Π-Ware: An Embedded Hardware Description Language using Dependent Types

Author: João Paulo Pizani Flor
<joaopizani@uu.nl>

Supervisor: Wouter Swierstra
<w.s.swierstra@uu.nl>

Department of Information and Computing Sciences
Utrecht University

Saturday 23rd August, 2014

Universiteit Utrecht

Table of Contents

Hardware design is hard(er)

- Strict(er) correctness requirements
  - You can't simply *update* a full-custom chip after production
    - Intel `FDIV`
  - Expensive verification / validation (up to 50% of development costs)
- Low-level details (more) important
  - Layout / area
  - Power consumption / fault tolerance

Universiteit Utrecht

Hardware design is growing

- Moore's law will still apply for some time
  - We can keep packing more transistors into same silicon area
- **But** optimizations in CPUs display diminishing returns
  - Thus, more algorithms *directly* in hardware

Universiteit Utrecht

Hardware Description Languages

- All started in the 1980s
- *De facto* industry standards: VHDL and Verilog
- Were intended for *simulation*, not modelling or synthesis
  - *Unsynthesizable* constructs
  - Widely variable tool support

Functional Programming

- Easier to *reason* about program properties
- Inherently *parallel* and *stateless* semantics
  - In contrast to imperative programming

Universiteit Utrecht

Functional Hardware Description

- A functional program describes a circuit
- Several *functional* Hardware Description Languages (HDLs) during the 1980s
  - For example, μFP [Sheeran, 1984]
- Later, *embedded* hardware Domain-Specific Languages (DSLs)
  - For example, Lava (Haskell) [Bjesse et al., 1998]

Embedded DSLs for Hardware

- Lava

Universiteit Utrecht

Dependently-Typed Programming

Dependently-Typed Programming (DTP) är en programmationstechnik...

Universiteit Utrecht

Research Question

"What are the improvements that DTP can bring to hardware design?"

Universiteit Utrecht

Methodology

- Develop a hardware DSL, *embedded* in a dependently-typed language (Agda)
  - Called **Π-Ware**
  - allowing simulation, synthesis and verification

Universiteit Utrecht

Dependently-Typed Programming

- Types can depend on values
  - Vec
- Types of arguments can depend on *values of previous arguments*
  - take

Universiteit Utrecht

Dependently-Typed Programming

- Dependent pattern matching
  - Example with Vec pattern forcing size pattern
- Programming language / Theorem prover
  - Types as propositions, terms as proofs [Wadler, 2014]
  - Example: $\_\leq\_$ and $3 \leq 4$.

Universiteit Utrecht

Agda syntax for Haskell programmers

- Liberal identifier lexing (Unicode **everywhere**)
  - a≡b+c is a valid identifer, $a \equiv b + c$ an expression
- *Mixfix* notation
  - _[_]:=_ is the array update function: arr [# 3] := true.

Agda syntax for Haskell programmers

- Implicit arguments
- "For all" sugar: $\forall\ n$ is equivalent to $(n\ :\ \_)$
  - Where _ means: guess this type (based on other args)
  - Example: $\forall\ n \rightarrow \mathsf{zero} \leq n$

Low-level circuits

- $\mathbb{C}'$
- “Untyped”

Atoms

- PiWare.Atom.Atomic
- Bool, `std_logic`, etc.
- Example: PiWare.Atom.Bool

Universiteit Utrecht

Gates

- PiWare.Gates.Gates
- Examples:
  - {NOT, AND, OR} (BoolTrio)
  - {NAND}
  - Arithmetic, Crypto, etc.
- Example: PiWare.Gates.BoolTrio

High-level circuits

- ℂ
- “Typed”

Universiteit Utrecht

Synthesizable

- ⇓W⇑ (pronouced `Synthesizable`)
  - $\mathsf{W}\ n = \mathsf{Vec}\ \alpha\ n$
- Example: $\Downarrow\mathsf{W}\Uparrow\ (\alpha \times \beta)$

Synthesis

- Work-in-progress
- Atom and Gates with VHDL *abstract syntax*

Universiteit Utrecht

Simulation

- Combinational
- Sequential

Examples

- `AndN`

Universiteit Utrecht

Problems

- Definition of $[\![\_]\!]$ blocks reduction

Universiteit Utrecht

Conclusion

Lorem ipsum...

Universiteit Utrecht

Future work

Lorem ipsum...

Universiteit Utrecht

Thank you!

Questions?

Universiteit Utrecht

References I

Bjesse, P., Claessen, K., Sheeran, M., and Singh, S. (1998).

Lava: hardware design in Haskell.
SIGPLAN Not., 34(1):174–184.

Sheeran, M. (1984).
MuFP, a language for VLSI design.
In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, LFP '84, pages 104–112, New York, NY, USA. ACM.

Wadler, P. (2014).
Propositions as types.
Unpublished note, `http://homepages.inf.ed.ac.uk/wadler/papers/propositions-as-types/propositions-as-types.pdf`.

Universiteit Utrecht