A Calculus for Hardware Description Languages

Sungwoo Park and Jinha Kim

Department of Computer Science and Engineering
Pohang University of Science and Technology
{gla,goldbar}@postech.ac.kr

In efforts to overcome the complexity of the syntax and the lack of formal semantics in conventional hardware description languages (most notably Verilog and VHDL), a number of approaches based on functional languages have been proposed. The merits of functional languages as hardware description languages can be attributed to the fact that basic building blocks for hardware circuits are equivalent to mathematical functions while functional languages lend themselves to creating and composing mathematical functions.

Functional hardware description languages are typically embedded into existing functional languages such as Haskell and ML, or into new functional languages designed specifically for hardware design such as $reFL^{ect}$. Since its semantics is unaware of hardware circuits, a host language represents hardware circuits as a special datatype. Eventually we convert such a datatype into netlists which describe connections in hardware circuits at the lowest level.

Converting a datatype representing hardware circuits into netlists is certainly a necessary step, but makes it an unnecessarily low level task to prove properties of hardware circuits. The reason is that netlists conceal high level descriptions written into a source program and are thus correspondingly more complex to analyze. Such an indirect analysis via netlists can be compared to an analysis of an ordinary functional program in which we first compile it into an assembly language and then analyze the output assembly program instead.

We present a new calculus, called $l\lambda$, which may serve as a "high level assembly language" for functional hardware description languages. $l\lambda$ is an assembly language in the sense that its definition consists only of a minimal set of primitive constructs each of which corresponds to a specific method of combining hardware components, *e.g.*, linking two separate hardware components or building feedback circuits. $l\lambda$ is still a high level language in that it makes no explicit use of low level constructs, such as ports and wires, characterizing netlists. As it extends the syntax and the type system of the standard lambda calculus, $l\lambda$ is particularly suitable as a substitute for netlists when designing functional hardware description languages.

Two characteristic features of $l\lambda$ are a denotational semantics and a linear type system. The denotational semantics enables us to translate an expression, without evaluating it, into a structural description of hardware components and their connections. It is sound in the sense that it maps expressions only to *realizable* hardware circuits which contain no input terminal connected with multiple wires. It is also complete in the sense that every realizable hardware circuit has a corresponding expression in $l\lambda$ as long as all its hardware components are connected via wires. The linear type system enables us to treat functions as first class objects as in conventional functional languages.

As a proof of concept, we extend $l\lambda$ with polymorphism and implement a Fast Fourier Transform circuit. An expression of a polymorphic type describes a family of hardware circuits with essentially the same layout of hardware components, but with different numbers of wires. It yields a specific hardware circuit when all its type variables are instantiated to sharable types. Thus polymorphism in $l\lambda$ offers a simple form of metaprogramming which is particularly useful for writing higher-order combinators.

As Sheeran [1] notes, *"functional programming and hardware design are a perfect match."* Thus it is actually no surprise to see that there is already an extensive literature on functional hardware description languages. What comes as a surprise, however, is that there has been little effort to formally interpret the lambda calculus, the core calculus for all functional languages, directly in terms of structural descriptions of hardware circuits. The development of $l\lambda$ has been motivated by a desire for such a formal interpretation of the lambda calculus.

References

1. Sheeran, M.: Hardware design and functional programming: a perfect match. The Journal of Universal Computer Science **11**(7) (2005) 1135–1158