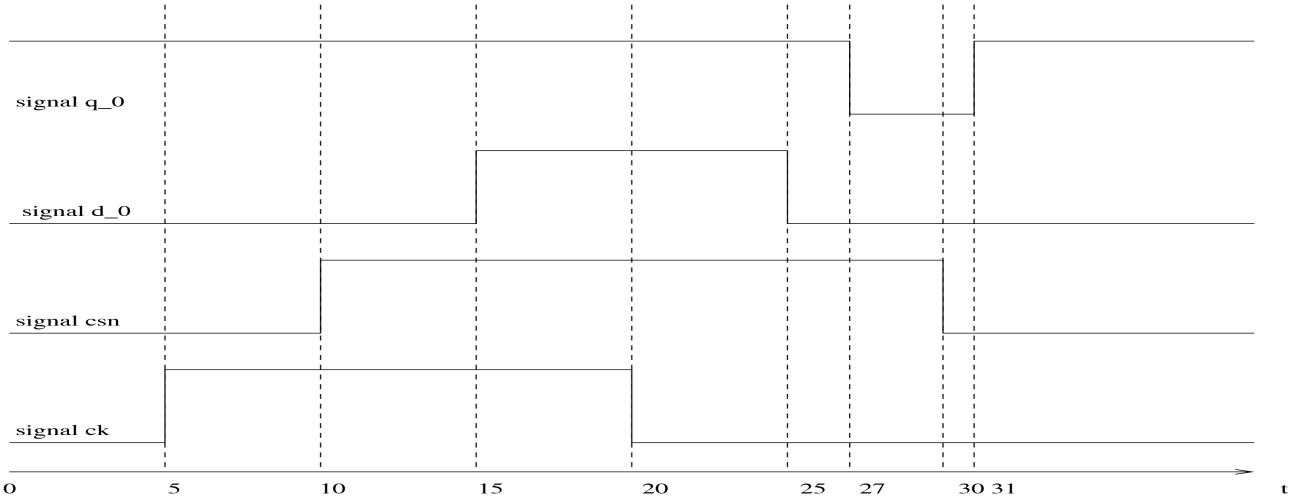
Environnement 1:



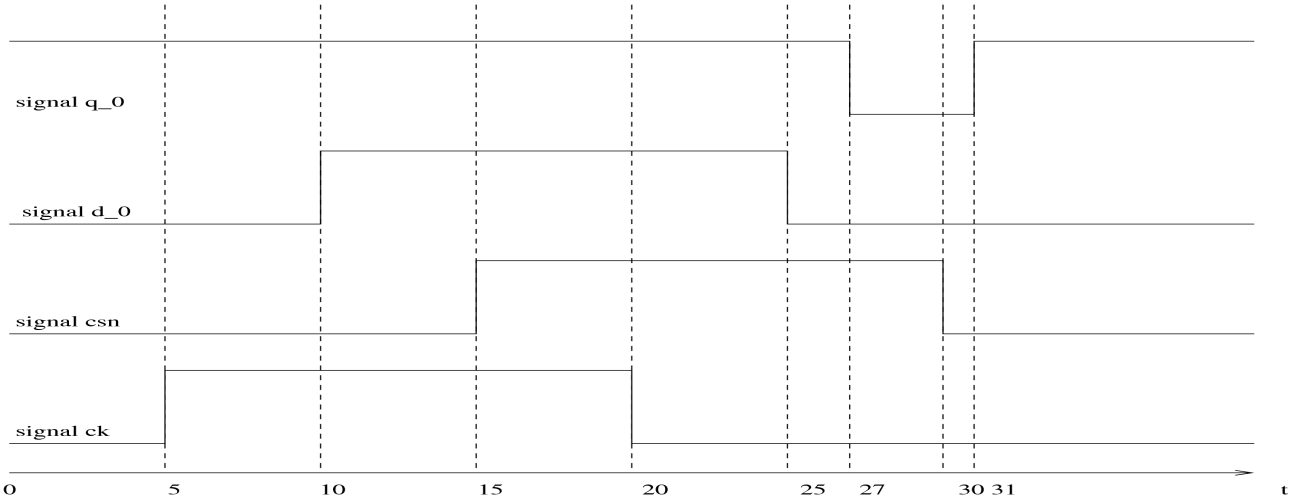
Environnement 2:



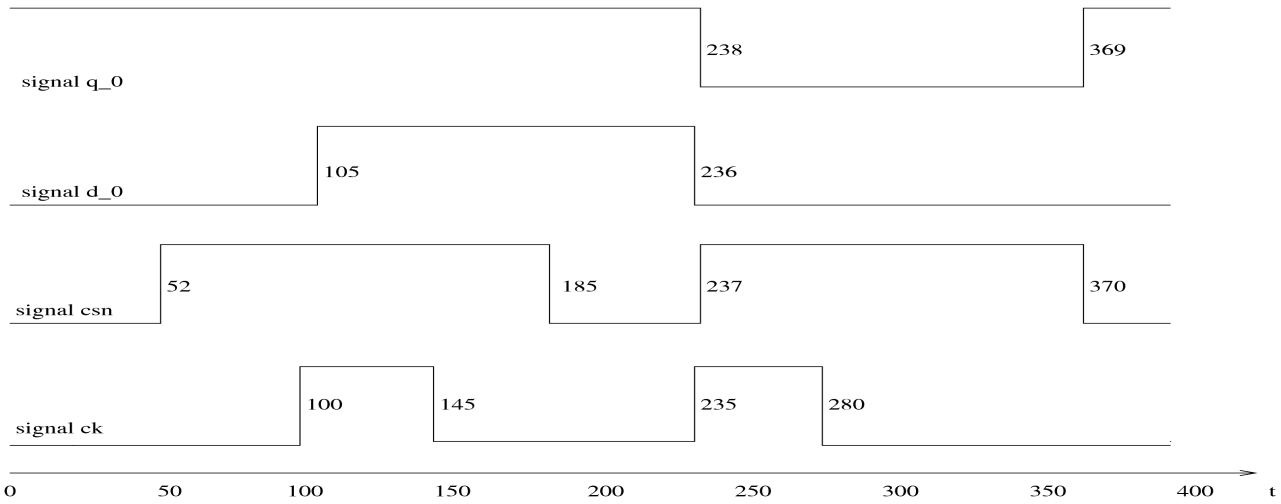
Environnement 3:



Environnement 4:



Environnement 5:



Annexe C

2. La description en VHDL LSV1 :

```
-- description SPSMALL selon BLUEBERRIES
-- modifications pour incorporer les délais dans les portes
--              - net13 supprime : gate 17 repliquee et delai de fils sur sortie incorpores
--              - wel supprime : gate 9 repliquee et delai de fil de sortie incorpores
--              - delai 19 (fil de sortie) incorpore dans gate 14
--              - delai 20 incorpore dans gate 12
--              - delai 3 incorpore dans gate 15

ENTITY SPSMALL_BLUEBERRIES IS
  PORT (
        Q_0 : out   BIT;
         CK : in    BIT;
        WEN : in    BIT;
        A_0 : in    BIT;
        D_0 : in    BIT
  );
END SPSMALL_BLUEBERRIES;

-- Architecture Declaration

ARCHITECTURE RTL OF SPSMALL_BLUEBERRIES IS
  SIGNAL D_h : BIT;                             -- entree data latchD
  SIGNAL WEN_h : BIT;                           -- entree data latchWEN
  SIGNAL A_h : BIT;                             -- entree data latchA
-- SIGNAL net13 : BIT;                          -- le fameux signal net13
  SIGNAL en_latchWEN : bit;                     -- enable de latchWEN, nommé enable_ext
  SIGNAL en_latchA : bit;               -- enable de latchA, aussi nommé net13b
  SIGNAL en_latchD : bit;                       -- enable de latchD, aussi nommé net13c
  SIGNAL D_int : bit;                           -- sortie latchD
  SIGNAL net45 : bit;                           -- sortie latchWEN
  SIGNAL net39 : bit;                           -- sortie latchA
  SIGNAL net13a : bit;                          -- retard net13
  SIGNAL net13d : bit;                          -- retard net13
  SIGNAL net13e : bit;                          -- retard net13
  SIGNAL net13f : bit;                          -- retard net13
  SIGNAL net45a : bit;                          -- retard net45
--  SIGNAL wel : bit;                           -- sortie gate 9
  SIGNAL wela : bit;                            -- retard wel
  SIGNAL welb : bit;                            -- retard wel
```

```
  SIGNAL row : bit;                          -- sortie gate 12
--  SIGNAL em : bit;                         -- sortie gate 14
  SIGNAL ema : bit;                          -- sortie gate 14
  SIGNAL D_inta : bit;                       -- retard D_int
  SIGNAL D_int_h : bit;                      -- retard D_int
  SIGNAL rowa : bit;                         -- retard row
  SIGNAL pass : bit;                         -- sortie point memoire
  SIGNAL passa : bit;                        -- retard pass
  SIGNAL net27 : bit;                        -- retard pass
BEGIN

  D_h <= D_0;                                        -- chaine d'inverseurs : delai 1
  A_h <= A_0;                                        -- chaine d'inverseurs : delai 5
  WEN_h <= WEN;                                      -- chaine d'inverseurs : delai 2

  -- net13 <= not CK;                        -- gate 17 / delai 4
  en_latchWEN <= not CK;                     -- gate 15 / delai 3
  en_latchA <= not CK;              -- delai 4 (not 17) +  delai 8 (retard)
  en_latchD <= not CK;                       -- delai 4 (not 17) +  delai 16 (retard)

  net13a <= not CK;                          -- delai 4 (not 17) +  delai 7 (retard)
  net13d <= not CK;                          -- delai 4 (not 17) +  delai 23 (retard)
  net13e <= not CK;                          -- delai 4 (not 17) +  delai 26
  net13f <= not CK;                          -- delai 4 (not 17) +  delai 29

  net45a <= net45;                           -- retard / net45a / delai 28

  -- wel <= net45a or net13a;                -- gate 9 / PAS DE DELAI
  wela <= net45a or net13a;                  -- delai nul (or 9) + delai 13
  welb <= net45a or net13a;                  -- delai nul (or 9) + delai 11

   row <= net13f or net39;                   -- gate 12 / PAS DE DELAI

  -- em <= not (welb or net13e or row); -- gate 14 / PAS DE DELAI

  D_inta <= D_int;                           -- retard / D_inta / delai 18
  rowa <= net13f or net39;                   -- delai nul (or 12) +  delai 20
  ema <= not ((welb or net13e) or row);      -- delai nul (nor 14) +  delai 19
  passa <= pass;                             -- retard / passa / delai 22
  D_int_h <= D_int;                          -- retard / D_int_h / delai 17

  Q_0 <= net27;                                      -- retard / net27a / delai 27

-- les process
```

```
 REG_latchD: PROCESS (en_latchD, D_h)          -- latchD / delai 15
             begin
             if en_latchD = '1' then
               D_int <= D_h;
             end if;
             end process;

 REG_latchWEN: PROCESS (en_latchWEN,WEN_h)-- latchWEN / delai 6
             begin
             if en_latchWEN = '1' then
               net45 <= WEN_h;
             end if;
             end process;

 REG_latch_A: PROCESS (en_latchA,A_h)          -- latchA / delai 10
             begin
             if en_latchA = '1' then
               net39 <= A_h;
             end if;
             end process;

 REG_mem_point: PROCESS (ema, D_int_h) -- point memoire / delai 21
             begin
             if ema = '1' then
              pass <= D_int_h;
             end if;
             end process;

 REG_mux_output: PROCESS (passa, D_inta, wela, rowa, net13d)
             begin                                  -- mux et buffer de sortie / PAS DE DELAI
             if net13d = '0' and wela = '0' then
               net27 <= D_inta;
             elsif ((net13d = '0' and wela = '1') and rowa = '0') then
               net27 <= passa;
             end if;
             end process;
END;
```