Generic Framework For Verifying Embedded Components

Lamia Eljadiri, Ismail Assayad

Abstract: Formal verification has become very useful and popular in last decade in area of embedded systems design and in analysis of critical systems. It can reveal common errors like deadlocks, starvation, check system invariants, but also verify more complex properties defined by LTL formulas whose writing may be very error prone for non expert users. To reduce the time-to-market for embedded systems and assist designers in the complexity of verification step at design time, we advocate the predevelopment of reusable behavioral properties for each family of embedded components to be verified. The proposed approach is to predefine the reusable properties by specifying them as logics on standard input/output signals and standard data values. Obtaining this reusable form enables them to be used for every new component in the product line, hence without the need to spend additional time to redo the verification setup every time a new component is used to create a new design. We have successfully predefined LTL reusable properties for widely used industrial embedded components families such as FIFOs and BUSes, and have performed generic verification using SPIN tool. This paper 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. Until now, there have been few research papers directed towards converting SystemC models to Promela language.

Index Terms: Algorithms, Automation, Embedded Components, Embedded Systems, Formal verification, Framework, LTL Properties, Promela, SystemC, SysVerPml, System design.

1. 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 a powerful technique because it is based on mathematical proofs to describe the absence or existence of errors, most of them demonstrated by the counterexample method. The formal verification it 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].

- *Lamia ELJADIRI is currently pursuing PhD degree program in LIMSAD laboratory, Faculty of Sciences, University Hassan II of Casablanca, Morocco, E-mail: lamia.eljadiri@taalim.ma*
- *Ismail Assayad is currently Doctor in LIMSAD laboratory, Faculty of Sciences, ENSEM, University Hassan II of Casablanca, Morocco, E-mail: iassayad@gmail.com*

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]. The SystemC language is a defacto-standard for embedded systems design and became the basic language of most of industrial production companies [9]. This allows research works to focus on checking systems specifications against programmed SystemC models which are first translated to equivalent formal model such as timed automata models and then checked using verification tools such as SPIN [10], SMV [11] and BIP [12]. SystemC simulation suffers the drawback of being incomplete. SystemC-level formal verification of implementations with respect to the rigorous specification stated in this paper brings designers another tool which can help them to purify the code and to ensure its correctness. Furthermore, unlike simulations such a process of verification can also lead to backward refinement of component specification through provided counter-examples debug, which is contributive to future reuse of the design under verification. There is another different way of using formal verification in system design which consists in the creation of an abstract model of the design instead of verifying SystemC programs. The problem of this approach which is out of the scope of this paper is to ensure that the abstract model is sound. Moreover using the SystemC-level way of verification advocated in this paper has the advantage of being accurate and thus permits to specify reusable full-specification based properties. This simple reusability is not always possible with the abstract-level way of verification. We successfully use the state-of-the-art automatic formal verification methods. More particularly, we use the model checking technique

[13][14][15], which takes into account all the possible behavior of the design. The essential prerequisite of the model checking technique is formal specification of verified properties. For this purpose, we use suitable temporal logics [16][17][18][19]. Strong expressiveness of these logics allows us to express all the typical requirements on the component behavior. Unfortunately, a nontrivial effort is required to specify these formulas and to use model checking tools to verify them. A lot of expert work has to be done manually to suitably express sophisticated behavioral properties to be verified. For this reason, model checking is not yet the standard method of verification. This paper is an extended version of our previous articles [37] and [38], it contains more details about our research work and its structure is the following. We first state the motivations, contributions and related works respectively. Second, we present the verification environment supported by our approach describing the different plug-ins used by the framework. Second, we state a few assumptions that define the subset of SystemC and the Promela automaton supported by our approach. Then, we present our general approach for the representation of SystemC in Promela language to making easier verification of generated properties. Then, we present the complete transformation procedure and an illustration of the verification flow that allows model checking of SystemC designs. Finally, 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.

2 MOTIVATION AND CONTRIBUTIONS

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 [20] 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 intersystem 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 [21], which has the advantage of permitting compositional verification of programs by keeping their incomplete interactions. To assisting the Model Checker and speeding up the process of verification the predefinition of properties to verify is an essential step. Our idea is based on these points;

1. To collect a predefined properties for components having the same interfaces. For example: properties for type components FIFO; BUS type components, ...
2. To create process automation especially a framework to automate the verification process from the SystemC program of a component to Promela models.

To overcome the verification issues previously explained, we propose in this paper the use of SystemC-level way of verification combined with predefined reusable specification properties. We specify the properties of verified components using the Promela language. More particularly, the properties are encoded as formula of LTL, Linear Temporal Logic [22] in our case, or CTL, Computation Tree Logic [23] for other tools. They are taken mostly from the standard specification of Input/Output component behavior. This article gives an overview of our Model Checker Platform named SysVerPml; the tool supports model checking of embedded systems components, it allows creating an abstract model of the design instead of translating SystemC programs to formal models on Promela language with integration of LTL generated properties extracting from state machines, and checking them using verification tool SPIN (Simple Promela INterpreter). In our SysVerPml system, we combined in a new prototyping set, data derived from observing the behavior of embedded systems whose operation is described by the programming language; On the other hand, to have a functional view of the system the properties to be verified are expressed by time formulas. These formulas are specified using system variables and discrete time operators describing the change of the possible values of the variables during the execution of this system. This new combination represents the originality of our work. It allows to analyze the behavior of an embedded system, according to different types of communication scenarios and model them in SystemC then to make an efficient conversion to Promela which will allow a verification of the generic or personalized properties chosen by the user and this gives us a structural view of the system that represents in detail the structure of the components and their interconnections, ignoring the details of calculation and communication, thereafter our verification tool produces a true result if the property is checked on the system, otherwise the tool provides a counter example which is the sequence of states that invalidates the property, which is the strength of this technique.

The advantages of our proposed approach are the following:

1. SystemC-level way of verification presented in this paper has the advantage of being very accurate because it is code-level, and will allow us to specify reusable properties which are reused whenever a new version of a component is developed.
2. Genericity scope covers new versions of components but also similar components, i.e., components implementing same standard specifications. The reason behind this is that the reusable properties are mostly defined using common behavior specifications implying standard component input/output signals and data values.

3 RELATED WORK

Many related works for SystemC-level verification exist in the literature [24][25]. In this paper the method we use for the conversion from SystemC to the target formal model is the one presented in [20]. The reason why we made this choice is that unlike the ones in existing related works the latter has the advantage of being composable and thus we can guarantee that component-level verifications are conservative with regards system-level verifications of safety properties. A lot of works are also done in the field of abstract-level way of verification [15][26][27]; however this approach is not suitable for the systematic use of predefined properties, because some strong abstracted component models may be too different from standard models and may even alter input/output interfaces. In addition to this loss of simple reusability,

abstracted models raise many issues in modeling soundness and verification accuracy. Finally, with regards the reusability of the predefined component properties for new versions and similar standard compatible components, this work is, at our best knowledge, the first reported one on the promotion and use of reusable properties in SystemC-level way of verification for embedded systems components.

4 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 [28], [29], [30], [31]. This IDE mainly allows providing an extensible platform for building software. 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). 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) predefines 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 [32]. 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 [33]. IPXACT 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 IPXACT format respects the syntax construction rules specified in XML's Abstract Syntax Tree (AST). IPXACT 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 IPXACT, we use Eclipse IPXACT plug-in 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 [21] 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. IPXACT 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 IPXACT model. In a previous work, we described [20] 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 [34]. 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 plug-in 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 [35]. 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 [36]. In our SysVerPml system, we combined in a new prototyping set, data derived from observing the behavior of embedded systems whose operation is described by the programming language; On the other hand, to have a functional view of the system the properties to be verified are expressed by time formulas. These formulas are specified

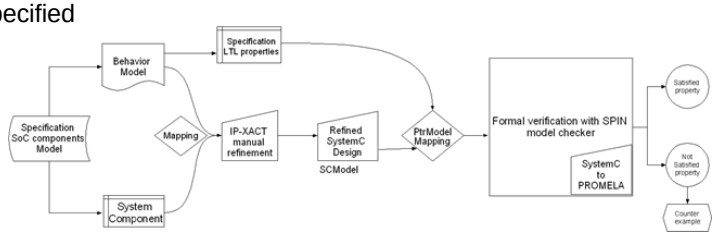

***Fig. 1**. Architecture of the model checker.*

using system variables and discrete time operators describing the change of the possible values of the variables during the execution of this system. As we observe in figure Fig. 1 our first component takes a set of SoC components as input and yields a corresponding the Abstract Syntax Tree (AST) in XML for each component of them including their description and

properties. The second component takes an IPXACT file as input and yields an association between ports and variables defining properties. We used the IP XACT which parses a given Abstract Syntax Tree (AST) in XML and generates SystemC design. The third one takes The AST in XML as input and generates a Promela model that is also in XML format and that can be used as input for the SPIN tool.

5 VERIFICATION PROCESS OVERVIEW

5.1 Input Component In SystemC

SystemC is a widely used C ++ library for the modeling of embedded systems and multimedia applications. It includes low-level descriptions of hardware such as RTL; and high level of SystemC-TLM descriptions which are functional and timed abstractions of hardware which require dedicated extension to the basic SystemC library. SystemC-TLM is a new level of description which is not present in other hardware description languages like Verilog that only support bit and signal types but no functional transaction data types. SystemC-TLM is however out of the scope of this paper because we only consider the SystemC-level way of verification for embedded components for the reasons already explained in a previous section. Considered SystemC-level descriptions (RTL, CABA, CA, BA) of components are highly accurate and at least faithful to the components interfaces and behavior specifications and thus fulfills the requirement to benefit from the simple reusability of predefined verification properties.

The main constructs of the SystemC language are:

- Modules are the fundamental building block in a SystemC program. Modules support multiple processes inside them. Modules can also be used for describing hierarchy: a module can contain sub-modules, which allows to break complex systems into smaller more manageable pieces.

Modules and processes can have a functional interface, which allows hiding implementation details of IP blocks.

1. Processes are used to describe functionality. SystemC provides three different processes to be used by hardware and software designers: methods (asynchronous blocks), threads (asynchronous processes) and clocked threads (synchronous processes).
2. Ports of a module are the external interface passing information to and from a module, and triggering actions within the module. Ports can be single-direction or bidirectional.
3. Signals create connections between module ports allowing modules to communicate. SystemC supports resolved and unresolved signals. Resolved signals can have more than one driver while unresolved signals can only have a single driver.

5.2 Verification Back-End

We use the SPIN tool and its input language Promela as the verification process back-end. To dot that SystemC program of the current component under verification is converted to an equivalent automata formalism based model which is in turn simply translated to a readable textual Promela description syntax. Finally SPIN model checker will be in charge of checking properties satisfaction through state space exploration. Properties are predefined based on the component standard Specification using formulas describing specified behaviors and expressed in temporal logic. Then the user has to map variables used in these formulas to actual names used in the program before launching the verification with SPIN.

5.3 Verification Properties In LTL

LTL is a temporal logic where a formula is composed of atomic propositions, logical operators, and temporal operators. Currently we do not have a GUI interface for visualization purposes. Nevertheless, properties associated to input components are inserted as LTL formula in the Promela file and may be easily identified by the designer at the top of the file by seeking the following Promela syntax:

```
Ltl    <Identifier1>     {<Component1_Formula1>}
Ltl    <Identifier2>     {<Component1_Formula2>}
...
Ltl    <IdentifierN>     {<Component1_FormulaN>}
```

As many LTL properties as needed are inserted for the current component in the Promela file corresponding to the description of that component. Then SPIN will check them all, one by one. In order to do that, translations are needed once again. Each formula is translated to generate a never claim, i.e. a Promela automaton representing the negation of the formula so that SPIN will seek to find a counter-example for this negation. At last, if a counter-example is found, the designer may use it to debug and refine the current component.

6 CASE STUDY

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 [37] and the verification results of Bus AMBA AHB have been published in [21]. In this dissertation we provide an application example related to memory SRAM (static RAM), this component have two views following the model described in the figure Fig. 2.

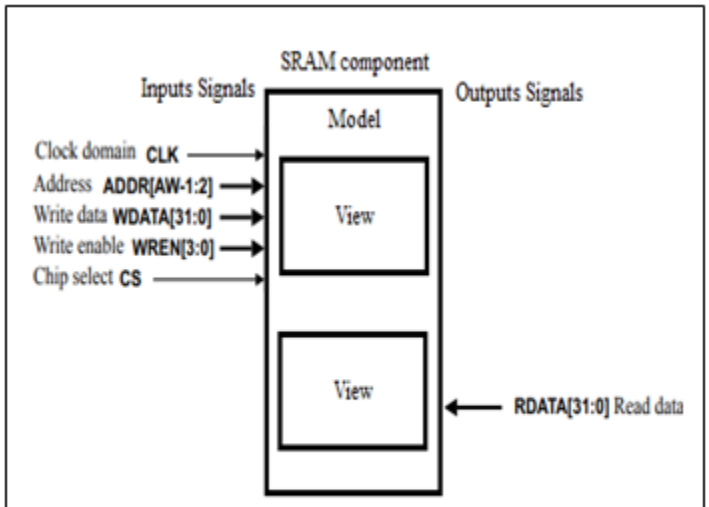

Fig. 2. The characteristics of theSRAM model

We report the IPXACT description introduced in the previous

part of this article. We use the namespace ipxact, bellow in figure 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>output lib</ipxact:libraryName>
    <ipxact:moduleName>sram output</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.*

We have the possibility to import the generated IPXACT 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. Furthermore, we can use easily the interface of JSPIN tool; downloaded from from the Github link [39]; 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.

6 TESTS OF THE PERFORMANCE VERIFICATION

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 componentTo 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 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. 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. 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.

AG (addr(microengine1_thread1_sram_enqueued[i]) =

addr(microengine 1_thread1_sram_done[i])) (4)

TABLE 1
VERIFICATION DATA

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

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

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

7 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] T. Gro¨tker, S. Liao, G. Martin, and S. Swan, System Design with SystemC. Kluwer Academic Pub., Hingham, MA, 2002.

[10] G. Holzmann, Spin Model Checker, the: Primer and Reference Manual, 1st ed. Addison-Wesley Professional, 2003.

[11] K. L. McMillan, Symbolic model checking. Kluwer, 1993.

[12] A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in bip," in SEFM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 3–12.

[13] J. Queille and J. Sifakis, "Iterative methods for the analysis of petri nets," in Application and Theory of Petri Nets, Selected Papers from the First and the Second European Workshop on Application and Theory of Petri Nets, Stasbourg 23.-26. September 1980, Bad Honnef 28.-30. September 1981, 1981, pp. 161–167. [Online]. Available: https://doi.org/10.1007/978-3-642-68353-4 27

[14] O. Lichtenstein and A. Pnueli, "Checking that finite state concurrent programs satisfy their linear specification," in Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ser. POPL '85. New York, NY, USA: ACM, 1985, pp. 97–107. [Online]. Available: http://doi.acm.org/10.1145/318593.318622

[15] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, Model Checking. Cambridge, MA, USA: MIT Press, 1999.

[16] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," ACM Trans. Program. Lang. Syst., vol. 8, no. 2, pp. 244–263, 1986. [Online]. Available: http://doi.acm.org/10.1145/5397.5399

[17] Z. Manna and A. Pnueli, The Temporal Logic of Reactive and Concur- rent Systems. New York, NY, USA: Springer-Verlag New York, Inc., 1992.

[18] R. Alur and T. A. Henzinger, Logics and models of real time: A survey. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 74–106. [Online]. Available: http://dx.doi.org/10.1007/BFb0031988

[19] S. Almagor, U. Boker, and O. Kupferman, Discounting in LTL. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 424–439. [Online].
Available: http://dx.doi.org/10.1007/978-3-642-54862-8 37

[20] 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).

[21] 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).

[22] Z. Manna and A. Pnueli, Temporal Verification of Reactive Systems: Safety. Springer-Verlag, 1995.

[23] M. Ben-Ari, A. Pnueli, and Z. Manna, "The temporal logic of branching time," Acta Informatica, vol. 20, no. 3, pp. 207–226, 1983. [Online]. Available: http://dx.doi.org/10.1007/BF01257083

[24] P. Herber, J. Fellmuth, and S. Glesner, "Model checking systemc designs using timed automata," in Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis, ser. CODES+ISSS '08. New York, NY, USA: ACM, 2008, pp. 131–136.

[25] I. Radojevic, Z. Salcic, and P. Roop, "Modelling heterogeneous embed- ded systems: from Esterel and SystemC to DFCharts," IEEE Design & Test of Computers, vol. 23, no. 5, pp. 348–357, May 2006.

[26] S. Campos, E. Clarke, W. Marrero, and M. Minea, “Verifying the performance of the pci local bus using symbolic techniques,” in Com- puter Design: VLSI in Computers and Processors, 1995. ICCD ’95. Proceedings., 1995 IEEE International Conference on, Oct 1995, pp. 72–78.
[27] D. Groe, H. M. Le, and R. Drechsler, “Proving transaction and system- level properties of untimed systemc tlm designs,” in Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010), July 2010, pp. 113–122.
[28] “Eclipse Platform Technical Overview”. Technical Report, Object Technology International (OTI) Inc… (2001). http://www.eclipse.org/whitepapers/eclipseoverview.pdf
[29] Holzner Steve, “Eclipse”, O’Reilly, (April 2004).
[30] Holzner Steve, “Eclipse Cookbook”, O’Reilly , (June 2004).
[31] Shavor Sherry, D’Anjou Jim, Fairbrother Scott, Kehn Dan, Kellerman John, McCarthy Pat, “The Java Developer’s Guide to Eclipse”,Addison-Wesley, (2003).
[32] “Managed Build Extensibility Reference Document (for CDT2.1)”, http://www.eclipse.org/cdt/
[33] “Guide for getting started with SystemC development”, by senior consultant Kim Bjerge, Danish technological institute (2007).
[34] 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).
[35] BEN-ARI, Mordechai, “Principles of the Spin Model Checker”. Springer, (2008). – ISBN 978–1–84628–769–5
[36] 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).
[37] Ismail Assayad, Lamia Eljadiri, Abdelouahed Zakari, “Systematic Verification of Embedded Components with Reusable Properties”. In WINCOM 2017, Rabat, Morocco, (November 01-04, 2017).
[38] Lamia Eljadiri, Ismail Assayad, “Generic Framework Architecture for Verifying Embedded Components”, IJACSA journal Volume 11 Issue 6 June 2020.
[39] https://github.com/motib