EverLost: A Flexible Platform for Industrial-Strength Abstraction-Guided Simulation*

(Tool Paper)

Flavio M. de Paula and Alan J. Hu

Department of Computer Science, University of British Columbia, Canada
{depaulfm,ajh}@cs.ubc.ca

Abstract. *Abstraction-guided simulation* is a general framework for automatically harnessing, during simulation, information from abstraction and model checking. EverLost is our platform for industrial-strength abstraction-guided simulation. EverLost takes an RTL Verilog design and preimage/abstraction information from any BDD-based abstraction/model-checking tool, and automatically generates code that implements abstraction-guided simulation and directly compiles with the design under the widely-used Synopsys VCS simulator. The platform enables flexible exploration of abstraction-guided simulation — different formal tools and guidance heuristics are easily inserted — while providing the capacity, speed, and Verilog compatibility of a leading industry-standard tool.

1 Abstraction-Guided Simulation

Automatic formal hardware verification continues to progress, through advances such as model checking [5, 8], symbolic model checking [3], bounded model checking [1, 2], and counterexample-guided abstraction refinement [7], which have greatly expanded the capacity of automatic verification tools. Conventional simulation, however, remains the primary workhorse for industrial hardware validation. Simulation provides unparalleled capacity for handling design size and complexity, but (or because) it performs no analysis of the design. Abstraction and model checking, on the other hand, derive considerable information about the structure of the state space of the design, but (therefore) suffer from capacity limitations.

Abstraction-guided simulation[1] is a general framework for automatically harnessing, during simulation, information obtained by model checking and abstraction of the design. Briefly, abstraction-guided simulation consists of the following:

– We assume the goal of verification is to find an execution sequence that reaches a specified set of states, e.g., error states or a hard-to-reach coverage target.

* Supported by an NSERC Discovery Grant. We would also like to thank Daniel Kroening and Himanshu Jain for their help with the *vcegar* tool.

[1] The idea of guiding state exploration via abstraction has been independently invented several times, e.g., as "tracks" [10], "abstraction database" [6], and "distance-guided simulation" [9]. Unlike other work, our emphasis is on working with the capabilities and limits of real, industrial simulation tools. We prefer the nomenclature "abstraction-guided" to "distance-guided" because the analysis of the abstract model gives not true distances, but only lower bounds on distances, and the challenge for good guidance heuristics is precisely to handle this inaccuracy.

– Any conservative abstraction technique is used to create a model small enough for symbolic model checking. The abstract model preserves existence of any paths to the error states, but may introduce paths that don't correspond to any concrete path.
– If formal verification succeeds (either finding no abstract error paths, or successfully concretizing an abstract error path), we are done. The interesting case for simulation is when formal verification fails (and attempts at abstraction refinement fail to create a tractable model), as can occur typically with large hardware designs.
– The model checker has computed a series of pre-images from the error states in the abstract model. From these pre-images, we can dump a sequence of BDDs, representing sets of abstract states whose shortest (abstract) path to an error state is *i* abstract states long. Visualize these sets as concentric "rings" around the error states. A concrete state that abstracts to an abstract state in ring *i* is at least *i* clock cycles away from an error state.
– During directed random simulation, the simulator can consult the abstraction information for guidance by periodically computing the abstraction of the current simulation state and querying which ring it is in. Thus, the simulator can benefit from considerable information computed by model checking the abstract model.

An analogy is to driving with a GPS navigation device: one's concrete location (GPS coordinates) goes into the device, which contains an abstract model of the terrain and provides optimum routing for the abstract model; problems arise when the abstract model is inaccurate (e.g., due to construction); in those cases, the user wanders semi-lost until the device computes a usable new route. The name "EverLost" is a play on a pioneering, widely-deployed in-car GPS navigation system.

Abstraction-guided simulation is a broad and flexible framework, so research is needed to explore trade-offs. Hence, we have created EverLost, as a flexible, yet industrial-strength platform for exploring abstraction-guided simulation. The key features of EverLost are:

– Direct connection into Synopsys VCS, one of the most widely used Verilog simulators, giving true industrial capacity, simulation speed, and language compatibility.
– Simple interfacing to any BDD-based abstraction/model-checking tool. All we need are the concrete state variables, the abstraction functions, and the BDD rings.
– Easy exploration of different guidance heuristics. Currently, we have implemented a simple parameterized stochastic search; this is an active research area.

2 EverLost Architecture

The three major components for using EverLost are the logic simulator, the abstraction/model-checking engine, and the EverLost tool itself (Fig. 1). For tight integration and highest performance, we had to target a specific logic simulator, although the tool could be retargeted easily. We chose Synopsys VCS, one of the most widely used industrially. For the interface with the abstraction/model-checking engine, we designed for maximum flexibility: all we require are a list of the design's latches, the abstraction map, and the BDD pre-images that are a by-product of model checking.

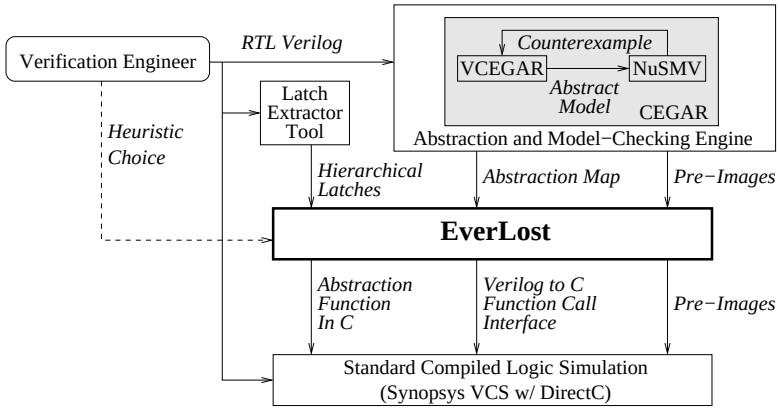

Fig. 1. Overall Tool Flow with EverLost

Given the needed inputs, EverLost generates a simulation guidance driver in C, the abstraction function in C, and a C interface in Verilog, which are passed to VCS along with the Verilog files and the BDD pre-images. The user can specify different simulation guidance heuristics via EverLost options.

The code generated by EverLost is compiled with VCS into a single executable. Internally, the simulator calls the EverLost driver every clock cycle. The EverLost code can read the current simulation state, possibly save it, and possibly evaluate it using the abstraction information. The EverLost code can then allow the simulation to continue, or it can force the simulator to jump to a particular state. In our current guidance heuristic, from a given state, the simulator explores n different traces for k cycles and picks the best state (i.e., the state that abstracts to the pre-image closest to the target states) from which to continue; one more parameter controls when to resort to a random walk to try to get around dead-ends.

3 Sample Results and Performance Overhead

We report some results from two publicly available designs: a USB 1.1 PHY [11] interface, and a full USB 2.0 Function Core [12]. We used VCEGAR [7] with NuSMV [4] as our formal engine; these tools are state-of-the-art, freely available, and support Verilog. When the designs were too big, the formal engine did not produce useful abstract models, so in some experiments, we used only a few sub-modules. VCS and EverLost, of course, had no capacity problems, including for the full Function Core.

When the formal engine provided enough pre-image rings, EverLost was able to guide simulation towards a target using up to an order of magnitude fewer simulation clock cycles than random simulation. For example, while verifying two usb_rx_phy coverage points, the formal engine generated 27 and 23 pre-images for (1) acknowledging receiving data and (2) proper synchronization. For (1), over 30 simulation trials, random simulation averaged 206K clock cycles and 2.1 seconds CPU time versus 5.3K clock cycles and 4 seconds for EverLost. For (2), also over 30 trials, random simulation

Module	Latches	Cycles	Standard	No-Op C-Calls	C-Calls+Heuristic	Overhead Ratio
usb_rx_phy	56	5M	96.3s	145.2s	490.0s	5.1
usbf_pl	696	250K	38.2s	232.3s	333.7s	8.7
usb	1785	15K	25.5s	379.5s	421.6s	16.5

Table 1. Simulation Overhead. The columns show, from left to right: the design, the number of latches, the length of each random trace, and the CPU times for the simulation when EverLost is absent, when only the C-Interface calls are added, and when both C-Interface calls and guiding heuristics are present, and the total overhead ratio.

averaged 1.4M clock cycles and 13 seconds versus 0.5M clock cycles and 1.25 seconds for EverLost.

Simulation overhead has two components: the overhead of calling/returning from the Verilog test bench to the C interface, and the time required by the heuristic to evaluate concrete states and choose an action. To measure the overhead, we ran extended random simulations on variously sized designs. Table 1 shows the results, averaged over 5 random runs each, with negligible standard deviations. Notice that as the design size increases, the predominant overhead is due to the interface between Verilog and C, rather than the guidance heuristic.

Future work includes reducing overhead and exploring guidance heuristics.

References

1. A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu. Symbolic model checking without BDDs. *Tools and Algorithms for Construction and Analysis of Systems*, LNCS 1579, pp. 193–207, 1999.
2. V. Boppana, S. P. Rajan, K. Takayama, M. Fujita. Model checking based on sequential ATPG. *Computer-Aided Verification: 11th Intl Conf*, LNCS 1633, pp. 418–430, 1999.
3. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Conf on Logic in Computer Science*, pp. 428–439, 1990.
4. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella. NuSMV 2: An OpenSource tool for symbolic model checking. *Computer-Aided Verification: 14th Intl Conf*, LNCS 2404, pp. 359–364, 2002.
5. E. M. Clarke, E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Workshop on Logics of Programs*, LNCS 131, pp. 52–71, May 1981.
6. S. Edelkamp, A. Lluch-Lafuente. Abstraction in directed model checking. *Workshop on Connecting Planning Theory and Practice*, pp. 7–13, 2004.
7. H. Jain, D. Kroening, N. Sharygina, E. Clarke. Word level predicate abstraction and refinement for verifying RTL verilog. *42nd Design Automation Conf*, pp. 445–450, 2005.
8. J.-P. Queille, J. Sifakis. Specification and verification of concurrent systems in Cesar. *5th Intl Symp on Programming*, LNCS 137, pp. 337–351, 1981.
9. S. Shyam, V. Bertacco. Distance-guided hybrid verification with GUIDO. *Design Automation and Test in Europe*, pp. 1211–1216, 2006.
10. C. H. Yang, D. L. Dill. Validation with guided search of the state space. *35th Design Automation Conf*, pp. 599–604, 1998.
11. USB 1.1 PHY. http://www.opencores.org/projects.cgi/web/usb_phy/overview.
12. USB 2.0 Function Core. http://www.opencores.org/projects.cgi/web/usb/overview.