Integrating Hardware Description Languages and Proof Systems*

K. G. W. Goossens
Laboratory for Foundations of Computer Science
Department of Computer Science
University of Edinburgh
Scotland, UK

1 The Problem

Simulation in Hardware Design and Testing

Hardware description languages (HDLs) have been used in industry since the 1960s to document and simulate hardware designs. Simulation of HDL descriptions is very useful to find design faults without the need to manufacture the design. A well known drawback of simulation, however, is that the number of input combinations (or test vectors) increases exponentially. For modern designs, it would take years to fully simulate a design. In practice a limited number of test vectors are used to probe the circuit, possibly failing to uncover faults.

Formal Hardware Verification

Research into *formal hardware verification* of hardware designs aims to address this problem. A circuit and its specification are given by mathematical descriptions. A mathematical proof system is then used to prove correctness of the design with respect to its specification. Although the hardware verification field is still young, notable achievements include the formally verified VIPER microprocessor which is manufactured and used commercially. However, most proposed methodologies are not automated, and need considerable expertise to be used. Although formal verification methods remove the exponential explosion occurring for simulation, verification takes substantial time.

A severe drawback is that many different notations and tools are employed. Industrial hardware description languages have not yet been used, alienating the hardware verification field from the industrial designers.

*This poster was presented at the IFIP 12th World Congress in Madrid, Spain, September 1992

2 The Solution

Our approach advocates the use of a HDL in conjunction with a proof system [3].

To be able to use a HDL in conjunction with a proof system, it must have a precise definition. We have provided a *formal semantics* for (a small subset of) a widely used industrial hardware description language called ELLA[1] [1]. Using a formal semantics it is possible to prove results about the behaviour of circuits.

This semantics has been embedded in the LAMBDA[2] [2] proof system. LAMBDA is a higher order logic proof system which allows specifications of designs to be stated in a natural and succinct form. Using the proof system, a number of results have been proved which confirm the correctness of the simulation model which is used in the industrial simulator for ELLA.

ELLA circuit descriptions reside *inside* the proof system which allows their formal manipulation. It is now possible for industrial users to describe designs in a formal setting, while retaining a familiar ELLA notation.

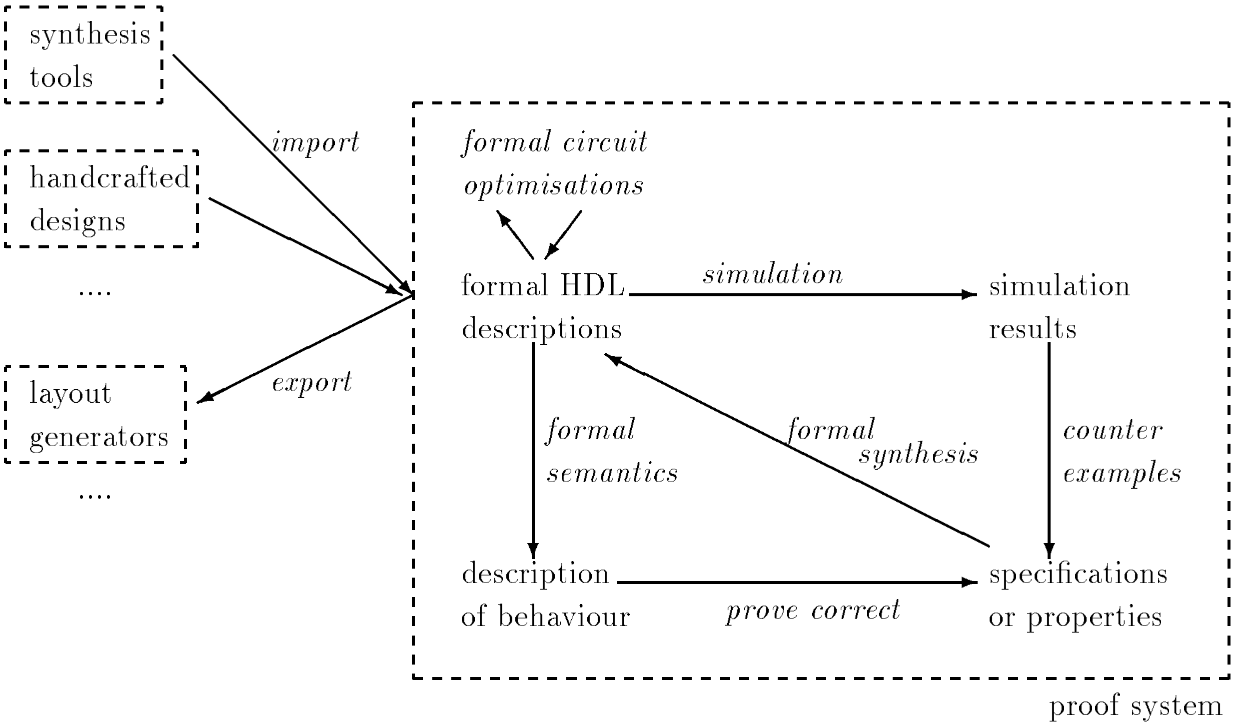

Figure 1: Embedding a HDL in a Proof System

We combine familiar notations (HDL) and techniques (*e.g.* simulation) with mathematical rigour (formal semantics, proof system). The fusion of a HDL and a proof system allows a number of different methodologies and tools to be integrated into one framework. Moreover, we can easily interface with existing tools because the HDL provides a common medium. We will discuss some of the applications in more detail below.

[1] ELLA is a trademark of the Secretary of State for Defence, United Kingdom.
[2] LAMBDA is a product of Abstract Hardware Ltd.

3 Proving General Properties About Circuits

Circuit descriptions are part of the proof system. Using conventional hardware verification methodologies we can prove, for example, that a circuit meets its specification. We can also prove properties about all circuits, classes of circuits (such as N bit adders) or individual circuits.

$$\vdash \ \forall c : \mathit{circuit.\ simulation\ of\ c\ cannot\ loop,\ \ and\ is\ monotone.}$$

Proving a property about a class of circuits may be harder than proving it for a particular member, but the proof needs to be done only once. We can also deal with circuits which are parametrised on word length (*e.g.* N bit adders), or other circuits (*i.e.* contain plug-in components).

$$\begin{aligned} \vdash \ \forall N.\ \mathit{adder}\ N\ (x,y)\ (s,c) \rightarrow \quad & s = \mathit{abs}^{-1}((\mathit{abs}\ x + \mathit{abs}\ y)\ \mathit{mod}\ 2^N) \wedge \\ & c = \mathit{abs}^{-1}((\mathit{abs}\ x + \mathit{abs}\ y)\ \mathit{div}\ 2^N) \end{aligned}$$

This example also illustrates the use of *data abstraction*.

4 Abstractions in Hardware Design

Hardware designs span many levels of abstraction from behavioural descriptions down to, for example, gate level. Most HDLs do not provide facilities to link different levels of abstraction. Although different levels may be simulated and their outputs compared, they are only "proven correct" through exhaustive simulation. Using a proof system different levels of abstraction may be related formally, thus ensuring that no errors are introduced when going from one level to another.

Data Abstraction The representation of integers by bitstrings above, is an example of data abstraction. By defining the abstraction function *abs*, boundary problems such as overflow often become more explicit.

Structural Abstraction Decomposition of hardware modules is essential in top-down design methodologies. Formal synthesis of hardware and refinement based synthesis are paradigms which have been used in conjunction with proof systems.

Temporal Abstraction Time is often observed differently at adjacent levels of abstraction. A microcode instruction may take several clock cycles, and a whole microcode program may implement a single instruction level operation. Temporal abstraction functions relate various time scales in a formal manner, facilitating proofs of correctness.

Behavioural Abstraction Often it is not desirable to specify the behaviour of a hardware module fully. We may not care what the behaviour of the component is for input combinations which will not occur. Loose specifications do not unnecessarily constrain implementations.

5 Formal Symbolic Simulation

Simulating circuit descriptions using simple values (**1** and **0**) leads to an exponential increase in the number of test vectors. Using extra values in the value domain, such as don't care **X**, and don't know **U**, somewhat relieves this problem. Symbolic simulation goes one step further and allows the use of variables x and formulae $x \vee y$ as inputs and outputs. The variable x is not a value in value domain, but ranges over all these values. Proof systems provide exactly the right environment for this sort of simulation. With the use of an industrial HDL, such as ELLA, we can perform any of the conventional (symbolic) simulations [4].

$$\vdash \text{simulate ANDGATE } x \ (y \vee \mathbf{1}) \Rightarrow x$$

ANDGATE is part of the proof system, and we could replace it by any proof system term, as long as it evaluates to a circuit. For example, we can simulate N bit adders, for arbitrary N.

A very useful application is *abstract hardware*. During the course of a design, we often want to simulate it, even when some subcomponents have not been implemented. Using variables to stand for unimplemented hardware we can still simulate the whole design, because we should know the specifications of the subcomponents. Using proof system facilities the specifications can be used instead of a future implementation to compute the outputs.

6 Formal Synthesis and Optimisation

Circuits may be produced inside the proof system, by *formally verified hardware generators*. The hardware generating function is proved correct once. All outputs from the function are then guaranteed to be correct. This output can then be exported out of the proof system to conventional tools such as layout generators. Of course, it is also possible to verify output from conventional synthesis tools, by importing the circuit description into the proof system. Our work fits in with interactive paradigms such as formal synthesis and refinement based synthesis.

Other applications include formally verified circuit optimisations. Semantically equivalent circuit descriptions may be swapped without changing the meaning of the circuit. It may also be possible to prove correct previously informal transformations and optimisations. Transformational design methodologies have also been used successfully in conjunction with proof systems.

References

[1] Computer General Electronic Design, The New Church, Henry St, Bath BA1 1JR, England. *The ELLA Language Reference Manual*, issue 4.0, 1990.

[2] Mick Francis, Simon Finn, and Ellie Mayger. *Reference Manual for the Lambda System*. Abstract Hardware Limited, version 3.2 edition, November 1990.

[3] K G W Goossens. Embedding a CHDDL in a proof system. In P Prinetto and P Camurati, editors, *Advanced Research Workshop on Correct Hardware Design Methodologies*, pages 359–374. North Holland, June 1991.

[4] K G W Goossens. Operational semantics based formal symbolic simulation. In *Higher Order Logic Theorem Proving and Its Applications*, September 1992. A longer version is available as LFCS Report ECS-LFCS-91-231.