T. Melham and J. O'Leary, 'A Functional HDL in *ReFLect*', in M. Sheeran and T. Melham (eds.),
Sixth International Workshop on Designing Correct Circuits: Vienna, 25–26 March 2006: Participants' Proceedings, ETAPS 2006 (Vienna, 2006).

A Functional HDL in ReFLect

TOM MELHAM
Computing Laboratory
Oxford University
Wolfson Building
Parks Road
Oxford, OX2 3QD, England

JOHN O'LEARY
Strategic CAD Labs
Intel Corporation
Mail Stop JF4-211
2111 NE 25th Avenue
Hillsboro, OR 97124-5961, USA

ReFLect [4] is a functional programming language designed and implemented at Intel's Strategic CAD Labs under the direction of Jim Grundy. The language is strongly typed and similar to ML, but provides certain *reflection* features intended for applications in industrial hardware design and verification. Like LISP, *reFLect* has quotation and antiquotation constructs that may be used to construct and decompose expressions in the language itself. Unlike LISP, these mechanisms are typed. The language also provides a primitive mechanism for pattern-matching, and in particular for defining functions over code by pattern-matching on the structure of *reFLect* expressions. The design of *reFLect* draws on the experience of applying an earlier reflective language called *FL* [1] to large-scale formal verification problems within Intel's Forte framework [8].

One of the intended roles of *reFLect* is to be the host language for a functional HDL. As with other work based on Haskell [2, 7] or LISP [5, 6], a key requirement is the ability to simulate hardware models by program execution. Circuit descriptions are just functional programs, which we can simply run to simulate the circuits on test case inputs. But in addition to this simulation capability, we also wish to execute various operations on the abstract *syntax* of circuit descriptions written in the language. We want to be able to write programs that 'see' the code of a circuit description. This allows us, for example, to program circuit design transformations [10] as functions that traverse code—or simply to generate netlists for other design tools.

This talk at DCC 2006 will illustrate how the reflection features of *reFLect* can provide both simulation and a handle on circuit structure (intensional analysis) within a single, unified language framework. We will present a small HDL embedded within *reFLect*. In the spirit of the approach pioneered by Sheeran in μFP [9], circuit descriptions are built up in this HDL from primitives using higher-order functions that implement various ways of composing sub-circuits together. The reflection features of *reFLect* will then be employed to allow a single-source circuit description in this language both to be executed for simulation and to be executed to generate circuit netlists.

Lava [2] is an HDL based on Haskell that also supports simulation by execution and circuit netlist generation with a single functional source. Lava achieves this using non-standard interpretation, laziness, and functional data structures with 'observable sharing' [3]. Our presentation at DCC will show how the reflection features of *reFLect* can be employed to achieve single-source simulation and netlist generation in another, perhaps more direct, way. We will also offer some speculations on capabilities available with our approach that seem beyond what can be achieved in Lava—for example the syntactic analysis of circuit descriptions before flattening into netlists.

References

[1] Mark D. Aagaard, Robert B. Jones, and Carl-Johan H. Seger. *Lifted-FL*: A pragmatic implementation of combined model checking and theorem proving. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics: 12th International Conference, TPHOLs 1999*, volume 1690 of *LNCS*, pages 323–340. Springer, 1999.

[2] Per Bjesse, Koen Claessen, Mary Sheeran, and Satnam Singh. Lava: Hardware design in Haskell. In *Functional Programming: International Conference, ICFP 1998*, pages 174–184. ACM, 1998.

[3] Koen Claessen and David Sands. Observable sharing for functional circuit description. In *Advances in Computing Science: 5th Asian Computing Science Conference, ASIAN 1999*, pages 62–73. Springer, 1999.

[4] Jim Grundy, Tom Melham, and John O'Leary. A reflective functional language for hardware design and theorem proving. *Journal of Functional Programming*, 16(2):157–196, March 2006.

[5] Steven D. Johnson. *Synthesis of Digital Designs from Recursion Equations*. MIT, 1984.

[6] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore, editors. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer, 2000.

[7] John Matthews, Byron Cook, and John Launchbury. Microprocessor specification in Hawk. In *Computer Languages: International Conference*, pages 90–101. IEEE Computer Society, 1998.

[8] Carl-Johan H. Seger, Robert B. Jones, John W. O'Leary, Tom Melham, Mark D. Aagaard, Clark Barrett, and Don Syme. An industrially effective environment for formal hardware verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(9):1381–1405, September 2005.

[9] Mary Sheeran. *μFP: An Algebraic VLSI Design Language.* PhD thesis, University of Oxford, 1983.

[10] Greg Spirakis. Leading-edge and future design challenges: Is the classical EDA ready? In *Design Automation: 40th ACM/IEEE Conference, DAC 2003*, page 416. ACM, 2003.