CandoGen – A Property-Based Model Generator

Martin Schickel, Volker Nimbler, Martin Braun, Hans Eveking

{schickel,nimbler,braun,eveking}@rs.tu-darmstadt.de

Computer Systems Lab – Darmstadt University of Technology – Germany

http://www.rs.tu-darmstadt.de/

Abstract

Property-based design has multiple applications in the domain of formal verification. We have developed a tool capable of automatically generating an exact implementation from a set of finite PSL properties.

1. Introduction

Automated design from properties has become more prominent during the last decade and quite rightly so.
During many stages of the design process, a component's formal specification is already present, although the actual implementation is not. In order to obtain a preliminary working model during the early design stages, a hardware designer will still have to sit down and build it. Work that will be more or less thrown away, once the design process reaches a more advanced stage. Property-based design can solve this problem by providing a simple means to automatically generate a model from the specification. It will neither be optimized nor be described in a sophisticated way, but still, it is a working model.

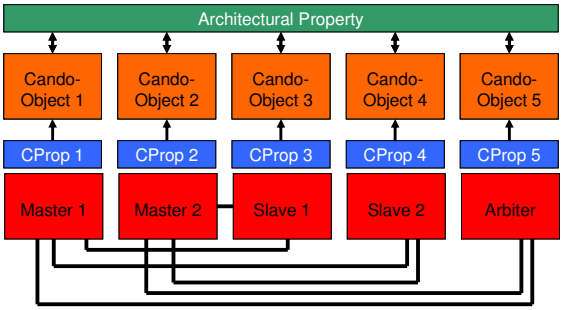

Figure 1: Verification using Cando-Objects

Models like these can also be used as property-based abstractions during the actual verification process: Once a component has been verified by a set of properties, the particular component can be replaced by an abstraction generated from the properties used to verify the component.

All parts of the original component which are not covered by the properties used to generate the model will be abstracted, hopefully reducing the abstraction in size. Also, the hardware generated from properties is mostly much more verification-friendly than optimized circuits that possibly also incorporate self-test-logic. Replacing components by these abstractions will therefore allow verifying larger systems than before. Figure 1 shows, how that may work for an ARM AMBA AHB Bus.

2. Cando-Objects

The hardware generated by the model generator will – contrary to what other projects, e.g. Prosyd [1], strive to – incorporate any ambiguity contained within the property set. If a signal value is not specified in certain states or logical ambiguities are present (e.g. $a \vee b$), then every possible behavior must be able to occur. Only by doing this, the abstraction will be fault-conserving, because eliminating the ambiguities corresponds to the introduction of new properties. We call this type of abstraction *Cando-Object*, because it "can do anything, show any behavior, which is not expressively forbidden" by the properties it was generated from.

Besides the already mentioned usage of Cando-Objects as a property-based abstraction of components, they can be used as early prototypes and even as property-based implementations, if the property-sets contain only a low level of ambiguity. Also, the result of the generation process reveals information on the signal assignment coverage.

In addition to the generation of VHDL models we aim at generating C++ models for software/hardware-co-simulation. By generating the early simulation models from verification properties, we can later easily make sure that the actual implementation corresponds to the model by simply verifying the properties.

3. The CandoGen Tool

In order to generate hardware from properties, we need two different information sets: For one, we need the properties to generate the hardware from. Currently two types of property specification language are supported by our tool: PSL and OneSpin Solution's ITL. Due to the way properties are transformed, the tool only supports properties covering a finite time window, which includes all types of safety properties (LTL: G), but excludes liveness properties (LTL: F).

Secondly, a black box description of the component is needed, since property specification languages do not incorporate information on signal types, signals, constants and the like. The information can be obtained directly from a VHDL description, be it a complete component

description which the user wants to abstract or an original black box, which the user wants to fill with the functionality specified by the properties.

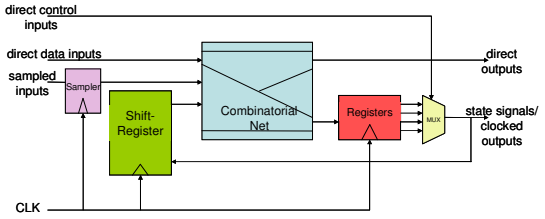

Figure 2: Internal Layout of a Cando-Object

The properties are then normalized by applying the algorithm described e.g. in [2]. This normalization procedure reveals potential inconsistencies between properties and disjoins them such that only one property specifies a particular signal's assignment in a particular state. While the specification is broken down to the bit-level if necessary to generate unambiguous signal assignments, the goal is to have properties at the bit-vector-level, since only then arithmetic operations can be represented compactly. After the normalization, which is the most time-consuming part of the whole process, is completed, a VHDL description of the Cando-Object is generated. All of this is done by the CandoGen tool, which was developed during the last three years for an x86-Linux platform. The resulting internal design always corresponds to the one displayed in Figure 2, which is basically a Mealy machine reordered for better feature visibility.

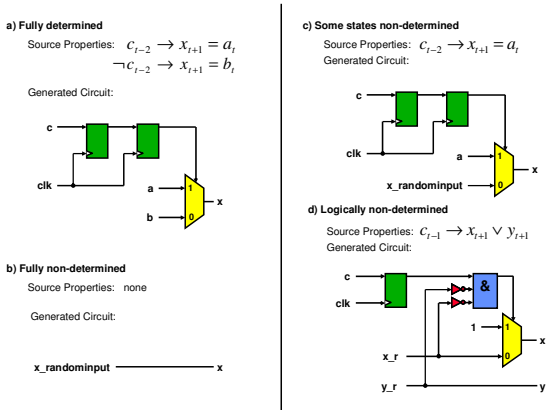

Figure 3: Realization of Non-determinism

The reason the above design deviates from the standard Mealy Machine description is to be able to incorporate asynchronous resets. While the properties normally deal with transitional systems, asynchronous resets require a combinatorial circuit to propagate a signal change within the same clock cycle. This is dealt with by providing two sets of signal assignments: One describes the assignment in case no reset will occur within the next cycle, the other describes the assignments in case a reset occurs. The reset signal (or, in general, any input signal not sampled at a clock edge) will then control a multiplexer that selects one of the assignment sets.

Another difficulty is the realization of non-determinism within a circuit. Since specifications (and therefore property-sets) mostly do not define every signal assignment in every situation, it often happens that a signal value is actually *don't care*. In order to allow all possible behaviors, additional input signals will be used to generate signal values for the cases, where a signal value is not defined. If the specification is logically non-determined (e.g. '*x or y must be set*'), free inputs will also be used. Only in this case, the random signals will be overridden by a particular valid result in case the signal assignment is not valid. Figure 3 illustrates the various ways to accommodate the various kinds of non-determinism.

4. Experimental Results

We have conducted experiments on various industrial designs, including ARM's AMBA AHB, the PCI Local Bus and a MIPS core.
For the complete AHB master the generation of a Cando-Object takes around 15s, for an AHB slave it is around 7s.

5. Conclusion

Although property-based design has not yet become a very important technique in the design and verification process, we have shown that there are variable promising applications that might be useful to supplement or replace tools currently in place. The CandoGen tool is a tool able to generate models from finite verification properties written in PSL/ITL. Future work includes the synthesis of complete processors in order to generate working models from the specifications of the memory interface and the assembly language.

5. References

[1] PROSYD Project Deliverable 2.1/1: *Property-based Design and Implementation*, 5/2005, www.prosyd.org

[2] M. Schickel, V. Nimbler, M. Braun, H. Eveking: *On Consistency and Completeness of Property-Sets: Exploiting the Property-Based Design Process*. In: Proc. of FDL, 2006

[3] M. Schickel, V. Nimbler, M. Braun, H. Eveking: An Efficient Synthesis Method for Property-Based Design in Formal Verification. In: Sorin Huss (Ed.): *Advances in Design and Specification Languages for Embedded Systems*, p. 163-182, Kluwer Acad. Publishers, Boston/Dordrecht/London, 2007, to appear

Acknowledgement

The research leading to the development of this tool was conducted within the scope of the FEST project, funded jointly by the German ministry of Research and Education and industry partners.