An Effective Approach for Model Checking SystemC Designs
(Tool Paper)

Razieh Behjati, Hamideh Sabouri, Niloofar Razavi
ECE Dept., University of Tehran, Tehran, Iran
{r.behjati, sabouri, n.razavi}@ece.ut.ac.ir

Marjan Sirjani
ECE Dept., University of Tehran, Tehran, Iran
School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics (IPM)
Niavaran Square, Tehran, Iran
msirjani@ut.ac.ir

Abstract

SystemC is a system level modeling language with the goal of enabling verification at higher levels of abstraction. In this paper, we propose a mapping from SystemC designs to Rebeca models supported by an automatic tool, Sytra. Rebeca verification tool set is then available for verifying LTL and CTL properties. The mapping is aimed to preserve the concurrent and event-driven nature of SystemC. This work is part of a project (Sysfier) to formally verify SystemC designs. The applicability of our approach is shown by a set of small and medium sized case studies, and the scalability of the approach is shown by the verification of a single-cycle MIPS design.

Keywords: System Level Formal Verification, SystemC Verification, Actor Model, Model Checking, Rebeca.

1 Introduction

SystemC [16] is an object-oriented system level language for embedded systems design and verification. It has emerged lately as the leading language for system-level models [23].

While a major goal of SystemC is to enable verification at a higher level of abstraction, the focus so far has been on simulation techniques. The performance of these techniques highly depends on the test bench coverage. Test bench generation is a time consuming task and is not reliable in validation of complex systems. To overcome this problem the tendency is now directing to the development of formal techniques to augment standard SystemC verification [7, 15, 19, 22]. Model checking [6] is a formal verification technique with the purpose of verifying specifications given in temporal logic formulas for concurrent systems.

The work presented in this paper is part of a project, named Sysfier. The goal of the Sysfier is to formalize SystemC semantics and provide an integrated environment for modeling and verification of SystemC designs. The components of this integrated environment, which is called Afra, are shown in Figure 1 (Section 3 briefly describes this tool). We formalized the semantics of SystemC in the context of Rebeca semantics. Rebeca (Reactive Objects Language) is an object-based, event-driven, concurrent modeling language based on the actor model and is supported by a set of model checking tools [12, 21]. The object-based nature of Rebeca which fits the object-oriented nature of SystemC, as well as the event-driven nature of both language makes Rebeca a suitable choice. Using the formalism, Sytra is developed to automatically generate Rebeca models from SystemC models. Rebeca models are then verified by the Rebeca model checking tool set, including Modere (Model-checking Engine of Rebeca) [12] and SyMon (SystemC Model-checking Engine) [3]. By verification of Rebeca models, the correctness of SystemC designs are verified against the properties given in temporal logic formulas, either Linear Temporal Logic (LTL) or Computational Tree Logic (CTL) [8, 17]. Modere is a direct model checker for Rebeca while SyMon is customized for model checking of Rebeca models based on the semantics of SystemC simulation kernel. These tools can also apply some automatic reductions on the model [13, 21] in order to diminish the state space explosion problem, which is one of the most challenging problems in the field of model checking. The contributions of the work are as follows:

• Proposing a formalism for SystemC using an actor-
Based modeling language as the intermediate language which establishes a basis for using and developing formal verification techniques and tools.

- Considering concurrent and event-driven features of SystemC as well as the notion of delta-time delays which allows to support SystemC designs in different and mixed levels of abstraction.
- Customizing and extending the intermediate language and the supporting tools slightly to generate simpler models and more efficient model checkers.
- Developing a new model checking engine based on the SystemC simulation kernel behavior, and incorporating specialized reduction techniques in the engine.
- Providing an integrated environment for modeling and verifying SystemC designs using Eclipse platform.

The focus of this paper is on modeling modularity and concurrency of SystemC using Rebeca and its formal semantics and also introducing the automatic tool, Sytra.

Outline of the paper: In the next section, SystemC and Rebeca are briefly introduced. Next we shortly introduce our project, Sysfier. The details of Sytra and the mapping including the discussion about its validity is presented in Section 4. We show the verification results for a set of case studies including a single-cycle MIPS in Section 5. Section 6 presents an overview of the related works. Finally we conclude the work in Section 7 and we also explain some directions for future works.

2 Preliminaries

2.1 SystemC

SystemC [16] is a system-level modeling language based on C++. The main motivation of developing SystemC is that a hardware model can be compiled using a regular C++ compiler, and then be simulated efficiently.

Modules are the basic building blocks of SystemC. The behavior of a module is specified by defining one or more processes. Processes are declared as special functions of modules and can be reactive to any input signal or to an event. A process should be either declared as a method or as a thread. The difference is that blocking statements are not allowed in method processes, making the whole body of the method to be executed atomically. Processes and modules can communicate via ports, signals, events and variables.

The simulation kernel of SystemC invokes the concept of delta-time delays to model the concurrent execution of the hardware systems. So, if processes that are running concurrently change the values of some signals, the values of the signals shall be updated simultaneously after a delta-time delay.

2.2 Rebeca

Rebeca is an object-based, event-driven, concurrent language based on the actor-model [5]. A Rebeca model consists of a set of rebecs (reactive objects) which are concurrently executed [20]. Rebecs are encapsulated active objects (object-based), with no shared variables. Each rebec is instantiated from a Reactive-Class and has a single thread of execution which is triggered by reading messages from an unbounded message queue (event-driven). Rebecs communicate by asynchronous (non-blocking) message passing. The messages which are sent by the sender rebec are put in the message queue of the receiver rebec. The receiver takes a message from top of its message queue and executes the corresponding message server atomically. Concurrency is modeled by interleaved execution of rebecs.

To be used as an intermediate language for SystemC verification, we have introduced a new version of Rebeca, called RebecaSys, which is equipped with three additional features. Global variables are added to support variable sharing, Wait statements are added to support synchronization and a new primitive data-type is added to simplify the translation of SystemC widely used data-types.
The goal of Sysfier is to develop an integrated environment for modeling and verifying SystemC designs by formalizing SystemC semantics and providing model checking tools. A tool, named Afra [1] (Figure 1), is developed to integrate Sytra, Modere and SyMon in addition to Rebeca and SystemC modeling environments. Afra gets SystemC models and LTL or CTL properties from the designer, and verifies the models. If a property is not satisfied, a counterexample is displayed. For verifying SystemC models, Afra translates SystemC codes to Rebeca using Sytra. It then utilizes the Rebeca verification tool set to verify the given properties. Where applicable, reduction techniques are used to tackle the state explosion problem.

Here, we briefly describe the main components of Afra.

- Sytra generates Rebeca models from SystemC models based on the proposed formalism. KaSCPar [2] is used as C++/SystemC parser.
- Modere [12] is a direct model checker of Rebeca. An important advantage of Modere over the existing verifiers is that since it is designed specifically for Rebeca, many abstraction and reduction techniques can be applied according to the computational model of Rebeca.
- SyMon [3] is a verification engine customized for verification of Rebeca models obtained from SystemC codes. The SystemC simulation kernel executes the processes one by one, with a non-preemptive scheduling policy, according to a pre-specified order. SyMon uses this order instead of executing all the possible interleaveings of processes and hence gains a significant amount of reduction in the size of the generated state space.

4 Sytra: Mapping and Validation

In this section we explain the details of the proposed mapping for each group of SystemC constructs as well as the details of modeling SystemC simulation kernel then at the end we discuss the validation of our mapping.

4.1 Key Idea

Here, we propose a mapping from SystemC to Rebeca which enables SystemC designs to be formally model checked. The mapping is well structured and natural and can be used for model checking complex designs which may include several inter-connected modules.

In this mapping, modules as principal structural building blocks of SystemC are translated to Reactive-Classes, which have the same role in Rebeca. Each process of a module is mapped to a message server in the corresponding Reactive-Class. In this way non-preemptive execution of processes is guaranteed. To provide the concurrency, whenever a module is instantiated, for each of its processes a rebec is instantiated. Each of these rebecs is responsible for the execution of one of the message servers.

There is a special Reactive-Class, Synchronizer, which models the behavior and functionality of SystemC simulation kernel. Only one rebec is instantiated from this class in each Rebeca model. The advantage of this approach is that the functionality of the SystemC simulation kernel is centralized in a separate Reactive-Class, relaxing the other Reactive-Classes from concerning the details of kernel functionality. Table 1 shows a brief overview of the proposed mapping.

In the following we will show in detail how our mapping to Rebeca deals with several SystemC constructs. We also explain the functionality of the Synchronizer rebec.

<table>
<thead>
<tr>
<th>SystemC construct</th>
<th>Rebeca Construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module instance</td>
<td>Reactive-Class</td>
</tr>
<tr>
<td>Method/Thread</td>
<td>One message server</td>
</tr>
<tr>
<td>Non-process functions</td>
<td>No explicit construct</td>
</tr>
<tr>
<td>Function call</td>
<td>A copy of function’s body</td>
</tr>
<tr>
<td>Signal</td>
<td>Two global variables</td>
</tr>
<tr>
<td>Event</td>
<td>One boolean global variable</td>
</tr>
<tr>
<td>Variable</td>
<td>One global variable</td>
</tr>
<tr>
<td>Simulation Kernel</td>
<td>Synchronizer rebec</td>
</tr>
</tbody>
</table>

Table 1. An Overview of the SystemC to Rebeca Mapping
chain are pointers to this signal. Therefore, the most reasonable way to concern this is to use the variable representing the attached signal, anywhere a port is accessed.

**Events, wait and notify**  Events are used for process synchronization in SystemC. In the proposed mapping, for each event a boolean variable, demonstrates the occurrence of the relating event, is considered in the global scope. statement is translated to an assignment statement which sets the value of this variable to true. A wait statement on the other hand, is translated to an assignment statement which sets the value of the variable to false, followed by a Rebeca wait statement.

**Function Calls**  Functions and function calls are also supported by the tool. The only limitation is that recursive functions (direct or indirect) are not allowed. A function call is substituted by an inline copy of the translation of its body in the Rebeca model. In addition, there are two blocks of assignment statements before and after this chunk of code to handle parameter passing and return values.

We support all the SystemC constructs except the features concerning dynamicity (like pointers), timing and SystemC utility classes. In addition, because of the model checking nature, the floating point variables and large integers (containing more than 16 bits) are not supported. These restrictions are not critical and harmful for modeling in SystemC while they help to avoid the state space explosion.

### 4.3 SystemC Simulation Kernel

In this section we discuss the Synchronizer rebec which mimics the behavior of the SystemC simulation kernel. This rebec has a special message server named nextDelta, which models a delta cycle. According to the algorithm executed by the simulation kernel, in each delta cycle, first some processes are activated and executed and then the signals are updated.

In our method, the activation of processes in delta cycles is modeled by putting messages in the queues of corresponding rebecs.

Using the sensitivity lists (lists of events which affect the execution of processes) of all the processes, the Synchronizer rebec finds all the ready to run processes (according to the changed signals). The Synchronizer rebec then puts the desired messages in the queues of the corresponding rebecs.

When no process is ready to run, current simulation cycle is finished. In order to start the next simulation cycle, the inputs are assigned non-deterministically to one of their possible values to make some processes ready to run. Non-deterministic assignments ensure that all possible combinations for all inputs are covered in the verification phase.

### 4.4 Validation of the Mapping

Regarding the main features of SystemC, here we discuss the validity of the proposed approach.

1. **Concurrency**  : The concurrency is provided by assigning to each process a different rebec. These rebecs are then executed concurrently.

2. **Non-preemptive execution**  : The atomic execution of a message server supports this feature. A message server loses the control of execution only when there are no more statements to be executed, or when it reaches a wait statement.

3. **Event-driven computational model**  : This feature is supported by means of message passing and the fact that the execution of a rebec is event-driven, i.e. taking a message from the top of the queue and executing it.

4. **SystemC simulation kernel**  : The functionality of the kernel is centralized in the Synchronizer rebec which is responsible for specifying the events in each cycle and sending messages to corresponding rebecs.

### 5 Case-Studies

We verified a set of small and medium sized case studies using Afra. Table 2 shows for each of these case studies its size (number of modules, processes, and variables) and the number of states generated using Modere and SyMon. All of these models were model-checked against 5-10 properties using both Modere and SyMon. Some bugs are found and the codes are fixed using the counter examples. In [4] the details of these case studies, including their descriptions, SystemC and Rebeca codes, the verified properties and model checking results are explained.

One of our medium-sized case studies is a SystemC design for a single-cycle MIPS processor named mmMIPS1. The SystemC design of the MIPS processor has 17 concurrent threads and a total of 96 signals, events and variables. The equivalent Rebeca model of this design has 18 concurrent rebecs and 136 global variables.

We have verified the model against a number of properties. Here, we show five sample properties for checking the correctness of the SystemC code and the LTL formula for some of these properties. The first property concerns that we should not read from and write into the memory simultaneously (G(!((read & write))). The second property ensures that the next value of pc is always pc + 4 except when the current instruction is a branch instruction.

---

1The model is from “http://www.es.ele.tue.nl/education/Computation/mmips-la/lah” which contains the mmMIPS processor as well as several other variants of it.
Table 2. Results for some small and medium sized case studies

<table>
<thead>
<tr>
<th>Name</th>
<th># of modules</th>
<th># of processes</th>
<th># of variables</th>
<th># of states (Modere)</th>
<th># of states (SyMon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFF</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>45</td>
<td>44</td>
</tr>
<tr>
<td>Shifter</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>53</td>
<td>32</td>
</tr>
<tr>
<td>BusArbiter</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>87</td>
<td>78</td>
</tr>
<tr>
<td>Latched ALU</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>133</td>
<td>132</td>
</tr>
<tr>
<td>2_by_4_decoder</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>164</td>
<td>163</td>
</tr>
<tr>
<td>Full adder</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>321</td>
<td>233</td>
</tr>
<tr>
<td>Fibonacci Generator</td>
<td>1</td>
<td>3</td>
<td>10</td>
<td>1847</td>
<td>1817</td>
</tr>
<tr>
<td>GCD calculator</td>
<td>7</td>
<td>12</td>
<td>30</td>
<td>536747</td>
<td>582</td>
</tr>
<tr>
<td>mmMIPS</td>
<td>17</td>
<td>17</td>
<td>96</td>
<td>–</td>
<td>345986</td>
</tr>
</tbody>
</table>

Table 3. Compositional verification results for MIPS.

<table>
<thead>
<tr>
<th>model</th>
<th># of states (Modere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property 1</td>
<td>1120</td>
</tr>
<tr>
<td>Property 2</td>
<td>45141</td>
</tr>
<tr>
<td>Property 3</td>
<td>15270</td>
</tr>
<tr>
<td>Property 4</td>
<td>11731</td>
</tr>
<tr>
<td>Property 5</td>
<td>13685</td>
</tr>
</tbody>
</table>

\(G((\text{op}\_\text{code} \neq \text{BEQ}) \rightarrow (next_{pc} = current_{pc} + 4))\).

The third property checks whether the result of an arithmetic operation (R-type) matches the expected value. The forth property addresses the load word and store word instructions. When the instruction is load or store the output of the ALU should be the address of corresponding data memory cell. The fifth property checks the correctness of the next value of the program counter in the case of branch instructions \(G((\text{op}\_\text{code} = \text{BEQ})) \rightarrow ((\text{op}\_\text{code} = \text{BEQ}) \land next_{pc} = \text{branch\_address}))\).

To avoid the state space explosion we used the customized model checker, SyMon, instead of Modere.

As another approach, we used the compositional verification technique of Rebeca [21]: decomposing a closed model and verifying one (open) component. Based on this technique five different models are extracted from the original model according to each property. For verifying each property we extract components which are involved in generating the values of variables that are used in the related property. Table 3 shows the number of states generated for verifying each property using this approach.

6 Related works

Different approaches are used in formalizing SystemC semantics and verifying the designs. In [18] and [19] the structure of the SystemC designs are captured by a component-based coordination language. In a parallel work with the current paper, a process algebraic semantics is proposed in [11]. The main differences are in the exploited formalisms: component-based visual language, actor-based language (Rebeca), and process algebra. The first approach shows the structure and behavior of a design together, where the last two approaches focused on behavior and model checking.

Considering the covered features of SystemC, amount of automation of the verification process, and efficient tool support, the closest approach to the approach presented in this paper is the one presented in [15]. In [15] the authors introduced a toolbox called LusSy for analysis of SystemC designs, including the transaction-level [9] specific constructs. Semantics of the SystemC designs are captured by means of parallel automata and the tool LusSy makes the process automatic. Then, the automata-based verification engines such as Lustre and SMV can be used to verify SystemC designs. The focus of [15] is only on the assertion-based verification which is restrictive compared with LTL and CTL model checking. The compositional verification approach, abstraction, and reduction techniques which are well established for Rebeca (and hence for Afra) are considered as future work for LusSy.

The first formal verification approach for SystemC has been presented in [10]. In this paper the authors have presented a new approach to reason about the behavior of SystemC designs based on the formal verification of properties specified in linear temporal logic (LTL). The work is based on reachability analysis of SystemC using BDDs, which is presented in [7].

A formalization for the semantics of SystemC, by means of Labeled Kripke Structures (LKS) is presented in [14]. The work is more focused on an abstraction-refinement approach based on automatic hardware/software partitioning.

In [22] the authors have described an encoding of SystemC designs into Promela. In their encoding they have considered SystemC designs at transaction level. Such a restriction means that neither built-in primitive channels nor
the so-called request update mechanism which deals with delta-time delays are supported. The SPIN model checker is then used to verify the given LTL properties.

The works mentioned in [14, 18, 19, 22], although valuable and applicable in many cases, lack a tool to automatically apply the given approaches on SystemC designs and obtaining the given formalisms. In addition, [14, 18, 19] also lack an efficient verification back-end. Our approach, in contrast, is implemented in an automatic tool.

7 Conclusion and Future work

In this paper we have introduced an approach for formal verification of SystemC designs. SystemC designs are transformed into Rebeca models as the intermediate code and Rebeca model checkers are then used to verify them.

We have developed a tool, Sytra, which automatically transforms SystemC designs into Rebeca models. The tool is integrated with Rebeca model checkers to form Afra, a SystemC verification tool. The results of verifying a number of small and medium sized designs using this tool are presented in this paper.

An interesting direction to continue the work is to equip the tool with additional automated state space reduction techniques. An ongoing work is considering transaction level constructs. We consider these constructs as black boxes and model their behavior as rebecs which is similar to the approach in LusSy.

Acknowledgements

The research of the authors is supported by a grant from Hi-Tech Industries Center. We also thanks to Iran Telecommunication Research Center for their grants on this project. We would like to thank Sander Stuijk and Hossein Hojjat for providing and helping us with mips case study. We also thanks to Iran Telecom-Hi-Tech Industries Center. We also thanks to Mohammad Mahdi Jaghoori, Amin Shali, and Ehsan Khamespanah. We would like to thank our research group at Formal Methods Lab. of ECE Department of University of Tehran specially to thank our research group at Formal Methods Lab. of ECE Department of University of Tehran specially

References