LE GRAFCET SYNCHRONE
POUR LA CONCEPTION DE CIRCUITS

Maurice DIAMANTINI[1], **Jean-Luc SCHARBARG**[2], **Lionel MARCÉ**[2]

[1] *École Nationale Supérieure de Techniques Avancées*
32 Bd Victor 75739 Paris Cedex 15, France
email: `diam@ensta.fr` *- tél: 01 45 52 54 91 - fax: 01 45 52 55 87*

[2] *Université de Bretagne Occidentale, Département d'Informatique*
6 Avenue V. Le Gorgeu - B.P. 809, 29285 Brest Cedex, France
email: `{scharbar,marce}@univ-brest.fr` *- tél: 02 98 01 69 57 - fax: 02 98 01 62 52*

RÉSUMÉ - Cet article présente une utilisation du modèle graphique *Grafcet* pour spécifier des séquenceurs en électronique synchrone, et une méthode pour synthétiser les grafcets obtenus dans des circuits FPGA de type *Xilinx*. Nous montrons par un exemple que le *Grafcet* peut être vu comme moyen de réaliser un encodeur spécialisé de machine d'états, plus pertinent du point de vue de l'utilisateur que l'encodage binaire ou one-hot. La seconde partie montre comment le langage synchrone *Signal* peut être utilisé entre le *Grafcet* et le circuit de façon à disposer de moyens de vérifications de propriétés.

ABSTRACT - *In this paper, we show how the graphical language* Grafcet *can be used for the specification of synchronous sequencers. We also give a method to synthetize resulting grafcets using reconfigurable circuits (for instance* Xilinx *components). We show on an example how* Grafcet *can be seen as a specialised encoder for states machines and why this encoder is more suitable than a binary or one-hot encoder, as far as users are concerned. The second part of the paper shows how the* Signal *synchronous language can be used between the grafcet description and the circuit, in order to be able to verify properties.*

1 Introduction

Un circuit intégré peut être découpé en deux parties, une partie contrôle et une partie calcul. La partie contrôle, moins consommatrice de ressources, est néanmoins la plus cruciale, dans la mesure où une erreur de conception peut avoir des conséquences catastrophiques (un non-fonctionnement total, une sous-exploitation de la partie flot de données ou un dysfonctionnement sporadique). Sa conception doit donc mettre en œuvre des modèles de haut-niveau et disposer de moyens de vérification.

Les modèles couramment utilisés actuellement sont les machines d'états [Ber87], en particulier les machines de Moore pour l'électronique synchrone (voir figure 1). Avec ces machines, les sorties du circuit modélisé sont déterminées à partir de l'état interne courant, lequel dépend de l'état précédent et des entrées du circuit. Le graphe d'états qui résulte de cette modélisation permet difficilement une association claire entre un état de la machine et une sortie. En fait, les machines d'états sont plutôt adaptées à la description d'une solution qu'à la spécification du comportement d'un séquenceur. Ainsi, une modification, même mineure, de la spécification du circuit entraîne le plus souvent une reconception complète de la machine d'états. D'autres modèles doivent donc être envisagés pour pallier à ces inconvénients.

Le *Grafcet*[1] est un modèle graphique très utilisé pour la spécification de la commande de processus

1. Nous utiliserons la notation de *Grafcet* pour parler du modèle, et nous écrirons un grafcet lorsqu'il s'agit d'une instance ou d'un schéma particulier de ce modèle.

industriels. Nous nous proposons de montrer comment il peut être utilisé pour spécifier des séquenceurs en électronique synchrone.

La première partie présente une méthode d'implantation de séquenceurs synchrones à partir d'une description grafcet. Cette méthode est illustrée par un exemple. La seconde partie montre comment l'utilisation d'un langage formel entre le *Grafcet* et le circuit permet de disposer de moyens de vérifications de propriétés.

2 Du *Grafcet* à l'architecture

Après une présentation succincte du *Grafcet*, nous montrons comment il peut modéliser des séquenceurs synchrones, puis nous proposons une implantation du *Grafcet* au moyen de logique reconfigurable via le langage VHDL.

2.1 Le modèle *Grafcet*

Le *Grafcet* [AA79] est un modèle graphique normalisé [IEC88, UTE93]. Son domaine d'applications privilégié est la commande de processus industriels. Il est déterministe et permet d'exprimer le parallélisme. Le modèle présenté ci-dessous n'est qu'un sous-ensemble du modèle *Grafcet*. Toutefois, ce sous-ensemble est suffisant dans le cadre des séquenceurs synchrones.

Un *Grafcet* (figure 2, partie droite) est un graphe biparti orienté comprenant des étapes et des transitions. Son comportement est défini par cinq règles, deux postulats et un algorithme d'interprétation.

Les étapes sont matérialisées par des carrés. À tout instant, une étape est active ou inactive. L'ensemble des étapes actives définit l'état (ou situation) du grafcet. Les étapes actives au départ (situation initiale) sont représentées par un double carré. Les changements de situation sont contrôlés par des transitions (trait épais horizontal) auxquelles sont associées des réceptivités (conditions booléennes dans lesquelles peuvent intervenir les activités d'étape X_i). *Toute* transition est franchie *dès* l'instant où sa réceptivité est vraie et où toutes ses étapes amont sont actives. Le franchissement d'une transition entraîne *instantanément* la désactivation des étapes amont *et* l'activation des étapes aval. Les sorties du système sont modélisées par des actions associées aux étapes. Une action est émise tant que l'étape à laquelle elle est associée est active. Lorsque plusieurs étapes actionnent une même sortie, le signal de sortie résulte du OU logique des actions associées.

2.2 *Grafcet* ou *Réseaux de Petri*

Par rapport aux *Réseaux de Petri Interprétés* [Moa85, DA91] desquels il est inspiré, le *Grafcet* est spécialement conçu pour spécifier des machines séquencielles. Comme eux, un grafcet forme un graphe biparti orienté, et permet d'exprimer le parallélisme. Cependant deux différences fondamentales existent :

1. dans le *Grafcet*, l'activité d'une étape est booléenne, tandis que le marquage d'une place de *Réseaux de Petri* est un entier. Le comportement qui en résulte est surtout différent lors de la réactivation d'une étape ;

2. le franchissement de toutes les transitions franchissables est simultané dans le *Grafcet*, alors qu'il s'effectue en séquence dans un *Réseau de Petri* : il s'ensuit un risque d'indéterminisme lorsqu'il y a conflit entre deux transitions validées par une même place.

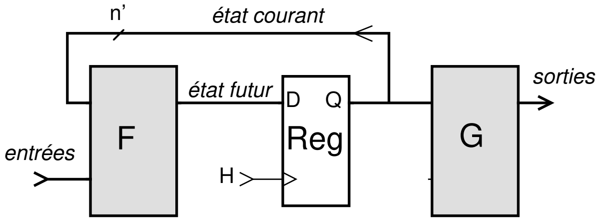
On peut toujours restreindre l'utilisation des modèles *Grafcet* et *Réseaux de Petri Interprétés* de façon à les rendre trés proches. Cependant, cette intersection appelé *Réseaux de Petri Interprétés de commande* ou *Grafcet sain* [DA91] suivant le point de vue, nous paraît être un nivellemment par le bas de la puissance des modèles respectifs.

D'une manière générale, de par sa vocation industrielle, le *Grafcet* privilégie les aspects de simplicité, en particulier au niveau des marques et de la représentation ; les conflits des *Réseaux de Petri* sont supprimés. En revanche, les *Réseaux de Petri* conservent un aspect très formel et rigoureux, fruit des différents travaux de recherche entrepris dans ce domaine.

2.3 L'exemple support

L'exemple support (`aminc_g7`) est illustré figure 2. Il décrit la partie contrôle d'un petit module (`aminci_bus`) dont le but est de transférer un flot de données d'un grand bloc de $k.N$ bits vers un petit bloc de N bits[2]. Les flux de données sont gérés par un protocole de dialogue asynchrone[3] du style "poignée de main" à l'aide des signaux Entrée Prête (`EP`) et Entrée Reçue (`ER`) pour le grand bloc, ainsi que Sortie Prête (`SP`) et sortie Reçue (`SR`) pour le

2. Par exemple pour transférer des informations présentées sur 64 bits vers un bus de 16 bits en quatre fois.
3. C'est le protocole qui est asynchrone, et non pas les signaux qui sont tous synchrones vis à vis de l'horloge `H`.

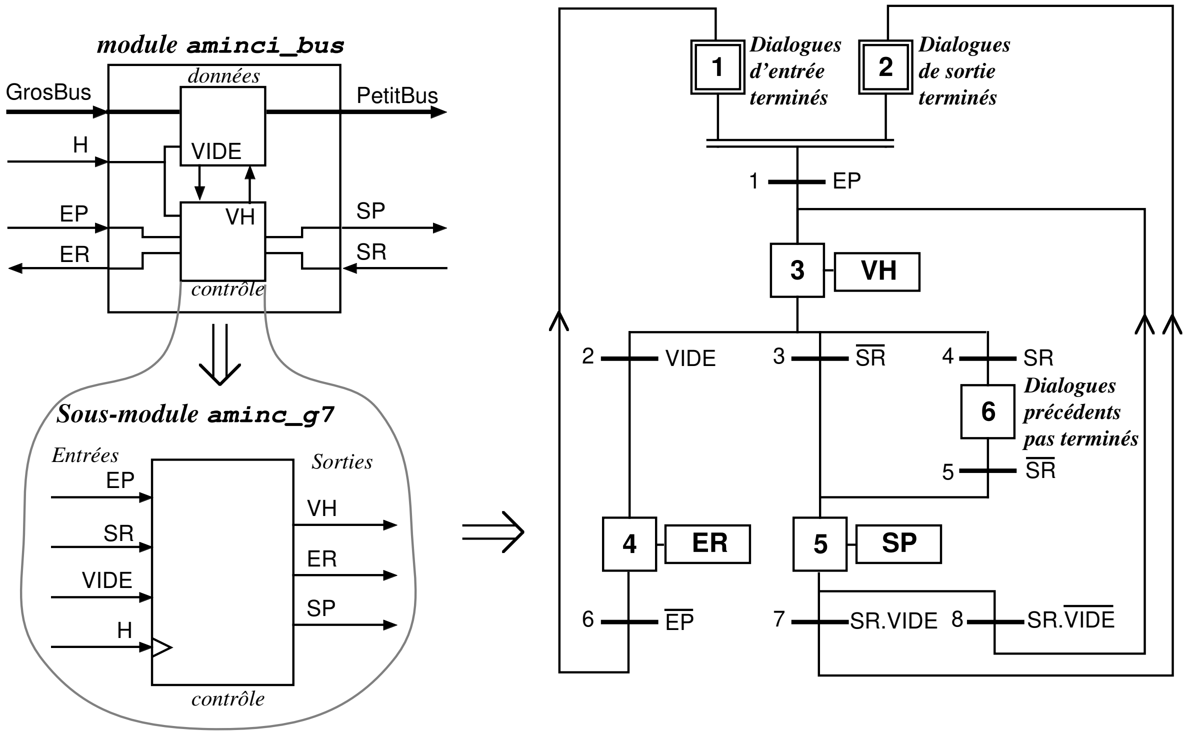


F : *fonction combinatoire définissant l'évolution des états,*

Reg : *registre de n' bascules mémorisant les n états avec $n = n'$ pour un* encodage one-hot *et $n = \lceil \log_2 N \rceil$ pour un* encodage binaire,

G : *fonction combinatoire calculant les sorties à partir de l'état interne.*

FIG. 1 – *Machine de* Moore *synchrone de n états et n' bascules*



Le franchissement de toute transition est implicitement conditionné par la présence de l'horloge $\uparrow H$.

Après initialisation, lorsque le signal `EP` *passe à 1, l'étape 3 est activée et la sortie* `VH` *est émise. L'étape 3 n'est active que pendant une période d'horloge car quelle que soit la valeur de l'entrée* `SR`*, au moins une des deux transitions 3 ou 4 sera franchie. Si* `SR` *était à 0 alors T_3 est franchie et* `SP` *s'active, sinon T_4 est franchie et l'étape 6 assure l'attente du passage à 0 de* `SR`*. Lorsque l'étape 5 devient active, la sortie* `SP` *reste émise jusqu'à ce que l'accusé-reception* `SR` *soit à 1. En fonction de l'entrée* `VIDE`*, soit* `VH` *est de nouveau émis, soit le cycle des dialogues de sortie est terminé. Parallèlement à ce cycle, si* `VIDE` *est à 1 lorsque l'impulsion* `VH` *est émise (ce qui est le cas après l'initialisation), le signal* `ER` *est émis tant que l'entrée* `EP` *n'est pas repassée à 0. Ce n'est que quand 1 et 2 sont actives (retour à l'état initial) qu'un nouveau cycle complet peut redémarrer (sur occurrence de* `EP`*).*

FIG. 2 – *Exemple de description d'un séquenceur par* Grafcet

petit bloc. Ce module contient un compteur initialisé à 0 (positionnant le signal `VIDE` à 1) indiquant qu'il est prêt à recevoir un gros bloc en entrée.

Le sous-module considéré (`aminc_g7`) comporte ainsi trois entrées et trois sorties (horloge non comprise). Sa fonction est de gérer les signaux de dialogues et de commander l'extraction de petits blocs par positionnement de l'impulsion `VH` (Validation d'Horloge).

Cet exemple illustre l'utilisation d'un conflit au sens des *Réseaux de Petri*. En effet, lorsque l'étape 3 est active, et si le signal `VIDE` est à 0 (ce qui est le cas après une initialisation), alors la transition 2 est franchissable en parallèle avec l'une des transitions 3 ou 4 (suivant l'entrée `SR`). On parle de parallélisme interprété (ou implicite) par opposition au parallélisme structurel dans lequel la distribution parallèle de l'activité d'une étape est explicitement représentée à l'aide de la double barre horizontale.

2.4 Implantation du *Grafcet*

Chaque étape du grafcet est implantée par une bascule RS synchrone avec mise à 1 prioritaire. Dans les circuits FPGA utilisés, cette bascule RS est réalisée par une bascule D, avec $D_i = S_i + X_i.\overline{R_i}$. D'autre part, les bascules sont initialisées à zéro à la mise sous tension. Une étape initiale est donc réalisée en utilisant une bascule D "inversée". Le schéma des deux types d'étapes est présenté figure 3.

Pour chaque transition, un signal de franchissement est calculé à partir du décodage de ses étapes amont, conditionné par la réceptivité associée à cette transition. Le signal ainsi obtenu valide la mise à 0 des étapes amont et la mise à 1 des étapes aval.

Par exemple pour l'étape 3 de la figure 2, les entrées S_3 de mise à 1 ou R_3 de mise à 0 de cette bascule sont :

$$\left\{ \begin{array}{lcl} S_3 & = & X_1.X_2.EP + X_5.SR.\overline{VIDE} \\ R_3 & = & X_3.VIDE + X_3.\overline{SR} + X_3.SR \\ & = & X_3 \quad \text{(dans ce cas particulier)} \end{array} \right.$$

Dans ce cas, l'entrée D_3 de la bascule D correspondante se réduit à :

$$D_3 \; = \; X_1.X_2.EP + X_5.SR.\overline{VIDE}$$

Notre démarche d'implantation consiste alors à traduire le grafcet obtenu en programme VHDL que les outils de CAO habituels peuvent simuler ou synthétiser. Cette opération actuellement manuelle est en voie d'automatisation.

2.5 Expérimentation

Le *Grafcet* a été utilisé depuis 1992 [Dia94] pour implanter plusieurs séquenceurs dans des circuits FPGA de la société *Xilinx*.

Les circuits utilisés sont du type XC4010-6 contenant 400 CLB (Bloc Logique Configurable) [Xil94]. Un CLB dispose (entre autres) de deux bascules D pour les registres, et de deux générateurs de fonctions à 4 entrées pour l'implantation de la logique combinatoire.

La figure 4 présente un extrait de la description VHDL structurelle synthétisable du grafcet illustré figure 2. La description des composants VHDL `etape_initiale` et `etape_normale` ne pose pas de problèmes particulier. Ce programme synthétisé occupe 4 CLB répartis en une couche.

2.6 Résultats

Le ***taux d'occupation*** d'un grafcet est pratiquement linéaire en fonction du nombre d'étapes. L'encombrement d'un grafcet (en CLB par étape) varie de 1/2 à plus de 1 suivant la connexité du graphe associé, et la complexité de ses réceptivités. Les ***performances*** du circuit synthétisé dépendent du délai du plus long chemin combinatoire correspondant à chaque signal d'entrée de bascule, ainsi que du nombre de périodes d'horloge dont dispose ce signal pour se propager avant de voir ses entrées combinatoires modifiées. Ce dernier point dépend de l'ensemble des scenarii d'entrées possibles. On peut évaluer les délais maximaux à environ 20 à 25 ns par couche traversée (délais de routage compris [4]) pour les circuits utilisés.

Un autre séquenceur, dans le cadre de la réalisation d'une carte de traitement d'images, ayant 14 entrées et 33 sorties a été décrit par un grafcet de 51 étapes. L'implantation de celui-ci a nécessité l'utilisation de 47 CLB. L'écriture de la table des états de ce séquenceur aurait comporté 2^{14} colonnes pour représenter les entrées !

2.7 Le *Grafcet* comme encodeur spécialisé

À l'usage, une représentation grafcet donnée peut être vue comme un encodage particulier d'une machine d'états, intermédiaire entre l'encodage binaire et l'encodage one-hot.

En effet, l'association "une étape $\longleftrightarrow$ une bascule" rapproche la réalisation d'un grafcet de l'encodage one-hot d'une machine d'états. Par ailleurs, le parallélisme du *Grafcet* (plusieurs étapes simultanément activables) diminue le nombre de bascules nécessaires à sa réalisation, ce qui est l'avantage d'un encodage binaire.

Finalement, l'encodage par grafcet est plus pertinent pour l'utilisateur grâce à la possibilité d'associer une sortie directement à une étape, lui donnant ainsi une signification physique. Dans l'exemple de la figure 2, les trois sorties `VH`, `ER` et `SP` sont directement associées à une étape, supprimant ainsi la nécessité du décodage des sorties (fonction `G` de la figure 1).

4. Ces délais maximaux moyens augmentent si le taux de remplissage du circuit est important.

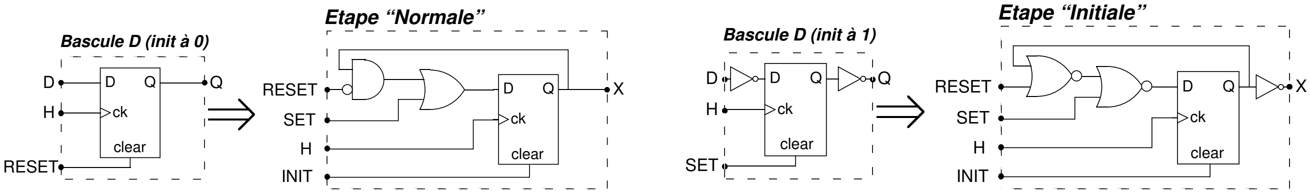


FIG. 3 – *Réalisation des deux types d'étapes (normale ou initiale) à l'aide de bascules D synchrones*

Source du composant `etape_normale`

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

entity etape_normale is
generic (
    DELAY : in time :=  0 ns
);
port (
    SET, RESET, H   : in  std_logic := 'X';
    Q               : out std_logic := '0'
);
end etape_normale;

architecture vhdl_lib of etape_normale is
    signal  D, Qtmp      : std_logic := '0';
begin
    -- On passe par un signal tampon car on n'a pas
    -- le droit de lire un signal de mode "out" :
    Q <= Qtmp after DELAY;

    -- Affectation de D avec RA1 prioritaire :
    D <= SET or (Qtmp and not RESET);

    -- Memorisation de D dans Qtmp :
    P1 : process
    begin
        wait until H'event and H = '1';
        if D = '1'  then Qtmp <= '1';
                    else Qtmp <= '0';
        end if;
    end process P1;
end vhdl_lib; -- etape_normale
```

Extrait du composant `aminc_g7`

```
...
entity aminc_g7 is port (
  EP, VIDE, SR, H : in  std_logic;
  ER, VH, SP      : out std_logic;
  -- le vecteur ETAPE facilite la simulation
  ETAPES          : out std_logic_vector (1 to 6)
  );
end aminc_g7;

architecture a1 of aminc_g7 is
...
begin
  -- connection des sorties :
  VH <= X3;
  ER <= X4;
  SP <= X5;
  ...
  -- etape initiale (dialogue d'entree termine)
  ETAPE1 : etape_initiale port map
              (X1_Set, X1_Reset, H, X1);
  X1_Set   <= X4 and not EP;
  X1_Reset <= X1 and X2 and EP;
  ...
  -- impulsion d'une periode (correspond a la sortie VH)
  -- ce composant est equiv. a une basc. D d'entree X3_Set
  ETAPE3 : etape_normale port map
              (X3_Set, X3_Reset, H, X3);
  X3_Set   <= (X1 and X2 and EP) or
             (X5 and SR and not VIDE);
  X3_Reset <= '1';
  ...
end a1;
```

FIG. 4 – *Extrait de la description* VHDL *du grafcet*

2.8 Validation

La sémantique du *Grafcet* est définie de manière informelle et aucun outil commercial de vérification n'est à notre connaissance associé au *Grafcet*. La pratique pour valider un grafcet est de le simuler. Malheureusement, il n'est pas réaliste d'envisager une simulation de tous les scénarii possibles sur le grafcet correspondant à un séquenceur de complexité moyenne. Une autre voie consiste à traduire le *Grafcet* dans un modèle possédant une sémantique formelle et auquel sont associés des outils de preuve de propriétés. Le paragraphe suivant propose une solution de ce type.

3 Utilisation d'un format pivot

Des schémas de traduction du *Grafcet* vers des modèles formels existent. On peut en particulier citer le langage synchrone *Signal* [Par94] et les automates temporisés [MLP96, L'H97]. Le compilateur *Signal* de l'Irisa permettant la génération de code VHDL, nous avons choisi d'explorer cette première voie. L'outil de preuve associé à *Signal* est Sigali.
Nous présentons succinctement le langage *Signal*, puis la modélisation du sous-ensemble du *Grafcet* utilisé dans le cadre des séquenceurs synchrones en *Signal*.

3.1 Le langage *Signal*

Le noyau d'instructions de *Signal* [GGtBn87] est composé de quatre opérateurs de base agissant sur l'ensemble des valeurs (dont $\perp$: absent) d'un signal :

- l'**opérateur fonctionnel** est un opérateur à un temps qui recouvre les opérations arithmétiques et booléennes classiques ; il impose le synchronisme des signaux utilisés à chaque instant logique donné t.

$$\forall t > 0 \; c_t = f(a_{1t}, \cdots, a_{nt})$$

- Le **retard** permet de retarder un signal d'un instant logique :

$$b := a\$1 \text{ init } i \Leftrightarrow \begin{cases} b_0 = i \\ b_t = a_{t-1} \quad \forall t \geq 1 \end{cases}$$

horloge	1	1	1	1	1	1
a	1	0	$\perp$	0	$\perp$	1
$a\$1$ init i	i	1	0	$\perp$	0	$\perp$

- Le **filtre** permet de réaliser le sous-échantillonnage d'un signal en fonction d'un signal booléen :

$$c := a \text{ when } b \Leftrightarrow c_t = \begin{cases} a_t & \text{si } a_t \neq \perp \\ & \text{et } b_t = \text{vrai} \\ \perp & \text{sinon} \end{cases}$$

horloge	1	1	1	1	1	1
a	1	0	$\perp$	0	$\perp$	1
b	1	$\perp$	0	1	$\perp$	0
a when b	1	$\perp$	$\perp$	0	$\perp$	$\perp$

- **La fusion** permet de sur-échantillonner (ou mélanger) des signaux d'horloges quelconques :

$$c := a \text{ default } b \Leftrightarrow c_t = \begin{cases} a_t & \text{si } a_t \neq \perp \\ b_t & \text{sinon} \end{cases}$$

horloge	1	1	1	1	1	1
a	1	0	$\perp$	0	$\perp$	1
b	1	$\perp$	0	1	$\perp$	0
a default b	1	0	0	0	$\perp$	1

Un processus *Signal* résulte de la composition parallèle (notée |) d'équations écrites à partir des opérateurs définis ci-dessus. En outre, il existe des directives de compilation, comme la directive *synchro* qui permet de spécifier les relations temporelles existant entre différents signaux.
A titre d'exemple, le processus *Signal* qui suit représente un compteur dont la sortie s est incrémentée à chaque fois que l'entrée top vaut 1 et est inchangée lorsque top vaut 0. Une évolution possible des signaux est présentée en-dessous.

```
process EXEMPLE = {
   ? logical top
   ! integer s
}
  (| s  := (zs+1) when top default zs
   | zs := s$1
   | synchro {s, top}
   |)
  where integer zs init 0
end
```

horloge	1	1	1	1	1	1
top	0	0	1	1	0	1
s	0	0	1	2	2	3
zs	0	0	0	1	2	2

Nous présentons maintenant une modélisation possible du *Grafcet* en *Signal*.

3.2 Modélisation du *Grafcet* en *Signal*

Un signal de sortie X_i est associé à chaque étape E_i. De même, un signal I_j est associé à chaque variable d'entrée I_j. Un cycle d'évolution d'un grafcet effectue la lecture des entrées, puis le calcul des réceptivités suivi du calcul de la nouvelle situation, enfin l'émission des sorties.
Le calcul des réceptivités a besoin des valeurs courantes des entrées et de la situation précédente. On va donc associer une équation *Signal* à chaque réceptivité. Si la réceptivité à calculer est :

$$R_l \;=\; f(I_1, \ldots, I_m, X_1, \ldots, X_n)$$

alors l'équation *Signal* associée est :

$$R_l \;=\; f(I_1, \ldots, I_m, X_1\$1, \ldots, X_n\$1)$$

où R_l est le signal associé à la réceptivité.
Le calcul de la nouvelle situation a besoin de la valeur des réceptivités et de la situation précédente. Une étape est active si elle est activée (elle était inactive à l'instant précédent et l'une au moins de ses transitions amont est franchissable) ou si elle reste active (elle était active à l'instant précédent et aucune de ses transitions aval n'est franchissable).
L'équation *Signal* correspondant à l'étape E_i est donc :

$$\begin{aligned} X_i \;=\; & \sum_{k=1}^{p} \mathcal{F}(T_k(E_i)) \quad \text{or} \\ & X_i\$1 \text{ and not } \sum_{j=1}^{m} \mathcal{F}((E_i)T_j) \end{aligned}$$

où les termes $T_k(E_i)$ et $(E_i)T_j$ représentent respectivement les transitions amont et aval de l'étape E_i.

Pour une transition T_l, on a le signal de franchissement :

$$\mathcal{F}(T_l) \;=\; R_l \text{ and } \prod_{q=1}^{r} X_q(T_l)\$1$$

où les termes $X_q(T_l)$ représentent les états des étapes amont de la transition T_l.

À chaque signal de sortie est associée une équation réalisant un OU logique entre les variables des étapes auxquelles est associée cette sortie. À titre d'exemple, l'équation *Signal* appliquée à l'étape 3 de la figure 2 vaut :

$$\begin{array}{rcl} X_3 & = & EP \text{ and } X_1\$1 \text{ and } X_2\$1 \quad \text{or} \\ & & SR \text{ and } \overline{VIDE} \text{ and } X_5\$1 \end{array}$$

Un outil pour la traduction automatique du *Grafcet* en *Signal* a été mis en œuvre. Le programme signal généré peut ensuite être exécuté à l'aide d'un autre outil qui permet en outre de suivre graphiquement l'évolution du grafcet de départ. Il faut à présent, d'une part définir les propriétés que l'on souhaite vérifier, d'autre part mettre en place la génération de bascules à partir du code signal, et évaluer la qualité du résultat obtenu.

4 Conclusion

Dans cet article, nous avons montré comment le *Grafcet* peut servir de langage de spécification pour les séquenceurs synchrones. Dans ce cadre, nous avons proposé une modélisation d'un séquenceur sous la forme d'un grafcet et la traduction de celui-ci en un ensemble de bascules. Nous avons également montré comment il est possible d'utiliser un langage pivot possédant une sémantique formelle (le langage *Signal*) lors de cette traduction. Cela permet d'utiliser les outils de preuve associés à ce langage.

Il faut à présent finaliser le passage du *Grafcet* aux bascules via *Signal*. Il faut en outre étudier l'utilisation d'autres langages pivot, en particulier les automates temporisés qui permettent une modélisation plus efficace du temps et donc la vérification de propriétés temporelles.

Enfin, il serait intéressant de généraliser la traduction à toutes les composantes du *Grafcet*. Cela permettrait de proposer une implantation de la commande d'un processus industriel à base de logique reconfigurable, permettant une bonne maîtrise des aspects temporels.

Références

[AA79] ADEPA and AFCET. *Le Grafcet.* Automation et Production. Cépadues, 1979.

[Ber87] J.M. Bernard. *Conception sructurée des systèmes logiques.* Eyrolles, 1987.

[DA91] R. David and H. Alla. *Du Grafcet aux réseaux de Petri.* Hermés, 1991.

[Dia94] Maurice Diamantini. *Implantation de l'algorithme DES dans des circuits FPGA de type Xilinx.* Mémoire, CNAM, Paris, 24 janvier 1994.

[GGtBn87] P. Le Guernic, T. Gautier, and L. Besnard. Signal a declarative language for synchronous programming of real time systems. In *Functional programming languages and computer architecture.* Springer-Verlag LNCS 274, 87.

[IEC88] Commission Électronique Internationale. *Établissement de diagrammes fonctionnels pour systèmes de commande*, 1988.

[L'H97] Dominique L'Her. *Modélisation du Grafcet temporisé et vérification de propriétés temporelles.* Thèse de doctorat, Université de Rennes 1, septembre 1997.

[MLP96] L. Marcé, D. L'Her, and P. Le Parc. Modelling and verification of temporized grafcet. In *CESA*, Lille, 1996.

[Moa85] M. Moalla. Réseaux de petri interprétés et grafcet. In *Technique et Sciences de l'Informatique (TSI)*, number 4, pages 17–30. AFCET, 1985.

[Par94] Ph. Le Parc. *Apports de la méthodologie synchrone pour la définition et l'utilisation du langage Grafcet.* Thèse de doctorat, Université de Rennes, janvier 1994.

[UTE93] Union Technique de l'Électricité. *Établissement des diagrammes fonctionnels pour systémes de commande - Diagramme fonctionnel GRAFCET Extension des concepts de base*, 1993.

[Xil94] Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124. *The Programmable Gate Array Databook*, 1994.