THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

Vérification formelle d'une méthodologie pour la conception et la production de systèmes numériques critiques

Formal verification of a methodology for the design and production of safety-critical digital systems

Présentée par Vincent IAMPIETRO

Le 16 décembre 2021

Sous la direction de David Delahaye et David Andreu

Devant le jury composé de

Sandrine Blazy, Professeure, Université de Rennes	Rapporteure
Frédéric Boniol, Maître de recherche, ONERA, Toulouse	Rapporteur
David Déharbe, Docteur, Ingénieur, Clearsy	Examinateur
Marc Pouzet, Professeur, ENS, Paris	Examinateur
David Delahaye, Professeur, Université de Montpellier	Directeur
David Andreu, MCF, Université de Montpellier/Neurinnov	Co-directeur

UNIVERSITÉ DE MONTPELLIER

Contents

Résumé

La production de circuits numériques complexes est devenue impossible sans l'aide des ordinateurs. La méthodologie HILECOP (HIgh LEvel hardware COmponent Programming) assiste les ingénieurs dans la conception et la production de circuits numériques dans le contexte des systèmes critiques, i.e. systèmes dont le malfonctionnement peut résulter en la perte de vies humaines, des catastrophes naturelles, des désastres économiques, etc. A titre d'exemple, la société Neurinnov[1] applique la méthodologie HILECOP pour la production de neuroprothèses, considérées comme des dispositifs médicaux hautement critiques par la loi de régulation de l'UE[2]. Dans HILECOP, les ingénieurs produisent un modèle de circuit numérique. Ils utilisent un formalisme graphique qui regroupe les diagrammes à composant et un type particulier de réseaux de Petri (RdP). Ensuite, le modèle est transformé en une représentation textuelle intermédiaire décrite en langage VHDL (Very high speed integrated circuit Hardware Description Language). Finalement, un compilateur/synthétiseur industriel génère un circuit numérique physique, i.e. un ASIC ou sur carte FPGA, depuis la représentation VHDL. Ici, l'utilisation des RdPs est liée au contexte des systèmes critiques. Les RdPs permettent la vérification de propriétés sur les modèles de circuits numériques grâce à l'application de techniques de *model-checking*. Cependant, une des transformations décrite dans la méthodologie HILECOP pourrait altérer le *comportement* (ou *sémantique*) des modèles initiaux, invalidant ainsi les précédentes étapes de vérification. Le but de cette thèse est de prouver que la transformation modèle-vers-texte de HILECOP, qui génère une description VHDL depuis un modèle de circuit numérique, préserve le comportement des modèles d'entrée, i.e.: pour tout modèle passé en entrée de la transformation, la description VHDL résultante se comporte de la même manière. Pour prouver cette propriété, nous nous inspirons des travaux menés sur la vérification formelle de compilateurs (notamment sur le compilateur C certifié CompCert [3]). Notre approche est celle de la vérification déductive interactive avec assistants de preuve. Dans ce contexte, les étapes pour établir la propriété de préservation de comportement de la transformation sont: (1) formaliser la sémantique d'execution de la représentation source, (2) de la représentation cible, (3) décrire la transformation, et (4) prouver un théorème de préservation sémantique. Même en suivant ce processus clairement détaillé, les spécificités de la transformation modèle-vers-texte de HILECOP (comparaît notamment aux compilateurs) apportent de nouvelles questions recherches et des challenges à chaque étape. Dans cette thèse, nous utilisons l'assistant à la preuve Coq pour nous accompagner tout au long du processus. Finalement, nous avons prouvé que la transformation de HILECOP préserve le comportement de tous modèles initiaux. La mécanisation complète de la preuve avec Coq est un travail en cours.

[1] https://neurinnov.com/

[2] https://eur-lex.europa.eu/eli/reg/2017/745/2020-04-24

Abstract

The complexity of digital hardware circuits makes it difficult to produce them without the help of computers. The HILECOP (HIgh LEvel hardware COmponent Programming) methodology aims at the assistance of engineers in the design and production of such digital circuits. The context of production is the one of *safety-critical* digital systems, i.e. systems which failure could result in direct human losses, natural catastrophes, economic disasters, etc. To give an example, the Neurinnov[3] company leverages the HILECOP methodology to produce highly critical medical devices known as neuroprostheses. In HILECOP, engineers rely on a graphical formalism, based on component diagrams and a particular kind of Petri nets, to produce a model of a digital circuit. Then, a computer program turns the model into an intermediary description written in VHDL (Very high speed integrated circuit Hardware Description Language). Finally, an industrial compiler/synthesizer transforms the VHDL description into a concrete physical circuit on an FPGA, or as an ASIC. The use of Petri nets permits the engineers to describe a *formal* model of a digital circuit. The mathematical foundations of Petri nets enable the use of model-checking techniques. Thus, proofs can be brought that the produced models verify certain *soundness* properties. However, even with a *sound* model of a circuit, one transformation step could alter the behavior of the initial model. The goal of this thesis is to bring the formal proof that the model-to-text transformation from a HILECOP high-level model to a VHDL description is *semantic preserving* (or *behavior preserving*); i.e. for all high-level model given as an input to the transformation, the resulting VHDL description behaves similarly. To perform this task, we draw our inspiration from the works pertaining to the formal verification of compilers for programming languages (especially from the certified C compiler CompCert [3]). Specifically, we are interested in proving the property of semantic preservation in the context of *deductive* verification with proof assistants. In this context, the steps to verify that a transformation is semantic preserving include: (1) the formalization of the execution semantics of the source representation, (2) of the target representation, (3) the formal description of the transformation, and (4) the proof of a corresponding semantic preservation theorem. In this thesis, these steps have been carried within the framework of the Coq proof assistant. Even though these steps are clearly set, the specificities of the HILECOP model-to-text transformation, compared to compilers for programming languages, bring some interesting research challenges. Finally, we have brought the informal *paper* proof that the HILECOP transformation is semantic preserving by demonstrating a related behavior preservation theorem. The full mechanization of the proof using the Coq proof assistant is an ongoing task.

[3] https://neurinnov.com/

Résumé étendu

0.1 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 [1], des variantes du langage C [7], ou encore les réseaux de Petri (RdPs) [6], 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. La présente 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.

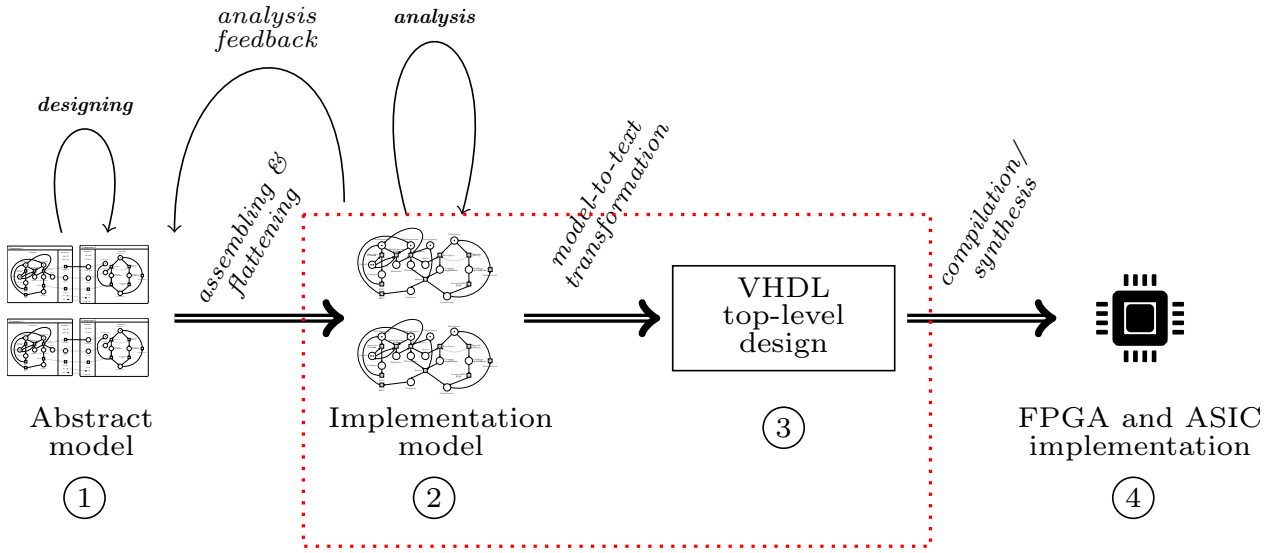

FIGURE 1: Principe de la méthodologie HILECOP; les double flêches horizontales représentent des phases de transformation; les simple flêches indiquent les autres types d'opérations ayant cours à une étape précise, ou entre étapes.

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 [5]. 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.

0.2 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 sur une ensemble de variables, ici, des *signaux* VHDL. Les conditions associées aux transitions sont des expressions Booléennes sûr la valeur des signaux. 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 donne un exemple de RdP interprété.

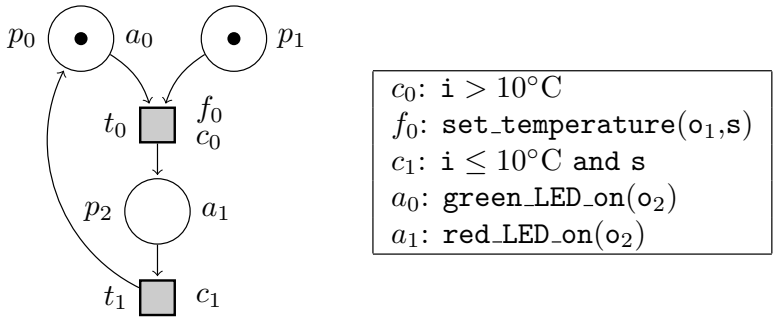

FIGURE 2: Un exemple de réseau de Petri interprété; sur le côté gauche, le RdP; sur le côté droit, les expressions Booléennes associées aux conditions et les opérations associées aux actions et fonctions.

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 3 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.

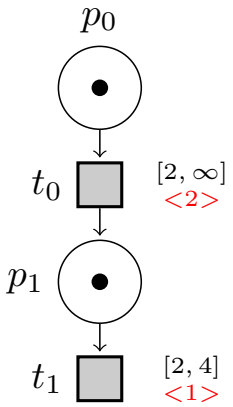

FIGURE 3: Un exemple de RdP temporel. La valeur des compteurs de temps apparaît en rouge.

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 4. Sur le front descendant (① de la Figure 4), 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 4), 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.

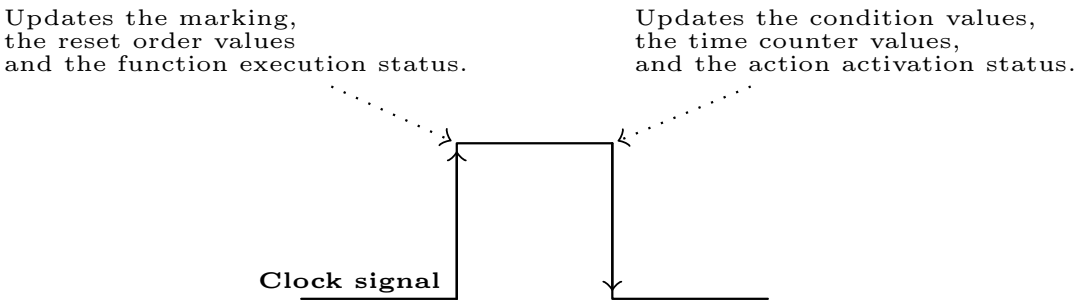

FIGURE 4: Evolution d'un SITPN synchronisée avec un signal d'horloge.

La structure et la sémantique des SITPNs ont été formalisées dans [2, 4]. 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, a lliste 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.

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[4]. Cependant, nous avons utilisées une autre version de l'implémentation des SITPNs en Coq pour effectuer la preuve de préservation sémantique. La dernière version est plus élégante et utilisent les types dépendants[5].

0.3 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

[4] https://github.com/viampietro/sitpns
[5] https://github.com/viampietro/ver-hilecop/tree/master/sitpn/dp

Algorithm 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
```

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 [**KB12**]. 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 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 [**VanTassel95**, **Borrione95**]. 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.

0.4 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.

Bibliography

[1] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, Oct. 23, 2014. 631 pp. ISBN: 978-0-12-800800-3.

[2] Hélène 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. 28, 2014.

[3] Xavier Leroy. "A Formally Verified Compiler Back-End". In: *Journal of Automated Reasoning* 43.4 (Nov. 4, 2009), p. 363. ISSN: 1573-0670. DOI: 10.1007/s10817-009-9155-4.

[4] Ibrahim 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. 15, 2018.

[5] The Coq Development Team. *Coq, Version 8.13.2*. manual. July 2021.

[6] Alex Yakovlev and Albert Koelmans. "Petri Nets and Digital Hardware Design". In: *Lecture Notes in Computer Science - LNCS*. Apr. 11, 2006, pp. 154–236. DOI: 10.1007/3-540-65307-4_49.

[7] Yana Yankova et al. "Automated HDL Generation: Comparative Evaluation". In: *2007 IEEE International Symposium on Circuits and Systems*. 2007 IEEE International Symposium on Circuits and Systems. May 2007, pp. 2750–2753. DOI: 10.1109/ISCAS.2007.378622.