ON THE FORMAL VERIFICATION OF A SYSTEMC PACKET SWITCH MODEL

Ali Habibi and Sofiène Tahar

Concordia University
Department of Electrical and Computer Engineering
1455 De Maisonneuve West
Montreal, Quebec, H3G 1M8, Canada
Email:{habibi,tahar}@ece.concordia.ca

ABSTRACT

In this paper, we present an approach to formally verify SystemC intellectual properties (IPs). We considered as illustrative case a Packet Switch model part of the SystemC library. We propose a verification methodology composed of two steps: (1) static code analysis using abstract interpretation; and (2) model checking. This latter is performed thanks to an integration of both the Property Specification Language (PSL) and the SystemC semantics in the Abstract States Machines (ASMs). We propose a technique based on a reachability algorithm part of the AsmL tool that translates the ASM code combining both the design and the properties into a finite state machine (FSM) representation. We use the generated FSM to run model checking on an external tool, here SMV. Our approach takes advantage from the ASM language capabilities to model designs at the system level as well as from the power of the AsmL tool in generating both a C# code and an FSM representation from an ASM model. The experimental results illustrate, in particular, a corner-case bug that we were able to detect in the design under verification.

1. INTRODUCTION

In order for system level language to model complex yet real System-on-Chip (SoC), it must be equipped with a library of Intellectual Properties (IP) including in particular bus structures, which will be used to interconnect devices such as processors, memories, etc. In this paper, we present a methodology to formally verify a Packet Switch model part of the SystemC library [13]. SystemC [12] is a system level language introduced 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 needs 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 Abstract State Machines (ASM) formalism [2] is used as a modeling language in a variety of domains both in academic and industry contexts. In ASM, large systems modeled on a high level of abstraction can be fit into the validation and verification process. There are many languages that have been developed for ASMs, the recent one is AsmL [11] which was developed at Microsoft Research. We choose this language, as a common level of abstraction, to define an abstract simulator, and then model designs and properties. The AsmL tester [11] can also be used to generate finite state machines (FSMs) or test cases.

The Accellera Property Specification Language (PSL) [1] was developed to address the lack of information about properties and design characteristics in RTL modeling. 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. PSL is formed from four layers: Boolean, temporal, verification and modeling layers.

The verification of SystemC designs is a serious bottleneck in the system design flow. Classical simulation does not guarantee the absence of errors. On the other hand, formal techniques, in particular model checking cannot handle neither the object-oriented (OO) nature of the library nor the complexity of its simulator. In order to overcome these two problems, we propose a bottom–up approach, where the design, which is originally given in SystemC, is reduced to an abstract form called *hypergraph* [14] based on abstract interpretation [3]. The generated hypergrapghs, which represent a reduced model of the design, are then used to construct an abstract model in ASM. Using our PSL modeling in AsmL [8] we are able to reason about the behavior of the design, and its correctness against its specification. We use the AsmL tool in order to generate an FSM for the reduced model (including the properties). This enables the verification of PSL properties for SystemC designs of a reasonable size using classical model checking tools. For this, we translate the generated FSM into the input language of the SMV model checking tool [10].

State machine representations were used in the verifi-

cation of SystemC designs either by applying model checking or guiding functional simulation. For instance, Drechsler *et al.* [4] proposed to use reachability analysis to verify certain properties of a SystemC design. Nevertheless, they restricted SystemC to a Verilog like language. Ferrandi *et al.* [5] proposed to use state machines to perform efficient functional verification of SystemC designs. They constructed an FSM directly from the code then used it to guide the test generation. In that work, the FSM generation was briefly described and does not consider the semantics of the SystemC simulator.

The rest of the paper is organized as follows: Section 2 describes the packet switch model. Section 3 describes the used verification methodology. Section 4 presents the experimental results. Finally, Section 5 concludes the paper.

2. PACKET SWITCH MODEL IN SYSTEMC

Figure 1 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 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 sends random data 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 2.

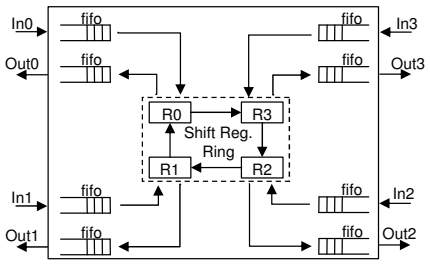

Figure 1. Switch Structure.

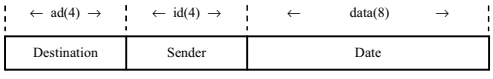

Figure 2. Packet Structure.

3. VERIFICATION METHODOLOGY

For the verification of the packet switch, we use the methodology given in Figure 3[1], where we start by a SystemC design, apply abstract interpretation and generate its hypergraphs. We then translate the events and processes based hypergraphs into ASM using an embedding for SystemC in ASM. We then compile the ASM model, including both the design and the PSL properties, using the AsmL tool and generate its FSM. This FSM is translated into the input language of the model checker, which will evaluate the correctness of the model. Similarly, the AsmL compiler can generate test scenarios, or C# models for verification by simulation.

The generation of the FSM from ASM is performed using the algorithm given in [6]. Since, the AsmL tool is provided as a black–box, we embed the state of every property (as Boolean) in every system's state. Once the FSM is generated, it will include, by construction, a Boolean state variable for evaluating the property. The last step in the verification process is to translate the FSM to the model checker input language, here SMV. Note that there is no restriction on the model checker as the final FSM is concrete and includes only Boolean variables to represent the state of the PSL properties.

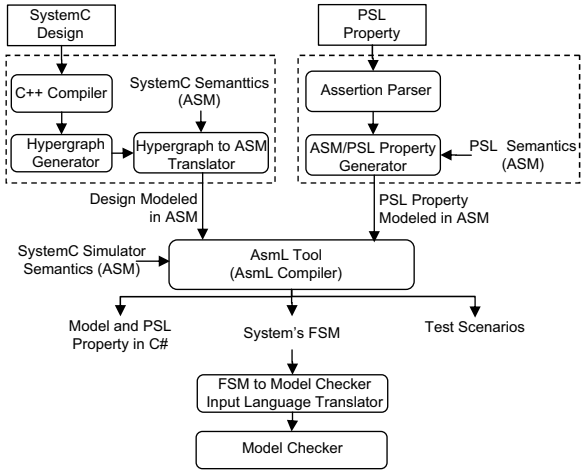

Figure 3. Methodology for Verifying PSL Properties for SystemC Designs.

Abstract syntax domains allow the syntactical manipulation of expressions in order to perform the analysis of the program. In order to allow both the interaction with the user and abstract debugging, we selected the *hypergraph* structure first introduced by [14] to represent the abstract environment. The hypergraph structure can be seen as a general automata connecting its states by branches. These branches can be considered as an extension to Binary Decision Diagrams (BDDs) more adapted to programs representation. In other terms, they offer a higher level of abstraction and flexibility by introducing the notion of confined hypergraph. Hypergraphs define a logical language

[1] A detailed description of the methodology can be found in [7].

and a control language for both simulation and program proofs. They offer a good solution to control the state explosion problem of model checking. In this paper, we use this hypergraph structure to generate the ASM code that will represent the abstracted version of the SystemC code.

The ASM model preserves the behavior of the original model with respect to the execution of the processes and the activation of the events. The abstraction of SystemC, Figure 3, includes two steps: (1) Hypergraph Generator, and (2) reduced SystemC to ASM Translator. We proved in [9] the soundness of this transformation.

Abstraction can be applied to the memory, the environment and the code itself. Since the environment is known statically at each program point, we can use the concrete program environment, which is generated during the compilation phase. It is hence seen as a function that associates with every variable a list of abstracted pointers referring to some locations in the stack.

PSL properties are embedded in ASM as assertions, the assertion here means the validity of the property. It provides a unique view of the property in every system's state. It also simulates the design with the property as a monitor. We build the assertion starting from basic Boolean components, sequences, and then verification units. We encapsulate sequences in the verification unit as an assertion which is embedded in the design. Given a set of Boolean items $x_1, x_2, \ldots, x_n$, and $y_1, y_2, \ldots, y_m$ belonging to the Boolean layer, and the sequences, S_1 and S_2 belonging to the temporal layer, we can define: $S_1 = \{x_1, x_2, \ldots, x_n\}$, and $S_2 = \{y_1, y_2, \ldots, y_m\}$ and then use assertions to check any PSL operation between S_1 and S_2 such as $S_1\ OP\ S_2$, where OP is a PSL operator (e.g., implication ($\Rightarrow$), or equivalence ($\Leftrightarrow$)).

4. EXPERIMENTAL RESULTS

4.1. Static Code Analysis

The direct hypergraph representing the switch includes twelve processes: four senders, four receivers, two clock processes (first clock used for input and output operation and second clock used as internal switch clock), a process for the internal clock of the switch and process for the switch core itself. Only the clocks $clock1$ and $clock2$ are active when the switch starts. The other processes are activated on the reception of a packet or after sending a packet. In parallel with the program environment, the events environment includes the list of all the system processes and their status. The simulation manager is connected to the entries of the program hypergraph. It can be seen as a procedure that determines the structure of the system according to the list of active processes. For example, if the senders 1 and 3 are active, then, only their relative code is analyzed.

The analysis phase relates the elements of the initial hypergraph to a list of general iterators representing the simulation cycle. In other terms, we replace the whole SystemC simulator by a number of loops and iterators that define statically the order of execution of the processes.

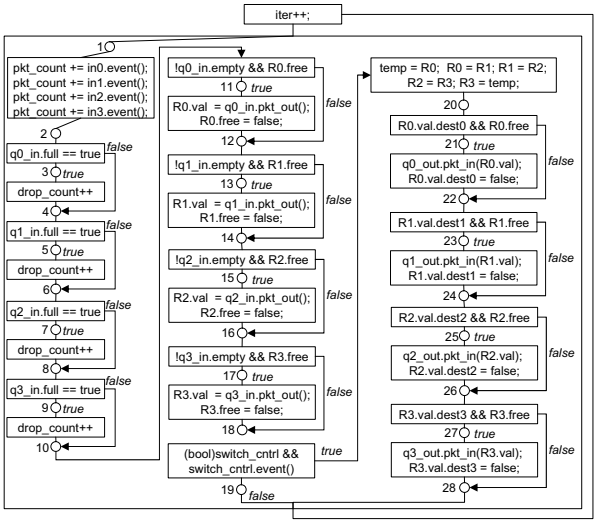

Figure 4. Reduced Hypergraph of the Switch's Main Method.

By applying our reduction techniques on the basic hypergraph of the switch core, we obtained the reduced hypergraph structure given in Figure 4, where the SystemC simulator is reduced to a while loop and that most of the internal variables of the switch are defined as functions of the loop iterators.

From the reduced hypergraph structure a number of properties can be deduced. For example, the total number of received packets pkt_count and the total number of dropped packets at the input FIFOs $drop\_count$ are defined by:

- $pkt\_count = \sum_{i=0}^{itr}\{in0.event()_i + in1.event()_i + in2.event()_i + in3.event()_i\}$
- $drop\_count = \sum_{i=0}^{itr}\{q0\_in.full()_i + q1\_in.full()_i + q2\_in.full()_i + q3\_in.full()_i\}$

where: $qX\_in.full()_i$ and $inX.event()_i$ are Boolean flags set to 1 when the input FIFO X is full and a packet is received from the sender X, respectively.

Although the previous properties may seem to be general, they can offer very precious information about the internal way the switch is working. At the same time, they allow the detection of behavioral errors. For example, the second property states that "the count of dropped packets is equal to the number of times the input queue (FIFO) is full", which is not correct. In fact, the correct property should state that "the count of dropped packets is the number of received packet at the entry of the FIFO when the input queue (FIFO) is full". In other words, we have to receive a packet when the FIFO is full to say that the packet was dropped. So, the condition to count the dropped packets must be changed from: "if((*q0_in.full* == true)" to "if((*q0_in.full* == true) && *in0.event*())".

Note also that the properties obtained from the analysis phase can be used to validate the switch's specification (e.g., maximum number of dropped packets). According to the reduced hypergraph, the switch core only uses the

packet's header to process the packet. Therefore, we can reduce the packet to its header (4 bits destination and 4 bits identifier), which would ease the use of model checking to verify some behavioral properties of the switch.

4.2. Model Checking

Using the above abstract mode, we verified a set PSL properties on the packet switch. For illustration purpose, two properties are given below:

The first property, $P1$, 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.

$$
\begin{array}{l}
Property P1: \\
\quad forall\ send\ in\ \{0,1,2,3\} \\
\quad\quad if\ Reg[send].free\ ==\ true\ and \\
\quad\quad\ Packet.dest0\ and\ not\ OutQueue[send].full\ and \\
\quad\quad\ not\ Packet.dest1\ or\ Packet.dest2\ or\ Packet.d3 \\
\quad\quad\quad then\ at\ next\ SWCLK: \\
\quad\quad\quad\ Reg[send].free\ =\ true \\
\quad\quad\quad\ OutQueue[0]\ =\ Reg[send]
\end{array}
$$

The second property, $P2$, 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).

$$
\begin{array}{l}
Property P2: \\
\quad forall\ send\ in\ \{0,1,2,3\},\ forall\ rec\ in\ \{0,1,2,3\} \\
\quad\quad if\ send\ ==\ rec\ and\ not\ InQueue[send].full\ and \\
\quad\quad\ Reg[send].free\ ==\ true\ and \\
\quad\quad\ OutQueue[rec].empty\ ==\ true\ then \\
\quad\quad\quad OutQueue[rec] = Reg[send]\ in\ 4\ SWCLK \\
\quad\quad\quad and\ OutQueue[rec] = Reg[send]\ in\ 1\ CLK
\end{array}
$$

The AsmL tool is used in order to generate automatically the FSM of the packet switch and having the evaluation of the properties as a state variable. The model checking resulted in successfully verifying the correctness of $P2$. $P1$, 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:

$$
\begin{array}{l}
if\ (R1.val.dest1 \| R1.val.dest1 \| R1.val.dest2 \| \\
\qquad R1.val.dest3) \quad R1.free = true;
\end{array}
$$

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:

$$
\begin{array}{l}
if\ \ (R1.val.dest0 \| R1.val.dest1 \| R1.val.dest2 \| \\
\qquad R1.val.dest3).
\end{array}
$$

5. CONCLUSION

In this paper, we presented an approach for formally verifying a SystemC packet switch model using a cascade of: (1) reduction techniques based on abstract interpretation; and (2) transformation to AsmL. We used a previously defined embedding of the semantics of SystemC components library and simulator in order to provide an abstract SystemC simulator in ASM. The SystemC design model is reduced based on abstract interpretation in terms of hypergraphs. An ASM model combining both the reduced hypergraph and the PSL property is input to the AsmL tool, as an FSM of the system including the properties. The verification of PSL properties is performed using the SMV model checker after translating the FSM model (given in *dot* format) into the input language of SMV. We have been able to verify a set of PSL sample properties and to detect a bug until today not found using simulation.

6. REFERENCES

[1] Accellera Organization. Accellera property specification language reference manual, version 1.01. www.accellera.org, 2004.

[2] ASM. website: www.eecs.umich.edu/gasm, 2005.

[3] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theo. Comp. Sc.*, 277(1-2):47–103, 2002.

[4] R. Drechsler and D. Große. Reachability analysis for formal verification of SystemC. In *Proc. Symposium on Digital System Design*, pages 337–340, Dortmund, Germany, 2002.

[5] F. Ferrandi, M. Rendine, and D. Sciuto. Functional verification for SystemC descriptions using constraint solving. In *Proc. Design, Automation and Test in Europe*, pages 744–751, Paris, France, March 2002.

[6] Y. Gurevich, B. Rossman, and W. Schulte. Semantic Essence of AsmL. Technical report, Microsoft Research Tech. Report MSR-TR-2004-27, March 2004.

[7] A. Habibi. *A Framework for System Level Verification: The SystemC Case*. PhD thesis, Concordia University, Montréal, Quebec, Canada, 2005.

[8] A. Habibi and S. Tahar. An Approach for the Verification of SystemC Designs using AsmL. In *Automated Technology for Verification and Analysis*, LNCS 3707, pages 69–83. Springer Verlag, 2005.

[9] A. Habibi and S. Tahar. On the Transformation of SystemC to AsmL using Abstract Interpretation. *Electronic Notes in Theoretical Computer Science*, 131:39–49, May 2005.

[10] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[11] Microsoft Corp. AsmL for Microsoft .NET Framework. research.microsoft.com, 2004.

[12] Open SystemC Initiative. www.systemc.org, 2005.

[13] OSI. *SystemC 2.0.1 language reference manual*. 2005.

[14] F. Vederine. *Analyses totales de programmes par interpretation abstraite*. PhD thesis, Ecole Polytechnique, Paris, France, 2000.