An Embedded Hardware Description Language using Dependent Types

João Paulo Pizani Flor <j.p.pizaniflor@students.uu.nl>
Wouter Swierstra <w.s.swierstra@uu.nl>

Department of Information and Computing Sciences
Utrecht University - The Netherlands

Sunday 14th December, 2014

Extended abstract

Computer hardware has experienced a steady exponential increase in complexity in the last decades. As we approach the physical limits of the chip manufacturing process, the performance of computing systems cannot be increased anymore just by making CPUs have higher frequency or more parallelism. There is an increased demand for application-specific integrated circuits that avoid the proverbial *von Neumann bottleneck* [1], and ever more algorithms enjoy hardware acceleration (such as 3D/2D renderers, video/audio codecs, cryptographic primitives and network protocols).

This demand puts pressure on the industry to make hardware design quicker and more efficient. At the same time, hardware design also requires very strong correctness guarantees: it is much harder to recall a batch of mass-produced circuits than to push a software update. These strong correctness requirements for hardware are commonly met using (exhaustive) testing and model checking, making the design-verify-fix loop also more costly than in the software world.

There is a long tradition [3, 4] of modeling hardware using functional programming to pursue higher productivity and strong correctness assurance. A particular trend in this direction is to use functional programming languages to *host* Embedded Domain Specific Languages (EDSLs) aimed at hardware description. One popular example is Lava [2], a hardware EDSL hosted by Haskell. Circuits modeled in Lava can be simulated, compiled to VHDL netlists, and verified for safety properties by calling an external SAT-solver.

We believe that the advantages brought to hardware design by FP-inspired techniques can be even greater if we use dependent types. Our project defines a Hardware Description Language (HDL) *embedded* in the dependently-typed general-purpose programming language Agda.

With the additional expressiveness provided by a dependent type system, we can have more static guarantees about circuit behaviour specified *in the circuit's type*. Therefore, certain classes of design mistakes (mismatching sizes, short-circuits, etc.) can be eliminated *by construction*, i.e, "wrong" circuits simply *cannot be built*. Additionally, in a dependently-typed setting, we can *interactively* prove properties of circuits, which provides some advantages over the fully-automated verification approach used widely in industry. For example, in Agda, we can prove (using induction) properties over whole *circuit generators*.

More specifically, with an HDL embedded in Agda, we can write *functional specifications* of circuit behaviour and prove that a certain circuit *implements* that specification. For instance, we could define the *functional specification* of adders to be a certain Agda function called `add` (which operates over machine integers). Then we can proceed to prove that different versions of adder circuits all *implement the behaviour* specified by `add`. Hardware designers often start with a simple (but inefficient) design, and successively add optimizations to achieve better performance. Using our embedded HDL, a designer can be *provably sure* that, at each optimization step, the *architecture* (and maybe performance) of the circuit changes, but the *functional behaviour* remains the same.

We use a deep-embedded representation for circuits, and the modelings is done at *architectural* level. Therefore, besides simulation, we also aim to able to extract *netlists* (in VHDL) from the our internal circuit representation. This extracted VHDL can be used as "entry point" for other tasks in the design chain, such as timing analysis, synthesis, etc.

References

[1] John Backus. Can programming be liberated from the von neumann style?: a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[2] Per Bjesse, Koen Claessen, Mary Sheeran, and Satnam Singh. Lava: hardware design in Haskell. *SIGPLAN Not.*, 34(1):174–184, September 1998.

[3] Peter Gammie. Synchronous digital circuits as functional programs. *ACM Computing Surveys (CSUR)*, 46(2):21, 2013.

[4] Mary Sheeran. Hardware design and functional programming: a perfect match. *J. UCS*, 11(7):1135–1158, 2005.