Generic Framework Architecture for Verifying Embedded Components

Lamia ELJADIRI[1]
LIMSAD Laboratory in Mathematics and Computer Science
Department, Faculty of Sciences FSAC
University Hassan II, Casablanca, Morocco

Ismail ASSAYAD[2]
LIMSAD Laboratory in Mathematics and Computer Science
Department, ENSEM
University Hassan II, Casablanca, Morocco

***Abstract*—This dissertation presents a framework for the formal verification of standard embedded components such us bus protocol, microprocessor, memory blocks, various IP blocks, and a software component. It includes a model checking of embedded systems components. The algorithms are modeled on SystemC and transformed on Promela language (PROcess or PROtocol MEta LAnguage) with the integration of LTL (Linear Temporal Logic) properties extracting from state machines in order to reduce verification complexity. Thus, SysVerPml is not only dedicated to verifying generated properties but also for the automation integration of other properties in models if needed. In the following, we will provide the answer to the problems of component representation on the design system, what properties are appropriate for each component, and how to verify properties.**

***Keywords*—Algorithms; automation; embedded components; embedded systems; formal verification; framework; LTL properties; Promela; SystemC; SysVerPml; system design**

I. Introduction

Verification can be applied to discover errors early in the SOC (System On Chip) design against properties expressed as part of the requirements. Worth to mention that the cost to find errors and to make correction in the product line increases ten times like what industry study demonstrates [1]; it is revealed that verification accounts for 55% in totality project time between 2012 and 2016.

The formal verification technology is divided into three methods: equivalence checking, model checking, and theorem proving [2], [3].

Equivalence checking is a technique based on mathematical approach to verify the equivalence of a reference or golden model to the implementation of the model [4].

Model checking is an algorithmic technique for determining whether a system satisfies a formal specification expressed as a temporal logic formula, where properties are the direct representation of a design's behavior [5].

Finally, the theorem proving method has the ability to decompose a problem especially the case of microprocessor verification. More details on theorem proving can be found in [6].

The three formal methods are generally used as formal verification techniques. However, model checking is particularly used in protocol verification. Model checking method [7] treats all the possible behavior of the design model.

The method called Undounded Model Checking (UMC) is based on the translation of the model checking problem into the satisfaction problem of a propositional formula, unlike the Bounded Model Checking (BMC) the encoding of the formulas is different. While the two techniques shares the encoding of the states and the transition relation of the model as explained on the article [8].

This article gives an overview of our Model Checker Platform named SysVerPml; the tool allows creating an abstract model of the design instead of translating SystemC programs to formal models, and then checking them using verification tool SPIN (Simple Promela INterpreter).

Model checking focuses on the state-space explosion problem. The main idea in our approach is that the number of states of a design is exponential to the number of variables and the width of each variable. To attain this first aim as explained in our previous article [9] the modeling methodology of a system must exhibit the execution semantics instead of encompassing it inside an execution-scheduler. Moreover, in order to allow new and old systems integration, any process interaction which might be useful for inter-system integration must not be cut in the final system model. The challenge of this approach is to guarantee that the abstract model is exact to the granularity of programs behaviors states. For that we use the code-level way of verification, as explained in the article [10], which has the advantage of permitting compositional verification of programs by keeping their incomplete interactions.

In the following, we first state the verification environment supported by our approach describing the different plug-in component used by the framework. Second, study case is taken as an example for the proposed method with the complete transformation procedure for the SRAM component. We conclude the resume with tests of the performance verification of our framework followed by conclusion.

II. Verification Environment

By the collaboration and exploitation of core integration technology, we can focus on core competencies to invent development technology as our platform SysVerPml will allow us. The SysVerPml tools have been developed over the Eclipse development environment. The open source integrated

development environment (IDE) Eclipse developed by IBM, Object Technology International (OTI), and eight other companies [11], [12[, [13], [14]. This IDE mainly allows providing an extensible platform for building software. As shown in Fig. 1 the major advantage is that it gives extensible facilities which makes possible to implement tools of our framework by plug–ins such us the use of SystemC plug-in, IPXACT plug-in, and JSpin Java GUI for SPIN (graphical user interface for the SPIN Model Checker).

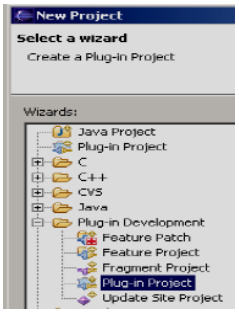

Fig. 1. Adding Plug-in Project from the Plug-in Development and Accepting Content Default Settings.

Nowadays, SystemC is an embedded system modeling language that has a lot of features and can be used to develop prototypes of embedded system. It is rich by its data types library and compilation environments of the C ++ language. It adds primitives to be able to write parallel processes, signals, clocks, as well as some concepts of a component language. SysVerPml has been designed to support SystemC plug-in. SystemC plug-in has been utilized to create a SystemC project based on C/C++ Development Toolkit (CDT) plug-ins in Eclipse Platform, SystemC, Cygwin packages required for building GCC compiler, and Managed Build System (MBS) pre-defines many useful macros and allows tool integrators and users to define additional macros. The CDT plug-ins supports a C/C++ Editor, Debugger, Launcher, Parser, Search Engine, Content Assist Provider and a Makefile generator [15]. To do the installation we followed the steps described at the guide for getting started with SystemC development, it contains a chapter for setup of Eclipse together with Cygwin and SystemC [16].

IP-XACT is another standard enabling the assembly of IP components (Intellectual Property blocks); it describes especially the interconnection interfaces, some communication components and associated protocols, using an Architecture Description Language (ADL). The ADL makes possible to define the interfaces of certain types of bus and protocols. The IP-XACT format respects the syntax construction rules specified in XML's Abstract Syntax Tree (AST). IP-XACT has been designed to address all these issues by providing a standardized data exchange format which has both the flexibility to represent SystemC models and the rigor to allow information to be automatically extracted and used in flow automation and advanced verification by Spin using Promela language.

In order to realize the transformation between SystemC and IP-XACT, we use Eclipse IP-XACT plug-in [17] as a means to import the IP component descriptions from the first model ScModel which provides database along with methods and structural information such as variables, functions, events, ports, processes, constructors and module instances, and from the second model PtrModel which include assertions with re-usable properties and the system declaration. So we realize the stream described in our previous article [10] and we pass the structural model conform to the SystemC behavioral model as a call parameter to retrieve a complete model as main file output. In this file, we create an instance of the embedded component with their attributes and the parameter configurations. Component properties are established by port-signal bindings.

IP-XACT is successful at ensuring syntactic formats compatibility and the interpretation's uniqueness of their descriptions to make component interoperability if needed, but it is not simulatable and it has neglected the behavioral aspects and components properties verification. Further, the purpose of this SystemC main file is to enable a simulation for the IP-XACT model. In a previous work, we described [9] that our translation to SystemC can also be seen as a translation into a set of automata. Each process and each function is translated into one produced automaton by composing produced SystemC models without any change. The SysVerPml framework enables to check safety properties for each SystemC program of the product line once at design time, without the need for additional time to redo the verification process every time programs are involved in the creation of new system prototypes as explained in our work [18]. After the simulation and gathering of results, a Promela file is generated. In this file, specifications can be given in Linear Temporal Logic (LTL) formulas;

The plugin consists of two main components, a compiler which compiles Promela code, and an interpreter. We used the graphical front-end JSPIN. The JSPIN tool executes SPIN commands in the background in response to user actions. It provides a clear overview of the many options in SPIN that are available for performing animations and verifications. JSPIN was built using the Java SWING library and consists of three adjustable panes, displaying text. The left one displays the Promela source files, the lower one messages from SPIN and JSPIN and the right one is used to display the output of printf statements and of data from animations [19].

As we shall see in the article JSPIN tool will attempt to do automatically verification limiting human intervention and returning one of three results; whether it be a state where properties are satisfied, or properties are not satisfied so a counterexample will be given, or Indeterminate if the state space is such that the tool cannot compute a result in a reasonable amount of time [20].

In order to demonstrate the importance of the SysVerPml framework the case studies of some embedded components have been published in preceding articles; the verification results of FIFO component have been published in [21] and the verification results of Bus AMBA AHB have been published in [10]. In this dissertation we provide an application example related to memory SRAM (static RAM), this component have two views following the model described in Fig. 2.

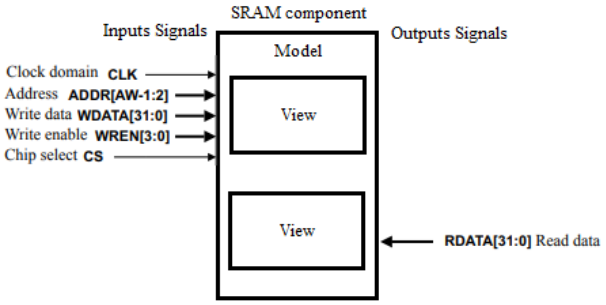

Fig. 2. The Characteristics of the SRAM Model.

We report the IP-XACT description introduced in the previous part of this article. We use the namespace ipxact, below in Fig. 3 we show the output view of the SRAM model, in which port is denoted with RDATA.

The component definition <ipxact:component> contains information to Promela file and the SystemC model about the component. This information is situated within a <ipxact:parameter> element, identified with the <ipxact:value> and <ipxact:name> tags.

Each component interface that uses SysVerPml mapping is defined in the generated file as: Inputs, outputs, the combination of inputs and outputs and the parameters.

We can combine inputs and outputs in a single component interface definition, but we havent't possibility to combine parameters and inputs/Outputs because these elements are defined in the pair name-value of <ipxact:parameter> which indicates to the SysVerPml generator that there is a SystemC mapping.

```
<ipxact:name> sram </ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:model>
  <ipxact:views>
    <ipxact:view>
      <ipxact:name>interface</ipxact:name>
      <ipxact:componentInstantiationRef>output-interface</ipxact:componentInstantiationRef>
    </ipxact:view>
  </ipxact:views>
<ipxact:instantiations>
  <ipxact:componentInstantiation>
    <ipxact:name>output-interface </ipxact:name>
    <ipxact:language> systemc </ipxact:language>
    <ipxact:libraryName>outputlib</ipxact:libraryName>
    <ipxact:moduleName>sramoutput</ipxact:moduleName>
    <ipxact:moduleParameters>
      <ipxact:moduleParameter parameterId="my_param" resolve="user" type="longint">
        <ipxact:name>my_param</ipxact:name>
        <ipxact:value>0</ipxact:value>
      </ipxact:moduleParameter>
    </ipxact:moduleParameters>
    <ipxact:fileSetRef>
      <ipxact:localName>fs-interface</ipxact:localName>
    </ipxact:fileSetRef>
  </ipxact:componentInstantiation>
</ipxact:instantiations>
<ipxact:ports>
  <ipxact:port>
    <ipxact:name>rdata </ipxact:name>
    <ipxact:wire>
      <ipxact:direction>out</ipxact:direction>
    </ipxact:wire>
  </ipxact:port>
```

Fig. 3. The IPXACT Document Tree.

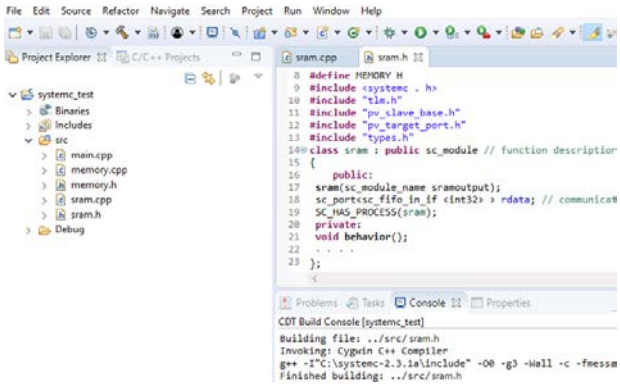

Fig. 4. The SystemC Interface Capture.

We have the possibility to import the generated IP-XACT file for use and update if needed by the use of the SysVerPml generator; the header file describes the design of Memory and it is entirely integrated into the SystemC model illustrating how information contained in IPXACT file can be used for a behavioral implementation as we observe in Fig. 4.

Furthermore, we can use easily the interface of JSPIN tool; downloaded from the Github link [22]; a tool that track bugs into the encoding programs and it can verify whether a specification is satisfied or make a counterexample of symbolic formulas. By the way, it makes possible to edit as well as to update the LTL formulas written inside of Promela model in respect of semantic transformation from SystemC model. JSPIN tool is an elementary part of our SysVerPml platform and it makes possible to run simulation and formal verification directly. We note well that JSPIN's main focus is the SpinSpider component. SpinSpider allows us to demonstrate the properties in case of concurrent processes.

The generated file in the PtrModel module is represented by the structured classes. These classes gave us the advantage to efficiently represent the semantic results and allow us to represent both the ports and the properties of the component.

III. Case Study

This section discusses the use of our approach to verify some properties of the SRAM design used in interaction with a CPU model which contains working microengines - a set of threads in each microengine – and all of them want access to SRAM component.

We have developed the translator, which takes the SystemC design as an input and generates the Promela encoding with the integration of properties as explained in the previous section. The translator uses IPXACT to extract from the SystemC design description that is useful for performing the transformation to Promela language.

Remember that the verification of the resulting Promela models from the SystemC models provided by JSPIN tool to completely verifying SRAM component.

To make length of this paper brief we express with LTL the most functional properties, such as non-starvation, safety and deadlock.

The non-starvation property for the events that are related to SRAM controller of a CPU means that if an SRAM access request comes from a thread 0 of a microengine 0 for example is enqueued, it is eventually committed in the next 400 SRAM occurrences. This property can be formulized with the LTL formula as shown in (1) in this way:

AG (microengine0_thread0_sram_enqueued ⇒XF [1:400] (microengine 0_thread 0_sram_done)) (1)

The safety property of the memory access is stored in a scheduling FIFO to handle the occurred order of the events sram_enqueued (the SRAM access request is enqueued), sram_dequeued (the SRAM access request is dequeued) and sram_done (the SRAM access request is committed), which makes necessary that always after an SRAM request by a thread 1 of a microengine 1 for example, it cannot be done before it is dequeued. As shown in (2) this property can be expressed with the LTL formula like this:

AG (microengine1_thread1_sram_enqueued ¬ microengine1_thread1_sram_done U microengine1_thread1_sram_ dequeued) (2)

The deadlock property to prevent problems with shared resource, for each SRAM access on CPU, the data readout and the memory address referenced must be similar, and always all the SRAM references represented by addr are made in execution with the same order. As shown in (3) the LTL formula can be expressed with the following:

AG (addr(sram_enqueued[i]) = addr(sram_enqueued_CPU[i]) ^ data(sram_done[i]) = data(sram_done_CPU[i])) (3)

We assume that the SRAM access request is put into a scheduling FIFO by a thread 1 of a microengine 1 for example and then eventually committed; always the memory address should be the same as shown in (4).

AG (addr(microengine1_thread1_sram_enqueued[i]) = addr(microengine 1_thread1_sram_done[i])) (4)

Table I lists the average values of performance metrics using by SPIN verification process. The average values were computed over the set of pre-defined specification properties to check without errors the functional properties of SRAM component.

TABLE I. VERIFICATION DATA

LTL formulas	**SPIN Metrics**			
	States generated	***Transitions number***	***Memory used***	***Verification time***
1	6700	$3.0*10^5$	1KB	100s
2	5739	$7.0*10^6$	50Bytes	24s
3	10267	$3*10^5$	40KB	6s
4	5710	$7.0*10^6$	12Bytes	60s

The units used in this table are; B= Bytes, s = second.

IV. CONCLUSION

In this paper, we have reported our effort to implement SysVerPml platform and the impressive component's modeling and checking gain obtained by transforming SystemC models to Promela encodings. This remarkable gain is achieved by modules which decomposes the implementation of our tool and make it modular. This modularity facilitates modifications inside of IPXACT description and LTL properties. We have provided an application example related to a sessions that implements the SRAM component.

REFERENCES

[1] H. D. Foster, "Trends in Functional Verification: A (2016) Industry Study", whitepaper, Mentor Graphics.

[2] Edmund M. Clarke and Jeannette M. Wing. "Formal Methods: State of the Art and Future". In ACM Computing Survey, volume 28–4, pages 626–643, (December 1996).

[3] Carl-Johan Seger. "An Introduction to Formal Verification". Technical Report 92–1, Department of Computer Science, University of British Columbia, Canada, (June 1992).

[4] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. "Embedded System Design: Modeling, Synthesis and Verification". Springer, (2009).

[5] E. M. Clarke and E. A. Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In Logic of Programs, Workshop, pages 52–71, London, UK, (1981). Springer-Verlag.

[6] M. J. C. Gordon and T. F. Melham, "Introduction to HOL: A Theorem Proving Environment for Higher Order Logic", Cambridge University Press, (1993).

[7] Orna Lichtenstein and Amir Pnueli. "Checking that finite state concurrent programs satisfy their linear specification". In Proceedings of the 12th ACM SIGACT-SIGPLAN POPL'85, pages 97–107, New York, NY, USA, (1985). ACM.

[8] Nina Amla, Robert Kurshan, Kenneth L. McMillan, and Ricardo Medel, "Experimental Analysis of Different Techniques for Bounded Model Checking", Springer-Verlag Berlin Heidelberg (2003).

[9] A. Ismail, E. J. Lamia, Z. Abdelouahed, and N. Tarik, "The behavior, interaction and priority framework applied to systemc-based embedded systems", in 13th IEEE/ACS International Conference of Computer Systems and Applications, AICCSA 2016, Agadir, Morocco, (November 29- December 2, 2016).

[10] Ismail Assayad, Lamia Eljadiri, "A platform for systematic verification of embedded components in IP-XACT, SystemC and Promela", In ICSDE 2018, Rabat, Morocco, (October 18-19, 2018).

[11] "Eclipse Platform Technical Overview". Technical Report, Object Technology International (OTI) Inc… (2001). http://www.eclipse.org /whitepapers/eclipseoverview.pdf

[12] Holzner Steve, "Eclipse", O'Reilly, (April 2004).

[13] Holzner Steve, "Eclipse Cookbook", O'Reilly , (June 2004).

[14] Shavor Sherry, D'Anjou Jim, Fairbrother Scott, Kehn Dan, Kellerman John, McCarthy Pat, "The Java Developer's Guide to Eclipse",Addison-Wesley, (2003).

[15] "Managed Build Extensibility Reference Document (for CDT2.1)", http://www.eclipse.org/cdt/

[16] "Guide for getting started with SystemC development", by senior consultant Kim Bjerge, Danish technological institute (2007).

[17] http://www.eclipse.org/dsdp/dd/ipxact/gettingstarted/QuickStart.html

[18] Lamia Eljadiri, Ismail Assayad, and Abdelouahed Zakari, "Generic Verification of Safety Properties For SystemC Programs Using Incomplete Interactions", In ICSDE 2018, Rabat, Morocco, (October 18-19, 2018).

[19] BEN-ARI, Mordechai, "Principles of the Spin Model Checker". Springer, (2008). – ISBN 978–1–84628–769–5

[20] Robert C. Armstrong, Ratish J. Punnoose, Matthew H. Wong, Jackson R. Mayo, “Survey of Existing Tools for Formal Verification”, Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, (December 2014).

[21] Ismail Assayad, Lamia Eljadiri, Abdelouahed Zakari, “Systematic Verification of Embedded Components with Reusable Properties”. In WINCOM 2017, Rabat, Morocco, (November 01-04, 2017).

[22] https://github.com/motib.