Abstract

This embedded tutorial surveys some possibilities for verification techniques that combine conventional simulation and ideas, techniques, and algorithms from formal verification, to obtain better functional test coverage of large designs.

1 Introduction

There is a desperate need for an immediate practical solution to the problem of verifying large designs. Simulation-based verification has not been able to keep up with increasing design complexity. In spite of increasing simulation and emulation speed, the number of problem cases that can be covered by manually generated functional test vectors or pseudo-random functional testing is declining. The amount of human and computational effort devoted to verification is increasing rapidly, time to market is being delayed, and bad bugs are showing up after tape-out and in products.

Many designers hope that formal verification will solve the problem. Formal verification will help (indeed, it will be indispensable) in niches where it is particularly effective. However, it will not provide a general solution to the verification problem in the near future. Except for equivalence checking tools, which are targeted at low-level design errors (e.g. in hand-optimized net-lists), formal verification tools do not currently scale well to large designs. Effective use of these techniques requires reducing the core computational problem by a variety of methods, none of which is supported well by current design practices. For example, model checking a small part of a design needs a detailed specification of the interface behavior of the block being verified, so that only legal inputs are checked. Real designs almost never have up-to-date, detailed specifications of the interfaces between low-level blocks, so verification often requires painful reverse engineering of the interface specification. Another approach to complexity reduction is to abstract the circuit behavior. But abstracting parts of a complex design is a frustrating process, because of false error reports resulting from unanticipated dependencies on design details which are changed by the abstraction.

I believe that design practices and formal verification techniques will evolve together to resolve these and other problems, at least in part. However, there is every reason to believe that it will be a long, difficult journey to the point where formal verification tools displace traditional simulation-based verification.

It is likely that the most practically effective verification methods in the medium-term future will be “semi-formal” methods, which combine ideas from conventional and formal verification to achieve much better test coverage of designs than conventional verification, while avoiding the scaling and methodology problems inherent in formal verification methods. Although work with along these lines has been going on for several years, the collection of ideas has not yet evolved into a “field of study.” Even a modest attempt to compare techniques reveals that many obvious ideas have not yet been tried. We should expect research in this area to bloom in the next few years, accompanied by rapid commercialization and use of the best ideas.

2 The spectrum

If we imagine a spectrum, as in Figure 1, ranging from conventional simulation to full formal verification. Near the simulation end of the spectrum is conventional simulation coupled with coverage analysis, which provides data on how thoroughly a design has been exercised. Coverage metrics can be used to guide the manual definition of test vectors, as well as evaluating the effectiveness of pseudo-random strategies.

Quite a variety of metrics have been proposed. HDL-based measures include line coverage, statement coverage, branch and conditional coverage, and path coverage. There are coverage metrics that record all the values that selected variables have taken, whether each bit has changed in each direction (toggle coverage). Other metrics are based on state machine coverage, including state and transition coverage of individual state machines or of combined state machines. Commercial tools are available to report on many of these coverage metrics. Coverage can also be based on an abstraction of the design, such as a state machine provided by the designer; this would allow a designer to indicate exactly the cases that he or she feels need to be exercised.

Another class of metrics comes from the viewpoint of manufacturing test: try to characterize a class of “design faults,” analogous to fault models from test, and measure the coverage of these “faults” achieved by a test case. Sometimes, fault models from manufacturing test are used, unmodified, as a rough measure of coverage for functional verification (this is easy because of the ready availability of fault simulators).

A bit further to the right on our spectrum are techniques that I will dub “smart simulation,” which generate functional tests based on a coverage metric. Test generation can be done off-line, storing a set of test vectors in a file, or on-line, generating new inputs to the design on-the-fly. Most existing research seems to be driven by the
desire to cover an abstract state machine, provided by the designer or as part of a requirements specification, or selected “corner cases” that are part of the specification of the design. It seems that a set of test vectors could be generated to maximize any given coverage model, although most such ideas have not been explored.

Even further towards the formal end of the spectrum are techniques that do “wide simulation” by symbolically representing large sets of states in relatively few simulations. A variety of symbolic representations could be used. Current systems use Boolean decision diagrams (BDDs) or logical expressions which have operators on higher-level data types than bits or bit vectors, such as integers, or uninterpreted functions to abstract away datapath elements.

Finally, relatively close to the formal verification end of the spectrum are model-checking techniques that explore the state space in parts, possibly using heuristics that try to find design errors early in search. As with the other approaches, only the initial possibilities have been explored.

3 Concluding remarks

It will be a major challenge to produce meaningful research results in this area, especially for academic researchers, because of the difficulty of quantifying the quality of results produced by a particular idea. The ultimate measure of quality is bug-finding ability on real designs, but measuring this reliably will require significant resources. Dealing with large designs requires lots of engineer or researcher time, especially when using prototype tools. Even getting access to realistic designs is difficult for researchers in academia. Moreover, conclusions drawn from a small number of designs will be suspect, because all of these techniques are likely to be very sensitive to the particular characteristics of a design and its designers.

It seems clear to me from conversations with my colleagues that semi-formal verification techniques will soon receive a great deal of attention from researchers. There will be commercial interest and major practical impact in the not-too-distant future as well.

What does this portend for formal verification? It will grow in importance, but not provide a universal solution. When applicable, formal verification provides value that other techniques cannot match. It can achieve vastly greater behavioral coverage, and (subject to assumptions about models and requirements) prove the absences of some classes of errors. However, hybrid verification methods may clear a long-term path to greater acceptance of formal verification by encouraging changes in design methodology that make formal verification easier.

A full version of this paper, with references, will appear on my website: http://verify.stanford.edu.