Processor Datapath Verification with SPASS

Thomas Hillenbrand Carsten Ihlemann

Max-Planck-Institut für Informatik, D-66123 Saarbrücken
{hillen,ihlemann}@mpi-sb.mpg.de

In the context of the VERISOFT project, we are currently verifying the datapath of a simple processor the architecture of which is regularly taught in computer science lectures at Saarbrücken [MP98]. The processor is a 32-bit RISC machine and similar in design to the well-known DLX of [HP90], but uses neither caching nor pipelining at the moment.

The specification is given via the semantics of a machine language that includes data transfer between memory and registers, arithmetical and logical operations, branching and jumping. Instructions are related to bit patterns on the specification level already. The hardware implementation is presented as a circuit diagram with registers and memory as building blocks.

Actually one might think that processor correctness, being an inductive property, were out of scope for a first-order theorem prover like SPASS [WBH$^+$02]. In our setting, however, this boils down to showing the following: Provided specification and implementation machine coincide in the contents of memory, registers and program counters, they will still do so after one step of execution. We started with some pen-and-paper proofwork to reassure ourselves that the proof obligations were really deductive, i. e. first-order, and knew 13 pages later that in fact they were, except for some properties of the type bitvector which is occuring frequently to model addresses, register contents and the like.

To formalize these bitvectors adequately was the first major challenge, with the need to capture inductive properties like associativity of append. Attempts with lists or even words over 0 and 1 were not very convincing because SPASS enumerated, with any bitvector pattern, commutation properties of append with other operations. Array-style or function-style axiomatizations have only been considered theoretically, because the idea of using n-place functions for vectors of length n led to a clause set which was quickly saturated. The reason behind this is that commutation properties only need to be established for bitvectors of fixed lengths; and in our application, there are indeed only finitely many different lengths which are known in advance.

Another important point is how to deal with definitions that contain case distinctions. For example, the second operand of arithmetical operations can be taken from either the instruction word or some register, which translates into

$$\mathrm{rop}(c) := \begin{cases} \mathrm{sxt}(\mathrm{imm}(c)) & \text{if comp.imm}(c) \\ c\,.\mathrm{GPR}[\mathrm{RS2}(c)] & \text{otherwise} \end{cases}$$

With a straightforward encoding, this single definition will be broken into two clauses during clausification, which is approach (I). Alternatively (II), one can define (fragments of) Boolean algebra on the term level up to an if-then-else-operator, such that the definition fits into a single clause. We are pursuing both approaches practically. Approach (II) more generally aims at axiomatizations that can finitely be saturated, which on the long run can be understood as pre-processing important theories to achieve much more efficiency. Some form of disproving might become possible as well this way.

As explained, the correctness theorem is broken down into three lemmata on preservation of contents equivalence, namely for memory, registers and program counters. For each of these lemmata, one has to prove that two particular expressions with case distinctions are equal, where the number of cases reaches up to six. Note that one has to show not only that the case results are equal, but also that the case conditions imply each other, which can become fairly non-trivial. In the pen-and-paper proof, these three lemmata are supported by a corpus of 15 propositions. Our current state is that the proof can be done automatically with this granularity in approach (I).

It would be nice if the propositions could be found automatically when proving the lemmata; but it is not clear to us yet wether this is realistic. If so, the next step would be to have the interactive theorem prover ISABELLE generating SPASS proof obligations for this application domain, incorporating the experiences from this case study. Here it is open wether our results will be sufficiently general.

Regarding previous work on hardware verification with first-order provers, the HWV domain of TPTP contains problems e. g. on gate construction [WOLB92] or on some FIFO buffer implementation [CHM02], but it seems as if the datapath level had not been reached yet. The general problem with first-order reasoning here is, however, that we are always dealing with a bunch of models, whereas in the application there is only one. Using bitvector decision procedures in some combination scheme might be a remedy.

[CHM02] K. Claessen, R. Hähnle, and J. Mårtensson. Verification of hardware systems with first-order logic. In G. Sutcliffe, J. Pelletier, and C. Suttner, editors, *Proceedings of the CADE-18 Workshop on Problems and Problem Sets for ATP*, number 02/10 in Department of Computer Science, University of Copenhagen, Technical Report, 2002.

[HP90] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, 1990.

[MP98] S. M. Mueller and W. J. Paul. *Computer Architecture: Complexity and Correctness.* Springer-Verlag, 1998.

[WBH$^+$02] Chr. Weidenbach, U. Brahm, Th. Hillenbrand, E. Keen, Chr. Theobald, and D. Topić. SPASS version 2.0. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, volume 2392 of *LNAI*, pages 275–279. Springer-Verlag, 2002.

[WOLB92] L. Wos, R. Overbeek, E. L. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications.* McGraw-Hill, 2nd edition, 1992.