Vers la vérification d'une méthodologie pour la conception de circuits numériques critiques

Vincent Iampietro[1], David Andreu[1,2], and David Delahaye[1]

[1]LIRMM, Université de Montpellier, CNRS, Montpellier, France
[2]NEURINNOV, Montpellier, France

1 Introduction

Pour répondre aux contraintes liées à la conception de circuits numériques critiques, et à l'augmentation constante de la complexité des systèmes, le domaine de l'Ingénierie Système à Base de Modèles (ISBM) a été développé. L'intérêt est de travailler sur des modèles de haut niveau avec un pouvoir d'expression et des qualités de compréhension et de lisibilité qui facilitent les interactions entre les acteurs de la conception du circuit (i.e, les ingénieurs). Plusieurs formalismes existent : le langage SysML [2], des variantes du langage C [9], ou encore les réseaux de Petri (RdPs) [8], pour citer les plus répandus. Une fois la conception terminée, les modèles sont physiquement synthétisés en suivant un procédé manuel ou automatique. Il reste alors à prouver que la phase de transformation préserve le comportement du modèle de conception. Le présent article décrit le travail d'une thèse en cours. Cette thèse s'intéresse à la vérification d'un processus d'aide à la modélisation et à la production de circuits numériques critiques : la méthodologie HILECOP (HIgh LEvel hardware COmponent Programming). Cette méthodologie est mise en oeuvre dans le cadre de la création de micro-contrôleurs intégrés à des dispositifs médicaux de type neuroprothèses. La Figure 1 en décrit les principales étapes.

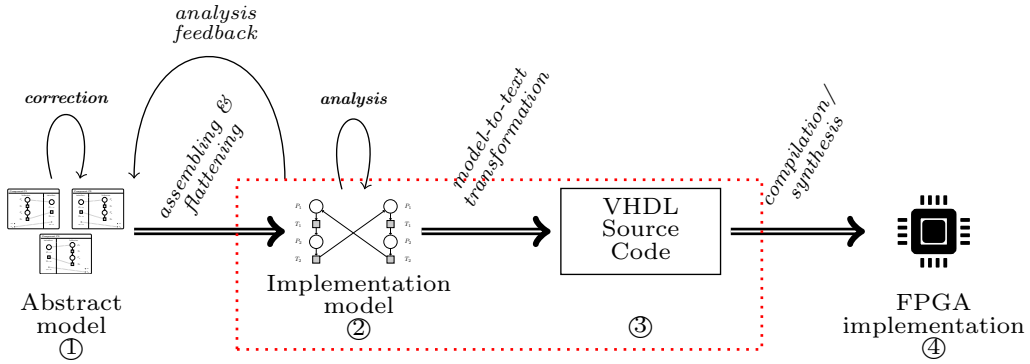


Figure 1 – La méthodologie HILECOP

Le concepteur de systèmes électroniques esquisse premièrement un modèle graphique de haut niveau de son circuit (①). Ce modèle s'appuie sur le formalisme des diagrammes à composants, avec l'addition des RdPs pour décrire le comportement interne des parties du circuit. Dans un deuxième temps, les parties du modèle sont assemblées et la structure des composants est effacée. Le résultat obtenu est un réseau de Petri global décrivant le système modélisé (②). Des outils d'analyse exploités par la méthodologie permettent alors de vérifier certaines propriétés du modèle (caractère borné, vivacité…) et présentent un compte-rendu au concepteur. Après plusieurs itérations du cycle analyse-correction, du code VHDL est généré à partir du modèle d'implémentation (③). Dès lors, la dernière étape de la méthodologie, qui opère la synthèse du circuit électronique depuis le code source VHDL, est prise en charge par un compilateur/synthétiseur industriel propriétaire (④).

L'objectif de la thèse est de prouver que la transformation du modèle d'implémentation en code VHDL (i.e, de ② vers ③ dans la Figure 1) n'introduit pas de divergences de comportement. Dans cette optique, il sera nécessaire de formaliser la sémantique des modèles de haut niveau (RdP), du langage cible (VHDL), et de décrire la transformation. Ensuite, la preuve de similarité comportementale devra être établie. L'intégralité de la démarche sera mécanisée avec l'assistant à la preuve Coq [6]. Même si cette démarche a été éprouvée pour la vérification de compilateurs,

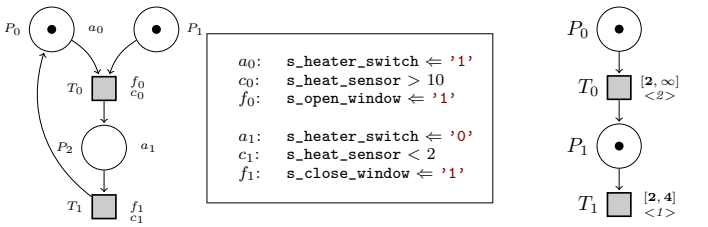
son application à la conception de circuits numériques est bien moins fréquente. L'intérêt scientifique provient de la distance qui existe entre le modèle d'exécution du formalisme source (SITPN) et celui du langage cible (VHDL). Cette distance devra être prise en compte lors de la preuve de préservation de comportement.

2 Un formalisme de haut-niveau : les réseaux de Petri

Du fait de leur statut de modèles formels et des possibilités d'analyse qui en résultent, les RdPs ont été retenus comme modèles de haut niveau de la méthodologie HILECOP. Le but de la méthodologie étant la conception et la production de circuits numériques *critiques*, les modèles se doivent d'être validés par analyse formelle. Afin d'augmenter l'expressivité des modèles, les RdPs HILECOP combinent plusieurs classes connues de RdPs (présentées ci-après), mais leur particularité réside dans leur exécution synchrone. Les RdPs HILECOP sont nommés SITPNs pour Synchronously executed Interpreted Time Petri Nets with priorities.

Les SITPNs sont des RdP interprétés; des actions peuvent être associées aux places d'un réseau, et des fonctions/conditions peuvent être associées aux transitions. Actions et fonctions définissent des opérations qui influencent l'environnement du RdP, et leurs effets sont mesurés à travers la valeur booléenne des conditions associées aux transitions. Dans un RdP interprété, une transition est franchissable si elle est sensibilisée et que toutes les conditions qui lui sont associées sont *vraies*. La Figure 2.(a) donne un exemple de RdP interprété où les actions, fonctions et conditions sont définies par du code VHDL.

Les RdP utilisés dans HILECOP sont temporels; une fenêtre de tir, i.e un intervalle de temps, peut être associée à une transition. Un compteur de temps est lancé lorsqu'une transition devient sensibilisée; celle-ci devient franchissable lorsque son compteur de temps a atteint l'intervalle de tir. La Figure 2.(b) donne un exemple de RdP temporel. La valeur courante des compteurs de temps est représentée entre chevrons en dessous des intervalles temporels associés. En résumé, une transition d'un SITPN est franchissable si elle est sensibilisée, si toutes les conditions qui lui sont associées sont vraies et si son compteur de temps est dans l'intervalle défini.



(a) RdP interprété.

(b) RdP temporel.

Figure 2 – Réseaux de Petri temporels et interprétés.

Contrairement au cas général, les SITPNs ont une politique de tir (i.e, une sémantique) *synchrone*. Fondamentalement, le tir des transitions d'un RdP est un phénomène indéterministe (si deux transitions sont franchissables au même instant, tous les ordres de tirs sont possibles), et asynchrone (dès qu'une transition est franchissable, elle peut être tirée sans attente). A contrario, l'évolution d'un SITPN est rythmée par le front montant et le front descendant d'un signal d'horloge, comme montré dans la Figure 3. Sur le front descendant (① de la Figure 3), toutes les transitions devant être tirées sont déterminées, ce après mise à jour des conditions et intervalles de temps; sur le front montant (② de la Figure 3), les précédentes transitions sont tirées, entraînant la mis à jour du marquage du réseau et l'exécution de fonctions. La sémantique d'évolution d'un tel réseau est synchrone et déterministe.

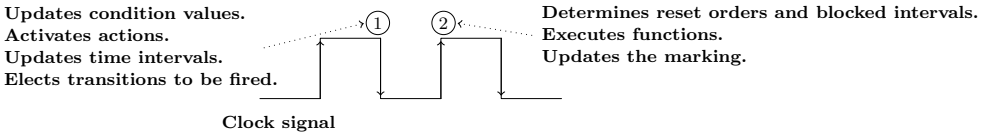


Figure 3 – Evolution synchrone d'un SITPN.

La structure et la sémantique des SITPNs ont été formalisées dans [4, 5]. La sémantique est exprimée comme un système états-transitions où les transitions sont étiquetées par les évènements d'un signal d'horloge. Il y a deux évènements possibles : le front montant et le front descendant du signal. L'état d'un SITPN décrit, entre autres, le marquage courant du SITPN, la valeur des compteurs de temps et des conditions associés aux transitions, la liste des transitions à tirer…La sémantique des SITPNs fixe les règles de changement d'état en fonction des évènements d'horloge. Par exemple, sur le front descendant d'horloge, la liste des transitions à tirer au prochain front montant est calculée; une règle stipule qu'une aucune transition non franchissable au front descendant n'appartient à l'ensemble des transitions à tirer.

Algorithme 1 : SimulationLoop(Δ, σ_{init}, $nbCycles$)

```
1  begin
       // Initialization phase.
2      σ1 = RunAllProcessesOnce(Δ, σinit)
3      σ2 = Stabilize(Δ, σ1)
       // Main loop.
4      Tc ← 0
5      while Tc ≤ nbCycles do
6          σ3 = ExecuteFallingEdgePss(Δ, σ2)
7          σ4 = Stabilize(Δ, σ3)
8          σ5 = ExecuteRisingEdgePss(Δ, σ4)
9          σ2 = Stabilize(Δ, σ5)
10         Tc ← Tc + 1
```

La première contribution de la thèse est l'implantation en Coq de la structure et de la sémantique des SITPNs. La sémantique a été implantée comme une relation inductive paramétrée par un SITPN, deux états (i.e, avant et après transition), et un évènement d'horloge. La relation présente deux cas de construction, un pour chaque évènement d'horloge considéré. Afin de tester notre implantation de la sémantique des SITPNs, un interprète a été conçu, i.e un programme qui simule les changements d'état d'un SITPN pour n cycles d'horloge, en partant de l'état initial du réseau. Cet interprète est prouvé correct et complet vis à vis de la sémantique des SITPNs pour une évolution sur un cycle d'horloge. L'intégralité de la formalisation et de la mécanisation est mise à disposition du lecteur[1].

3 Un langage cible : VHDL

Il existe plusieurs techniques permettant la synthèse physique d'un RdP. Cependant, la technique la plus étudiée est la transformation vers la langage VHDL. Cette technique a donc été retenue par la méthodologie HILECOP. Le langage VHDL permet les descriptions structurelle et comportementale de circuits électroniques, à des fins de simulation ou de synthèse physique. En VHDL, un *design* décrit un composant électronique en termes d'interface entrée-sortie (l'*entité*) et de comportement interne (l'*architecture*). Le comportement d'un design s'exprime de deux manières : via l'interconnexion d'instances d'autres designs (des sous-composants), ou à l'aide de *processus*. La spécificité du langage VHDL tient à l'exécution concurrente des processus et des sous-composants décrivant une architecture de design. Un processus définit un bloc d'instructions séquentielles; il observe un certain nombre de signaux qui composent sa liste de sensibilité. Le changement d'état d'un signal de cette liste entraîne l'exécution du bloc d'instructions du processus. Conceptuellement, un signal VHDL représente une connexion physique sur un circuit électronique. Les signaux sont les principaux véhicules des valeurs dans les programmes VHDL.

La sémantique de VHDL est décrite dans une prose informelle dans le manuel de référence du langage (MRL). De fait, interpréter un programme VHDL, qui décrit un *design* de circuit, revient à simuler le design décrit. Dans le MRL, la sémantique de VHDL est donc définie sous la forme d'une boucle de simulation. La boucle de simulation spécifie la dynamique d'exécution des blocs concurrents qui composent une architecture de design, ainsi que la propagation des valeurs au travers des signaux.

La littérature propose de nombreuses formalisations de la sémantique de VHDL [3]. Certaines formalisations expriment la boucle de simulation telle qu'exhibée dans le MRL; d'autres choisissent de s'abstraire de cette boucle, et optent pour une formalisation alternative basée sur des modèles permettant la gestion de la concurrence et du temps (automates temporels, réseaux de Petri, logique d'intervalles temporels...).

La méthodologie HILECOP opère la génération d'un design VHDL dans l'optique de sa synthèse physique. Dès lors, nous ne considérons qu'une partie *synthétisable* du langage que nous définissons et nommons $\mathcal{H}$-VHDL. De plus, les designs VHDL générés par la méthodologie HILECOP décrivent des circuits synchrones, i.e, dont l'exécution est rythmée par un signal d'horloge. La prise en compte d'une sous-partie synthétisable et du synchronisme nous a permis d'exprimer la sémantique des programmes $\mathcal{H}$-VHDL en termes d'une boucle de simulation bien plus simple en comparaison de celle exprimée dans le MRL. L'Algorithme 1 décrit notre boucle spécifique de simulation pour un design $\mathcal{H}$-VHDL.

La boucle de simulation de l'Algorithme 1 est paramétrée par un design VHDL (Δ), l'état initial du design (σ_{init}) qui contient les valeurs des signaux et les états courants des sous-composants du design Δ, et le nombre de cycles de simulation à effectuer ($nbCycles$). Durant la phase d'initialisation, tous les processus décrivant le

[1] https://github.com/viampietro/sitpns

comportement du design sont exécutés une fois. Cette phase est suivie d'une phase de stabilisation de la valeur des signaux. La phase de stabilisation correspond à la propagation des valeurs entre signaux interconnectés, ce jusqu'à ce que la propagation n'induisent plus aucun changement. La boucle principale de simulation décrit l'alternance entre l'exécution des processus dits *séquentiels*, i.e qui sont sensibles aux évènements d'horloge, et des processus *combinatoires*, qui s'exécutent de manière continue jusqu'à stabilisation des signaux.

Au stade actuel des travaux, une formalisation de la sémantique de $\mathcal{H}$-VHDL a été effectuée sous la forme d'une sémantique opérationnelle à grands pas, et sa mécanisation en Coq a été réalisée. Cette sémantique s'inspire des travaux de formalisation esquissés dans [7, 1]. La sémantique formalisée prend également en compte la phase d'élaboration du design, préliminaire à la simulation. L'élaboration génère l'environnement de simulation, i.e un couple Δ, σ_{init} qui se trouve en paramètre de la boucle de simulation (voir Algorithme 1). Durant la phase d'élaboration, une vérification de type est effectué sur le code VHDL. La vérification de type s'assure que la partie déclarative et la partie comportementale du design VHDL respectent certaines règles de typage définies par le MRL. Par exemple, pour une instruction d'affectation de valeur à un signal, l'expression affectée doit être du même type que le signal cible.

4 Conclusion

Le but de la thèse est de vérifier formellement une partie de la méthodologie HILECOP, usitée dans le cadre de la conception de circuits numériques critiques. Spécifiquement, le travail de vérification porte sur la phase transformant un modèle de conception, à base de RdPs, en *design* VHDL. La finalité de ce travail sera la spécification et la démonstration d'un théorème de préservation de comportement pour cette phase de transformation.

Jusqu'ici, la sémantique des SITPNs, modèles de haut niveau de HILECOP, et la sémantique de $\mathcal{H}$-VHDL ont été implantées à l'aide du langage Coq. Concernant les SITPNs, deux éléments déjà existant dans ce formalisme restent à prendre en compte : les macroplaces, qui permettent d'exprimer la gestion d'exceptions dans les SITPNs, ainsi que la possibilité de spécifier des domaines d'horloge différents au sein d'un même modèle; c'est le cas des systèmes Globalement Asynchrones Localement Synchrones (GALS).

La transformation d'un SITPN en un modèle VHDL est en cours de programmation avec le langage Coq. Enfin, la dernière étape de ce travail sera d'établir la preuve de préservation de comportement.

Références

[1] D. Borrione and A. Salem. Denotational semantics of a synchronous VHDL subset. *Formal Methods in System Design*, 7(1–2) :53–71, Aug 1995.

[2] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML : The Systems Modeling Language*. Morgan Kaufmann, Oct 2014.

[3] C. D. Kloos and P. Breuer. *Formal Semantics for VHDL*, volume 307. Springer Science & Business Media, 2012.

[4] H. Leroux. *Méthodologie de conception d'architectures numériques complexes : du formalisme à l'implémentation en passant par l'analyse, préservation de la conformité. Application aux neuroprothèses*. PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc, Oct 2014.

[5] I. Merzoug. *Validation formelle des systèmes numériques critiques : génération de l'espace d'états de réseaux de Petri exécutés en synchrone*. PhD thesis, Université Montpellier, Jan 2018.

[6] The Coq Development Team. *Coq, version 8.9.0*. Inria, Jan. 2019. `http://coq.inria.fr/`.

[7] J. P. Van Tassel. *An Operational Semantics for a Subset of VHDL*, volume 307, page 71–106. Springer US, 1995.

[8] A. Yakovlev and A. Koelmans. *Petri nets and digital hardware design*, page 154–236. Apr 2006.

[9] Y. Yankova, K. Bertels, S. Vassiliadis, R. Meeuws, and A. Virginia. Automated hdl generation : Comparative evaluation. In *2007 IEEE International Symposium on Circuits and Systems*, page 2750–2753, May 2007.