Intel® Core™ i7 Processor Execution Engine Validation in a Functional Language Based Formal Framework

Roope Kaivola

Intel Corporation, JF4-451, 2111 NE 25th Avenue, Hillsboro, OR 97124, USA

Abstract. Formal verification of microprocessor components has been pursued in Intel processor development projects in various forms for over a decade. Usually formal verification has been used to supplement more traditional coverage oriented testing activities. For the Intel® Core™ i7 design we took a step further and used formal verification as the primary validation vehicle for the core execution cluster, the component responsible for the functional behaviour of all microinstructions. We applied symbolic simulation based formal verification techniques for full datapath, control and state validation for the cluster, and dropped coverage driven testing entirely [2]. The project, involving some twenty person years of verification work, is one of the most ambitious formal verification efforts in the hardware industry to date, and shows that under the right circumstances, full formal verification of a major design component is a feasible, industrially viable and competitive validation approach.

Technically the verification work was carried out in the Forte verification framework, originally built on top of the Voss system [1]. It is based on a strongly typed ML-like [4] lazy functional programming language *reFlect*. Most of the verification code is written in reFlect: specifications, whether they are functional specifications or relational constraints, verification facilities, analysis routines etc. The execution of an individual verification task in the framework amounts to the evaluation of a reFlect program, and the entire verification initiative involves significant software engineering aspects [3]. In the reFlect language binary decision diagrams are first-class objects: the type Bool includes not just the constants T and F, but arbitrary BDD's. For verification purposes, a very important feature of the language is that it allows *symbolic evaluation* of objects containing BDD's and symbolic circuit simulation using BDD's. Similar facilities exist for non-canonical graph representations of Booleans, used for interfacing with satisfiability solvers.

References

1. Hazelhurst, S., Seger, C.-J.H.: Symbolic trajectory evaluation. In: Kropf, T. (ed.) Formal Hardware Verification. LNCS, vol. 1287, pp. 3–78. Springer, Heidelberg (1997)
2. Kaivola, R., et al.: Replacing testing with formal verification in Intel Core i7 processor execution engine validation. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 414–429. Springer, Heidelberg (2009)
3. Kaivola, R., Kohatsu, K.: Proof engineering in the large: formal verification of Pentium® 4 floating-point divider. Int'l J. on Software Tools for Technology Transfer 4, 323–334 (2003)
4. Paulson, L.: ML for the Working Programmer. Cambridge University Press, Cambridge (1996)