On the Design and Verification Methodology of the Look-Aside Interface

Ali Habibi, Asif Iqbal Ahmed, Otmane Ait Mohamed and Sofiène Tahar
Department of Electrical and Computer Engineering
Concordia University
1455 de Maisonneuve, West
Montreal, Québec, H3G 1M8, Canada
Email: {habibi,a_ahmed,ait,tahar}@ece.concordia.ca

Abstract

In this paper, we present a technique to design and verify the Look-Aside (LA-1) Interface standard used in network processors. Our design flow includes several refinements starting from an informal UML specification until getting to an RTL modeled in Verilog. We integrate the verification of the LA-Interface in the design flow by considering two intermediate levels: (1) Abstract State Machines (ASM); and (2) SystemC. The first one serves the verification by model checking of a set of PSL properties, while the second includes a set of assertions to be verified by simulation. To evaluate the performance of our approach, we used the RuleBase model checker to verify the same properties; and the OVL library to verify the same assertions.

1. Introduction

New IPv6 systems and carriers increasingly demanding detailed lookups on packets and flows, the interface between network processors and other components, such as external co-processors and memory devices, is taking the spot light in the networking sector. Currently, the Look-Aside 1 (LA-1) interface [7] is the defacto standard for linking these components [3]. It is being the key to several networking-specific applications, including packet forwarding, packet classification, admission control, and security.

Designing LA-1 directly at the RTL level is possible, however, it is time and effort consuming and certainly an error-prone process. In this paper, we propose an approach to design and verify the LA-1 interface in a hierarchical flow using an object-oriented (OO) methodology and several refinements levels. We aim to design an IP (Intellectual Property) module that can be used as a Look-Aside Interface or as a validation unit for LA-1 compatible devices. For this reason, we propose to use a top-down design and verification approach.

In our model, UML presents the top level to capture a precise specification of the design and its properties. Then, comes Abstract State Machines (ASM) [6], a high level language providing a variety of features that allow the description of the relevant state of a system in a high level of abstraction, to represent a formal and precise model of the design. At this level, we introduce the first step of the verification process which is model checking. The properties of the under verification model are in PSL (Property Specification Language) [2], a standard language for properties specification. PSL language supports specifying a large class of real design properties that range from simple to complex ones in a hierarchical yet very well organized structure consisting of four layers: Boolean, temporal, verification and modeling layers.

Once the model verified at the ASM level we translate it to SystemC [8], a relatively new system level language for embedded system design and verification. The PSL properties modeled in ASM are also translated to C# and combined to the SystemC model in order to perform assertion based verification using simulation. Finally, we derive the synthesizable RTL implementation in Verilog from the SystemC design.

In order to prove the correctness of the refinement process, with respect to the verified properties, we re-verify the same PSL properties for the RTL Verilog code using the RuleBase [4] model checker. For instance, such a proof will set that both the high level model in ASM and the low RTL level satisfy the same set of properties. In addition to that, comparing our model checking methodology to RuleBase represents a good evaluation of the integration of the verification at the early design stages (ASM level for our case). We also used the Open Verification Library (OVL) to verify the same assertions as those integrated in the SystemC model in order to compare the simulation performance of SystemC against the Verilog/OVL based approach.

As related work to ours, we cite the approach proposed by Kanna *et* al. [10] to implement a PCI bus as a Verilog monitor and to verify its properties using SMV [5]. In [10], the bus was implemented in Verilog with all the properties embedded as part of the code. This makes its modification or upgrade a very complex task. Besides, the Verilog model they verified includes only 2 agents (one master and one slave), which considers a relatively small design in comparison to LA-Interface with four banks.

The rest of this paper is organized as follows: Section II introduces the SystemC, PSL and ASM. Section III presents the LA-1 Interface. Section IV describes the proposed LA-1 Design Methodology. Section V details the verification approach. Section VI contains the experimental results. Finally, Section VII concludes the paper.

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

2. SystemC, PSL and ASM

2.1. SystemC

SystemC is a set of C++ class definitions and a methodology for using these classes [9]. SystemC introduces channels, interfaces, and events to enable communication and synchronization between modules or processes. 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.

2.2. PSL

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 [2]:

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.

2.3. ASM

An ASM model by definition encodes only those aspects of the system's structure that affect the behavior being modeled. ASM provides a variety of features that allow the description of the relevant state of a system in a very economical, high-level way. Each abstract state machine represents a particular view of the distinct operational steps that occur in the real system being modeled. AsmL [6] is a language used to model systems in ASM. It is not, however, restricted to ASM but offers a rich programming language. It is integrated with Microsoft's software development environment including Visual Studio, MS Word, and Component Object Model (COM), where it can be compiled and connected to C# or to the .NET framework [6].

3. Look-Aside Interface

The LA-1 standard is the look-aside interface to network-processing elements (NPEs). It targets look-up-tables and memory-based coprocessors and emphasizes as much as possible on the use of the existing technology. It is based on QDR and Sigma RAM technologies. Although modeled on an SRAM interface, the LA-1 specification aims to accommodate other devices as well, such as classifiers and encryption co-processors.

The LA-1 interface major features include:

- Concurrent read and write operation
- Unidirectional read and write interfaces
- Single address bus
- 18 pin DDR data output path transfers 32 + 4 bits of even byte parity per read.
- 18 pin DDR data input path transfers 32 + 4 bits of even byte parity per write
- Byte write control for writes

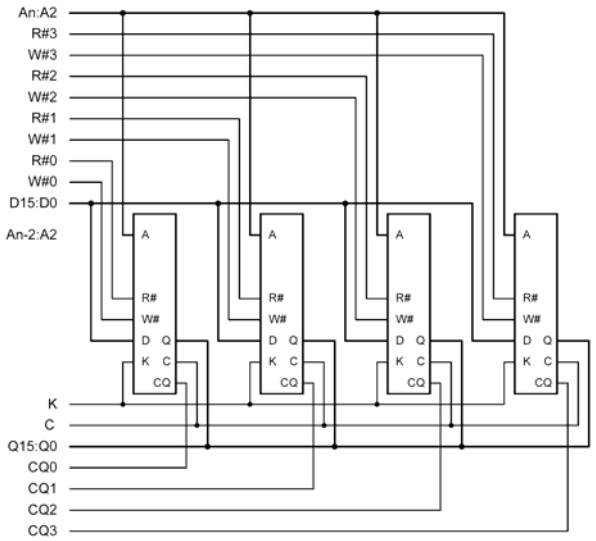

Figure 1. Look-Aside Interface (4 Banks) [7].

The LA-1 interface requires a master-clock pair. The master clocks (K and K#) are ideally 180 degrees out of phase with each other, and they are outputs for the host device and inputs for the slave device. A write cycle is initiated by asserting WRITE_SEL (W#) low at rising edge of K (K clock). The address of the Write cycle is provided at the following edge of K (K# clock which 180 degrees out phase from clock K). A read cycle is initiated by asserting READ_SEL (R#) low at rising edge of K (K clock) and the read address is presented on the same rising edge.

4. LA-1 Design

Our approach is shown in Figure 2, where we start with an informal specification for the intended design developed in UML. This step provides a better view of the design components and their interactions. Based on the UML model (class diagram, sequence diagram, etc.), we develop an ASM model, which will enable the verification of the design using formal tools (model checkers for our case). When the verification terminates with an error, we update UML specification and re-capture them to ASM. Finally, when the verification passes without errors, we map the verified ASM model to a SystemC implementation. This latter is then refined and verified using assertion based verifications (ABV). Finally, we map the SystemC model to a low level syntehsizable HDL implementation (Verilog for our case).

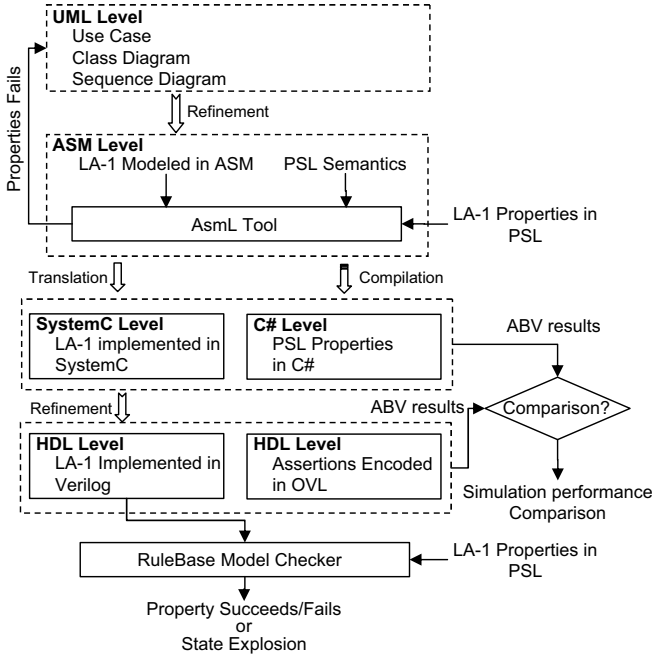

Figure 2. Look-Aside Interface Design and Verification Methodology.

We combined both the verification and design processes in the same path. We applied model checking at higher level of abstraction (ASM) to avoid state explosion problems. On the other hand, we used RuleBase at the lowest level (HDL) for comparison purpose against using ASM. ABV is applied to both SystemC and Verilog models in order to: (1) compare the simulation performances of the SystemC simulator against the commercial verilog simulators; and (2) to evaluate our ABV technique in comparison to the OVL/Verilog methodology [1].

4.1. UML LA-1 Specification

We designed the LA-Interface considering a structure based on four principle classes: Write Port, Reading Port, SRAM Memory and a Light Simulator (optional classes include properties). Such an architecture guarantees that the final design can be used in two different ways:

- A stand-alone IP: to integrate larger SoC.
- A Verification Unit: to validate other LA-1 Interface compatible devices.

In addition to the classes involved in the design, we embedded a light semantics of a light synchronous Verilog like simulator. As first impression we may consider that such an addition module in the system presents an overhead to the ASM model; nevertheless, its size is relatively small compared to the complete ASM design. Besides, such a module offers a higher level of programming and instantiation flexibility. For example, it allows to upgrade the design from 1 bank to 4 banks (actually, for any number N of banks) by just a matter of object instantiation. Basically, this is one of the main objectives to use a purely object-oriented approach to design the LA-1 Interface at the ASM level. On the other hand, in order to verify precise clocked properties at earlier stages of the design, it is mandatory to embed certain semantics of the simulation cycles.

In order to enable a better representation of the properties at the UML level, we propose to use a modified sequence diagram where new notation are included to enable specifying information principally to the methods activation clocks, execution cycles and duration of execution. An illustration of the modified sequence diagram, for the read mode scenario, is displayed in Figure 3. Such a diagram says that a read scenario starts by putting a read request at the clock K which causes the ReadPort to request the data from the SRAM in the next cycle at the same clock K. After formatting the data, the ReadPort releases it in two consecutive steps at the next rising edges of K and $K\#$.

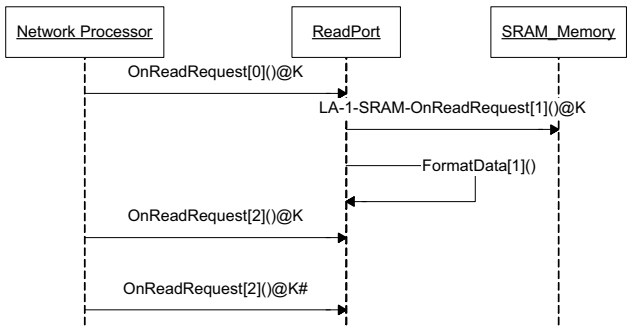

Figure 3. Sequence Diagram for the Reading Mode.

4.2. ASM Model

The ASM model includes both the design and the properties. All the design classes are mapped from the UML model. However, a number of updates is required to guarantee a correct generation by the AsmL tool of the model's FSM. For instance, the firstly explored action by the reachability analysis algorithm embedded in the tool must initialize all the model's objects. An execution with non-initialized objects may results to an exception or a misbehavior of the exploration algorithm. We also need to add

at the entry of every method a pre-condition defining the rules filtering the states where the method can be executed. This is one of the most critical issues in defining the ASM model because a wrong definition of the pre-condition may totally change the behavior of the system and consequently modify the verification results. The pre-condition rules define also a way of communication between the embedded light simulator and the other modules. Actually, one of the basic rules relates the method to the clocks.

Figure 4 illustrates the ASM semantics of the light Verilog simulator. It includes in particular an initialization function that is executed after pre-condition "all the all the system modules have been initialized" (condition: SystemFlag = STARTED). The same method includes a second pre-condition setting that the method is only executed at the initialization phase (condition: SimStatus = INIT). This illustrates how the pre-condition constructor is used to manage the exploration algorithm performing the reachability analysis.

```
class SimManager
  public var m_K as ClockEvent = CLK_UP        // Main system clock
  public var m_Ks as ClockEvent = CLK_DOWN // Negation of the main clock
  public var m_E as BANK_ID = BANK_0

  public SimManager()

public SimManager_Init()
    require SystemFlag  = STARTED and SimStatus = INIT
    me.m_K  := CLK_UP   //set first clock to high
    me.m_Ks := CLK_DOWN //set second clock to down
    forall  w in WPORTS
      if(w.m_E = m_E) then
        w.LA1_WP_OnReceiveData_Depth  := any rec | rec in {true,false}
      else
        w.LA1_WP_OnReceiveData_Depth  := false
    forall r in RPORTS where r.m_E = m_E
      if(r.m_E = m_E) then
        r.LA1_RP_OnReadData_Depth     := any rea | rea in {true,false}
      else
        r.LA1_RP_OnReadData_Depth     := false
    forall  s in SRAMS
      s.LA1_SRAM_OnWriteData_Depth  := false
    SimStatus := CHECKING_PROP

  public SimManager_Restart()
    require SystemFlag  = STARTED and SimStatus = STOPPED
    SimStatus := INIT
```

Figure 4. Light Verilog Simulator in ASM

The LA-1 Interface properties are extracted from both the sequence diagrams and the class diagram. Once mapped to ASM classes, assertions need to be instantiated correctly. Actually, some properties are referring to certain objects (Bank 2 for e.g.) which requires a particular instantiation of certain modules. Besides, some methods may require additional variables to enable recording certain system's variables. For example, in a read scenario, the assertion needs to record the number of clock cycles since the initial request has been issued.

In this work, we used the PSL to specify the design's properties. Actually, in addition to the fact that PSL is the standard for properties specification, our choice was guided by our previous work where we defined a deep embedding of PSL in ASM. PSL properties were implemented in a hierarchical way: (1) with all the components defined as objects; and (2) every PSL layer *extends* its lower layer using the inheritance feature of AsmL. This OO embedding of PSL promotes re-using the verification classes in other designs.

4.3. SystemC

The SystemC design is directly obtained from the ASM model using a syntax transformation. For instance basic types (including Boolean, integer, byte, etc.) are one-to-one mapped between C++ and AsmL with exactly the same operations and semantics. Every class from the ASM model is translated to SystemC module. The pre-conditions in the ASM methods are included in the SystemC Constructor section as triggering conditions for the SystemC methods.

The object instantiation in the ASM model is translated to a naming mapping in the SystemC design and is integrated in the design's main procedure *sc_main*. We will discuss in Section 5 a technique to make sure both the ASM and systemC models behaves exactly the same way for the same inputs.

4.4. Verilog

A synthesizable Verilog implementation is derived from the SystemC design. For the case of the LA-1 Interface, we map each class to a Verilog module. The transformation is straight thanks to the basic types used in the LA-1 SystemC design (which only includes enumerated types, Boolean and integer). Multiple banks model is obtained from the single one by instantiating the Read, Write and Memory modules. The connection between the control signals is performed using tristate buffers.

5. Verification Approach

5.1. Using AsmL for Model Checking

The AsmL tool is not a model checker for PSL properties. However, it includes a general algorithm implementing reachability analysis (also called state space exploration [6]). The AsmL tool generates the model's FSM by executing the model program in a special execution environment, keeping track of the actions it performs and recording the states it visits. This process is called exploration. Usually a model program implies so many states and transitions that it is not feasible to include them all in the FSM, so you must limit the number of states and transitions that the tool explores. The FSM that the tool builds is usually only a portion – an under-approximation – of the huge FSM that would result if the model program could be explored completely. As a result, the test suite generated from the FSM usually does not cover all possible states and transitions of the model program.

By adapting the exploration algorithm we've been able to implement a model checking procedure for PSL properties. For instance, we encode a PSL property (modelled in ASM) as two state variables P_{status} and P_{value} meaning respectively that the property is under verification or already verified and giving the current value of the property. In other terms, a property is: (1) correct if $P_{status} = true$ and $P_{value} = true$; (2) incorrect if $P_{status} = true$ and $P_{value} = false$; and (3) having an undefined value if $P_{status} = status$ (this is the case when a temporal property over several cycles is being verified in an intermediate state).

Verifying PSL properties by covering for all the system state space is not always feasible due to state explosion problem. Defining a smart configuration to guide the state exploration is a very important step towards enabling model checking using AsmL. For instance, defining the domains, which are defined as finite collections of values from which method arguments are taken, are the most important issues to consider. For instance, for an integer input that can only take a value in the range from 5 to 23, considering all possible integer values for the type $AsmL.Integer$ is a waste of time; besides, this may get the verification to output wrong results.

In order to provide counter-examples when the verification fails we define for every property a filter stopping condition defined by the conjunction: "$P_{status} = true \,\&\, P_{value} = false$". The generated portion of the state machine from the initial state until getting to the stop error point forms a complete path for a counter-example, an information very useful in order to debug and correct the design's errors.

In order to guarantee that the translated SystemC model is conform to the original ASM model we perform a conformity check using AsmL. In fact, AsmL tool performs a conformance test by executing the program under test, called the implementation (SystemC model for our case), together with the model program in ASM. The tool executes the exploration algorithm in the same time on both the ASM model and a binary executable generated from the SystemC design. It then verifies if for all the possible inputs, both models behave the same. For instance, this is phase is sometimes time consuming, however, it is quite important to make sure the ASM to SystemC mapping preserves the system's properties.

5.2. Using RuleBase For Model checking

RuleBase is one of the first model checkers supporting PSL (actually it supports even its ancestor Sugar). We find it quite complicated to verify PSL properties using this model checker due to the important effort that we spent defining the environment and writing the behavioral model including all the design levels of hierarchy. We used the same properties that we verified at the ASM level with PSL. The main objective is to compare the efficiency of performing model checking at earlier stages of the design flow.

5.3. ABV using SystemC and C#

As it was shown in Figure 2, we propose to integrate PSL assertion to SystemC designs as external monitors implemented in C#. These latter are directly compiled from the PSL properties modeled in ASM. They are then integrated with the SystemC model. The integration requires, in particular, updating the SystemC design to interface to the assertion monitor. For instance, the signals (variables) that are used in the assertion must be seen as external signals in the model so that they can be input to the assertion monitor. So, the list of variables involved in the assertion is required in order to make them visible to the assertion 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, an instance of the assertion is bound 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#.

5.4. ABV using OVL

The Accellera Open Verification Library (OVL) provides designers, integrators and verification engineers with a single, vendor-independent interface for design verification using simulation, semi-formal and formal verification techniques [1]. Assertion monitors are instances of modules whose purpose is to verify that certain conditions hold true. An assertion monitor is composed of an event, a message, and a severity. Event is a property that is being verified by a monitor. An event can be classified as a temporal or static property. A static property is a property that must be valid at all times, whereas a temporal property is a property that is valid only during certain times. Message is the string that will be displayed in case of an assertion failure. Severity represents whether the error captured by the monitor is a major or a minor problem.

Using OVL assertions is promoting for small size HDL designs. Nevertheless, when the complexity of the design becomes important, writing OVL itself is not an easy task. Writing the assertion for the reading mode, for example, requires encoding all the atomic operations in separate modules which gets to complex final design in the simulation since every call to an OVL will load the correspondent module as part of the simulated design.

6. Experimental Results

In following, we describe our results on the verification of the LA-1 Interface using the proposed methodology. The ASM and SystemC experiments were performed on a 2.4 GHz Pentium IV and 512 MB of RAM (PC 2700) while the RuleBase and OVL experiments were performed on 2 X UltraSPARC-III+ machine with 2 900Mhz processors and 4096M of RAM.

6.1. Model Checking

Table 1 shows the CPU time required to verify all the interface properties combined together. The number of states and transitions refers to generated FSM. This illustrates the efficiency of our approach to handle model checking of relatively complex case study thanks to raising the abstraction level. We note that the number of nodes and transitions is not the total number present in the global system FSM. It represents the FSM that's generated according the AsmL configuration parameters including the domains, methods, variables and actions.

Table 2 gives a summary of the model checking results using RuleBase for the Read Mode operation. We notice that even though the tool succeeds to verify the property for up to 3 banks the required time is relatively big. So, considering more complex designs is out of the scope of the tool which is confirmed by the raise of the state explosion problem when considering 4 banks. It is not adequate to compare both levels ASM and Verilog considering the gap of abstraction between the two models, however, these

Number of Banks	CPU Time (s)	Number of FSM	
		Nodes	Transitions
1	8.093750	93	106
2	18.093750	212	185
3	28.078125	277	318
4	40.734375	369	424

Table 1. Model Checing Using AsmL.

Number of Banks	CPU Time (s)	Memory (in MB)	Number of BDDs
1	675.13	58	812785
2	2530.75	226	9.2×10^7
3	15264.21	590	52.4×10^7
4	State Explosion		

Table 2. Model Checking Using RuleBase: Read Mode.

results confirm that applying model checking cannot be reused the classical way for low level HDL models but needs to be applied at the early stages of the design's flow.

6.2. Simulation

Table 3 compares the average of execution time per cycle for the assertion based verification of SystemC design with assertions in C# and the Verilog design with assertions in OVL. For both cases we considered the same assertions describing the Reading Mode. We notice that the SystemC simulation runs always at least 20 times faster the Verilog design. In other terms, 1 day simulating SystemC is equivalent to more than 20 days simulating Verilog (and OVL) for LA-1 Interface. Besides, we notice that the larger is the system, the faster is the SystemC simulation in comparison to Verilog. This clearly confirms the adequation of using SystemC for the design and also for the verification by simulation of designs at the system level.

Number of Banks	Average of execution Time/cycle (δ in s)		Ratio = δ_{OVL}/δ_{SC}
	SystemC (δ_{SC})	OVL (δ_{OVL})	
1	21.22×10^{-9}	0.44×10^{-6}	20.73
2	26.25×10^{-9}	1.90×10^{-6}	72.38
3	30.07×10^{-9}	2.91×10^{-6}	96.77
4	38.73×10^{-9}	3.68×10^{-6}	95.01

Table 3. Simulation Results

7. Conclusion

In this paper, we presented a design and verification approach using UML, ASM and SystemC to create a Verilog IP for the Look-Aside Interface de-facto standard for interfacing network processors to other components. We first defined the interface's model in UML, then, we translated it to ASM in order to verify, using model checking, a set of PSL properties. After verifying all the PSL properties, we mapped the ASM code to SystemC and the PSL properties to C# assertions (modules) in order to enable ABV at the SystemC level. Finally, we implemented the SystemC model in a synthesizable Verilog RTL. By proving the writing and reading modes properties using RuleBase for the RTL implementation, we also proved the correctness of our refinement from ASM to RTL (for one, two and three banks). Besides, experimental results showed some limitations of the tool and hinted that using model checking is more efficient when integrated at the early design phases. We also, inserted the same assertions, used at the SystemC level, to the Verilog implementation using the OVL library. Simulation results displayed a superior performance of SystemC over the OVL based approach. Finally, we believe that the proposed approach improves the design and verification of IP blocks.

As future work, we consider proving the soundness of the complete refinement process from ASM to RTL. This will allow reusing the verification results that can be proved at any level for the other lower levels; thus, reduce the complexity of verifying RTL.

References

[1] Accellera Organization. Open Verification Library, Assertion Monitor Reference Manual, v 03.06.06., 2003.

[2] Accellerab Organization. Accellera Property Specification Language reference manual, Version 1.01., 2004.

[3] H. Bhugra. La-1b: Moving the look-aside interface forward. *CommsDesign*, August 2002.

[4] IBM Haifa Research Laboratories. *RuleBase Formal Verification Tool (Version 1.5). Users Guide.* May 2003.

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

[6] Microsoft Corporation. AsmL for Microsoft .NET Framework, Microsoft., 2004.

[7] Network Processing Forum. *Look-Aside (LA-1) Interface, Implementation Agreement, Revision 1.1.* Kluwer Academic Publishers, April 15, 2004.

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

[9] OSI. *SystemC 2.0.1 language reference manual.* 2004.

[10] K. Shimizu, D. L. Dill, and A. J. Hu. Monitor-based formal specification of PCI. In *Formal Methods in Computer-Aided Design*, pages 335–353, Austin, Texas, November 2000.