Computer Science Project Proposal

Building HDL abstractions from an RTL model in Haskell

Alex Horsman, Homerton College

11th October 2012

Project Supervisor: Dr. Simon Moore

Director of Studies: Dr. Bogdan Roman

Project Overseers: Dr. Peter Robinson & Dr. Robert Watson

Introduction

Currently, the only widely supported hardware description languages are Verilog and VHDL. While these languages are a vast improvement on older techniques of manual circuit design, as designs become ever more complex, more powerful languages will be needed.

Existing alternatives these low level languages usually provide a higher level model which is then compiled to Verilog. However in many cases, while the model is effective at designing certain kinds of system, it can be awkward to use for others. Additionally, the compiler may not be able to make use of microarchitectural tricks that would usually be used by a designer at a lower level.

In order to provide multiple powerful abstractions, while still allowing microarchitectural freedom where it is desired, the aim of this project is to start with a low level model, and build higher level models on top of it. These higher level models can then be connected together through a common interface, such that each component of a system can be created in an appropriate abstract model.

Haskell has many properties useful for this goal. Its syntax is very flexible, allowing arbitrary binary operators to be defined, and providing a notation for monadic operations. In addition, its type system allows very powerful abstractions and generalisations to be made, while maintaining type safety.

Starting Point

There are a few of relevant libraries in the Haskell package database which I may use in my project. In particular there are some libraries for working with VHDL and Verilog. It is likely I will also make use of other more general libraries from this package database as well. In addition, during the summer, I created a library for abstracting connections in Bluespec, and I may use a similar design in my language's library.

Substance and Structure

The core of the project will consist of three components. A representation for RTL circuit models, a generator to transform the representation to Verilog code, and a series of library functions for working with the representation, which should be designed to form an embedded language.

The RTL representation will essentially represent a subset of Verilog, indicating what registers exist, what logic connects them, and what signal edges trigger an update. Other features of Verilog may be included if it turns out their reimplementation at a higher level is either less efficient, or cumbersome. For example, it may prove useful to be able to describe a module hierarchy, to assist the Verilog compiler by providing synthesis boundaries.

As the representation will correspond closely to Verilog, generating this code will involve a relatively simple transformation. Each register is given a name and defined, similarly intermediate logical values must be named and assigned, and finally register update logic is produced as a set of always blocks representing the different triggers.

While it would be possible to use the constructors of the internal representation as a language to describe hardware directly, this is unlikely to be particularly convenient, and so a set of library functions will be written to make it easier to build circuit representations.

One key feature of the library will be to provide a common interface for communicating between circuits described at different levels of abstraction. The suggested interface is structural groups of input and output wires, which can be constrained and abstracted as appropriate in each higher level model. For example, a model based on guarded channels could constrain its wire interfaces to have appropriate guard signals, and expose these as channels internally.

Once the library is created, the project will, as an extension, explore various higher level abstractions built on top of this base. For example, one simple model which covers many use cases, is single clock synchronous circuits, where all register updates are triggered by the same clock transition.

In order to test the system, various hardware examples will also need to be created throughout the project. These will be compared with implementations

in other languages, in terms of amount of code necessary, as well as the efficiency of the produced design.

Success Criterion

1. An internal representation of RTL circuit description must be designed and implemented.
2. A program must be written to transform this representation into valid Verilog source code.
3. A set of functions must be written, which provide an embedded language for creating RTL circuit descriptions, in the internal representation.
4. Simple hardware examples should be created in the language to demonstrate the project works.
5. Hardware implemented using the language should be compared with implementations in other languages, in terms of code length as well as logic elements used.

Plan of Work

1. **19th October - 2nd November:** Read papers on Lava, and other related languages. Research Verilog synthesis semantics. Set up computers with appropriate simulation and synthesis software, and familiarise myself with its usage.
2. **2nd November - 16th November:** Design and implement the underlying RTL representation.
3. **16th November - 1st December:** Create the Verilog generator, and test with some very simple hardware examples.
4. **1st December - 13th January (Michaelmas vacation):** Implement a simple set of functions for building RTL circuits, and use them to create some slightly more complex examples.
5. **13th January - 27th January:** Write progress report. At this stage, the project should be capable of describing simple RTL systems, and producing correct Verilog to implement them.
6. **27th January - 17th February:** Implement an abstraction for groups of signals.

7. **17th February - 16th March:** Implement a simpler interface for creating synchronous systems.
8. **16th March - 21st April (Easter vacation):** Explore further extensions, perform evaluation experiments and start writing dissertation.
9. **21st April - 3rd May:** Complete dissertation.
10. **3rd May - 10th May:** Checking, editing and printing.
11. **10th May:** Submission one week before the deadline.

Resources Required

I will be using my own computers for the majority of the project work. For backup, I will keep my project in a git repository, and regularly push it to a number of locations, including Github, Dropbox and my external hard drive.