A Method of EFSM Model Extraction from HDL Descriptions: Application to Hybrid Verification

Sergey Smolov
Software Engineering Department
Institute for System Programming of Russian Academy of Sciences
25, Alexander Solzhenitsyn, Moscow, 109004, Russia
E-mail: smolov@ispras.ru

Functional verification is a challenging task in a digital hardware design process. Traditional approaches to functional verification of digital hardware are: *formal* (i.e. model checking, property checking, equivalence checking and so on) and *simulation-based* (i.e. an observation of system behavior in a simulated environment). In spite of their advantages none of them can be fully substituted by another because of some limitations. The formal approach often requires a manual implementation of unit's formal model and therefore – expert skills from engineer in this area. As for the simulation-based approach, the important problems here are estimation of generated tests' quality (in terms of source code coverage or requirements coverage) and equivalence checking between the design under test (DUT) and the reference model.

An essential way to alleviate the problems discussed above lies in the usage of *hybrid approaches*. A typical hybrid approach comes from a combination of two or more verification engines to produce more effective tests for target design [1]. The way of combining depends on the current verification task and today there is no unified methodology here. It seems natural to combine different verification engines basing on the DUT representation. This representation should be powerful enough to meet a huge variety of well-known verification tasks and possibly should be extendible for being adapted to new ones.

One of the widely used ways to represent DUT is to use well-known abstract representations so-called *models*. Several types of models are used in hardware verification, like Flow\Dependency Graphs, Petri nets, Finite State Machines (FSM) [2]. Some of them are built from requirements (properties, use cases, etc.); others are extracted from source code. The last approach seems to be less error prone, but it implies strong requirements of model extraction methods in terms of their applicability to hardware design description formats and coding styles. This research is focused on models of the second type and their application to simulation-based and formal verification. Specifically, static, compile-time derivation of Extended Finite State Machines (EFSMs) from design source code written in a hardware description language (HDL) is considered.

The EFSM formalism extends the classical FSM model by adding (1) *input* and *output parameters*, (2) *registers* and (3) *transitions' guards* and *actions* defined over registers and input parameters. The main idea is to clearly separate *control* and *data path* (exactly as it is acknowledged in hardware design and synthesis). EFSM-based models are intensively used in functional test generation (to generate conformance test suites [3] or to cover unlikely corner cases [4]) as well as in formal verification, especially in *model checking* (to check properties or to find errors and inconsistencies in a design model).

Since existing methods of automated EFSM model extraction require additional information from user ([5], [4]) a new one is proposed [6]. It consists from the following stages: (1) HDL code is parsed, and the *abstract syntax tree* (*AST*) is built; (2) the AST is traversed, and the *intermediate representation* (*IR*) that is based on control flow graph is elaborated: (a) *clock inputs* (*CIs*) are identified; (b) *implicit state registers* (*ISRs*) are detected and replaced with the *explicit state registers* (*ESRs*); (3) the IR is transformed into the set of *guarded actions* (GA); (4) data flow analysis of the GAs is performed, and the *ESRs* are recognized; (5) the conditions on the ESRs are analyzed, and the EFSM states are identified; (6) the EFSM transitions (with guards and actions) are generated.

The major advantage of the approach is its high automation – no information except HDL code is required. The method uses heuristics for identifying states and clock signals and extracts the EFSM from the control flow graph-based representation. For every process defined in the HDL description, a single EFSM is usually built; all EFSM models of the description are defined over the same set of variables. It should be emphasized that EFSM actions have the "flat" syntax, which means that each action is a linear sequence of assignments. Another advantage of the method is its flexibility: since state detection heuristic can be easily changed, end user can extract different EFSMs from the same process depending on the verification task.

The method can be applied to multi-process hardware designs – in such case a multi-EFSM model will be extracted. For every single process of HDL description the related EFSM will be extracted. Since processes can use common variables, corresponding EFSMs have them too.

To use the proposed model extraction method in hybrid verification an extendible framework is needed. In this work a new toolkit is proposed that is called HDL Retrascope [7]. The HDL Retrascope is a toolkit for Reverse Engineering and Transformation of digital hardware designs written in Hardware Description Languages (HDL) like VHDL or

Verilog. It written in Java and is organized as an extendable framework with the ability to add new types of models as well as tools for their analysis and transformation. The HDL Retrascope toolkit operates with three main categories of entities: *engines*, hardware design inner representations (i.e. *models*), and *result* representations. The system of engines includes the following: 1) parsers elaborate HDL code of DUT and produce a *control flow graph* model; 2) transformers convert models of one type to equivalent models of another type; 3) generators performs some analysis tasks upon target models and produce the result. Hardware models are equivalent representations of DUT in terms of behavior but differ from each other in terms of structure. Along with control flow graph representation the toolkit supports several kinds of models: Clocked Guarded Atomic Action (CGAA) [8], EFSM, High Level Decision Diagrams (HLDD) [9] and so on. Result representations are auxiliary entities for transmitting additional data to engines. The main use case of the toolkit is as follows: accepting the input parameters (through command line or IDE based on Eclipse), selecting internal engines that are needed to achieve the result (i.e. a "tool chain") then applying them to input data in the selected order.

With the help of proposed EFSM extraction method promising hybrid verification approaches can be constructed through using HDL Retrascope framework. The first one that was implemented is a new test generation method aimed at generation of test sequences that cover all the EFSM transitions [10] and based on state traversal. The complexity of EFSM state graph traversal caused by data dependencies between actions and guards (some guard may depend on variable, which is defined in some action). We proposed some techniques for operating with such dependencies (so-called "counters" – there is well-known example for such dependency, when for the specified EFSM state an incoming transition contains an incremental assignment to the variable, and outgoing transition has a guard that includes an equation with this variable) in combination with some classical techniques, like symbolic execution and constraint solving. Experiments were made on ITC'99 benchmark designs and they have shown that the tests are generated by the proposed method (called RETGA – Retrascope EFSM-based Test Generation Algorithm) allow reaching better transition coverage (and, consequently, high statement and branch coverage too) with less number of test vectors than the known methods ([4]) do.

Another use case of EFSM extraction method in context of hybrid verification was connected with model checking based approach [11]. The implemented test generation method is based on automated extraction of HLDD models from hardware design's source code. This kind of models was chosen because of its similarity to the input language of nuXmv model checking tool [12]. Extracted models are then automatically translated into an input format of the nuXmv model checking tool and checked by it. The idea of extracted EFSM models usage was to treat them as "specifications" and to generate tests that provide high coverage of the source code in terms of statements and branches using negations of EFSM transition guards as conditions for model checker. Functional tests are extracted from the model checker execution results. Experiments on ITC'99 benchmark designs show that the generated tests have close characteristics to RETGA-based, but in some cases an additional adjustment of model checker was needed (selecting bounded model checking or classical symbolic model checking, etc.).

Tests are generated on the basis of EFSM models extraction can be applied to low-level representations of hardware designs, like BLIF (Berkeley Logic Interchange Format). At the gate level several fault models are well known like stuck-at faults, bridge faults and so on. Results on applying EFSM-based tests for HDL descriptions to low-level digital circuits aimed at different fault models will be published soon.

It is also supposed to extend HDL Retrascope toolkit with other engines for DUT transformation to input formats of popular external verification engines (like SPIN) and for requirements analysis (PSL). These engines could be used for an automated translation of HDL descriptions into model checker formats and applying formal verification techniques to them.

References

[1] J. Bhadra, M.S. Abadir, L.C. Wang and S. Ray, "A Survey of Hybrid Techniques for Functional Verification", IEEE Design & Test of Computers, vol. 24(2), pp. 112-122, 2007.

[2] S. Smolov, "A Survey of Methods for Model Extraction from HDL Descriptions" [in Russian], Trudy ISP RAN [The Proceedings of ISP RAS], vol. 27(1), pp. 97-123, 2015.

[3] A.Y. Duale, M.U. Uyar, "A Method Enabling Feasible Conformance Functional Test Sequence Generation for EFSM Models", IEEE Transactions on Computers, 53(5), pp. 614-627, 2004.

[4] G. Guglielmo, L. Guglielmo, F. Fummi, G. Pravadelli, "Efficient Generation of Stimuli for Functional Verification by Backjumping Across Extended FSMs", Journal of Electronic Testing, 27(2), pp. 37-162, 2011.

[5] J.C. Giomi, "Finite State Machine Extraction from Hardware Description Languages", ASIC Conference and Exhibition, pp. 353-357, 1995.

[6] S.A. Smolov, A.S. Kamkin, "A Method of Extended Finite State Machines Construction from HDL Descriptions Based on Static Analysis of Source Code" [in Russian], Saint Petersburg State Polytechnical University Journal. Computer Science, Management, Telecommunications, pp. 60-73, 2015.

[7] HDL Retrascope toolkit. http://forge.ispras.ru/projects/retrascope

[8] A. Brandt, J. Gemünde, M. Schneider, S.K. Shukla, J.P. Talpin, "Representation of synchronous, asynchronous, and polychronous components by clocked guarded actions", Design Automation for Embedded Systems, vol. 18(1), pp. 63-97, 2014.

[9] R. Ubar, J. Raik, A. Jutman, M. Jenihhin, "Diagnostic Modeling of Digital Systems with Multi-Level Decision Diagrams", Design and Test Technology for Dependable Systems-on-Chip, pp. 92-118, 2011.

[10] I. Melnichenko, A. Kamkin, S. Smolov, "An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs", Proceedings of ISP RAS, vol. 27(3), 2015.

[11] M. Lebedev, S. Smolov, "A Model Checking Based Method of Functional Test Generation for HDL Descriptions", Proceedings od Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE), pp.84-89, 2016.

[12] D. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuXmv symbolic model checker", Proceedings of the 16th International Conference on Computer Aided Verification (CAV), 2014, № 8559, pp. 334-342.