Automatic Equivalence Checking for SystemC-TLM 2.0 Models Against their Formal Specifications

Mehran Goli[1] Jannis Stoppe[1,2] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{mehran jstoppe drechsle}@informatik.uni-bremen.de

***Abstract*—The necessity to handle the increasing complexity of digital circuits has led to the usage of more and more abstract design paradigms. In particular, the *Electronic System Level* (ESL) has become an area of active research and industrial application, especially via SystemC and its *Transaction Level Modeling* (TLM) framework. Additionally, the usage of formal specification languages such as the *Unified Modeling Language* (UML) prior to the implementation (even at higher abstraction levels) is now a broadly accepted workflow.**

Utilizing this layered approach leaves the translation from the specification to the implementation to the designer, leaving the question unanswered how the equivalence of these should be verified. This paper proposes a novel, non-intrusive and broadly applicable approach to automatically validate the equivalence of the structural and behavioral information of a SystemC-TLM 2.0 model and its formal specification.

I. Introduction

One approach to handle the increasing complexity of circuits and systems is the utilization of more abstract description means. Specifying systems at the *Electronic System Level* (ESL) is an established approach to cope with this complexity. System design at the ESL with SystemC [1] – a C++ based ESL design library – has become a de-facto standard [15].

Additionally, formally specifying designs prior to their ESL implementation using modeling languages such as the *Unified Modeling Language* (UML) [14] is an established workflow that is used to ensure compliance with constraints. Based on the great potential of UML to complement current C++-oriented languages, system designers commonly use UML for the modeling phase in the design process [12], [13]. By this, the structure and behavior of the design can be specified using the UML. Although these descriptions can be exploited to generate ESL code stubs or initial implementations, manual refinement of the resulting ESL implementation is required afterwards. Due to this manual process, new errors might be introduced or design decisions made for the formal specification may have to be altered or reverted. The final ESL implementation might differ from the formal specification in both structure and behavior. Therefore, it is necessary to validate that the implemented ESL design matches the original formal description.

This paper introduces a method that focuses on checking the compliance of a given design in ESL against its formal specification. The suggested methodology

- retrieves the static information of an ESL model's debug symbols which is required to extract run-time information,
- retrieves the run-time information which refers to the order of components activation and the data to describe the model's architecture at simulation time and finally
- compares this extracted information with the original specified formal description.

A case study illustrates the advantages of the proposed approach. To do this, the method is applied to check the compliance of a SystemC-TLM 2.0 implementation against its formal specification.

II. Related Work

Several approaches for validation of SystemC-TLM 2.0 based design have been proposed, with the majority of them employing assertion-based verification at the ESL. For these approaches, system properties to be verified are expressed as assertions (written in languages like *Property Specification Language* PSL or System Verilog) and checked using static (model checking) [10], [7], [11] or dynamic (simulation based) [16], [5], [4] techniques. They require complex specification to describe TLM behavior and specifically focus on verification of a SystemC-TLM 2.0 implementation against TLM rules and do not cover its compliance checking against a corresponding formal specification.

In [9], formal specifications (specified using SysML, a UML dialect, and the *Object Constraint Language* OCL) are translated to SystemC-TLM 2.0 models to ensure compliance. The method is limited to only ensure that the generated model adhered to the rules – it does not validate any model's structural data that are not generated from that description or those are altered manually afterwards.

The method in [8] proposes an automatic two-step compliance checking technique for TLM 2.0 models based on a UML profile. The first step checks the static TLM 2.0 rules during the code generation process. The second step generates and adds the required TLM 2.0 components to the model's source code, allowing the transactions to be recorded during simulation of the model. A sequence diagram validation is performed against the TLM 2.0 rules in an off-line TLM 2.0 compliance checking process. Although the method can verify some behavioral parts of the model, it is limited to models which are using the UML profiling and automatic code generation steps during the design process. Thus it does not support the validation of existing or manually altered models to a given specification.

The approach in [3] presents a method for protocol compliance checking of SystemC-TLM 2.0 models. First, it generates a graph representation from compact user specifications of protocol sequences. Afterwards, the protocol checker as well as the design's modules are instantiated. While the method checks equivalence between a SystemC-TLM 2.0 model's behavior and its formal specification, neither the equivalence of the model's structure nor the expected sequence of component activations is validated. Additionally, the usage of protocol checkers makes the method intrusive as they are inserted to the original source code.

Another recently published method [19] takes advantage of a SystemC data extraction method [18] to retrieve the information of a SystemC model. It compares the extracted data to the model's formal specification. The extracted information only describes the structure of the model and not its behavior. Therefor, the method is restricted to only check the structure of two models.

Existing solutions have two major limitations in terms of equivalence checking of a given ESL design's behavior against its formal specification and restricting the language. The first limitation is that most of them only check the compliance of two models' structure and not its behavior. The second limitation is that most of them can only be applied to a restricted range of SystemC designs and do not support TLM constructs.

III. Proposed Methodology

In order to check equivalence between an ESL model and its formal specification, both architectural and behavioral information, need to be validated. The former refers to the object instances that define the structure of the model (such as modules or signals), the latter refers to

978-3-9815370-8-6/17/$31.00

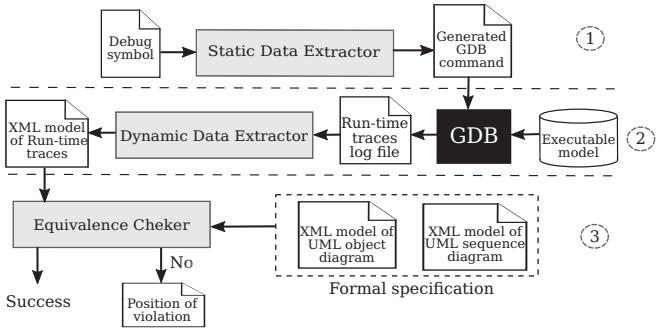

Fig. 1. Architecture of proposed methodology.

the simulation of the design that needs to comply with the constraints that are given in the specification (such as the order in which methods are invoked to follow a certain communication protocol).

The formal description of the architecture and the behavior of an ESL model is done based on block definition diagrams and sequence diagrams, respectively. The block definition diagram formally describes the structure of the model in a hierarchy format including components and their attributes and member functions. The sequence diagram consists of a set of scenarios which describes the expected sequences of components' activation for different transactions. In particular, each scenario includes a set of sequence numbers indicating the order of components' activations during the model execution from the time that a transaction is created until it ends.

Based on this idea, the structure and the behavior of the ESL design require to be extracted first. Afterwards, the extracted information is translated to a structural format in order to be comparable with the corresponding formal specification.

In order to analyze SystemC designs, an approach using the *GNU debugger* (GDB) [17] as a gateway to the runtime information was recently proposed [6]. This approach was adapted to retrieve the information required to validate the SystemC implementation against its specification. The architectural view of the proposed methodology is presented in Fig. 1 including the flow of data extraction demonstrated in Phases 1 and 2.

1) Static information is extracted by invoking GDB for a compiled SystemC design (that includes the debug symbols). This data is used to (again automatically) generate a set of GDB instructions that is required for the next phase.
2) The original design is executed via GDB using the previously generated instructions to retrieve the run-time information of the model. These instructions make GDB pause the execution at selected events (such as function calls) and write information about the execution state to a log.
3) This information (that has been stored in the previous phase from the ESL model's execution) is compared to the corresponding formal description. If the given models match, the simulation adheres to the formal specification, thus indicating that the implementation did not violate any previously specified constraints (so the result is a successful validation). Otherwise, the position of violation is reported as a log file.

IV. Implementation

This section describes the implementation of the proposed methodology in detail.

A. Retrieving Static information

The *Static Data Extractor* module retrieves the static information of a model from its generated GDB debug symbols as shown in Fig. 1, Phase 1. It stores the extracted data into a more manageable data format. It includes a design's modules, their member functions and attributes in a hierarchical model.

```
<ESL_RUN_TIME_TRACE Design_name = "AT–example">
. . .
 <SEQUENCE>
  <SEQUENCE_NUMBER> 10 </SEQUENCE_NUMBER>
  <ROOT_MODULE>AT_interconnect</ROOT_MODULE>
  <FUNCTION_NAME>nb_transport_fw</FUNCTION_NAME>
  <INSTANCE_NAME>top.interconnect</INSTANCE_NAME>
  <LINE_OF_CODE>
   <SOURCE_FILE>at_interconnect.h</SOURCE_FILE>
   <LINE_NUMBER> 115 </LINE_NUMBER>
  </LINE_OF_CODE>
  <SIMULATION_TIME> 0 </SIMULATION_TIME>
  <TRANS_ID> 0x7054f0 </TRANS_ID>
 </SEQUENCE>
. . .
</ESL_RUN_TIME_TRACE>
```

Fig. 2. A part of XML model of the *AT-example* run-time trace.

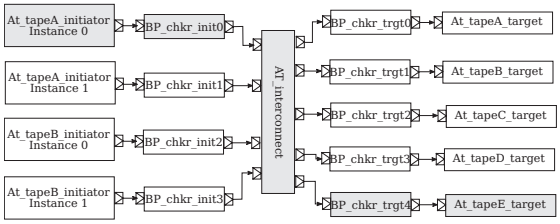

Fig. 3. Architecture of the *AT-example*.

The extracted static information is used as the foundation to retrieve the dynamic information in the next step during execution of the executable model. Based on the extracted static information, the *Static Data Extractor* module automatically generates a *GDB Command File (GCF)* which is required to extract the dynamic information in the second phase. This GCF is a text file including GDB commands that later controls the behavior of the debugger.

B. Retrieving Run-time Information

Based on the commands within the GCF, breakpoints are set for each modules' functions and global functions within the design. These pre-defined breakpoints are scripted to automatically execute a GDB function which is defined separately for them in the GCF. These contain instructions for the debugger to retrieve the dynamic information of the SystemC model during its execution. More precisely, the structure of the model is extracted based on the commands of each pre-defined breakpoint. This information consists of the modules' root names, instance names, variables and the binding information of the respective ports as well as the functions' names, their return types and local variables. Based on the instruction in each pre-defined GDB function the model's behavior is retrieved, including the current function's name, line of code, source file name, the simulation time and (for TLM models) the transaction reference address.

The execution continues until the next pre-defined breakpoint that refers to the next function is hit, over time retrieving a vast amount of run-time information. As illustrated in Fig. 1, Phase 2, this information is stored in the according log file.

In the next step, the extracted information is translated to an XML model. The XML model presents the structure of the design in a hierarchical format in which modules and global functions are the root elements and their member functions and attributes are their child elements. The behavior of the model is presented as a set of scenarios separated by the transaction lifetime (i.e. the time that a transaction is created until it ends). Each scenario is defined as a set of sequence numbers where each sequence number indicates the order of components' activation during the model execution. Fig. 2 shows a part of behavioral information of the *AT-example* (which is introduced in detail in Section V) in XML format.

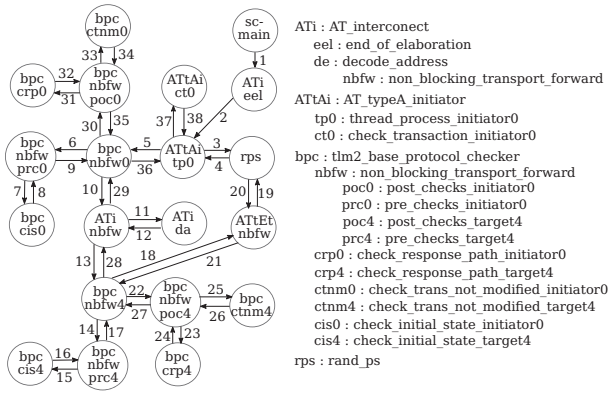

Fig. 4. Sequence diagram of the *AT-example* including initiator module typeA, base protocol checkers, interconnect module and target module typeE.

C. Equivalence Checking of Models

Checking the compliance of a SystemC-TLM 2.0 design's structure with its specification is done by comparing the structural information of the XML model of run-time traces from Phase 2 to its formal presentation. As the structure of an ESL design is identified by the root elements which are modules and global functions, a bijective (one-to-one) mapping algorithm is proposed to check the equivalence of two models. Based on this, a root element in the XML format of the ESL's formal specification is selected to be mapped to the corresponding root element in the XML model of its run-time traces. Whether an element of the specification matches an element from the implementation is determined by several criteria:

- their names must be equal,
- their child elements must be equal, i.e.
  - their variables' names and types,
  - their ports' names and types and
  - their functions' signatures (name, return type and parameters) and their local variables.

If no fitting root element is found in the XML model of run-time traces, the according implementation element is reported as a violation. Otherwise, the algorithm continues with the next element.

The ESL model's behavior compliance is tested by checking for each transaction whether the expected sequences of components' activation (i.e. function calls and according responses) matches the sequence specified formally. Fig. 4 shows a scenario of activations for the *AT-example* depicted using a UML sequence diagram. It describes the expected behavior of the gray components in Fig. 3 which presents the architecture of the *AT-example*. Based on Fig. 4, each transaction created by an instance of initiator module type `AT-typeA-initiator` must pass through the `AT-interconnect` module to reach the target module of type `AT-typeE-target`. The transactions created in this scenario are passed as a function argument with a unique reference address that can be used as a transaction ID in each scenario.

The equivalence checking algorithm checks for each unique transaction in the extracted run-time traces of an ESL model whether or not it complies with the expected scenario. The algorithm itself is illustrated in Algorithm 1. The transactions' reference addresses are stored in the unique transaction list. For each transaction in the list, a matching scenario is then selected from the formal specification part if available. If a fitting scenario is found, the success is indicated to the user and, for each extracted transaction, a corresponding (XML formatted) log is generated from the transaction's run-time traces. Otherwise, while the algorithm continues for all transactions and scenarios, the scenarios that could not be matched are reported as violations.

Algorithm 1: Equivalence checking algorithm

```
foreach sequence in XML_runtime_trace do
    if trans_address not in transaction_unique_list then
        add (trans_address, transaction_unique_list);
foreach transaction T in transaction_unique_list do
    foreach scenario in scenario_list do
        if scenario in XML_runtime_trace then
            equivalence_list [scenario] = true;
        else
            equivalence_list [scenario] = false;
```

V. Evaluation

All execution times have been measured on a PC equipped with 8 GB RAM and an Intel core i7-2760QM CPU running at 2.4 GHz.

The proposed approach has been applied to several ESL models. All case studies which are presented in Table I are taken from the standard examples which have been provided by Doulos [2] and OSCI. As none of the case studies have available formal specification, we manually created UML descriptions of each ESL design based on its text book specification.

As a representative to prove the benefits of the proposed method, the *AT-example* (a standard SystemC-TLM 2.0 Approximately-Timed (AT) design consisting of initiators, targets and an interconnect) is evaluated in detail.

A. Case Study: AT-example

Fig. 3 shows the structure of the *AT-example* design. It consists of 19 modules: four initiators, one interconnect, five targets and nine checkers. The difference between type `A` and `B` initiators and among type `A-E` targets is based on the various cases of permitted phase transitions of the SystemC-TLM 2.0 based protocol.

In this example, a part of the design including an instance type of `AT-typeA-initiator` module, the interconnect module, an instance type of `AT-typeE-target` and two TLM base protocol checkers `BP-chkr-init0` and `BP-chkr-trgt4` is considered.

The scenario of transactions exchanged among TLM 2.0 modules (i.e. the gray components in Fig. 3) has been depicted in Fig. 4 using a UML sequence diagram. It describes the expected order of components' activation during the model execution.

In Fig. 4, nodes represent the architectural information of components taking part in interactions while edges indicate the execution sequence of actions. The architectural information of each node is described in a hierarchical format including the module's name and the function's name (e.g. AT-interconnect::end-of-elaboration).

This specification information is compared to the data that is retrieved from the execution. As both, method calls and their order are retrieved from the simulation (and this information directly corresponds to the nodes and transitions shown in the diagram), they can easily be mapped to the specification data. The final step, a comparison of all specified transaction protocols to a given specific simulation trace is thus easily achievable. The *AT-example* is confirmed to match the specification. However, the more interesting question is how the simulation runtime is affected by applying the information retrieval approach. Thus, several other SystemC examples were instrumented using the proposed approach.

Table I summarizes the results obtained from the case studies. The first two columns show the ESL models as well as the number of lines of code for each design, respectively. The *LT-example* and *AT-example* are the standard TLM 2.0 based models which are provided by Doulos. Other designs are the SystemC standard models which are provided by OSCI. For each case study, *#Seq* is the number of retrieved sequences which refers to the number of components' activation during model's execution, *#uTrans* is the number of unique transactions. For a TLM model it indicates the number of different scenarios. The parameter *#Time* shows the total time of compilation and execution of models using either the proposed method (a)

TABLE I
CASE STUDIES

ESL Designs	#Line	#Seq	#uTrans	Time (s)[a]	Time (s)[b]
LT-example[1]	166	34	1	13	2
AT-example[1]	1851	738	10	183	21
fir[2]	233	126	-	39	4
pkt-switch[2]	1023	1954	-	109	8
risc-cpu[2]	1960	1130	-	82	11

[1] Provided by Doulos [2] Provided by OSCI.
[a] Total time of Models' compilation and execution using the proposed method.
[b] Total time of Models' compilation and execution without instrumentation.

and without any further instrumentation (b). It demonstrates that the proposed method can provide the designer and the equivalence checker with precise run-time information for an ESL model in reasonable time.

B. Integration and Discussion

The proposed method is able to automatically retrieve a significant amount of information to be used for the validation of both the structure and the behavior of an ESL implementation against a corresponding formal specification.

Unlike the approaches that focus on validation of static aspect of an ESL implementation, the proposed method checks the equivalence of the ESL models' behavior as well as their structure. Compared to methods which rely on code generation to retrieve run-time information by inserting some extra modules to the ESL implementation of a model, the proposed method extracts the detailed run-time information without any modification of the user's implementation. This means that the proposed method is non-intrusive and can be applied to future versions of SystemC-TLM 2.0 without any further changes. Thus, it can be combined with setups that already rely on a modified SystemC library. Moreover, it not only covers the equivalence check of a SystemC model against a corresponding formal specification, but also completely provides the feature for TLM 2.0 based designs. This makes the approach applicable for the validation of a wide range of applications in ESL designs against their formal specification.

The performance of the proposed method is measured concerning the time that is spent

1) extracting the run-time information of an ESL model and
2) comparing the extracted information to the corresponding formal specification.

The first step is the major part of the total execution time. It is related to Phases 1 and 2 of the proposed method as shown in Fig. 1. This time frame, even for complex TLM designs such as *AT-example* and SystemC models such as *pkt-switch*, is within reasonable boundaries in comparison with their pure compilation and execution time using GCC as presented in Table I. The proposed approach thus shows a trade-off between the precise information extraction to be used for the compliance checking purpose and the execution time.

The second step is related to the equivalence checking algorithm. For checking the validation of each expected scenario, the time complexity is $O(t \cdot m \cdot log\ n)$, where t is the number of unique transactions generated during the execution of the model, m is the number of sequences in the expected formal specification sequence diagram and n is the number of sequences in the run-time trace XML file. For the given examples, the time required to match the models is negligible.

VI. CONCLUSION

In this paper, we presented a novel method for the validation of a given ESL implementation against its formal specification. For this purpose, a non-intrusive approach was proposed which automatically extracts both the structure and behavior of a given ESL model, translates them to a structural format and then compares it with the corresponding formal specification. The presented method can be applied without any modification to the source code of the model or the library. A case study illustrated the usefulness of the presented methodology. Additionally, the extracted information in this method can be used for SystemC-TLM 2.0 visualization approaches.

VII. ACKNOWLEDGMENTS

The research reported here was supported by the German Federal Ministry of Education and Research (BMBF) under grants 01IW13001 (SPECifIC) and 01IW16001 (SELFIE), German Research Foundation (DFG) within the Reinhart Koselleck project DR 287/23-1, and University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

REFERENCES

[1] IEEE Standard SystemC Language Reference Manual. *IEEE Std 1666-2005*, pages 1–423, 2006.
[2] J. Aynsley. TLM-2.0 base protocol checker. https://www.doulos.com/knowhow/systemc/tlm2/at_example. Accessed: 2016-01-30.
[3] M. Bawadekji, D. Große, and R. Drechsler. TLM protocol compliance checking at the electronic system level. In *Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 435–440, 2011.
[4] W. Ecker, V. Esen, T. Steininger, M. Velten, and M. Hull. Interactive presentation: Implementation of a transaction level assertion framework in SystemC. In *Design, Automation and Test in Europe (DATE)*, pages 894–899, 2007.
[5] L. Ferro and L. Pierre. Formal semantics for PSL modeling layer and application to the verification of transactional models. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1207–1212, 2010.
[6] M. Goli, J. Stoppe, and R. Drechsler. AIBA: an Automated Intra-cycle Behavioral Analysis for SystemC-based design exploration. In *proceedings of the IEEE International Conference on Computer Design (ICCD)*, 2016.
[7] A. Habibi and S. Tahar. Design and verification of SystemC transaction-level models. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 57–68, 2006.
[8] V. Jain, A. Kumar, and P. Panda. Exploiting UML based validation for compliance checking of TLM 2 based models. *Design Automation for Embedded Systems*, pages 93–113, 2012.
[9] V. Jain, A. Kumar, and P. R. Panda. A SysML profile for development and early validation of TLM 2.0 models. In *European Conference Modelling Foundations and Applications (ECMFA)*, pages 299–311, 2011.
[10] D. Karlsson, P. Eles, and Z. Peng. Formal verification of SystemC designs using a petri-net based representation. In *Design, Automation and Test (DATE)*, pages 1228–1233, 2006.
[11] Moy, Maraninchi, and Maillet-Contoz. LusSy: A toolbox for the analysis of systems-on-a-chip at the transactional level. In *Application of Concurrency to System Design (ACSD)*, pages 26–35, 2005.
[12] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, and Y. Vanderperren. UML for ESL design: basic principles, tools, and applications. In *International Conference on Computer-Aided Design (ICCAD)*, pages 73–80, 2006.
[13] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. a UML 2.0 profile for SystemC: Toward high-level SoC design. In *ACM International Conference on Embedded Software (EMSOFT)*, pages 138–141, 2005.
[14] J. Rumbaugh, I. Jacobson, and G. Booch, editors. *The Unified Modeling Language Reference Manual*. 1999.
[15] C. Schulz-Key, M. Winterholer, T. Schweizer, T. Kuhn, and W. Rosentiel. Object-oriented modeling and synthesis of SystemC specifications. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 238–243, 2004.
[16] H. Sohofi and Z. Navabi. Assertion-based verification for system-level designs. In *Fifteenth International Symposium on Quality Electronic Design*, pages 582–588, 2014.
[17] R. Stallman. *Debugging with GDB*. Free Software Foundation, 2011.
[18] J. Stoppe, R. Wille, and R. Drechsler. Data extraction from SystemC designs using debug symbols and the SystemC API. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 26–31, 2013.
[19] J. Stoppe, R. Wille, and R. Drechsler. Validating SystemC implementations against their formal specifications. In *Symposium on Integrated Circuits and Systems Design, (SBCCI)*, pages 1–8, 2014.