Formal Verification of Synchronization Issue in System-Level Design with Automatic Abstraction

Thanyapat Sakunkonchak and Masahiro Fujita
Department of Electronic Engineering, University of Tokyo
7-3-1, Hongo, Bunkyo, Tokyo, 113-8656, Japan
Phone:+(81)-3-5841-6764, Fax:+(81)-3-5841-6724
E-mail: thong@cad.t.u-tokyo.ac.jp, fujita@ee.t.u-tokyo.ac.jp

Extended Abstract

Semiconductor technology has been growing rapidly, and entire systems can be realized on single LSIs as embedded systems or System-on-a-Chip (SoC). Designing SoC is a process of the whole system design flow from specification to implementation which is also a process of both hardware and software development. Concurrency is becoming common-exist in describing a system design, both from the hardware and software aspects. The collaboration of parallel execution of behaviors/processes is fundamental to meet the design requirement and such collaboration is properly accomplished by realizing with the synchronization of those behaviors/processes. In a system design, synchronization might be oftenly exist and distributed throughout the design. Verifying the synchronization correctness in such a case may be difficult and significantly time-consuming.

The size of the design is one of the major problems in the verification field. When the design becomes larger, it is usually unable to verify due to the memory explosion or the limitation of the capability of the verification tools. Boolean programs [2] are the software verification technique that use the idea of the abstraction of the original program codes, namely those in which all variables and parameters have boolean type. Verification of the boolean programs, with the fact that the boolean programs are the subsets of the original ones, if the results are satisfied, we can directly imply that the original program codes are also satisfied.

SpecC has been proposed as the standard system-level design language based on C programming language which covers the design levels from specification to behaviors. It can describe both software and hardware seamlessly and a useful tool for rapid prototyping as well. Recently the semantics of SpecC has been reviewed and clarified.

In this paper, we develop and demonstrate a technique for the verification of synchronization issues in SpecC language. The proposed technique applies the idea of the boolean program. The abstracted SpecC code (to avoid confusion with the boolean programs, let us called ‘boolean SpecC’) is containing the statements that can be expressed in terms of inequalities of timing of those statements $(Tas \leq T1s < \ldots)$. Hence, we can make use of the difference decision diagrams (DDDs) [1], a kind of the decision diagram which can represent the inequalities efficiently, in order to verify the synchronization issues of the SpecC programs. SpecC programs are firstly parsed and translated into the boolean SpecC (the boolean programs which are generated from SpecC), then, translate those boolean SpecC into DDD graphs. The idea here is to abstract any conditions in `if` statements of the original programs with user-defined *predicates* and translate them into boolean domain. All statements other than event manipulation and conditions for `if` or `switch`, and so on, are removed (or abstracted away). Thus only boolean variables and event manipulation statements remain in the generated boolean programs. Here we use boolean programs as a kind of abstracted descriptions from original SpecC descriptions and verify them with DDDs, concentrating on verification of only synchronization issues in SpecC descriptions. Boolean variables are generated based on user-defined *predicates*, which define *abstraction functions* in verification process. Right now we are just assuming that *predicates* are given by designers (who are describing their designs in SpecC), but in the future we plan to develop automatic generation of *predicates* as well.

When verifying the synchronization of SpecC with DDDs, if the result turns out to be true, then verification terminates and the synchronization is satisfied. When the result is false, however, the counter-example must be provided. This counter-example gives the trace back to the unsatisfied source in the original program. The idea for the automatically refinement of the predicates is proposed. We believe that the implementation of this refinement of predicates would help the designers to save a lot of time to find errors.

References

[1] J. Mϕller, J. Lichtenberg, H. R. Anderson, and H. Hulgaard, “Difference Decision Diagrams,” *Technical report IT-TR-1999-023, Department of Information Technology, Technical University of Denmark.*, February 1999.
[2] T. Ball and S. K. Rajamani, “Boolean Programs: A Model and Process For Software Analysis,” Microsoft Research, http://research.microsoft.com/slam