Assertion-Based Verification of Transaction Level Models

B. Niemann
Fraunhofer Institut für
Integrierte Schaltungen
nmn@iis.fraunhofer.de

Ch. Haubelt
Universität Erlangen-Nürnberg
Hardware-Software-Co-Design
haubelt@cs.fau.de

Abstract. Transaction Level Modeling with SystemC has become a de-facto industry standard for modeling of system-on-chip designs. The correctness of these models is therefore of crucial importance. In this paper, we propose a novel methodology to apply Assertion-Based Verification to Transaction Level Models. The novelty is based on two contributions: (1) Using Aspect-Oriented Programming techniques permits *transaction recording* without having to modify the existing TLM. (2) Mapping of transactions to Boolean signals and automatic *event clock creation* enables the concise formulation of assertions. In contrast to other approaches, our methodology allows direct Assertion-Based Verification of the TLM without the necessity of an additional abstract model.

1 Introduction and Related Work

In this paper, we propose a methodology to use assertions for the verification of Transaction Level Models (TLMs). Specifically, we show how to use SystemVerilog Assertions [1] for the functional verification of a SystemC [2] TLM. Instead of treating a transaction as a complex data structure, we associate with each transaction a Boolean signal which is $true$ during the transaction and $false$, otherwise. This reduces the problem of verifying transaction sequences and transaction properties to the well known problem of verifying sequential systems communicating over Boolean signals. Aspect-Oriented Programming is being used because it allows to add transaction recording to an existing TLM without having to modify the original source code. Using other approaches like e. g. the transaction recording capabilities of SCV [3], the TLM code needs to be instrumented with additional code. The sweet spot of our approach is to allow a clear separation of implementation and verification code.

Peranandam et al. [4] use an abstract model, called *Message Sequence Model* (MSM), to describe the functionality of a TLM. In addition to that they create a formal specification based on properties given in Linear Time Logics (LTL) [5]. Based on symbolic simulation they can verify whether or not a property holds for the MSM. Coverage information is obtained, meaning the percentage of functionalities of the MSM covered by the LTL properties. Our approach is different in that we do not require an abstract model of the TLM. We directly execute the SystemC implementation of the TLM, thereby obtaining a transaction trace. Using this transaction trace, properties written as SystemVerilog Assertions are verified and moreover functional coverage information is being extracted.

Agosta et al. [6] observe that Aspect-Oriented Programming (AOP) technology is available for C++ and could therefore also be used for system-on chip design using SystemC. Kasuya [7] promotes the usage of aspects in JEDA (a functional verification language) for debugging, performance and coverage measurement, or fault injection. AOP elements are also available in verification languages like e [8], or OpenVera [9].

2 Preliminaries

Transaction Level Models, as used in this work, are a six tuple $S = (M, M_I, M_T, N, T, I)$, where M is a set of modules, $M_I \subset M$ the set of initiator and $M_T \subset M$ the set of target modules. A module needs not necessarily belong to only one category, however $M_I \cup M_T = M$. N is the set of all possible interface method names. The set of transactions is described as $T \subseteq M_T \times N$ and associates target modules and interface method names. Finally, the function $I : T \rightarrow M_I$ maps each transaction to an initiator, meaning that we do not allow an interface method to be called by multiple initiators.

SystemVerilog Assertions (SVA) provide a convenient way of describing sequences and temporal properties. Our methodology to use SVA at transaction level is based on the observation that system functionality can be described in terms of two sequences of transactions and an implication. If a certain sequence of transactions (the precondition) has occurred, then another sequence of transactions (the antecedent) has to occur. For a thorough treatment of using SVA properties, sequences and assertions see e. g. [10].

Aspect-Oriented Programming is an advancement of object-oriented programming, adding additional modularity and expressiveness. The idea behind AOP is to allow a modular implementation of so called *cross-cutting concerns*. Basically this means to locate functionality that affects multiple classes in one common place which exists in parallel to the original code; this place is called an *aspect*. Aspects can be *woven* into the original code using *aspect weavers*. AOP is the ideal mechanism to implement transaction recording for two reasons: (1) it allows the implementation of one generic tracing code in a central place that can be woven into all transactions being recorded and (2) the original code of the TLM can be kept unchanged. The recording functionality is automatically added at the appropriate places by the aspect weaver. We have chosen AspectC++ [11], a C++ flavor of AOP for the implementation of the transaction recording mechanism.

3 Aspect-Oriented Transaction Recording

Figure 1 shows the flow we use to apply SVA to SystemC TLMs. The first step is to weave aspects into the original SystemC code and to compile the resulting model. The aspects contain the transaction recorders that map interface method calls to Boolean signals. Then, the simulation is run, thereby obtaining a Value Change Dump (VCD) file. The next step exploits a feature of most commercial HDL simulators that allows the conversion of a VCD file to a Verilog module. Once converted, the transaction trace is analyzed and elaborated together with the SVA code. Finally, the simulation is run inside the HDL simulator. Assertion failures and passes can be analyzed and coverage information can be obtained.

The algorithm implementing the mapping of transactions to Boolean signals is outlined below:

1. Associate with each transaction a Boolean signal that is $true$ during the execution of the transaction and $false$ otherwise.
2. To be able to detect un-timed non-blocking transactions wait at least one delta cycle before resetting the signal. This requires the transaction to be executed in the context of an `SC_THREAD` process.
3. For each target module create a *local event clock* that toggles after the execution of a method of the target module. This clock is used to trigger the SVA assertions.

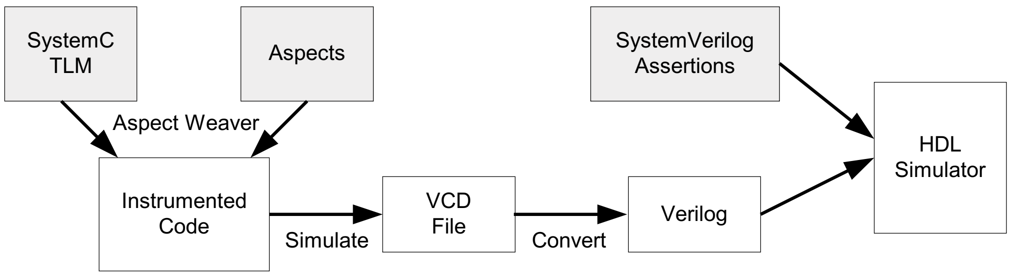

Figure 1: Our proposed approach to Assertion-Based Verification of Transaction Level Models. After transaction recording using AOP and simulation, the resulting VCD file is converted into a Verilog description. This flow allows to apply SystemVerilog Assertions to a SystemC TLM.

4. Create a global event clock that toggles at the end of each transaction regardless of the target module.
5. Map a delta cycle to the smallest simulation time unit supported by the simulator or viewing tool.

To be able to successfully use our methodology, the TLM has to fulfill two prerequisites. Firstly, transactions being traced may only be called in the context of a `SC_THREAD` process. Secondly, the number of delta cycles between two timed events has to be less than the number of time units (using the smallest possible resolution) between those two events.

4 Case Study: Traffic Light System

A simple traffic light system is used as an example to illustrate our methodology (see Figure 2). The system consists of a controller module `tl_ctrl` and two traffic lights `tl1` and `tl2`, respectively. Each traffic light module implements three interface methods, `put_red()`, `put_yellow()`, and `put_green()`.[1] The controller module is responsible to set the traffic lights by calling their interface methods.

The AspectC++ code below shows an aspect called `trace_transaction` generating wrappers for all methods of class `tl_device` starting with `put_` regardless of the return type or the parameters. Assuming that both traffic lights (`tl1` and `tl2`) are instances of the same class, `tl_device`, the aspect generates wrappers for the transactions `put_red`, `put_yellow` and `put_green`.

```
aspect trace_transaction {
 advice execution(% tl_device::put_%(...)) : around {
  // before executing interface method, set signal to true
  ...
  // execute interface method
  tjp->proceed();
  // after executing interface method, set signal to false
  ...
}};
```

Before the transaction is executed, a signal corresponding to the transaction is asserted. Then the transaction itself is executed using the AspectC++ construct `tjp->proceed()` and finally the respective signal is reset again.

[1]We use OSCI-TLM Standard naming conventions as described in [12].

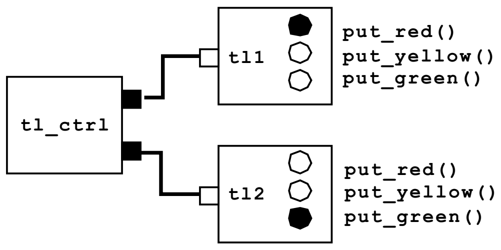

Figure 2: Block diagram of a simple traffic light system.

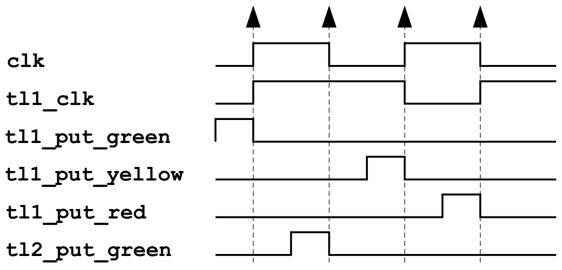

Figure 3: Waveform dump showing parts of the traffic light system.

Figure 3 shows selected waveforms generated for the traffic light system. As stated in Section 2, a transaction is uniquely identified by the interface method name $n \in N$ and the target name $m \in M_T$. Therefore, the signals associated with a transaction are labeled with the interface method name n prefixed by the target name m. This signal is $true$ during the execution of the corresponding transaction. The local event clock of target m is named `<m>_clk` and the global event clock is named `clk`. It can be seen that the global event clock `clk` toggles at the end of each transaction, while the local event clock `tl1_clk` only toggles at the end of transactions related to target `tl1`. The local event clocks are a very important part of our methodology, as they allow to localize and simplify the formulation of properties. Without the local clocks, the formulation of properties would be much more difficult as we do not know (and do not care) how many other transactions occur in the entire system between two consecutive method calls of one target module.

For the traffic light, we require that after a green light, a yellow light has to show up; then, after a yellow light, a red light has to occur, and finally after a red light, a green light should show up. The property for the transition from green to yellow formulated using SVA is shown below:

```
property p_tl1_gy;
  @(tl1_clk) tl1_put_green |=> tl1_put_yellow;
endproperty
a_tl1_gy : assert property( p_tl1_gy );
```

It states that after `tl1_put_green` has become true, with the next edge of `tl1_clk`, `tl1_put_yellow` has to be asserted; the property is checked only at edges of the automatically generated local event clock for the target.

Functional coverage information can be generated on the fly, once properties have been defined for the different transaction sequences. To obtain coverage data for the functionality that after a green light a yellow light has to occur, the following has to be added to the assertion code:

```
c_gy : cover property( p_tl1_gy );
```

Note that this kind of coverage should not be confused with coverage in the context of formal verification which tries to indicate whether enough properties have been defined to exhaustively describe an implementation [13].

5 Conclusions

In this paper, we have shown a methodology for the application of assertions at transaction level using SystemC and SVA. In addition to that, a novel non-intrusive technique for transaction recording has been demonstrated which is based on the application of AOP to SystemC. Currently post-processing of a VCD file is used to check the assertions.

Future work will be targeted towards replacing the post-processing step by a real co-simulation. Another important improvement is the capability to handle parameter values in connection with transactions. To this end, we will add the possibility to create additional (non-Boolean) signals for the parameter values.

References

[1] Accelera Organization. SystemVerilog 3.1. Accelleras Extensions to Verilog. `http://www.eda.org`, 2003.

[2] Thorsten Groetker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Kluwer Academic Publishers (now Springer), Dordrecht, 2002.

[3] OSCI SystemC Verification Working Group. SystemC Verification Standard Specification. `http://www.systemc.org`, 2003.

[4] Prakash M. Peranandam, Roland J. Weiss, Juergen Ruf, and Thomas Kropf. Transaction Level Verification and Coverage Metrics by Means of Symbolic Simulation. In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 260–269, Aachen, 2004. Shaker Verlag.

[5] Thomas Kropf. *Introduction to Formal Hardware Verification* . Springer-Verlag, Berlin, Heidelberg, 1999.

[6] G. Agosta, F. Bruschi, and D. Sciuto. Aspect Orientation in System Level Design (short paper). In *Proceedings International Forum on Specification and Design Languages (FDL'05)*, 2005.

[7] Atsushi Kasuya. Verification Applications of Aspect-Oriented-Programming. In *Proceedings of the Design & Verification Conference and Exhibition (DVCon)*, 2004.

[8] IEEE Standards Activities Department. IEEE P1647/D5 Functional Verification Language 'e'. `http://www.ieee1647.org`, 2005.

[9] Synopsys. OpenVera Language Reference Manual: Testbench. `http://www.open-vera.com`, 2005.

[10] Srikanth Vijayaraghavan and Meyyappan Ramanathan. *A Practical Guide for SystemVerilog Assertions*. Springer Science+Business Media, Inc., New York, 2005.

[11] Andreas Gal, Wolfgang Schroeder-Preikschat, and Olaf Spinczyk. AspectC++: Language Proposal and Prototype Implementation. In *OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, 2001.

[12] Adam Rose, Stuart Swan, John Pierce, and Jean-Michel Fernandez. Transaction Level Modeling in SystemC. OSCI TLM Working Group, 2005. `http://www.systemc.org`.

[13] Hana Chockler, Orna Kupferman, Robert P. Kurshan, and Moshe Y. Vardi. A Practical Approach to Coverage in Model Checking. In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 66–78. Springer-Verlag, 2001.