Toward the Formal Verification of HILECOP: Formalization and Implementation of Synchronously Executed Petri Nets*

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

[1] LIRMM, Univ Montpellier, CNRS, Montpellier, France
[2] NEURINNOV, Montpellier, France
Firstname.Lastname@lirmm.fr

While designing critical digital systems, the use of formal models is necessary as they help us to assess the design soundness with the help of their mathematical foundations. However, the formal model is then frequently hand-coded by the engineer, sometimes with automatic generation of a code skeleton to be completed. While automatic generation facilitates the task and limits the risk of error, it remains to be proved that the properties highlighted in the design model are preserved in its implemented version, while considering the impact of its execution on the target as well.

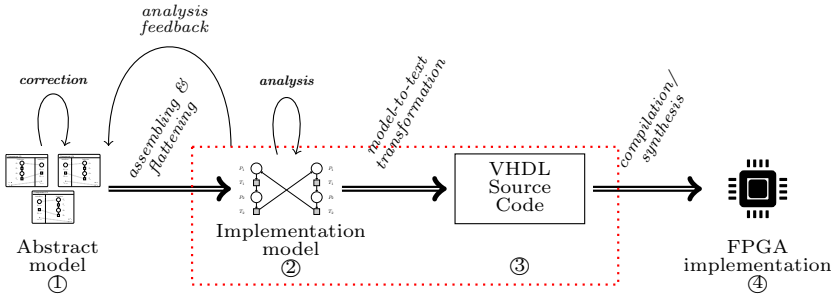

Fig. 1: Workflow of the HILECOP Methodology

We propose to address the problem within the framework of the HILECOP (HIgh-LEvel hardware COmponent Programming) methodology. As shown in Fig. 1, the HILECOP methodology describes a full fledged process to design, analyze, and synthesize critical digital systems. In HILECOP, the models of digital systems are built leveraging a modular, component-based graphical formalism (see ① in Fig. 1). The internal behavior of the model components are described with a particular kind of Petri nets that we call SITPNs (Synchronously executed Interpreted Time Petri Nets with priorities). The SITPN formalism allows us to analyze the models (see ② in Fig. 1). Up to that point, the HILECOP

methodology provides a way to design sound models that ensure properties such as boundedness and liveliness. From Step ② to Step ③, the SITPN model is then transformed into VHDL code. Currently, there is no proof that the structure and behavior of the implementation model are preserved in the generated VHDL code. Using theorem proving and the Coq proof assistant [4] in particular, we propose to formally verify the above statement.

This verification task is somewhat similar to the verification of compilers for programming languages (see [2], for an example). In this kind of verification, the steps to establish the proof of behavior preservation are well-known: first, formalize and implement the semantics of the source and the target languages; second, implement the transformation; finally, prove that for all source program, the generated program has the same behavior. However, contrary to usual programming languages, our source formalism is very abstract, and the target language is very specific, as VHDL describes both the structure and the behavior of hardware circuits. These particular aspects bring a certain originality to our work.

To complete the verification of HILECOP, the first step consists in formalizing the semantics of SITPNs, and implementing it in Coq. The SITPN semantics is a state-transition system. It has been thoroughly formalized in [1,3]. We provided only minor changes to this semantics, most of them to correct redundancies. SITPNs combine multiple classes of PNs such as time and interpreted PNs. However, the singularity of SITPNs resides in the synchronous execution of the models. The evolution of the overall state of an SITPN depends on the two events of a clock signal: the falling edge and the rising edge. These two events label the transitions of the state-transition system. The election of the transitions to be fired takes place on the falling edge and the update of the marking takes place on the rising edge. The major impact of synchronous execution is that all transitions are fired at the same time.

Our first contribution to the verification of HILECOP has been to implement the SITPN semantics in Coq. To increase our confidence in our implementation of the SITPN semantics, we have also developed an SITPN token player in Coq and proved that it is sound and complete with respect to the SITPN semantics. The full formalization and proof are available to the reader[3].

References

1. H. Leroux. Handling Exceptions in Petri Net-Based Digital Architecture: From Formalism to Implementation on FPGAs. *IEEE Transactions Industrial Informatics*, 11(4):897–906, Aug. 2015.
2. X. Leroy. Formal Verification of a Realistic Compiler. *Communications of the ACM (CACM)*, 52(7):107–115, July 2009.
3. 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.
4. The Coq Development Team. *Coq, version 8.11.0.* Inria, Jan. 2020. `http://coq.inria.fr/`.

[3] `https://github.com/viampietro/sitpns`