Modeling and Analyzing Timing Faults in Transaction Level SystemC Programs

Reza Hajisheykhi[1], Ali Ebnenasir[2], and Sandeep S. Kulkarni[1]

[1] Michigan State University
East Lansing, Michigan 48824, USA
{hajishey, sandeep}@cse.msu.edu
[2] Michigan Technological University
Houghton, Michigan 49931, USA
aebnenas@mtu.edu

1 Introduction

In order to increase design productivity of SoC (System on Chip) systems, there is a need to move from implementation-driven design at Register Transfer Language (RTL) [1] to higher levels of abstraction. This move introduces a shift in the development of electronic systems, which has been put into practice as Electronic System Level (ESL) design. The main advantages of ESL are to enable: system-level design, hardware/software co-design, architecture exploration, virtual prototyping, and co-simulation/co-verification. The importance of ESL has become reality with *Transaction Level Modeling* (TLM) standard TLM-2.0 [2]. The C++-based system modeling language *SystemC* [3] perfectly supports TLM and hence is well-accepted for ESL design in industry. Since the TLM models serve as references for the RTL implementation, it is necessary to study the effect of faults in such models. Also in SoC systems, scheduling and timing play an important role. Hence, analyzing the system in the presence of timing faults is essential. This paper studies the effect of timing faults on SystemC TLM programs.

Formal verification and, in particular, model checking of finite models created from SystemC programs has been studied extensively in the past decade. Nonetheless, none of the studies addresses the issue of timing faults in the context of SystemC TLM programs. Specifically, [4] focuses on fault-free behaviors and does not support timing faults. Although [4] permits modeling of timing properties (not faults), the generated models are often too large to perform significant analysis. The results in [5] focus on untimed behaviors and does not address timing faults. And, [6] does not address timing faults considered in this paper. The objective of our paper is to develop a methodology to support analysis in the presence timing faults. Since SystemC TLM programming is typically considered in two forms: loosely-timed (LT) and approximately-timed (AT), we consider both coding styles. The LT coding style utilizes b_transport for communication between modules. As the name suggests, in the LT coding style, there is a loose dependency between timing and data. The AT coding style utilizes nb_transport for communication between modules. This captures the global time more precisely. In this coding style, there is a stronger dependency between timing and data.

To model timing faults in SystemC TLM programs, our approach uses three steps: (1) *model extraction*, (2) *timing faults modeling*, and (3) *analyzing the model in the presence of timing faults*.

For the first part, we use the ideas in [4] and [5] to extract an UPPAAL timed automata model from the SystemC TLM program. Towards this end, each component in the SystemC TLM program is mapped to one or more timed automata. These timed automata interaction captures the communication between SystemC TLM components. This translation is based on a set of rules that identify the semantics of b_transport and nb_transport as well as different primitives in the transaction.

For the second part, we consider timing faults and inject them into the extracted timed automata model. Some of the timing faults we consider include transactions (or parts of transactions) that are delayed or finish too quickly. The effect of these faults under LT and AT coding style is different. Specifically, in LT coding style, the effect of a delay would affect all subsequent transactions. However, the order of transactions will not change. By contrast, in AT coding style, such a fault would cause a change transactions to finish in a different order. Also, the choice of timed automata can cause modeling of these faults more complex. For example, in a timed automaton, a clock can either increase continuously (at the fixed rate for all clock variables) or be reset, but it cannot be set to an arbitrary value. This constraint needs to be satisfied in generating the UPPAAL model.

In the third part, we use UPPAAL toolset to verify the extracted model and analyze how the model behaves in the presence of timing faults. Specifically, we evaluate whether the UPPAAL model that is subjected to timing faults satisfies (1) deadlock-freedom, (2) timing constraints for the original program, and (3) relaxed timing constraints.

Contributions of the paper. We present:

- A set of transformation rules to extract the UUPPAAL timed-automata from SystemC TLM program.
- An approach for modeling timing faults in timed automata for SystemC TLM programs.
- An analysis of the fault-affected model for both loosely-timed and approximately-timed coding styles.

2 Model Extraction and Fault Analysis

Model Extraction for LT Coding Style. In a transaction using b_transport interface, we need to consider the timing since one of the sending arguments is *delay*. If the transaction is from the Initiator to the Target, this argument describes the point of time in the future where the communication actually starts, and if the transaction is from the Target to the Initiator, *delay* argument illustrates the ending time of the communication. Hence, we define a a synchronization channel in the UPPAAL model to synchronize the Initiator and Target of a transaction. Moreover, in order to guarantee deterministic execution and increase the timing accuracy, a SystemC TLM program that uses loosely-timed coding style benefits from explicit synchronization points by utilizing calls to *wait()* function. This function is a synchronization-on-demand method that yields the control to the SystemC scheduler. To transform this function into the UPPAAL model, we define a variable *global-clock* that plays the role of the global time in a SystemC TLM program. when the wait function is called, we add the *delay* arguments to the global-clock variable and reset the local clock.

Model Extraction for AT Coding Style. Having the approximately-timed coding style and nb_transport interface in a SystemC TLM program, we utilize our rules from [5] along with the ideas from [4] to extract the UPPAAL model.

Fault Modeling and Impact Analysis. Timing faults could perturb the state of a system to an illegitimate configuration. In other words, the timing faults cause an action/operation to be executed either too early or too late, and, as a result, the operation cannot be performed properly. We introduce a variable $delay_t$ to the fault-free UPPAAL model that can model both early and late timing problems. The maximum value of $delay_t$ (default value is 1) is identified by the designer. The UPPAAL model is further modified to non-deterministically increase the delay argument in b_transport by $delay_t$ in all processes. Regarding executing an action too early, we also introduce a new transition between two locations, let say L_i and L_j, that increases the clock variable by a value less than that of the original transition between L_i and L_j.

We considered the effect of such timing faults in models that utilize both the LT (loosely timed) and AT (approximately timed). Regarding LT model, we presented an approach for model extraction that extended the approach in [5] by modeling of timing faults. This approach also utilized UPPAAL (as opposed to Promela considered in [5]) so that accurate timing properties can be verified. Regarding AT model, we utilized the approach in [4]. We further simplified the resulting UPPAAL model by removing components unaffected by timing faults. This abstraction enabled verification of timing faults in the AT model.

We considered the faults when components execute too early or too late. To evaluate our approach, we considered on-chip memory mapped busses. In this example, both in LT and AT coding style, the time for verification in the presence of timing faults was small. However, there was a substantial gap between the verification time of LT and AT examples. For example, the average verification time for the faults where the Initiator executed too late in the LT model was 7.5 ms, whereas in the AT model was 17 s.

3 Future Work

One future work in this area is to automate the analysis of timing faults completely. Currently, our analysis of timing faults is rule-based and identifies the changes that need to be done to the original UPPAAL model. We are currently working on automating these rules to develop a fully automated approach that takes the description of the timing faults from users and automates the analysis.

References

1. D. E. Thomas, E. D. Lagnese, J. A. Nestor, J. V. Rajan, R. L. Blackburn, R. A. Walker, Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench, Kluwer Academic Publishers, Norwell, MA, USA, 1989.
2. Transaction-Level Modeling (TLM) 2.0 Reference Manual, `http://www.systemc.org/downloads/standards/`.
3. Open SystemC Initiative (OSCI): Defining and advancing SystemC standard IEEE 1666-2005, `http://www.systemc.org/`.
4. P. Herber, M. Pockrandt, S. Glesner, Transforming systemc transaction level models into uppaal timed automata, in: S. Singh, B. Jobstmann, M. Kishinevsky, J. Brandt (Eds.), MEMOCODE, IEEE, 2011, pp. 161–170.
5. A. Ebnenasir, R. Hajisheykhi, S. S. Kulkarni, Facilitating the design of fault tolerance in transaction level systemc programs, Theor. Comput. Sci. 496 (2013) 50–68.
6. R. Hajisheykhi, A. Ebnenasir, S. Kulkarni, Analysis of permanent, transient, and message faults in transaction level systemc programs, Tech. Rep. MSU-CSE-13-6, Department of Computer Science, Michigan State University, East Lansing, Michigan (July 2013).