Assertion Based Verification of PSL for SystemC Designs

Ali Habibi, Amjad Gawanmeh and Sofiène Tahar
Department of Electrical and Computer Engineering
Concordia University
1455 de Maisonneuve, West
Montreal, Québec, H3G 1M8, Canada
Email: {habibi,amjad,tahar}@ece.concordia.ca

Abstract— **In this paper, we present an assertion based verification approach for SystemC designs based on embedding the Property Specification Language (PSL) using Abstract State Machines (ASM). Our approach utilizes an existing embedding of PSL in ASM in order to enable modeling PSL assertions at the ASM level. Here, we propose to compile PSL assertions into C# code, and integrate them with the SystemC design. Assertions are then verified by simulating the new model that combines the original design and the integrated assertions. This enriches the SystemC language with a powerful and expressive assertion specification layer, and improves the verification of SystemC designs by targeting specific properties during simulation.**

I. Introduction

SystemC [8] is a relatively new system level language proposed to overcome the problem of the growth in complexity and size of systems combining different types of components, including microprocessors, DSPs, memories, embedded software, etc. SystemC meets the need for a system level language that can fill the gap between hardware description languages (HDLs) and traditional software programming languages. SystemC comprises C++ class libraries and a simulation kernel used for creating behavioral and register transfer level (RTL) designs. The verification of a SystemC design is a more serious bottleneck in the design cycle. Going further in complexity and considering hardware/software systems will be out of the range of the nowadays used simulation based techniques. In order to improve classical simulation, many proposals offer to use assertion based verification.

In conventional HDL designs, an assertion is a conditional statement that checks for specific behavior and displays a message if it occurs. Assertions are generally used as monitors looking for bad behavior, but may be used to create an alert for desired behavior as well. Assertions are added during verification to monitor conditions that are otherwise hard to check using simulation. They are used to simplify the debugging of complex design problems. Assertion monitors can be thought of as internal software test points that wait for a particular problem to happen and then alert the designer when it does.

In order to provide an efficient language to write assertions, the Accellera organization proposed the Property Specification Language (PSL) [1], which addresses the lack of information about properties and design characteristics. It provides means of specifying design properties using a concise syntax with clearly defined formal semantics. PSL permits specifying a large class of real design properties that range from simple to complex ones since it consists of four layers: Boolean, temporal, verification and modeling layers.

In this paper, we provide a framework to verify PSL properties as assertions for SystemC designs based on an embedding of PSL in Abstract State Machines (ASM) proposed in [3]. ASM [2] is a formal specification method for software and hardware systems that has become successful for specifying and verifying complex systems. The ASM methodology is mathematically precise, yet general enough to be applicable to a wide variety of problem areas [5]. The ASM thesis asserts that any computing system can be described at its natural level of abstraction by an appropriate ASM. There are many languages that have been developed for ASMs, the recent one is AsmL [6], which was developed at Microsoft. This tool offers to generate a C# code from the ASM model. We propose to model the PSL assertions in ASM and then translate them into a C# code using the AsmL tool. Finally, we integrate those assertions as monitors in the SystemC design model in order to check certain behaviors during simulation. We experimented our approach on a number of case studies including a packet switch, which we report in this paper, and where a bug in the original SystemC code has been found.

The rest of this paper is organized as follows: Section II briefly describes SystemC library and the PSL language. Section III presents our approach to verify PSL assertion for SystemC designs. Section IV illustrates our approach on a packet switch case study. Finally, Section V concludes the paper.

II. SystemC and PSL

SystemC is a set of C++ class definitions and a methodology for using these classes [7]. SystemC introduces channels, interfaces, and events to enable communication and synchronization between modules or processes. An interface specifies a set of access methods to be implemented within a channel, where channels provide the implementation for these interfaces. An event is a flexible synchronization primitive that is used to construct other forms of synchronization. Events in SystemC occur at a given simulation time.

0-7803-8558-6/04/$20.00 ©2004 IEEE.

The core language consists of an event-driven simulator as the base. It works with events and processes. The other core language consists of modules and ports for representing structures. Interfaces and channels are used to describe communications. The primitive channels are built-in channels such as signals, semaphores and FIFOs. SystemC provides data types for hardware modeling and certain types of software programming as well.

PSL is an implementation independent language to define properties. It does not replace, but complements existing verification methodologies like VHDL and Verilog test benches. The syntax of PSL is very declarative and structural which leads to sustainable verification environments. PSL consists of four layers based on the functionality of interest [1]:

The modeling layer is used to model behavior of design inputs for formal verification tools, and to model auxiliary parts of the design that are needed for verification.

The verification layer is used to tell the verification tool what to do with the properties described by the temporal layer.

The temporal layer is used to describe properties of the design, as well as simple general properties. This layer can describe properties that involve complex temporal relations. Temporal expressions are evaluated over a series of evaluation cycles.

The Boolean layer is used to build expressions for the other layers, specifically the temporal layer. Boolean expressions are evaluated in a single evaluation cycle.

PSL is a hierarchical language, where every layer is built on top of the layer below. This approach allows the expressing of complex properties from simple primitives. A property (also called assertion) is built from three types of building blocks: Boolean expressions, sequences, which are themselves built from Boolean expressions, and finally subordinate properties. Sequences, referred to as SEREs (Sequential Extended Regular Expressions), are used to describe a single– or multi–cycle behavior built from Boolean expressions.

III. Assertion Based Verification Approach

To support SystemC, PSL can be either integrated as part of SystemC, or put on top of the library. The first approach presents a radical change of SystemC requiring the addition of new constructors and functionalities to the library (like *assert* and *assume*). Besides, the SystemC simulator and semantics must be updated in order to manage, support and verify the assertions and their verification process. Although, this choice may seem to be very efficient, considering the object-oriented aspect of SystemC and its modular structure, it is easier, yet probably more efficient, to add assertions on top of SystemC. In fact, any assertion can be seen as a monitor keeping track of some of the design signals, performing a verification operation and giving a status flag as an output. The open question with the second approach is how to update the design in order to connect the assertion monitors.

The classical way to add PSL assertions to SystemC is to code them in C++ (similar to [4]). However, this option has many drawbacks especially that the C++ language is not adequate to write logical and sequential properties and formulas as defined in PSL. Besides, to make sure that the embedding of PSL in C++ is correct, we must put an important additional effort to validate the new assertion's layer. It will hence be more efficient to model the assertion in a language, like ASM, which offers two very important features: (1) it can model state machines, and (2) it can be translated to a C# code, which supports any other language in the .NET framework (in particular C++).

A. PSL Assertions Integration in SystemC

Figure 1 describes our methodology to integrate and verify PSL assertions for SystemC designs, which consists of the following three main steps:

1) Updating the SystemC design to interface to the assertion monitor.
2) Generating the assertion as a C# code from its ASM description.
3) Integrating the assertion in the design.

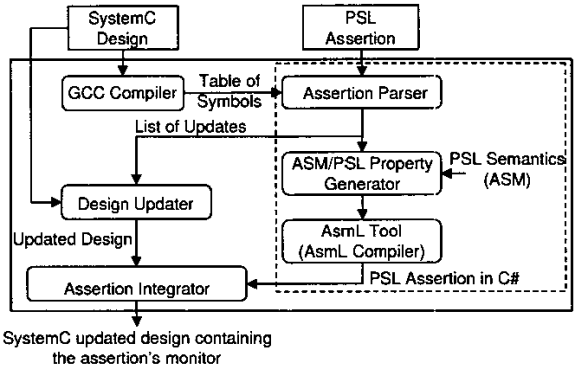

Fig. 1. Methodology to verify PSL assertions for SystemC designs

Generating the table of symbols from the SystemC design is important in order to validate the variables (names and types) that are used in the assertion. In fact, while compiling the assertion, we are concerned with, first, its syntactical correctness, and second, its semantical validity. In this latter, we check the type and the naming of the assertion variables.

Once the assertion's structure verified, we translate it to its equivalent ASM code. In our embedding of the assertion in ASM, we defined a one to one mapping between the PSL assertion and their ASM embedding (see Section III-B). Hence, the transformation is purely syntactical, which guarantees the correctness of the embedded assertion.

In the validation phase of the assertion structure, we also generate a list of updates required to prepare the design to integrate the assertion. For instance, the signals (variables) that are used in the assertion must be seen as external signals so that they can be input to the assertion monitor. So, we provide the Design Updater with a list of variables as defined by their unique identifier in the table of symbols. Then, the Design Updater modifies the SystemC design to make the required

variables visible to the monitor. This transformation does not affect the behavior of the code as it will only be accessed in a read–only mode.

Once the code is updated and the assertion is generated, the Design Integrator will add the required instantiation of the assertion to bind it to the existing SystemC design modules. The assertion monitor, acting as part of the design, can do the following: (1) stop the simulation when the assertion is fired; (2) write a report about the assertion status and all its variables; and (3) send a warning signal to other modules (if required). We note that the internal code of the assertion is C# so the designer can update it or do any other functionalities that can be coded in C#.

B. Embedding PSL in ASM

We embed PSL properties into the design code itself, where all the parameters of PSL properties are defined as objects. The objective of the embedding is to reuse PSL properties, as embedded in ASM, at lower design levels. In fact, the AsmL tool can automatically compile them into a C# or .NET code, which can be compiled and executed with the concrete SystemC level for example.

We defined the embedding of PSL properties in a hierarchical way, where all components are defined as objects and every PSL layer extends its lower layer using the inheritance feature of AsmL. We embedded the first three layers in ASM by defining classes for all types and expressions including their methods. The embedding of the Boolean layer mainly includes (1) the expression type class, which contains the basic types of the language, (2) PSL expressions, which constructs properties using the implication and equivalence operators, and (3) the PSL functions, which include all the functions defined by PSL to operate at this layer. The embedding of the temporal layer includes SEREs and Properties, which contain the operations necessary to create properties from sequential expressions. The embedding of the verification layer includes verification directives, which are used to specify how the assertion will be interpreted (assertion, requirement, restriction or assumption). It also includes the verification unit [1], which is a compact way to encapsulate several properties together. More details about the PSL embedding in ASM can be found in [3].

IV. Case Study: Packet Switch

In this section, we apply the above approach on a packet switch design from the SystemC library [8] [1].

A. Packet Switch

Figure 2 provides a general structure of a 4x4 multi-cast packet switch. The switch uses a self routing ring of shift registers to transfer cells from one port to another in a pipelined fashion, resolving output contention and efficiently handling multi-cast cells. Input and output ports have FIFO buffers of depth four each. Input and output signals are 16-bit

[1] A detailed description of the case study, SystemC code, ASM code, AsmL configuration files, and the generated FSMs for the packet switch are available at http://hvg.ece.concordia.ca/Research/SoC/ASM.

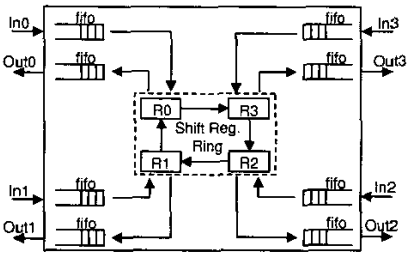

Fig. 2. Switch structure

packets. Each input port is connected to a sender process. Each output port is connected to a receiver process. The sender and receiver processes are given distinguished *id* numbers during instantiations. A sender process, writes a random value to data, and sends it to one or more of the four receivers. Sender processes send packets at random intervals, varying from 1 to 4 units of its clock. A receiver process is activated whenever a packet arrives. Then, it displays the content of the packet and the receiver *id*. The switch operates on an external clock, CLK, and an internal clock, $SWCLK$, which is four times faster. Input and output signals are 16-bit packets with the structure given in Figure 3.

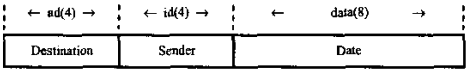

Fig. 3. Packet structure

B. Assertions

For illustration purposes, we consider the following three PSL assertions for the packet switch.

The first assertion, $A1$, is intended to verify that if there is only one recipient for the packet, and the output queue is not full, then the register that holds the packet should be free in the next internal clock, and the packet should be received at the output queue.

$AssertionA1$:
  $forall\ send\ in\ \{0,\ 1,\ 2,\ 3\}$
    $if\ Reg[send].free\ ==\ true\ and$
      $Packet.dest0\ and\ not\ OutQueue[send].full\ and$
      $not\ Packet.dest1\ or\ Packet.dest2\ or\ Packet.dest3$
      $then\ at\ next\ SWCLK$:
        $Reg[send].free\ =\ true$
        $OutQueue[0]\ =\ Reg[send]$

The second assertion, $A2$, is intended to check the shortest path when sending from sender i to receiver i, where the input queue is not full. This operation should be performed in four internal clocks ($SWCLK$) or equivalently one external clock (CLK).

$AssertionA2$:
  $forall\ send\ in\ \{0,\ 1,\ 2,\ 3\},\ forall\ rec\ in\ \{0,\ 1,\ 2,\ 3\}$
    $if\ send\ ==\ rec\ and\ not\ InQueue[send].full\ and$
      $Reg[send].free\ ==\ true\ and$

$$OutQueue[rec].empty == true\ then$$
$$OutQueue[rec] = Reg[send]\ in\ 4\ SWCLK$$
$$and\ OutQueue[rec] = Reg[send]\ in\ 1\ CLK$$

The third assertion, $A3$, is intended to check the (worst) longest path when sender 0 transmits to receiver 3, input queue 3 has only one free slot, all other input queues are full, and the output queue 0 is full. This assertion is specified as follows:

$$if send == 3\ and\ rec == 0$$
$$and\ InQueue[3].size == 3$$
$$and\ InQueue[0].full\ and\ InQueue[1].full$$
$$and\ InQueue[2].full\ and\ OutQueue[0].full\ then$$
$$OutQueue[0] == Reg[3]\ in\ 8\ to\ 19\ SWCLK$$

C. Experimental Results

The AsmL tool is used in order to generate automatically the corresponding C# code for the above PSL assertions. Figure 4 shows, as the example of the integration of the generated C# model for assertion $A1$ with the SystemC model. The connection to the existing objects in SystemC model is done using read-only signals extracted from the packet main module and the switch clock generator.

The simulation of the new model that combines the original design and the integrated PSL properties resulted in successfully verifying the correctness of assertions $A2$ and $A3$. Assertion $A1$, however, was violated, indicating a bug in the SystemC packet switch model. This bug showed, after further inspection of the code, that the switch will free any packet coming from senders 0, 2 and 3 and having at least two destinations including port 1 before routing it to output port (different from port 1). The erroneous code is the following:

$$if\ (R1.val.dest1 \| R1.val.dest1 \| R1.val.dest2 \| R1.val.dest3)$$
$$R1.free = true;$$

where the condition to free the register does not check if the packet is having as destination the port 0 and uses a double copy of the check about the port 1 ($R1.val.dest1$). The correct condition should be:

$$if\ (R1.val.dest0 \| R1.val.dest1 \| R1.val.dest2 \| R1.val.dest3).$$

V. Conclusion

In this paper, we introduced a new approach to verify the Property Specification Language (PSL) assertions for SystemC designs. An assertion is a conditional statement that checks for specific behavior and displays a message if it occurs. Our approach utilized the embedding of PSL in Abstract State Machines (ASM) in order to enable modeling PSL assertions at the ASM level. These assertions are translated into a C# code using the AsmL tool. Finally, we integrated those assertions as monitors in the SystemC design model in order to check certain behaviors during simulation.

We applied this approach on a number of case studies, including a packet switch from the SystemC library, for which

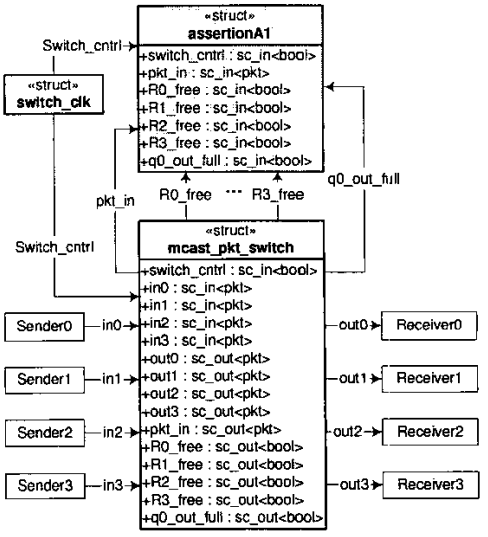

Fig. 4. Integrating assertion $A1$ with SystemC model of the packet switch

we defined various PSL assertions in ASM and generated the corresponding C# code. Then we simulated the new model that combines the original design and the integrated PSL properties. We were able to identify a bug in the SystemC packet switch model through the violation of one assertion. This bug was unknown before and was not caught by traditional simulation.

We believe that this approach improves the verification of SystemC design by targeting specific properties during simulation. It also complements other verification techniques, such as simulation and formal verification, in particular model checking, in order to develop a framework for SystemC verification. In such a framework, the same language, namely PSL, can be used for defining properties, assertions, or even test cases. ASMs represent a very good means to integrate these latter with the original SystemC design in order to improve the verification process. As a future work, we also plan to apply the same approach on other languages such as SystemVerilog.

References

[1] Accellera Organization. Accellera Property Specification Language Reference Manual, version 1.01. http://www.accellera.org, 2004.

[2] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[3] A. Habibi, A. Gawanmeh, and S. Tahar. Embedding of PSL in ASM with an Application to SystemC Verification. Technical Report, Department of Electrical and Computer Engineering, Concordia University, May 2004.

[4] A. Habibi and S. Tahar. On the Extension of SystemC by SystemVerilog Assertions. In *IEEE Canadian Conference on Electrical and Computer Engineering*, IEEE, Niagara Falls, Ontario, Canada, May 2004.

[5] J. Huggins. Abstract State Machines. http://www.eecs.umich.edu/gasm, 2003.

[6] Microsoft Corporation. AsmL for Microsoft .Net Framework (version 2.1.5.7). http://www.research.microsoft.com/foundations/asml, 2004.

[7] Open SystemC Initiative. *SystemC 2.0.1 language reference manual*. 2003.

[8] Open SystemC Initiative. http://www.systemc.org, 2004.