Research Proposal

Mantle: Building HDL abstractions from an RTL model in Haskell

Alex Horsman

28th March, 2013

Introduction

The focus of my research would be continuing development of my Part II project, a hardware description language named Mantle[1], which explores new abstractions for hardware design. The end goal would be to develop an effective and practical language for use in computer architecture research, as well as a solid base for further language developments.

Background

Currently, the only widely supported hardware description languages are Verilog and VHDL. While these languages offer a vast improvement on older techniques of manual circuit design, as hardware designs become more complex, the limited abstraction capabilities of these older languages are no longer sufficient. Various new standards and extensions have been created, such as SystemVerilog, but tool support for these is limited, especially among synthesis tools.

[1] https://github.com/aninhumer/mantle

Existing alternatives these low level languages usually provide a higher level model which is then compiled to one of the existing ones. However in many cases, this limits the effectiveness of the language to a particular class of designs. For example, a model based on data flow may be effective for designing DSP hardware, but a control flow model may be better for stateful network hardware. Additionally, limitations of the higher level model may make it awkward or even impossible to use certain micro-architectural tricks to optimise a design.

While it is possible to combine different languages to complete different parts of a design, the productivity cost and design overhead of interfacing between them can make this impractical. Ideally we would like a single language which can support multiple models and abstractions, and interface between them easily and efficiently. Furthermore, it should be possible to introduce new models of hardware as libraries without requiring any changes to the language.

Developments

One alternative HDL which demonstrates some interesting abstractions, is Lava (from which Mantle takes its name). This language is based on a data flow model, where components are represented as functions from input signals to outputs, and state is introduced by delay elements.

This model makes describing feedback rather messy, as it requires exposing additional inputs and outputs, and using an explicit combinator to close the loops. An alternative style uses co-recursive functions, which is slightly more elegant, but requires a non-conservative extension to the Haskell compiler, and breaks referential transparency.

Another interesting language is Bluespec, which is based on a model of guarded atomic actions. The language is used to describe a set of conditional transactions on the register state, which conceptually occur sequentially. The compiler schedules these, exploiting parallelism as far as possible, and constructs a design which implements equivalent behaviour. Additionally, it includes an imperative sublanguage for describing state machines, making it very effective for implementing control flow oriented designs.

However a few shortcomings have been observed in its use at Cambridge. In

particular, its interface system imposes certain restrictions on the way modules can interact, which make sense within its model, but do not correspond well to users' intuitions of hardware modules. For example, it is impossible to describe a module with a connected pair of input and output channels, despite the fact that this corresponds to a trivial layout in hardware.

Proposal

With the approach taken in the design of Mantle, I hope to avoid the shortcomings that existing languages have presented, and create a more powerful tool for ongoing research.

Design Motivation

The core of Mantle is a low level model of hardware, corresponding closely to that of existing RTL languages. Notably, this model includes support for arbitrary register update triggers, which permits the language to describe asynchronous designs, although the focus of development so far has been on synchronous ones. Higher level abstractions can then be built up from this, using Haskell as a meta-language. As the representation corresponds closely to Verilog, code generation involves only a relatively simple transformation, which is transparent to the user at this level.

It is clear from existing work, that a consistent and flexible framework for interfacing between components is critical to modularity. This framework should allow powerful compositions at a high level, while still permitting precise control where necessary. In order to achieve this Mantle uses hierarchically structured groups of input and output wires for interfacing between modules. This provides a clear mapping to hardware at a lower level, and the abstraction capabilities of Haskell allow powerful dynamic combinators to be created for use with more complex models.

Haskell has many properties which made it a good target for embedding Mantle. Its syntax is very flexible, allowing arbitrary binary operators to be defined, and providing a notation for monadic operations, which encompass a wide variety of common syntax patterns from other languages. In addition,

its type system admits very powerful abstractions and generalisations, while also allowing safety properties to be enforced.

Initial Results

Mantle is currently a flexible meta-programming system, with the underlying representation and code generation now reasonably stable. A convenient syntax for RTL level programming, similar to Verilog, has also been created.

A few simple extensions which demonstrate potential for powerful abstractions have already been created, including an abstraction for automatically propagating clock and reset signals around synchronous circuits. This allows the common case to be handled as easily as in other languages which only permit synchronous designs.

In addition, the interface structure has shown to be able to create guarded channels, and a wide range of corresponding combinators very easily. Many of these functions also fit neatly into existing abstraction patterns commonly used in Haskell, such as Applicative Functors, suggesting a good fit with the idioms of the language, and potential for many more ideas to be drawn from existing work.

Goals

First Year

The initial target would be to take Mantle from its existing state as an interesting proof of concept, to a complete set if tools for hardware development. This would include simulation of the underlying model, with associated debugging tools. One intended feature would be the ability to create behavioural simulations for components, both to test their functionality against, and also to accelerate wider system testing. These could even be derived directly from any higher level abstract models used to implement them.

In order to demonstrate the effectiveness of these tools, and also to give their development direction, a simple but realistic CPU design would be implemented in Mantle alongside them. This would likely be based on one of the

existing example designs in the lab, such as the TTC or Cyan processors. Additional features could also be included where they demonstrate new abstractions made possible in Mantle. For example, caching could be added in a modular way that allows multiple designs to be tested, or even composed into a multilayer cache.

Further Extensions

From this point development would focus on identifying language problems faced by hardware researchers in the department, and seeking solutions to them in Mantle. While the cost of porting existing projects would likely be prohibitive, it should still be possible to isolate particular components, and use Mantle to accelerate the exploration of their design space. The capability to use abstract models, and dynamic circuit generators could permit more relevant design properties to be parametrised. For example, by using a generic notion of expressions, large blocks of combinational logic could be pipelined to different depths automatically, allowing comparisons of cycle latency against critical path length to be made more rapidly.

Additionally, a more powerful language may expose possibilities for hardware architectures, which were previously considered infeasibly complex. For example asynchronous designs offer many potential improvements in power consumption, by eliminating the necessity of global clock signals. However there have been relatively few large hardware designs constructed in this way. It may be possible to create a new model for expressing their design in a convenient and modular fashion, and avoiding the majority of timing issues involved.

Long Term

The eventual goal of the project would be to produce a language mature and flexible enough to replace Bluespec as the primary language used for hardware development within the department. Reaching this point would obviously be a significant undertaking, likely beyond the duration of a PhD, but there would be many advantages to an alternative. The commercial nature of Bluespec is not at all ideal for research, as it imposes a significant barrier to contributions or extensions from other institutions. Furthermore,

the focus of Bluespec's development is on the needs of industry, sacrificing many advanced features to maintain similarity to existing languages. By developing an open source alternative, long term development could proceed more rapidly, and more readily respond to the needs of research.

Another possible goal for the project, would be to develop Mantle as a teaching language for hardware design. Many students are put off hardware development by the existing languages and tools, which do not compare well with modern software development equivalents. Additionally, the incidental complexity of the existing processes, can mean many relatively simple hardware design concepts are impractical to include in exercises. By creating new abstractions as necessary, exercises using Mantle could be tailored to focus on teaching particular concepts, without introducing unnecessary details.

Conclusion

The design of HDLs presents a rather different set of challenges and constraints to those of programming languages, and while there has been much research into abstractions suitable for the latter, relatively few abstractions have been created for hardware design. This is despite the fact that the field of hardware design includes some of the largest and most complicated technology projects in existence. For this reason I believe exploring the possibilities of HDL design will prove a very fruitful area of research.