Towards System-level Assertions for Heterogeneous Systems *

Muhammad Hassan[1], Thilo Vörtler[2], Karsten Einwich[2], Rolf Drechsler[3], and Daniel Große[4]

[1] Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
muhammad.hassan@dfki.de
[2] COSEDA Technologies GmbH, Dresden, Germany
{thilo.voertler,karsten.einwich}@coseda-tech.com
[3] Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
drechsle@informatik.uni-bremen.de
[4] Institute for Complex Systems, Johannes Kepler University, 4040 Linz, Austria
daniel.grosse@jku.at

Abstract. Heterogeneous systems are today *System-on-Chips* (SoCs) with integrated hardware and software, where the hardware consists of digital and *Analog Mixed-Signal* (AMS) parts. To manage the enormous verification challenges at the system-level, SystemC-based virtual prototyping is heavily employed. However, a practical system-level assertions library for heterogeneous systems is not available which prevents the full potential of AMS assertion-based verification from being exploited.
In this paper, we present a system-level assertions library with an intuitive API, full SystemC compatibility, software and transaction support, and heterogenous characteristics, all mandatory to specify complex AMS behavior. We demonstrate our prototypical implementation for an industrial case-study using ARM fast models, a temperature software, environment models and control software and assertions. We will make our system-level assertions library available as open source.

Keywords: System-level Assertions · SystemC/AMS · Functional Verification · Assertions library · Virtual prototyping · Hetereogeneous systems · ARM Fast Models

1 Introduction

Driven by growth opportunities in various application domains, e.g. *Internet-Of-Things* (IOT), many semiconductor vendors are shifting their focus towards a more integrated solution of high-performance *Analog/Mixed-Signal* (AMS) designs. Due to this industry shift, most *System-On-Chips* (SOCs) today are AMS containing analog sensors, mixed-signal converters, and digital processors running *Software* (SW) on top, tightly integrated on a single die. One characteristic

* This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project AUTOASSERT under contract no. 16ME0117.

of such SOCs is that each subsystem interacts simultaneously with each other by internal connections and reacts to inputs coming from outside. Digital systems behavior usually exhibits discrete changes in time and value, whereas analog circuits usually exhibit continuous changes. While this shift has resulted in high-performance and low-area devices, it has significantly increased the efforts required to develop and verify these highly complex devices and achieving the required *Time-To-Market* (TTM) simultaneously. Nowadays, *Assertion-Based Verification* (ABV) in combination with coverage analysis [31, 19, 20] and constrained randomization techniques [44, 18] is widely used to perform functional verification of digital designs at *Register Transfer Level* (RTL). ABV defines temporal properties in order to verify the functional correctness of the design with respect to expected behaviors. Consequently, the bugs are found at their source. Furthermore, design observability and controllability is also improved. Applying the ABV methodology to AMS designs can bring the same benefits that the digital design community has enjoyed. However, late availability of RTL in the design process exacerbates the situation.

In this regard, the emergence of *Virtual Prototypes* (VPs) at the abstraction of *Electronic System Level* (ESL) has modernized the design and verification of AMS SOCs in many ways [11, 28, 36]. Essentially, a VP is a software simulation model of the entire *Hardware* (HW) platform, created by composing models of the individual *Intellectual Property* (IP) blocks (i.e. Instruction Set Simulators, bus and peripheral models, etc.). For this purpose, the C++-based system modeling language SystemC together with *Transaction Level Modeling* (TLM) techniques [22] and mixed-signal extension SystemC/AMS [4] are being heavily used in industrial practice [11, 29, 28, 3, 36, 20]. Overall, the adoption of VPs has led to significant improvements on the design and verification of SOCs. Because of earlier availability and significantly faster simulation speed as opposed to RTL, the VPs enable HW/SW co-design and verification very early in the development flow. Serving as reference for (early) embedded SW development and HW verification, the functional correctness of VPs is very important. Hence, a whole VP as well as its individual components are subjected to rigorous verification.

However, one of the main challenges is the availability of a practical assertions library for system-level design verification which enables ABV methodologies. When speaking of unavailability, we also broadly include advanced testbench concepts based on the *Universal Verification Methodology* (UVM), or in the future even more abstract based on *Portable Stimulus Specification* (PSS). Regardless of the specific solution, a system-level assertions library is missing which satisfies the following: 1) expressiveness to represent complex behaviors of heterogeneous systems, 2) compatibility to SystemC, TLM, and SystemC/AMS, 3) capture of complex analog-digital interactions, and 4) integration of complex heterogeneous characteristics like continuous time, frequency analysis etc.

Contribution: In this paper we present a system-level assertions library for heterogeneous systems, an advanced *ABV* environment for SystemC, TLM, and its mixed-signal extension SystemC/AMS. To overcome the limitations of

state-of-the art libraries (see Section 2 for more details), the proposed SystemC assertions library provides the following features:

- New assertions specification API: An intuitive, user-friendly, and expressive *Application Programming Interface* (API) to specify complex behaviors of non-trivial heterogeneous systems has been developed.
- Compatibility: The library is compatible with SystemC and its extensions, TLM and SystemC/AMS.
- Complex behaviors: Various complex behaviors can be captured like, 1) complex analog-to-digital, 2) digital-to-analog, 3) digital-to-digital, and 4) analog-to-analog.
- SW and TLM Support: The assertions library supports the checking of TLM interface and SW/HW interactions.
- Heterogeneous characteristics: The library integrates heterogeneous characteristics like continuous time, frequency analysis, slopes, equations, attenuations, *Differential Algebraic Equations* (DAE), digital signals, temporal logic, variables, and events. These characteristics are necessary for expressing complex properties.
- Improved usability: Debugging of failed assertions is supported.

Considering all these features, we develop a new system-level assertions library for bridging the gap of ABV for heterogeneous systems. The running example and experiments on a real-world model of ARM V8 based CPU using ARM Fast Models demonstrates the capabilities of the library to improve the system verification in a significant way.

The paper is organized as follows: Section 2 gives a survey of current approaches concerning heterogeneous/AMS verification. Section 3 discusses the running example along with assertions to setup the environment. Section 4 describes our contribution, the implementation, and discusses the approach. This includes syntax and semantics of the system-level assertions library. In Section 5 we demonstrate the benefits of our methodology with experiments. Finally, we conclude and mention future work in Section 6.

2 Related Work

SystemC is widely used for system-level design and verification, however, it still lacks native temporal assertions support. Several approaches have been proposed for digital SystemC-based models/VPs. Besides basic work on the temporal language itself [40], these approaches can be divided into two categories, formal assertion-based verification (e.g., [12, 17, 42, 24, 41, 16, 26, 7]) and simulation-based verification (for example, [6, 13, 35, 9, 39, 14, 5]). The formal approaches aim to fully explore the state space based on abstract representations of system-level models. However, these approaches typically run into the state space explosion problem. Furthermore, the aforementioned simulation-based methods only consider purely digital models.

In [35, 6, 9] new approaches for transaction level assertions are introduced. However, in [35] transactions are mapped to signals and therefore the approach is restricted only to transactions which are invoked by suspendable processes. In [6] transactions are recorded and written into a trace to do post processing. Trace based assertion checking however requires that everything to be recorded must be annotated in the code and the creation of simulation data bases can become very resource intensive.

Various works have also been presented for the specification and verification of analog circuits [38, 27, 37, 30, 45, 34]. Here, too, a distinction is made between formal and simulation-based methods. One focus of the work was in particular to develop suitable extensions for the specification of assertions. It should be noted, however, that the aforementioned works only target analog components and usually only address the implementation level. The overall heterogeneous systems (incl. SW) and environment considered here are not supported.

In the area of digital HW/SW co-design and verification, various formal approaches have been proposed, for example [15, 43, 33]. However, these so far assume only implementation-level descriptions for the hardware part (e.g., in Verilog or VHDL). Furthermore, due to the huge state spaces in analog domain, only small problems can be handled. Recently, abstraction techniques have been developed and the hardware parts are abstracted to C level [32, 21]. However, these methods consider only pure digital designs.

Heterogeneous characteristics like continuous time, frequency analysis, slopes, equations, attenuations, DAE, digital signals, temporal logic, variables, and events are insufficiently integrated in all known specification languages. However, these characteristics in combination with a special time definition are necessary for expressing complex properties. Therefore our work considers all these conditions to develop a new system-level assertions library for bridging the gap of ABV for heterogeneous designs.

3 Preliminaries

3.1 Assertion Based Verification

ABV is an established technique used nowadays to verify SOCs [10]. To enable ABV, a language is required based on the general notion of *Property Specification Language* (PSL) [23], *Linear Temporal Logic* (LTL), Finite LTL (FLTL), or Computation Tree Logic (CTL) [25]. Based on the specification assertions (properties) are typically manually created and capture the design intent. The basic function of an assertion is to specify a set of behaviors that is expected to be true for a given *Design Under Verification* (DUV). Assertions are included in the DUV via monitors and they compare the temporal behavior of the assertions against the DUV during simulation. Assertions are used in the validation environments of TLM, RTL, and gate level and offer the following advantages, 1) detect design errors at their source and increase observability, 2) actively monitor a design to ensure correct functional behavior, and 3) can be used for functional

and formal verification. The widely used assertions library for RTL, *SystemVerilog Assertion* (SVA) [8] unifies simulation and formal verification semantics to drive the design for verification methodology. It takes a layered approach to define the properties of the DUV. More precisely, properties are composed of four layers: 1) the **Boolean layer** consists of propositions and Boolean connectives, 2) the **sequence layer** adds operators for temporal reasoning to the Boolean layer. 3) the **property layer** defines operations on sequences, and 4) the **verification layer** provides indicators for the verification tools on how to apply the properties. Most often assertions use implication operators which define some specific sequence of events (known as *antecedent*) which should occur before another sequence of events (known as *consequent*) should occur.

The first three layers define the actual property (intended or error state) that relates to parts of the DUV whereas the fourth layer is used to control the high-level behavior of the verification tools.

3.2 System-level Running Example

For brevity, we refrain from giving a proper introduction to SystemC, TLM, and SystemC/AMS. Instead, we present here a heterogeneous system as a running example (Fig. 1) that will be used to showcase the main ideas of our approach throughout this paper. The SystemC, TLM, and SystemC/AMS constructs and semantics necessary to understand the example will be explained as needed. The running example models a temperature control system covering multiple domains, i.e. SW, digital HW, and analog behavior. The system is modeled in SystemC/AMS using different *Models of Computation* (MoC), in particular *Timed Data Flow* (TDF) and *Electrical Linear Networks* (ELNs). The overall system as shown in Fig. 1 consists of the following components:

- an ARM V8 based CPU using ARM Fast Models implemented as SystemC TLM [2] with Linux operating system and SW running on top,
- four ADT7420 temperature sensors implemented as SystemC/AMS TDF and discrete event model [1],
- an *Advanced Microcontroller Bus Architecture* (AMBA) bus that acts as a bridge device to connect temperature sensors and ARM processor (created in SystemC TLM) – (*COS_AMBA_DEVICE* in Fig. 1),
- an environment model (*Thermal_Network*) that builds 3 connected rooms and an ambient temperature modeled as a sinus (*SIN_SRC_TDF*), i.e. each sensor senses a different temperature (implementation as SystemC/AMS ELN and discrete event model), and finally
- a heater model implemented as SystemC/AMS ELN that can be used to increase the temperature.

The communication between SW running on the ARM8 and the connected sensors is done via registers connected to the bus of the processor. The SW configures the sensors by writing to addresses on the bus, which in turn creates TLM transactions. These TLM transactions are written into the corresponding registers of the ADT7420 sensors. The AMBA bus (*COS_AMBA_DEVICE*) also

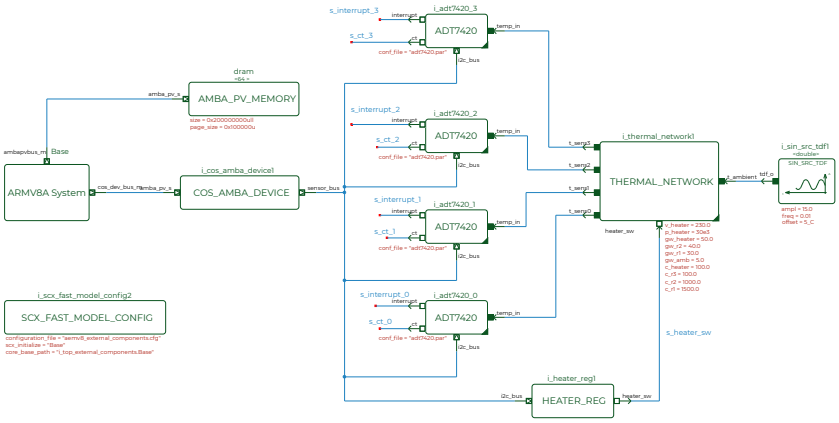

Fig. 1. Schematic of running example: temperature control system

translates the AMBA-PV transactions used by ARM Fast models. Additionally, I^2C transactions of the sensor model are also translated. To showcase the features of proposed system-level assertions library, the running example considers the following scenario for demonstration purposes:

- booting a Linux operating system on the ARM processor,
- a control SW is executed on top of Linux. The control SW continuously measures (monitors) the temperature sensor output,
- if the SW detects that the temperature value falls below a programmed threshold value, it switches the heater to *ON* state,
- otherwise, when the temperature exceeds a certain programmed threshold, the heater is switched to *OFF* state.

3.3 Assertions for System-level Running Example

A lot of assertions can be defined for the running example introduced in Section 3.2. However, for the purpose of demonstrating the features of the proposed system-level assertions library, we focus on only one. The concrete assertion states that:

- When the temperature of Room 1 *t_r1* (SystemC TDF signal) is above the threshold *t_threshold* (SW-controlled TLM register value), the heater has to be switched off (*heater_sw*) within 1 ms.

How this heterogeneous assertion can be expressed in our proposed assertions library can be seen in Listing 1.1. Please note, we introduce all ingredients (in particular, API, layers, ...) from the users-perspective for the proposed system-level assertions library in the next sections.

```
1  auto heater_off = (t_r1 > t_threshhold) ->* (true | delay(1_SC_MS)
       | (heater_switch==false));
2  heater_off.default_sampling(1_SC_MS);
```

Listing 1.1. Concrete assertion for temperature control system example

4 System-Level Assertions Library for Heterogeneous Systems

In this section, we introduce the proposed system-level assertions library and its components for bridging the gap of ABV for heterogeneous systems. First, we provide a brief overview of the library. Then, we describe the intuitive API and the layered architecture of the assertions library in detail while always providing an example.

4.1 Overview

The system-level assertions library is developed with an intuitive, user-friendly, and an expressive API. As a result, complex behaviors of heterogeneous systems can be captured easily. These behaviors are not only limited to events taking place at one point in time in one domain, rather also temporal behaviors across different domains, e.g., TLM and analog. To enable the API expressiveness, a layered architecture inline with SVA layered architecture [8] is used, i.e., boolean layer, sequence layer, property layer, and verification layer. At the back-end, first the assertion is divided into different layers and expressions, then multiple SystemC processes are spawned to monitor the signals and events specified in the expressions. The library uses linear time model where the assumption is that the time is linear. Each assertion is synchronized to the sampling ticks (notion of discrete time) of DUV as defined by SystemC/AMS semantics, unless specified. The assertion is evaluated at each sampling tick. If the specified expressions evaluate to *true*, the assertion is satisfied. Additionally, the complete trace of assertion evaluation is displayed to the verification engineer.

In the following sections, the components of system-level assertions library are explained in detail.

4.2 Application Programming Interface

The API of the library is designed to enable the expressiveness required for specifying cross-domain behaviors, e.g., TLM and analog. Hence, dedicated functions like *delay(...)*, *repeat(...)*, *default_sampling(...)* etc are defined to specify the behaviors and make the library user-friendly. Additionally, operators (e.g., pipe (|), ->*) are introduced to enable specification of sequences in SystemC.

The concrete assertion (specified in Listing 1.1) is interpreted in light of the proposed API as follows: an assertion property *heater_off* is created. The

property joins 2 sequences via an overlapping implication operator (->*). The sequences are, 1) antecedent – (t_r1 > t_threshhold), 2) consequent – *(true | delay(1_SC_MS) | (heater_switch==false))*. The sequences comprise of 4 boolean expressions in total, 1) *(t_r1 > t_threshhold)*, 2) *true*, 3) *delay (1_SC_MS)*, 4) *(heater_switch==false)*. Furthermore, the sampling time of the assertion is written in Line. 2, i.e., *1_SC_MS*.

4.3 Boolean Layer

The boolean layer describes the behaviors of primitive elements relative to each other at a particular point in time. The primitive elements in our proposed library are SystemC events, variables, SystemC/AMS signals. These primitive elements are related using arithmetic, logical, or relational operators. Consequently, they form an expression, e.g., a relational expression. In Listing 1.1 the expression *(t_r1 > t_threshhold)* compares an analog signal *t_r1* with a digital threshold value *t_threshhold* stored in TLM register. If the relational condition is satisfied, the expression is evaluated to *true*. A non-comprehensive list of boolean expressions is shown in Table 1.

Table 1. Non-comprehensive list of supported boolean expressions by system-level assertions library

Operator	Name	Data type
+= -= /= *= &= |=	Binary assignment operators	int, double
< <= > >=	Binary relational operators	int, double
+ − ∗ /	Binary arithmetic operators	int, double
&& || == !=	Binary logical operators	int, double
+ - ! ++ −	Unary operators	int, double

4.4 Sequence Layer

The sequence layer builds on top of boolean layer to specify the temporal relationship between primitive elements (boolean expressions) over time. The sequence layer also specifies sequences as either a combination of simpler sequences using sequence operators or as basic boolean expressions correlated by events. The proposed API introduces the pipe operator (|) to represent the continuity of a sequence. This increases readability as well as user-friendliness of the assertion property. Additionally, the API introduces *delay(...), repeat(...)* operators to specify temporal assertions. As a result, a sequence can comprise of delay operators (Section 4.4), boolean expressions, and event expressions. To determine a match of the sequence, the boolean expressions are evaluated at each successive sample tick, defined by a sampling event (SystemC/AMS sampling points) that gets associated with the sequence. If all expressions of the sequence are true,

then a match of the sequence occurs. For example, the assertion in Listing 1.1 has the expressions

(true | delay(1_SC_MS) | (heater_switch == false))

The expressions are interpreted as follows: a signal is asserted – *true*, followed by a delay operator – *delay(1_SC_MS)*, and after the delay of 1 ms, the expression *(heater_switch == false)* is evaluated. The sequence returns *true* only if all the expressions evaluate to *true*. A non-comprehensive list of supported sequence operators is shown in Table 2.

Table 2. Non-comprehensive list of supported sequence operators by system-level assertions library

Operator	Description
delay	Specifies delay from current sampling point until the next
and	Sequence *and* operation
or	Sequence *or* operation
repeat	Repetition operator

Delay Operator The system-level assertions library introduces delay operator – *delay(delay_cycles)* and *delay(min_delay_cycles, max_delay_cycles)* which takes delay time as input. The function of delay operator is to create a relationship between boolean expressions over a period of time or between the given time constraints.

Repeat Operator The library also introduces repeat operator – *repeat(value)* and *repeat(min_value, max_value)* which takes a repetition value as input for how many times the sequence should be repeated. It helps in cases when a certain set of expressions are expected to be true over multiple time points.

Sequence "and/or" Operators The system-level assertions library introduces the sequence *"and/or"* operators. The sequences are evaluated in parallel. In case of *"and"* operator, if one sequence evaluates to *"false"*, the evaluation stops and the assertion fails. On the other hand, in case of *"or"* operator, the library waits for all sequences to be evaluated.

4.5 Property Layer

The property layer allows for more general behaviors to be specified, i.e., specification of properties as either a combination of simpler properties using property operators or as an implication built up from several sequences. In particular, properties allow users to invert the sense of a sequence (e.g., when the sequence

should not happen), disable the sequence evaluation, or specify that a sequence be implied by some other occurrence. The properties and their respective sequences (including boolean expressions) are evaluated on each sampling event (sampling tick) of the system's default sampling time, unless specified. In this concrete assertion (defined in Listing 1.1), the property sampling time is set to 1 ms (*heater_off.default_sampling (1_SC_MS)*). As a result, the assertion property in Listing 1.1 is evaluated every ms. The property layer supports implication operators, *"not"*, and *"and/or"* operators.

Implication Operator An implication refers to a situation in which in order for a behavior to occur, a preceding sequence must have occurred. This preceding sequence in this case is known as *antecedent*. The succeeding behavior is known as *consequent*. Evaluation of an implication starts through repeated attempts to evaluate the antecedent. When the antecedent succeeds, the consequent is required to succeed for the property to hold. Thus, in other words, an antecedent sequence implies a consequent property expression, as follows

$$antecedent -> * consequent$$
$$where -> * = overlapping\ implication\ operator$$

Non-overalapping Implication The *delay(...)* operator is used to implement non-overlapping implication.

Overlapping Implication ->* In the system-level assertion library we introduce an overlapping implication operator (->*). This means that if the antecedent sequence is evaluated to *true*, the consequent sequence is evaluated at the same tick.

As shown in Listing 1.1, if the expression *(t_r1 > t_threshhold)* is true, the sequence *(true |delay(1_SC_MS) |(heater_switch==false))* should be true in next sampling ticks. A non-comprehensive list of supported property operators is shown in Table 3

Table 3. Non-comprehensive list of supported property operators by system-level assertions library

Operator	Description
Not	the evaluation of the property returns the opposite of the evaluation of the underlying property expression
and	The property evaluates to true if, and only if, both property expression 1 and property expression 2 evaluate to true.
or	The property evaluates to true if, and only if, at least one of property expression 1 and property expression 2 evaluates to true.

4.6 Verification Layer

The verification layer specifies which properties are to be asserted or covered. This layer always associates properties with corresponding verification directives. A verification directive can be parameterized by the severity level and an info string; further on it can be specified if the property should be asserted, covered or both. The proposed library supports only *assert* at the moment.

5 Experiments

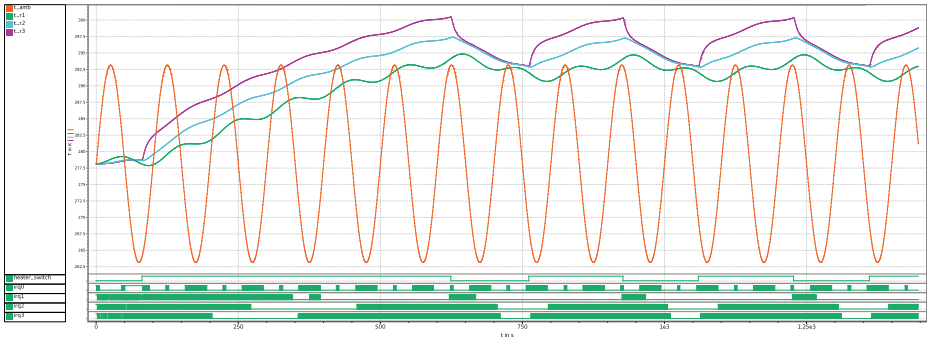

Fig. 2. Simulation results running the temperature control SW

This section describes the experimental evaluation on a real world model integrating an ARM V8 CPU via ARM Fast Models (as described in Section 3.2). Fast Models are accurate, flexible programmer's view models of ARM IP, allowing one to develop software such as drivers, firmware, OS and applications prior to silicon availability. They allow full control over the simulation, including profiling, debug and trace. As mentioned, the complete model is implemented as a mixture of a SystemC TLM model and a SystemC/AMS model.

Several assertions were created to verify the behavior of temperature control system. The behaviors to verify included but not limited to: 1) SW and TLM interactions, 2) analog and TLM interactions, 3) analog-digital, 4) digital-analog, 5) digital events, and 6) analog-analog interactions etc. In the following, we detail the results of the concrete assertion from Listing 1.1.

Partial simulation results of the temperature control system SW are shown in Fig. 2. The *orange* sinus signal is the ambient temperature (*SIN_SRC_TDF*) which oscillates between 262 K and 293 K. The *green* waveform signal (*t_r1*) is the temperature of room 1. The *blue* waveform signal (*t_r2*) is the temperature of room 2. The *purple* waveform signal (*t_r3*) is the temperature of room 3. At the bottom of Fig. 2, digital signals – *heater_switch* and interrupts (irq0-irq3) from temperature sensors are displayed.

After booting the Linux OS (approx. 30s) the control SW gets started. The heater (*heater_switch*) gets turned on as the temperature in room one (*t_r1*) is below the minimum temperature of 292 K. It can be seen how the temperature slowly increases in all rooms. When the temperature is above the maximum threshold of 294.15 K the heater gets turned off. As a consequence, the room temperatures start to decrease. The sensors have been programmed to generate an interrupt whenever the temperature is above or below a threshold value (stored in register).

We could see the assertion was satisfied throughout the simulation. However, if we decreased the *delay(...)* from 1 ms to a smaller value, the assertion was always violated. This is expected and in accordance with the specifications. They require that the *heater_switch* should be turned off within 1 ms after the threshold temperature is crossed. The reason for 1 ms is because of the inherent delays due to reading and writing of registers in different connected devices, and can be summarized as follows:

- the temperature sensor senses the temperature,
- the sensed temperature is written into the register,
- SW reads the temperature from the ARM processor,
- SW checks whether the sensed temperature value is above the threshold value,
- and writing the heater switch control register depending on the comparison result.

Hence, using the proposed intuitive system-level assertion library, it is possible to check complex behaviors of the heterogeneous systems, e.g., digital, analog and SW behavior.

6 Conclusion

In this paper, we presented a practical system-level assertions library for heterogeneous systems. The library comprises of an intuitive and user-friendly API and offers full compatibility with SystemC, TLM, and SystemC/AMS. As a result, the library supports specification of SW, TLM, and complex interactions, all necessary to represent complex AMS behavior. The system-level assertions library prototype was used to verify the industrial model using ARM Fast models, a temperature control system SW, environment models, temperature sensors, and assertions. We will make our system-level assertions library prototype available as open source.

References

1. Analog Devices ADT7420 Data Sheet Rev. A (Sep 2017), https://www.analog.com/en/products/adt7420.html
2. ARM Fast Models Version 11.17 User Guide (Feb 2022), https://developer.arm.com/documentation/100965/1117/

3. Barnasconi, M., Adhikari, S.: ESL design in SystemC AMS: Introducing a top-down design methodology for mixed-signal systems. In: DAC. pp. 1–5 (2017)
4. Barnasconi, M., Grimm, C., Damm, M., Einwich, K., Louërat, M., Maehne, T., Pecheux, F., Vachoux, A.: SystemC AMS extensions user's guide. Accellera Systems Initiative (2010)
5. Bombieri, N., Fummi, F., Guarnieri, V., Pravadelli, G., Stefanni, F., Ghasempouri, T., Lora, M., Auditore, G., Marcigaglia, M.N.: Reusing rtl assertion checkers for verification of systemc tlm models. Journal of Electronic Testing **31**(2), 167–180 (2015)
6. Chen, X., Luo, Y., Hsieh, H., Bhuyan, L., Balarin, F.: Assertion based verification and analysis of network processor architectures. Design Automation for Embedded Systems **9**(3), 163–176 (2004)
7. Cimatti, A., Narasamdya, I., Roveri, M.: Software model checking SystemC. TCAD **32**(5), 774–787 (2013)
8. Committee, D.A.S., et al.: Ieee standard for systemverilog unified hardware design, specification, and verification language standard ieee 1800. http://www. edastds. org/sv/ (2005)
9. Ecker, W., Esen, V., Steininger, T., Velten, M., Hull, M.: Implementation of a transaction level assertion framework in SystemC. In: DATE. pp. 894–899 (2007)
10. Foster, H.D., Krolnik, A.C., Lacey, D.J.: Assertion-based design. Springer Science & Business Media (2004)
11. Grimm, C., Barnasconi, M., Vachoux, A., Einwich, K.: An introduction to modeling embedded analog/mixed-signal systems using SystemC AMS extensions. In: DAC. vol. 23 (2008)
12. Große, D., Drechsler, R.: Formal verification of LTL formulas for SystemC designs. In: ISCAS. pp. V:245–V:248 (2003)
13. Große, D., Drechsler, R.: Checkers for SystemC designs. In: MEMOCODE. pp. 171–178 (2004)
14. Große, D., Groß, M., Kühne, U., Drechsler, R.: Simulation-based equivalence checking between SystemC models at different levels of abstraction. In: GLSVLSI. pp. 223–228 (2011)
15. Große, D., Kühne, U., Drechsler, R.: Hw/sw co-verification of embedded systems using bounded model checking. In: GLSVLSI. pp. 43–48 (2006)
16. Große, D., Le, H.M., Drechsler, R.: Proving transaction and system-level properties of untimed SystemC TLM designs. In: MEMOCODE. pp. 113–122 (2010)
17. Habibi, A., Tahar, S.: Assertion and model checking of SystemC. In: North American SystemC Users Group Meeting, San Diego, California, USA (2004)
18. Haedicke, F., Le, H.M., Große, D., Drechsler, R.: CRAVE: An advanced constrained random verification environment for SystemC. In: SoC. pp. 1–7 (2012)
19. Hassan, M., Große, D., Le, H.M., Drechsler, R.: Data flow testing for SystemC-AMS timed data flow models. In: DATE. pp. 366–371 (2019)
20. Hassan, M., Große, D., Vörtler, T., Einwich, K., Drechsler, R.: Functional coverage-driven characterization of RF amplifiers. In: FDL. pp. 1–8 (2019)
21. Huang, B.Y., Ray, S., Gupta, A., Fung, J.M., Malik, S.: Formal security verification of concurrent firmware in SoCs using instruction-level abstraction for hardware. In: DAC. pp. 1–6 (2018)
22. IEEE Std. 1666: IEEE Standard SystemC LRM (2011)
23. IEEE Std. 1850: IEEE Standard for Property Specification Language (PSL) (2005)
24. Karlsson, D., Eles, P., Peng, Z.: Formal verification of SystemC designs using a petri-net based representation. In: DATE. pp. 1228–1233 (2006)

25. Kropf, T.: Introduction to formal hardware verification. Springer Science & Business Media (1999)
26. Lämmermann, S., Ruf, J., Kropf, T., Rosenstiel, W., Viehl, A., Jesser, A., Hedrich, L.: Towards assertion-based verification of heterogeneous system designs. In: DATE. pp. 1171–1176 (2010)
27. Lämmermann, S., Weiss, R., Ruf, J., Kropf, T., Rosenstiel, W., Jesser, A., Hedrich, L.: An assertion-based verification methodology for SystemC-AMS designs. In: The 15th Workshop on Synthesis And System Integration of Mixed Information Technologies. pp. 434–439 (2009)
28. Lora, M., Vinco, S., Fraccaroli, E., Quaglia, D., Fummi, F.: Analog models manipulation for effective integration in smart system virtual platforms. TCAD **37**(2), 378–391 (2018)
29. Ma, K., Van Leuken, R., Vidojkovic, M., Romme, J., Rampu, S., Pflug, H., Huang, L., Dolmans, G.: A precise and high speed charge-pump pll model based on systemc/systemc-ams. International Journal of Electronics and Telecommunications **58**, 225–232 (2012)
30. Maler, O., Ničković, D.: Monitoring properties of analog and mixed-signal circuits. International Journal on Software Tools for Technology Transfer **15**(3), 247–268 (2013)
31. Mehta, A.B.: System Verilog Assertions and Functional Coverage: Guide to Language, Methodology and Applications. Springer Nature (2019)
32. Mukherjee, R., Purandare, M., Polig, R., Kroening, D.: Formal techniques for effective co-verification of hardware/software co-designs. In: DAC. pp. 1–6 (2017)
33. Nguyen, M.D., Wedler, M., Stoffel, D., Kunz, W.: Formal hardware/software co-verification by interval property checking with abstraction. In: Proceedings of the 48th Design Automation Conference. pp. 510–515 (2011)
34. Ničković, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: Amt 2.0: qualitative and quantitative trace analysis with extended signal temporal logic. International Journal on Software Tools for Technology Transfer **22**(6), 741–758 (2020)
35. Niemann, B., Haubelt, C., et al.: Assertion-based verification of transaction level models. In: MBMV. pp. 232–236. Citeseer (2006)
36. Pêcheux, F., Grimm, C., Maehne, T., Barnasconi, M., Einwich, K.: SystemC AMS based frameworks for virtual prototyping of heterogeneous systems. In: ISCAS. pp. 1–4 (2018)
37. Radojicic, C., Grimm, C., Schupfer, F., Rathmair, M.: Verification of mixed-signal systems with affine arithmetic assertions. VLSI Design (2013)
38. Steinhorst, S., Hedrich, L.: Model checking of analog systems using an analog specification language. In: DATE. pp. 324–329 (2008)
39. Tabakov, D., Vardi, M.: Monitoring temporal SystemC properties. In: MEMOCODE. pp. 123–132 (2010)
40. Tabakov, D., Vardi, M., Kamhi, G., Singerman, E.: A temporal language for SystemC. In: FMCAD. pp. 1–9 (2008)
41. Vardi, M.Y.: Formal techniques for SystemC verification. In: DAC. pp. 188–192 (2007)
42. Weiss, R.J., Ruf, J., Kropf, T., Rosenstiel, W.: Efficient and customizable integration of temporal properties into SystemC. In: Applications of Specification and Design Languages for SoCs, pp. 101–114. Springer (2006)
43. Xie, F., Liu, H.: Unified property specification for hardware/software co-verification. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007). vol. 1, pp. 483–490. IEEE (2007)

44. Yuan, J., Pixley, C., Aziz, A.: Constraint-based Verification. Springer (2006)
45. Zivkovic, C., Grimm, C., Olbrich, M., Scharf, O., Barke, E.: Hierarchical verification of AMS systems with affine arithmetic decision diagrams. TCAD **38**(10), 1785–1798 (2019)