Improving System-Level Verification of SystemC Models with SPIN[*]

Martin Elshuber, Susanne Kandl, and Peter Puschner

Institute of Computer Engineering
Vienna University of Technology
Treitlstr. 3, 1040 Wien, Austria
`{martine,susanne,peter}@vmars.tuwien.ac.at`

Abstract

SystemC is a de-facto industry standard for developing, modelling, and simulating embedded systems. As embedded systems become more and more integrated into many aspects of human lives (e.g., transportation, surveillance systems, . . .), failures of embedded systems might cause dangerous hazards to individuals or groups. Guaranteeing safety of such systems makes formal verification crucial. In this paper we present a novel approach for verifying SystemC models with SPIN. Focusing on system-level verification we reuse compiled and executable code from the original model and embed it into the verifier generated by SPIN. In contrast to most other approaches, which require a complete model transformation, in our approach the transformation focuses only on the relevant parts of the model while leaving functional blocks untransformed. Our technique aims at reducing the state vector size managed by the verifier of SPIN, at improving state exploration performance by avoiding unnecessary model transformation steps, and at concentrating on verifying properties that emerge from the composition of multiple functional units.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases SystemC, SPIN, Promela, System-Level Verification

Digital Object Identifier 10.4230/OASIcs.FSFMA.2013.74

1 Introduction

Nowadays computer systems are more and more introduced into many aspects of human lives. Especially when they contain safety-relevant features, failing may cause dangerous hazards. Consequently, formally verifying that certain properties of the system hold under all circumstances becomes a central task in system design.

With the growing complexity of state-of-the-art computer systems, manual proofs often turn out to be infeasible and error prone. To circumvent this problem, tools have been developed that analyse models at different abstraction levels (e.g., system specification, system implementation, ...) in order to formally prove that system properties match the desired behaviour of the developed product.

SystemC: *SystemC* is a de-facto industry standard for modelling systems at system level, and can be used to model software and hardware aspects in a single language. *SystemC* is an add-on library to C++. *SystemC* extends C++ by constructs similar to Hardware

[*] This work has been partially funded by the ARTEMIS Joint Undertaking and the National Funding Agency of Austria for the project VeTeSS under the funding ID ARTEMIS-2011-1-295311.

Description Language (HDL) languages and a scheduler. Such models can be compiled to native machine code for most of the existing hardware architectures, thus allowing fast and accurate simulation of the system.

State-of-the-art verification: Although simulation is a proper method for detecting many bugs in a system, it cannot be used to verify whether a property of a system holds for every possible system state or not. Formal verification, on the other hand, guarantees the validity of a property for all possible system states.

Amongst others, we emphasise two reasons that make it difficult to formally verify *SystemC* models: *(1)* the constructs introduced by *SystemC* use a scheduler to execute and schedule processes activated on specific events. *(2) SystemC* allows to freely use C++ constructs like class inheritance, library functions, or Standard Template Library (STL), ...

State-of-the-art techniques address these problems by transforming the model to another language and use existing tools. This transformation requires *(ad 1)* to model the scheduler explicitly, thus increasing the overall state space, and *(ad 2)* to restrict the *SystemC* model to a specific subset of *SystemC* (e.g., prohibiting class inheritance).

This paragraph gives a short summary of existing verification tools. The *SystemC* category in the 2nd International Competition on Software Verification (SV-COMP) 2013 [2] provided the *SystemC* benchmarks already transformed to *C*. Cimatti et al. [6] describe the transformation of *SystemC* to *C*, thus reducing *SystemC* verification to software verification. The implementation presented in this work uses the tool *Pinapa* [11]. *Pinapa* is a predecessor of *PinaVM* [10], the tool we are referring to in this work. The *SystemC* category of SV-COMP 2013 was won by *UFO* [1], a framework for software verification working on LLVM Bitcode (LLVM BC). Second and third place were assigned to two *CPAchecker*-based verifiers [3]. Also worth to mention are Bounded Model Checking (BMC) [4] approaches used for example by *CBMC* [7]. *Scoot* [5] is an extension to *CBMC* allowing *SystemC* verification. SPIN [8] is a popular tool for proving properties of asynchronous distributed systems specified in the language *Promela*. PAT (Process Analysis Toolkit) [12] is a modular toolkit for verification and simulation of concurrent systems.

A glance at our approach: In system-level verification we concentrate on the composability aspects of systems consisting of several functional blocks. Assuming that each functional block works as specified, we are interested in verifying properties that emerge from the composition of those blocks. The approaches mentioned above aim to exhaustively verify the system with all implementation aspects included.

Our approach solely transforms the interaction of functional blocks into the formal language *Promela* and executes code within a functional block as a single transition. Based on a model analysis done by *PinaVM* at LLVM BC level, we split the model into several functions which are embedded into the SPIN verifier and executed atomically. With this technique we can use the model checking capability of SPIN on a model that represents the relevant aspects of the system, whereas details within single blocks of the system are hidden. Thus it is possible to focus on the verification of system properties without considering functional details which may easily cause a state space explosion during the verification. In [13] a similar approach for multi-threaded C programs, where SPIN orchestrates the search, is proposed.

The remainder of this paper is structured in four further sections. Section 2 gives a more detailed overview on existing technology reused during this work. Section 3 describes the verification process of our approach. In the following sections we discuss the advantages and disadvantages of the concept and conclude with a summary of the paper.

2 Prerequisites

This section gives a brief introduction to existing technology (namely *SystemC*, SPIN, and *PinaVM*) we build our approach on top.

SystemC: *SystemC* is a library on top of C++. For hardware aspects *SystemC* modules are defined. They contain input-, output-, bidirectional ports for communication, and processes acting on these ports. Software aspects can be implemented using classic C++. The important fact is that *SystemC* processes are executed non-preemptively. The effect is that all modifications to the system state semantically take place at the instant when the process preempts itself. As already mentioned above, state-of-the-art verification of *SystemC* models often requires a transformation from *SystemC* to another language, and to model the scheduler separately.

SPIN: SPIN is an on-the-fly model checker, which can be used to run and verify models described in the language *Promela*. The interesting part is the way SPIN verifies a model. It first translates the *Promela* model into a C verifier which has to be compiled and run to execute the verification. In the verifier the *Promela* model is translated into a transition system represented as a `switch/case` statement. The verifier then searches the transition system for errors and reports paths if a problem was found. To allow backtracking the state vector is stored and compared against states which have already been investigated. SPIN also implements various performance features, like partial order reduction, and techniques reducing the memory requirements for storing the state vector. *Promela* provides constructs for in-lining C-Code into the verifiers transition system.

PinaVM: *PinaVM* is a tool, developed by Verimag [10], able to analyse the structure *SystemC* models. It detects which *SystemC* modules are created and how they interact with each other. Thus simplifying the translation into arbitrary languages. *PinaVM* roughly works in several phases:

Phase 1: Use LLVM to create an LLVM BC of the model.

Phase 2: Analyse the created functions in Phase 1 and find out where each *SystemC* construct is used.

Phase 3: Execute the models initialisation code generated in Phase 1 detecting the instantiated *SystemC* classes.

Phase 4: In this instant it is known what the *SystemC* model looks like (instantiated modules; Phase 2), how the interact (instantiated ports; Phase 2) and which code parts manipulate the structures (Phase 1). This information is passed to a back-end, which transforms the model to the desired format usable by existing model checker infrastructure.

PinaVM also provides a back-end for *Promela* including efficient encodings [9] for *SystemC* constructs like `wait`, `notify`, ... The *Promela* back-end translates every instruction of the LLVM BC into a corresponding *Promela* construct. We, on the other hand, only transform the *SystemC* constructs to *Promela*, and in-line the rest of the code directly into the SPIN verifier.

3 Our Approach

Our verification process is based on the given infrastructure described in Section 2. We plan to use *PinaVM* for analysing *SystemC* code, to add our own transformation technique, and to use SPIN for verifying the resulting *Promela* model. The difference to *PinaVM* using the

Promela back-end, is that we do not translate the whole model to *Promela*. We only model the *SystemC* constructs in *Promela* and include calls to the compiled *SystemC* model. These calls also modify the verifier's state vector.

Figure 1 depicts the steps executed to create a verifier binary with our approach. The gray blocks denote the steps that can be done by existing tools. These are SPIN, *PinaVM* or *LLVM*. Implementations for the white boxes are currently missing and are subject for the work to be done during the thesis. Most steps (except *Create Verifier C-Code* and *Compile & Link Verifier*) are done by PinaVM or an extension of it. PinaVM itself is based LLVM BC and uses *LLVM* libraries to handle the code.

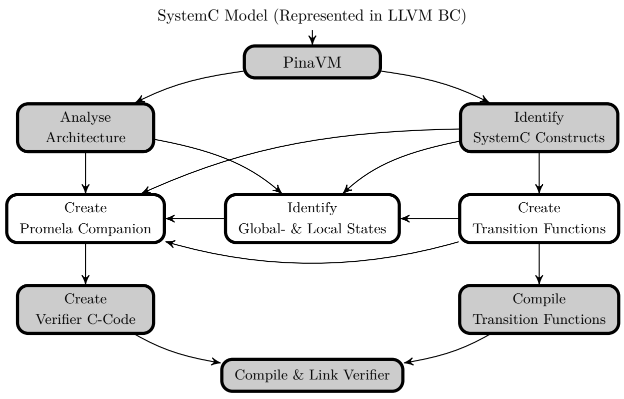
SystemC Model (Represented in LLVM BC)
PinaVM
Analyse Architecture
Identify SystemC Constructs
Create Promela Companion
Identify Global- & Local States
Create Transition Functions
Create Verifier C-Code
Compile Transition Functions
Compile & Link Verifier

Figure 1 From *SystemC* to the SPIN verifier executable.

The output of *PinaVM* to the back-end is the LLVM BC, enriched with information on *SystemC* constructs, as well as the system architecture instantiated during model initialisation. The text below describes the actions taken within each block:

- During **Analyse Architecture** the initialisation code of the model is analysed by *PinaVM* in order to detect the kind, number, and interaction channels and the *SystemC* modules.
- During **Identify SystemC Constructs** all functions are analysed by *PinaVM* to detect and mark *SystemC* constructs such as `wait` and `notify`, but also `write` and `read`.
- **Create Transition Function:** The *SystemC* model is split into transition functions callable by the verifier. These functions are compiled separately and finally linked to the verifiers binary.
  The original functions are divided into code regions, such that each code region either contains no *SystemC* construct at all, or it consists solely of a single *SystemC* construct. Each code region that contains no *SystemC* construct is converted into a separate function. This function returns a reference to the next code region to be executed, and it receives parameters according to the values read or modified during execution.
  The resulting functions have the same semantics as the original functions when called in a proper order. Furthermore each function returns at each point the *SystemC* scheduler might preempt the execution of the original thread.
- **Identify Global- & Local States:** Depending on the model the states of the system have to be identified. *Promela* distinguishes three kinds of state variables. *Global* state variables are instantiated only once, *Local* state variables are instantiated per process and thus can be stored multiple times in the state vector, and finally *hidden* states are never stored in the state vector, thus they cannot be restored on backtracking.
  In our approach variables that are read and written solely within a code region, can be totally hidden from the verifier and are stored on the function's stack or optimised into a processor register. Variables that are written in a code region, but possibly read by another code region have to be instantiated somewhere in the verifier. To decide which type of variable has to be used, control flow analysis has to be done.
  A variable can be declared *hidden*, if and only if all statements between (and including) writing and reading are executable in an atomic manner. Because *SystemC* processes

are non-preemptive, this is the case if no blocking *SystemC* construct can be executed between writing and reading.
The rest of the variables have to be stored in the state vector, and are *global* or *local*. Variables are declared local if they are declared locally in the *SystemC* process parenting the code region. They are declared globally otherwise. Member variables of *SystemC* classes can also be declared globally, because from the architecture analysis phase it is known how many objects are instantiated.

- **Create *Promela* Companion:** The *Promela* Companion is the actual input file to SPIN. It encodes
  - *SystemC* constructs similar to [10, 9],
  - definitions of all *hidden*, *local* and *global* state variables accessible by other Promela constructs like LTL formulas, and
  - calls to the transition functions as well as the control flow among them (`goto` statements, and C in-lining).
- Finally the verifier executable can be generated by using SPIN to **create the verifier C-Code** and LLVM to compile the transition functions, the verifier, and link all together.

4 Discussion

The main aim is to create a source-code driven verification system, allowing us to verify properties on system level while disregarding details within functional blocks. A requirement of the implementation of each functional block is that it is free of memory bugs such as buffer under- and overflow, access violations, and so on.

Furthermore, the structure of the model architecture has to be static. This means that no dynamic *SystemC* constructs must be created, except during the initialisation code. This requirement stems from they way *PinaVM* works.

SystemC verification is often driven by translating the model into plain C or similar languages and by adding a scheduler to the translated model. This has the disadvantage of introducing additional states and thus adding complexity to the verification process. With our approach we reuse at least parts of the process scheduler of SPIN, thus aiming at an improvement of the verification process.

We also expect improvements in the memory requirements, by reducing the size of the state vector. The expected effect is mainly caused by hiding states from the verifier, thus disallowing it to backtrack to system states that are irrelevant for system-level verification. Assume a model does some computation inside a loop, whereas the result is passed to other functional blocks solely after the loop. The intermediate results (e.g., temporary variables) do not have to be included in the system states. Thus we expect an improvement in both the state vector size and the number of states that have to be investigated. The effect of the latter one is expected to be smaller as partial order reduction also can be done by SPIN.

By avoiding the complete transformation of the model from C++ to *Promela* and then back to C again, we think that we can improve the search speed of the verifier, and thus increase the number of states investigated per second. This is because each transition in a *Promela* model is selected by a switch statement. The size of the switch statement and the number of entries within it is expected to be reduced.

So far we implemented a prototype that automatically extracts the transition functions of simple *SystemC* models. First experiments showed that the size of the state vector is reduced and the number of explored states is kept at a similar amount compared to the *Promela* backend of PinaVM.

5 Conclusion

In this paper we presented an idea for a novel approach to improve the formal verification process on system-level of a *SystemC* model. The *SystemC* model is transformed into a formal automaton model by interpreting the *SystemC* constructs and assigning precise semantics to them. By a straight-forward transformation the whole functionality of the *SystemC* model is represented in the resulting formal model with the consequence that all the complexity of the system description is part of the verification process. In our approach the model transformation is realized in such a way that functional details within a block of the system model are hidden and only the aspects of the model that are relevant for system-level verification are considered for the verification process. This principle should enhance the verification process by saving time and memory within the model checking process by SPIN.

References

1 Aws Albarghouthi, Yi Li, Arie Gurfinkel, and Marsha Chechik. Ufo: A Framework for Abstraction- and Interpolation-Based Software Verification. In *CAV*, pages 672–678, 2012.

2 Dirk Beyer. Second Competition on Software Verification - (Summary of SV-COMP 2013). In *TACAS*, pages 594–609, 2013.

3 Dirk Beyer and M.Erkan Keremoglu. CPAchecker: A Tool for Configurable Software Verification. In *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 184–190. Springer Berlin Heidelberg, 2011.

4 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded Model Checking, 2003.

5 Nicolas Blanc, Daniel Kroening, and Natasha Sharygina. Scoot: A Tool for the Analysis of SystemC Models. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 467–470. Springer Berlin Heidelberg, 2008.

6 A. Cimatti, A. Micheli, I. Narasamdya, and M. Roveri. Verifying SystemC: A software model checking approach. In *Formal Methods in Computer-Aided Design (FMCAD), 2010*, pages 51–59, 2010.

7 Edmund Clarke, Daniel Kroening, and Flavio Lerda. A Tool for Checking ANSI-C Programs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

8 Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23:279–295, 1997.

9 Kevin Marquet, Jeannet Bertrand, and Matthieu Moy. Efficient Encoding of SystemC/TLM in Promela. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2011*, pages 1039–1044, 2011.

10 Kevin Marquet and Matthieu Moy. PinaVM: A SystemC front-end based on an executable intermediate representation. In *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '10, pages 79–88. ACM, 2010.

11 Matthieu Moy, Florence Maraninchi, and Laurent Maillet-Contoz. Pinapa: an extraction tool for SystemC descriptions of systems-on-a-chip. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT '05, pages 317–324. ACM, 2005.

12 Jun Sun, Yang Liu, JinSong Dong, and Jun Pang. PAT: Towards Flexible Verification under Fairness. In *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer Berlin Heidelberg, 2009.

13 Anna Zaks and Rajeev Joshi. Verifying Multi-threaded C Programs with SPIN. In *Proceedings of the 15th international workshop on Model Checking Software*, SPIN '08, pages 325–342. Springer-Verlag, 2008.