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

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

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

Présenté par Vincent IAMPIETRO

Le Date de la soutenance

Sous la direction de David Delahaye et David Andreu

Devant le jury composé de

[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]

UNIVERSITÉ
DE MONTPELLIER

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

Acknowledgements **iii**

1 The HILECOP methodology **1**
1.1 Designing critical digital systems 1
1.2 Introducing the HILECOP methodology 2
1.3 Verifying the HILECOP methodology 6

Bibliography **9**

List of Figures

1.1 A Model-Based Systems Engineering process. 2
1.2 Workflow of the HILECOP methodology. 3
1.3 An example of HILECOP high-level model. 3
1.4 Global Petri net model. 4
1.5 Generation of a VHDL design from a Petri net. 5
1.6 A view of the HILECOP software. 7

List of Tables

List of Abbreviations

SITPN	**S**ynchronously executed **I**nterpreted **T**ime **P**etri **N**et with priorities
VHDL	**V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage
PCI	**P**lace **C**omponent **I**nstance
TCI	**T**ransition **C**omponent **I**nstance
GPL	**G**eneric **P**rogramming **L**anguage
HDL	**H**ardware **D**escription **L**anguage

For/Dedicated to/To my…

Chapter 1

The HILECOP methodology

In this chapter, we present the context of our work, and more specifically, the subject of our verification task, i.e. HILECOP, a methodology for the design and implementation of critical digital systems. In Section 1.1, we motivate the use of Model-Based Systems Engineering (MBSE) and formal methods in the design and production of safety-critical digital systems; in Section 1.2, we give an overall presentation of the HILECOP methodology which applies both the principles of MBSE and formal methods; in Section 1.3, we point out the specific transformation phase, intervening in the HILECOP methodology, that we propose to verify and discuss on how we propose to verify it and which research questions does it give rise to.

1.1 Designing critical digital systems

According to Moore's law [12], the complexity of digital integrated circuits is always increasing. To give an example, the cut-of-the-edge *AMD Epyc Rome* microprocessor (2019) is made out of 50 billions of transistors. Composing billions of transistors on a wired circuit is no more a task for humans but is very suited to computers. However, engineers need to think about the design of digital circuits in a way that is understandable for humans. Therefore, they need high-level views of the circuits they are designing in order to work together and to communicate about the designs. The domain of Model-Based Systems Engineering (MBSE) [10] proposes a framework to help engineers to design and produce digital circuits, in a well-documented, safe and reliable way. Comparable to what Model Driven Engineering (MDE) does in the world of software engineering, models are first order concepts in MBSE. A model represents a simplified view of real object. As illustrated in Figure 1.1, a MBSE process describes a way to design a digital circuit starting from a high-level view of the system. This high-level view can follow a graphical formalism such as SysML [6] or Petri nets [13], or a textual one such as SystemC [4] or VHDL [2]. Then, the MBSE process describe many refinements phases (the green arrows in Figure 1.1) during which the input model will be transformed; at each refinement phase, the model goes down in abstraction towards its final implementation as a hardware circuit. A refinement phase, which is also a transformation phase, can be performed automatically or manually.

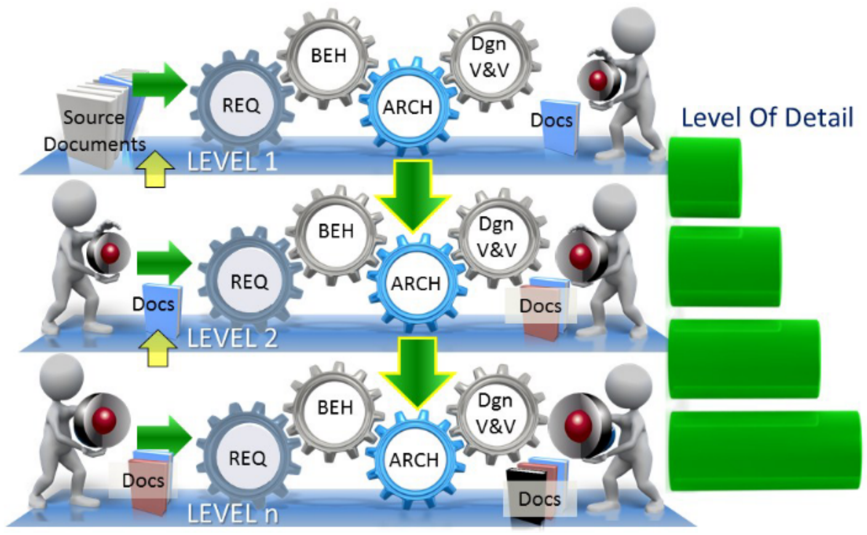

FIGURE 1.1: A Model-Based Systems Engineering process; REQ stands for requirements, BEH for behavior, ARCH for architecture, Dgn V&V for design verification and validation. This figure is an excerpt from [10].

In the case where the digital circuit being designed is a safety-critical system, i.e. on its behavior depends the life of people, an MBSE process will often employ formal models, i.e. models with a formal mathematical definition, as the design formalism. Thus, these models enable a certain extent of mathematical reasoning to prove that safety properties are met during the design V&V phase (cf. Figure 1.1).

1.2 Introducing the HILECOP methodology

The INRIA CAMIN team (former DEMAR team) has developed a new technology of neuroprotheses [7]. Neuroprotheses are medical devices which purpose is to electro-stimulate the nerves of patients suffering from moving disabilities. The nerves are responding to the stimulation, i.e an electric influx, in order to activate the muscles and so that the patient can recover some movements. Thus, controlling the intensity and the form of the electric signal sent to the patient's nerve is a critical point of the device overall functioning. These two parameters are controlled by a digital hardware circuit (i.e. a microcontroller) that is a part of the neuroprosthesis. Therefore, the design of such digital systems becomes utterly critical as a faulty circuit could result in the injury of patients. To assist the engineers in the design and the implementation of these critical digital systems, the CAMIN team came up with a process called the "HILECOP methodology" [1]. This methodology follows the principles of a MBSE process and relies on several transformations going from abstract models to concrete FPGA implementations through the production of VHDL code. Figure 1.2 details the global workflow of HILECOP.

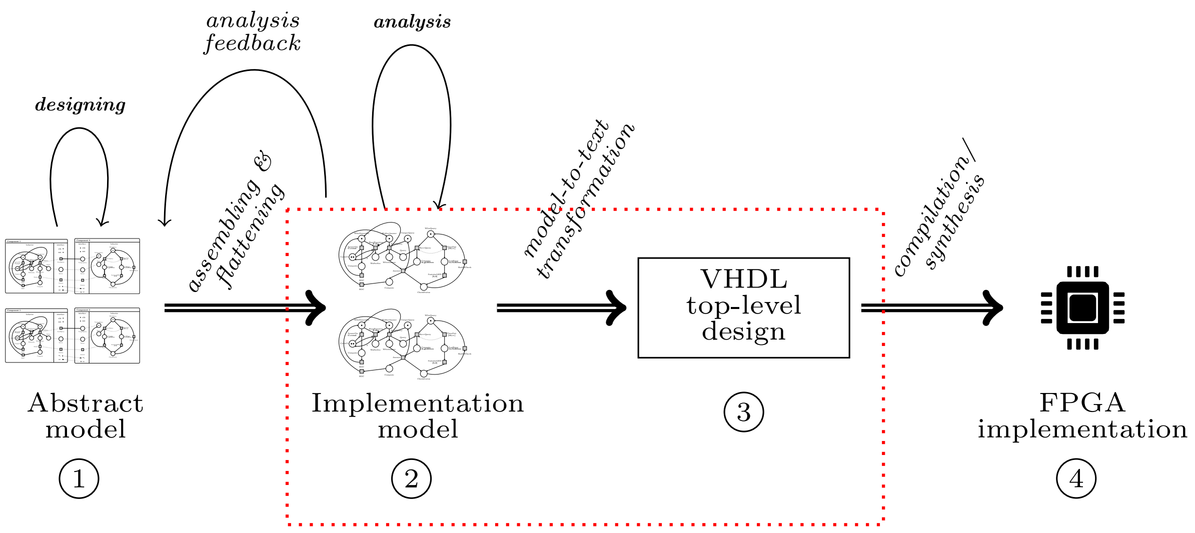

FIGURE 1.2: Workflow of the HILECOP methodology; horizontal double arrows indicate the transformation phases, i.e. the refinement phases in MBSE terms; simple arrows indicate different kinds of operations performed at a given step.

In Figure 1.2, Step 1 corresponds to the design phase of a critical digital system. At this step, the engineers produce a model of the wanted system; the leveraged model formalism is a graphical formalism specially designed for the methodology and based on component diagrams. Figure 1.3 provides an example of such a model.

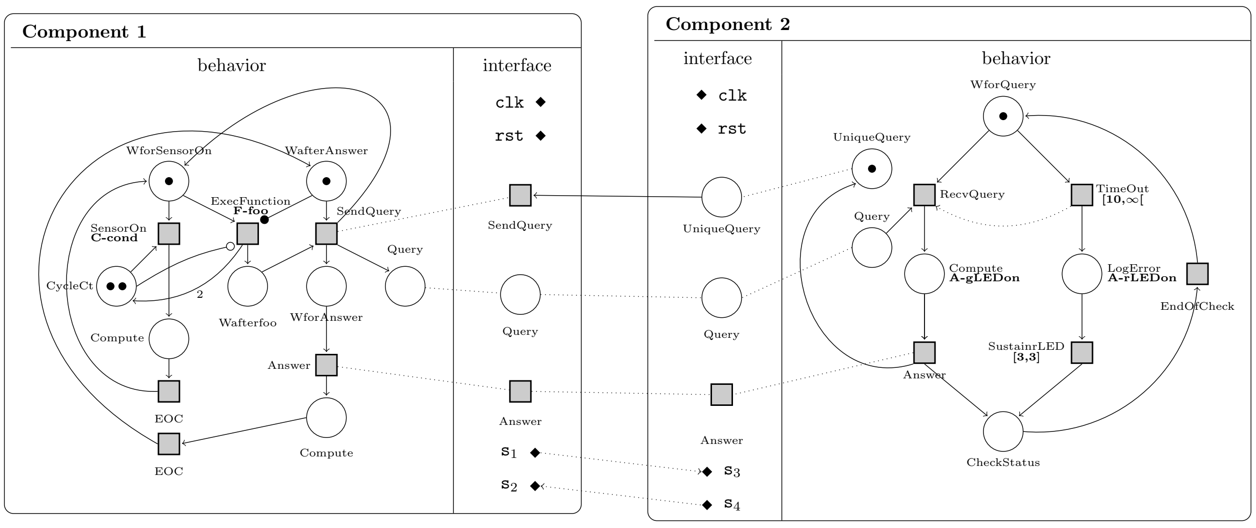

FIGURE 1.3: An Example of HILECOP high-level model. Black diamonds represent VHDL signals.

As shown in Figure 1.3, a component of the HILECOP high-level model formalism is represented by a box having an internal behavior and an interface that permits the connection to other components. The internal behavior of a component is defined with a specific kind of Petri Net (PN) model. These PNs and their distiguishing features will be thoroughly presented in Chapter **??**. The component interface exposes places, transitions, and signals, which are references to the elements of its internal behavior, to the outside so that multiple components can be assembled. Each component has a clock and a reset input port (`clk` and `rst`) in its interface. This is the mark that the HILECOP has been built for the design of synchronous digital systems. To a certain extent, VHDL signals can be integrated to the high-level components to represent a direct wiring between components. A component behavior can also be defined through the composition of other components. In that case, we talk about a composite structure.

Next, in Figure 1.2, the transformation from Step 1 to Step 2 flattens the model. The internal behaviors are connected according to the interface compositions, and embedding component structures are removed. Figure 1.4 gives the result of the flattening phase for the model of Figure 1.3.

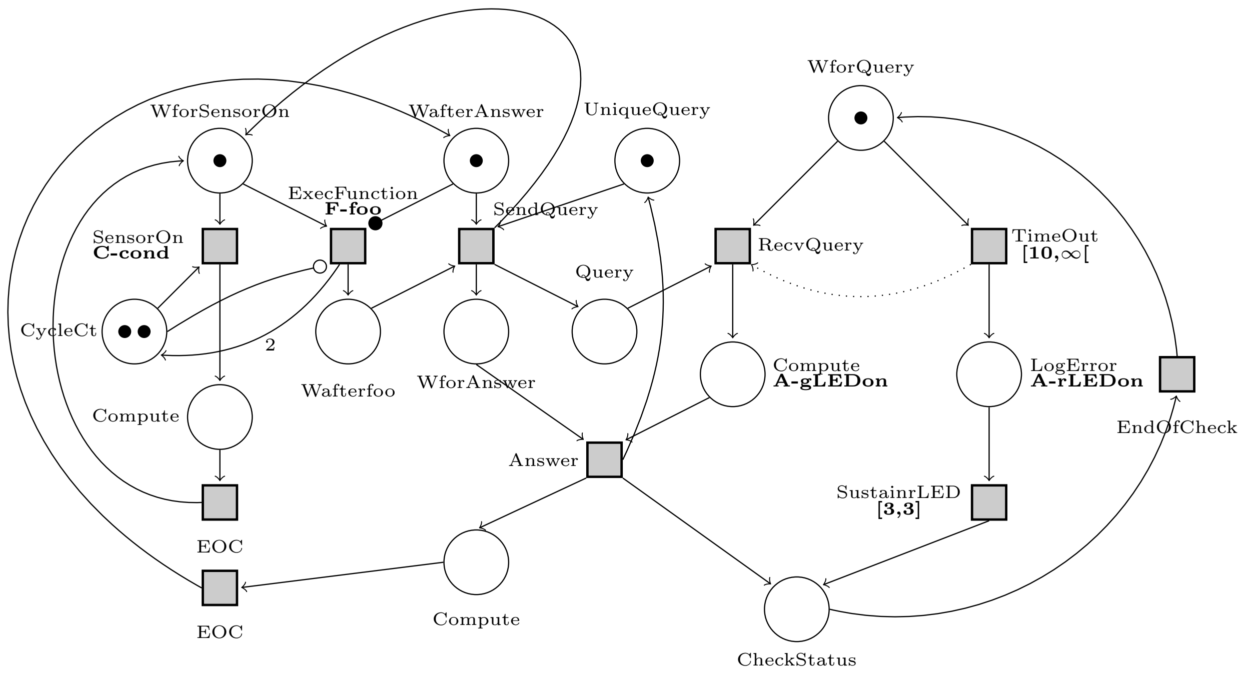

FIGURE 1.4: A global Petri net model obtained after the flattening of a HILECOP high level model.

The PN formalism is a formal model and therefore permits to apply mathematical reasoning on its instances. Particularly, a PN model can be analyzed, and a proof that a given model meet some properties can be automatically produced through the direct analysis of the structure or through the use of model checking techniques. This feature of PNs has been one of the reason of the adoption of this formalism as the basis of the design of critical digital systems. A whole thesis has been dedicated to developed new methods to analyze the HILECOP PN models [11]. In fact, the transformation of the abstract model is a bit different in preparation of the model analysis. The transformation adds new information to the flattened model for the purpose of the analysis. Figure 1.4 only gives the flattened version of the model that is not produced for the analysis but in preparation of the next transformation into VHDL design. The analysis phase is here to convince the engineers that they are designing a safe system. The analysis process is a round trip between Step 1 and Step 2. It aims at producing a model that is conflict-free (see Section **??** for more details about the definition of a conflict), bounded, and deadlock-free, using model-checking techniques.

After several iterations, the model should reach soundness and is then said to be implementation-ready.

From Step 2 to Step 3, VHDL source code is then generated by means of an automatic model-to-text transformation. The generated code describes a VHDL design, i.e. a textual description of a hardware system, which has an interface defining input and output ports and an internal behavior called an architecture. Details about the syntax and the semantics of the VHDL language will be given in Chapter **??**. Figure 1.5 succinctly illustrates the transformation happening between Step 2 and Step 3. The transformation from Step 2 to Step 3 will be thoroughly presented in Chapter. add ref

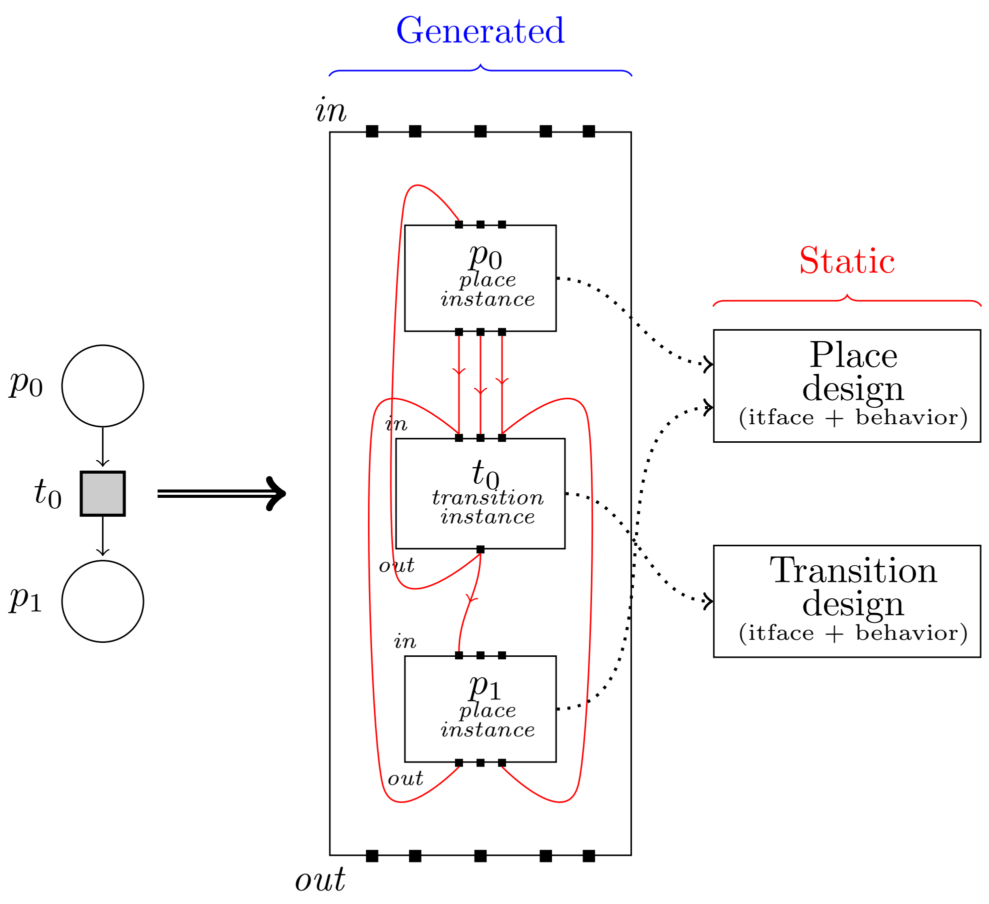
Generated
in
out
Static
p_0
t_0
p_1
p_0 place instance
t_0 transition instance
p_1 place instance
in
out
in
out
Place design (itface + behavior)
Transition design (itface + behavior)

FIGURE 1.5: Generation of a top-level VHDL design from a Petri net. On the left, the input PN and on the right the generated VHDL top-level design. Dotted arrows shows the relation between the component instances and their source design.

For the purpose of the HILECOP methodology, two VHDL designs have been defined: the place design that is a hardware description of a PN place (circle nodes in a PN) and the transition design that is a hardware description of a PN transition (square nodes in a PN). Like all VHDL designs, the place and the transition design have an input and output port interface, and their own internal behavior. A VHDL design is a mould to describe a hardware component. Thus, a design can be instantiated in the behavior of other designs in order to obtain more complex behaviors. As illustrated in Figure, the transformation from Step 2 to Step 3 creates a place design (or component) instance (PCI) and a transition design (or component) instance (TCI) for each place and transition of the input Petri net. Then, the PCIs and TCIs are connected together through their input and output port interfaces. These connections mimic the arc connections between the places and the transitions of the input PN.

From Step 3 to Step 4, the VHDL compilation/synthesis and the FPGA programming are finally performed using industrial tools. At the end of Step 4, the designed circuit is physically built on an FPGA device. What happens between Step 3 and Step 4 appears as a black box in the whole HILECOP methodology. Therefore, we are not interested in detailing this transformation phase.

1.3 Verifying the HILECOP methodology

The HILECOP methodology is useful to design and implement critical digital systems. The use of Petri nets as the base model of the process is one of its major advantage. All the analysis tools that accompany the Petri net formalism and that permit to prove the safety properties of the models qualify the HILECOP methodology as a formal method for the design and implementation of critical digital systems. However, even with input models that are proved to be safe and sound, the advantages provided by the use Petri nets would be lost if one of the transformation happening during the process changed the input definition of the circuit in a way that would alter its behavior. Thus, the engineers would have specified a perfectly correct digital system but would never obtain the expected circuit on a physical FPGA device. Therefore, in order to reinforce the confidence in the HILECOP methodology as a reliable formal method to produce critical digital systems, the goal of this thesis is to verify, i.e. to establish the formal proof, that the model-to-text transformation from Step 2 to Step 3 (i.e. the framed part with red dotted lines in Figure 1.2) preserves the behavior of input models into the generated VHDL designs. We choose to carry out this task as a verification task (see Section **??** for an explanation on the difference between validation and verification tasks). It means that we want to prove a behavior preservation theorem stating that: for all input models of our transformation the generated output designs act similarly at the execution time. Chapter **??** formally presents our behavior preservation theorem, and thus, what we mean about the similarity of execution between our input and output representation.

One could argue that to qualify the entire HILECOP methodology, one has to verify all the transformations happening in the methodology, i.e. consider also the transformation from Step 1 to Step 2, and the transformation from Step 3 to Step 4. In our defence, we shall say that:

- The transformation from Step 1 to Step 2 changes the structure of the component-based input model. However, if we had to set a formal semantics for the HILECOP high-level models then it would probably be through the definition of the global PN obtained by transformation (i.e. a kind of translational semantics). Therefore, we argue that there is no need to verify that the transformation from Step 1 to Step 2 is semantic preserving.
- The transformation from Step 3 to Step 4 is performed by industrial tools. We rely on these tools because they are widely used in the industry for the development of critical systems.

Now that we have clarified the nature of the verification task we want to achieve, we can state our research question as follows:

CAN WE PROVE THAT THE MODEL-TO-TEXT TRANSFORMATION DESCRIBED IN THE HILECOP METHODOLOGY IS SEMANTIC PRESERVING?

Even though answering this question is interesting, however, its formulation as a question that one can answer only by yes or no does not stimulate the scientific interest of verification task. However, our verification task resembles to the verification of compilers for programming languages. Compiler verification has been widely explorated, and many works are accessible in the literature [5]. The major source of inspiration of this thesis has been the work done on the CompCert certified C compiler [9]. Thus, we argue here that the scientific interest of our research comes from the comparison between the methods used to perform our verification task and the methods used to perform similar verification task in other domains such as compiler verification. Thus, we can complement our research question with the following ones:

- What are the similarities and the differences between the HILECOP transformation and other transformation situations (compilers, model transformations...)?
- Is there a strategy to perform the verification of the HILECOP transformation?
- How far the correspondence holds between this strategy and the strategy used in other transformation situations such as compiler verification?

To achieve the formal verification of HILECOP, our approach is similar to what has been done for the CompCert compiler. The idea is to formalize the semantics of the source and target languages, and verify that the transformation preserves the semantics of any input model. In the thesis, we propose both to perform the formalization work on "paper" and to mechanize it within the Coq proof assistant [3].

In the case of HILECOP, some specificities of the source and target languages introduce additional technical difficulties in the process of formal verification. A first difference pertains to HILECOP's high-level formalism (the input language), which is quite abstract. This formalism depends on PNs, and thus is not a common programming language.

A second difference is about the VHDL language (the output language). Similarly to the PN models used in HILECOP, the VHDL language is not a common programming language as its purpose is both the structural and behavioral description of hardware circuits. Although previous work has been conducted toward the formalization of the VHDL semantics [8], a semantics that is able to both handle all the constructs in the generated programs, and facilitate the proof of behavior preservation, still needs to be designed.

To further motivate the necessity of the verification task, the development of neuroprotheses by the INRIA CAMIN team is at the base of the creation of the Neurrinov company[1]. The Neurrinov company is now looking towards the industrial development of such neuroprotheses. We hope that once the verification performed on the HILECOP methodology, it will help to obtain the CE certification[2] necessary to qualify the neuroprotheses as eligible for the medical market.

Moreover, the HILECOP methodology comes with a working implementation based on the Eclipse framework. This sofware is currently used by the engineers of the Neurinnov company to design the digital systems having a part in the neuroprotheses. Figure 1.6 gives a view of the existing HILECOP software.

FIGURE 1.6: A view of the HILECOP software implemented on top of the Eclipse framework.

To the purpose of formal verification, we will implement the HILECOP model-to-text transformation leveraging the functional language of the Coq proof assistant. However, after the mechanization of the proof of semantic preservation, we could use the extraction feature of the Coq proof

[1]http://neurinnov.com/

[2]http://data.europa.eu/eli/reg/2017/745/2020-04-24

assistant to produce the implemented transformation as an OCaml program. Then, we will able to connect this program to the existing HILECOP software in order to use the verified version of the transformation.

Bibliography

[1] David Andreu, David Guiraud, and Guillaume Souquet. “A Distributed Architecture for Activating the Peripheral Nervous System”. In: *Journal of Neural Engineering* 6.2 (Apr. 1, 2009), p. 026001. ISSN: 1741-2560, 1741-2552. DOI: 10.1088/1741-2560/6/2/026001. URL: https://iopscience.iop.org/article/10.1088/1741-2560/6/2/026001 (visited on 06/08/2020).

[2] Peter J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, Oct. 7, 2010. 933 pp. ISBN: 978-0-08-056885-0. Google Books: XbZr8DurZYEC.

[3] Yves Bertot and P. Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Berlin ; New York: Springer, 2004. 469 pp. ISBN: 978-3-540-20854-9.

[4] David C. Black et al. *SystemC: From the Ground Up, Second Edition*. Springer Science & Business Media, Dec. 18, 2009. 291 pp. ISBN: 978-0-387-69958-5. Google Books: Op2a6yiO9jwC.

[5] Maulik A. Dave. “Compiler Verification: A Bibliography”. In: *ACM SIGSOFT Software Engineering Notes* 28.6 (Nov. 1, 2003), p. 2. ISSN: 01635948. DOI: 10.1145/966221.966235. URL: http://portal.acm.org/citation.cfm?doid=966221.966235 (visited on 05/22/2020).

[6] 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. Google Books: Ze60AwAAQBAJ.

[7] David Guiraud et al. “An Implantable Neuroprosthesis for Standing and Walking in Paraplegia: 5-Year Patient Follow-Up”. In: *Journal of Neural Engineering* 3.4 (Sept. 2006), pp. 268–275. ISSN: 1741-2552. DOI: 10.1088/1741-2560/3/4/003. URL: https://doi.org/10.1088/1741-2560/3/4/003 (visited on 06/11/2021).

[8] Carlos Delgado Kloos and P. Breuer. *Formal Semantics for VHDL*. Springer Science & Business Media, Dec. 6, 2012. 263 pp. ISBN: 978-1-4615-2237-9. Google Books: eJ3xBwAAQBAJ.

[9] 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. URL: https://doi.org/10.1007/s10817-009-9155-4 (visited on 01/21/2020).

[10] David Long and Zane Scott. *A Primer for Model-Based Systems Engineering*. Lulu.com, 2011. 126 pp. ISBN: 978-1-105-58810-5. Google Books: pCaoAwAAQBAJ.

[11] 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. URL: https://tel.archives-ouvertes.fr/tel-01704776 (visited on 02/10/2020).

[12] Gordon E. Moore. “Cramming More Components onto Integrated Circuits, Reprinted from Electronics, Volume 38, Number 8, April 19, 1965, Pp.114 Ff.” In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (Sept. 2006), pp. 33–35. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2006.4785860.

[13] Carl Adam Petri. “Kommunikation mit Automaten”. In: *http://edoc.sub.uni-hamburg.de/informatik/volltexte/20 petri.pdf* (1962). URL: https://edoc.sub.uni-hamburg.de//informatik/volltexte/2011/160/ (visited on 06/10/2021).