Assertion-Based Monitoring in Practice – Checking Correctness of an Automotive Sensor Interface

Thang Nguyen[1,⋆] and Dejan Ničković[2]

[1] Infineon Technologies AG, Austria
Thang.Nguyen@infineon.com
[2] AIT Austrian Institute of Technology GmbH, Vienna, Austria
dejan.nickovic@ait.ac.at

Abstract. In this paper, we evaluate the assertion-based monitoring technology for mixed-signal systems by applying it to real-world case study from the automotive domain.

We first motivate the case study by presenting the state-of-the-practice verification and validation work-flow typically used in the automotive industry. We identify the shortcomings of this work-flow, and propose a more rigorous and automated methodology based on monitoring correctness of simulated mixed signal designs with respect to assertions, which formalize in Signal Temporal Logic (STL) the requirements from the design specification.

We apply the assertion-based monitoring framework for mixed signal designs to check the correctness of Distributed System Interface (DSI3) in a modern airbag system-on-chip application. We present all the relevant steps in our proposed work-flow, evaluate the results and discuss the framework's benefits as well as its identified missing features.

1 Introduction

A modern car is a system-of-systems (SoS) that merges a number of embedded elements that are often developed independently. The systems in a car are heterogeneous, combining digital controllers with analog sensors and actuators. They interact with their physical environment and are interconnected through the vehicle physics, as well as communication protocols. This results in complex interactions generating emergent behaviors that are not predictable in advance. Many components in a car, such as the airbag systems, are *safety critical*. Hence, correct system integration in the automotive domain is crucial to achieve high standards with respect to safety.

Due to the heterogeneity and the complexity of components and sub-systems in modern cars, verification and validation (V&V) poses a major challenge in the

⋆ The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement Nr. 295311 and the Austrian Research Promotion Agency FFG under the program "Forschung, Innovation und Technologie fr Informationstechnologien (FIT-IT).

F. Lang and F. Flammini (Eds.): FMICS 2014, LNCS 8718, pp. 16–32, 2014.

automotive domain and represents the main bottleneck in the design process. Verification by simulation and manual testing are the dominant methods used in the V&V practice of the automotive industry. However, these techniques have the weakness of being ad-hoc, inefficient and prone to human errors.

The research community has investigated a number of approaches that address V&V issues for mixed-signal systems. Formal verification of systems combining continuous and discrete dynamics has been mainly studied by the *hybrid systems* [18,2] community. It consists in computing over-approximations of reachable sets of states of the circuit, modeled as a hybrid automaton (differential equations with mode switching). Despite the important progress achieved in this research field in recent years [13], such technique [6,14,17,24,1] still cannot scale up to the size and complexity of transistor-level circuit models. In addition to hybrid system verification, there are other orthogonal analytical approaches to study similar systems. For instance, static analysis and abstract interpretation were used to develop a framework for inferring continuous time properties of systems consisting of synchronous components that interact by quasi-synchronous composition [5].

Assertion-based monitoring is a promising technology for verification of analog and mixed-signal (AMS) designs, i.e. designs that consist of interacting digital and analog components. It successfully exports some well-established ingredients from digital verification to the AMS domain, while retaining the relative simplicity and scalability of the simulation-based verification. In essence, assertion-based monitoring frameworks consist of an assertion language used to formalize the requirements that describe the correct interaction between analog and digital components, including timing constraints due to the communication delays. The formal assertions are then automatically translated into *monitors*, programs that read simulation traces of the design-under-test and check for the assertion satisfaction/violation.

Signal Temporal Logic (STL) [19,20] is an assertion language extending Linear Temporal logic (LTL) [22]. LTL enables declarative, formal and compact specification of reactive system requirements. Its original use was for evaluating sequences of states and events in digital systems. A typical property stated in temporal logic is `always (req -> eventually! ack)`. This property says that it is always the case that a request `req` eventually triggers an acknowledgment `ack`. STL extends LTL to specification of properties involving both digital and real-valued variables defined over dense time. Offline monitoring of STL was implemented in the tool AMT [21]. The monitoring flow based on using STL for formalizing assertions and monitoring them with AMT is depicted in Figure 1. This specification language has been successfully used in the past for monitoring in various application domains, such as analog circuits [16], biochemical reactions [7], synthetic biological circuits [4] and music [11]. STL has also been extended in several other directions. In [11], the authors developed a first attempt of time-frequency logic-based (TFL) specification, and successfully applied it to detect music patterns. TFL expresses frequencies as atomic predicates (using sliding FFT to evaluate the intensity of the signal around a frequency) and time

using intervals and the classic temporal operators. The classic qualitative semantics of STL was recently extended with more powerful and precise notions of quantitative semantics [12,10,9] (or robustness degree), providing a real value measuring the level of satisfaction or violation for a trajectory of the property of interest. Several tools, such as BIOCHAM [23], S-TaLiRo [3] and Breach [8], are available to perform robustness analysis on the time series collected in wet-lab experiments or produced by simulation-based techniques.

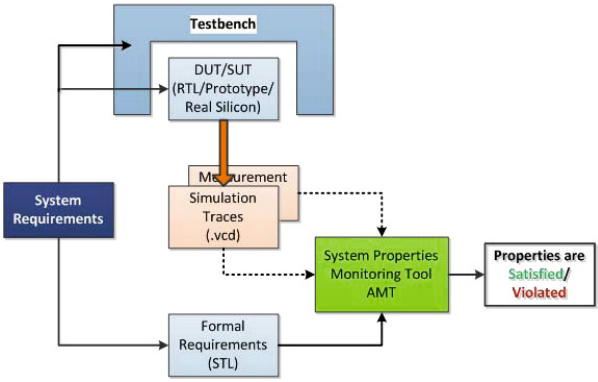


Fig. 1. Assertion-based monitoring flow with STL assertion language and AMT tool

In this work, we apply the assertion-based monitoring framework from Figure 1 to check the correctness of a sophisticated automotive sensor interface integration in a modern system-on-chip (SoC) airbag system, developed by Infineon Austria AG. The correct integration of the SoC with its sensor interface is specified in the Distributed System Interface (DSI3) protocol standard [15]. We present the work-flow of the case study in which we use STL to formalize DSI3 requirements and AMT tool to monitor the simulation traces. We evaluate the case study results and discuss the lessons that we learned regarding the applicability of this approach in industry.

2 Verification Flow in the Automotive Domain – State-of-the-Practice

Figure 2 illustrates the state-of-the-practice verification work-flow by Power Train and Safety department at Infineon Technology Austria AG. The work-flow describes as well collaboration between Tier-1 (system developer and integrator) and Tier-2+ (HW - Hardware and SW Software element developers) teams. The work-flow starts with the requirements and specifications phase at the Tier-1 level. In this step system functionalities and related HW/SW components are defined. The HW requirements are provided to Tier-2 supplier, e.g.

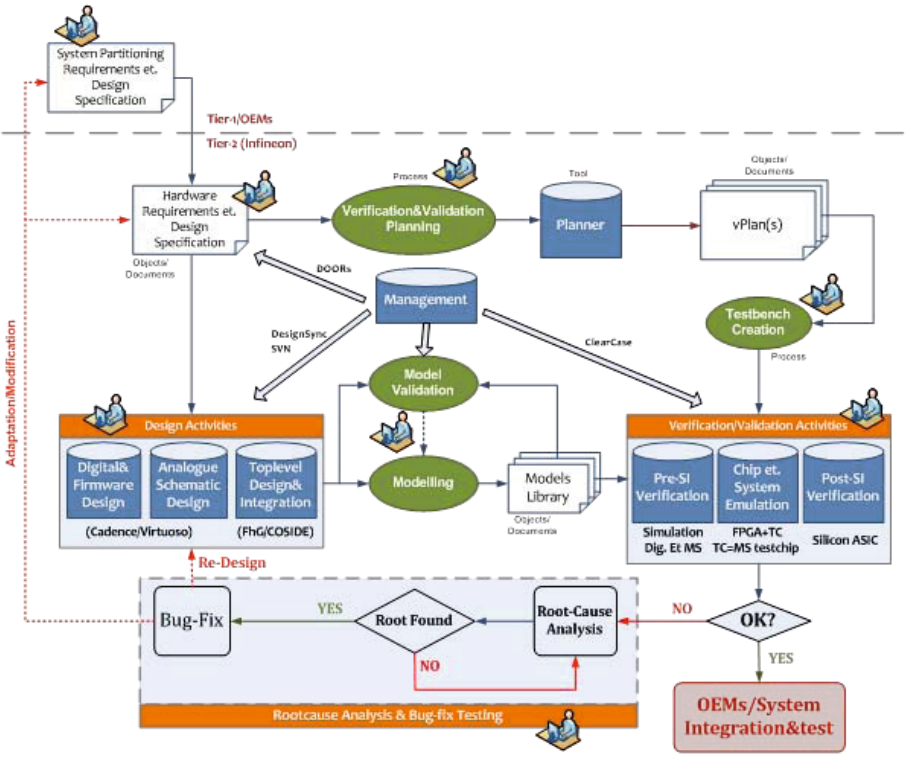


Fig. 2. Verification workflow for complex mixed signal IC development

Infineon so that HW concept and design specification can be further defined. Right after this phase, the design and verification/validation activities will be launched (almost in parallel). The design activities covers conceptual and design work, which including digital and firmware design, analog schematic design and top-level integration. Most of the tasks defined during design phase are mainly done under the Cadence Virtuoso/AMS-Designer tooling environment, whereas for the proof-of-concept, the COSIDE (Complex System Integrated Development Environment) from Fraunhofer IIS is used. Nearly at the same time, verification engineering team also based on the hardware requirements and design specification starts their verification and validation planning process. This process (with the support from some planning tools, e.g.: in-house tool) results in a verification plan which then used by verification/validation engineer for test bench creation. The verification plan is categorized with different verification approaches including:

Pre-silicon verification covers all type of simulation at different design level (block, module and chip top-level) using different techniques from mixed-signal to mixed-abstraction simulation.

Emulation at integrated circuit (IC) and system level uses FPGA with mixed-signal test chip as an early prototype for verifying many scenarios that are very impractical or impossible to simulate. These scenarios are usually long term test, stress test or sensor data transmission test over a long period of time (e.g.: could result in data interception of million sensor message). This approach is an innovative approach, developed by Infineon and its customer. The approach has been recently accepted as publication in the SAE Journal of Passenger Car - Electrical and Electronic System (SAE Society of Automotive Engineer).

Post-silicon Verification refers to verification of the real IC in the lab. It is an extension of those test scenarios which could not be done using emulation system. This is due to the fact that emulation system is mainly designed to cover certain safety critical functions (e.g.: sensor interfaces or the deployment interfaces) but not the full design functionalities. Through extensive tests done in the lab, the post-silicon verification should maximize the test coverage at HW component level before being delivered to the system integrator Tier-1 supplier.

Finally, the rootcause analysis and bug-fix testing is considered as an undesired part of the verification activities. However, when a bug is found, rootcause analysis and bug-fix testing could significantly contribute to increase the verification as well as the project timing and effort. This is because the bug-fix could be a change in the design (re-design) of a modification/adaptation in the specification. In any case, this would trigger the verification regression run, meaning cost in time and effort. Despite the fact that verification/validation activities for mixed-signal IC development are well established, the verification work-flow above still involves simulation and manual testing methods used in the practice of the automotive industry. These methods consist in verification engineers creating input stimuli, executing simulation models and observing the waveforms for correctness. They are known for the following weaknesses: ad-hoc, inefficient and prone to human errors. In addition, it is widely accepted that for complex mixed-signal multi-cores System-on-Chip (SoC) IC products, verification accounts for around 60%-70% of the total development. This is especially true for automotive safety critical SoC product with a high number of analogue interfaces to the physical components, e.g.: an airbag SoC chipset in an automotive airbag system application. As such, any approaches which could help to reduce design and verification effort, improve time-to-market and product quality, e.g.: formal verification, boost up verification runs using hardware acceleration platform and verification automation are of extreme interest.

3 Signal Temporal Logic

In this section, we give a brief overview of the Signal Temporal Logic (STL) that we use to formalize the case study requirements. For the full details regarding the assertion language and the monitoring algorithms for STL, we refer the reader to [20].

We consider the STL logic with both *future* and *past* operators, interpreted over a finite multi-dimensional signal w. A signal w is a partial function $w : \mathcal{T} \to \mathbb{B}^m \times \mathbb{R}^n$, where $\mathcal{T}$ is the interval $[0, d)$ denoting a time domain of duration d. Let $X = \{x_1, \ldots, x_m\}$ be the set of real valued variables and $P = \{p_1, \ldots, p_n\}$ the set of STL propositions. We denote by $w|_x$ and $w|_p$ the projection of the signal w to a real-valued or propositional variable $x \in X$ or $p \in P$. A *Boolean constraint* over X is a predicate of the form $x \circ c$, where $x \in X$, $\circ \in \{\texttt{<}, \texttt{<=}, \texttt{=}, \texttt{>=}, \texttt{>}\}$ and $c \in \mathbb{Q}$. The syntax of an STL formula φ over X and P is defined by the grammar

$$\begin{aligned} \alpha &:= p \mid x \circ c \\ \varphi &:= \alpha \mid \texttt{not}\ \varphi \mid \varphi_1\ \texttt{or}\ \varphi_2 \mid \varphi_1\ \texttt{until!}_I\ \varphi_2 \mid \varphi\ \texttt{since!}_I\ \varphi_2 \end{aligned}$$

where $p \in P$, $x \in X$, $c \in \mathbb{Q}$ is a constant and I is an interval of the form $[a, b]$, $[a, b)$, $(a, b]$, (a, b), $[a, \infty)$ or (a, ∞) where $0 \leq a \leq b$ are rational numbers. As in LTL, basic STL operators can be used to derive other standard Boolean and temporal operators, in particular the time-constrained `eventually!`, `once!`, `always`, and `historically` operators:

$$\begin{array}{ll} \texttt{eventually!}_I\ \varphi = \mathit{true}\ \texttt{until!}_I\ \varphi & \texttt{once!}_I\ \varphi = \mathit{true}\ \texttt{since}_I\ \varphi \\ \texttt{always}_I\ \varphi = \texttt{not eventually!}_I\ \texttt{not}\ \varphi & \texttt{historically}_I\ \varphi = \texttt{not once}_I\ \texttt{not}\ \varphi \end{array}$$

The semantics of an STL formula φ with respect to an n-dimensional signal w is described via the satisfiability relation $(w, t) \models \varphi$, indicating that the signal w satisfies φ at time t, according to the following recursive definition, where $\mathcal{T}$ is the time domain.

$$\begin{array}{ll} (w,t) \models x \circ c & \leftrightarrow w|_x[t] \circ c \\ (w,t) \models p & \leftrightarrow p[t] = 1 \\ (w,t) \models \texttt{not}\ \varphi & \leftrightarrow (w,t) \not\models \varphi \\ (w,t) \models \varphi_1\ \texttt{or}\ \varphi_2 & \leftrightarrow (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\ (w,t) \models \varphi_1\ \texttt{until!}_I\ \varphi_2 & \leftrightarrow \exists\, t' \in (t \oplus I) \cap \mathcal{T}\ (w,t') \models \varphi_2 \text{ and} \\ & \quad\ \forall\, t'' \in (t, t')\ (w, t'') \models \varphi_1 \\ (w,t) \models \varphi_1\ \texttt{since!}_I\ \varphi_2 & \leftrightarrow \exists\, t' \in (t \ominus I) \cap \mathcal{T}\ (w,t') \models \varphi_2 \text{ and} \\ & \quad\ \forall\, t'' \in (t', t)\ (w, t'') \models \varphi_1 \end{array} \tag{1}$$

A formula φ is satisfied by w if $(w, 0) \models \varphi$.

Example 1. An example of a property that can be expressed in STL is a mixed signal stabilization property that has the following requirements:

- The absolute value of a continuous signal x is always less than 5;
- When the (Boolean) trigger rises, within 600 time units $\texttt{abs}(x)$ has to drop below 1 and stay like that for at least 300 time units.

This property is illustrated in Figure 3 and expressed in STL as:

$$\begin{aligned} &\texttt{always}\ (\texttt{abs}(x) < 6\ \texttt{and} \\ &\qquad (\texttt{rise}(\mathit{trigger})\ \texttt{->}\ \texttt{eventually!}_{[0,600]}\ \texttt{always}_{[0,300]}\ (\texttt{abs}(x) < 1))) \end{aligned}$$

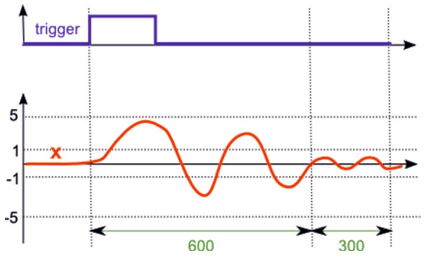
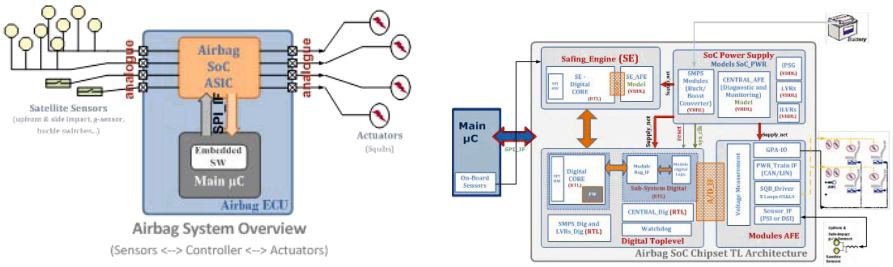


Fig. 3. Example: stabilization property

4 Case Study

4.1 Case Study Description

The increasing number of airbags in a vehicle, the requirement to comply with stricter safety requirements, while costs must be reduced has brought automotive airbag system application to a new approach with SoC design, shown in Figure 1. Consequently, verification has dramatically increasingly challenges the design of complex mixed-signal System-on-Chip (SoC) products. This is especially true for automotive safety critical SoC products with a high number of analogue interfaces to the physical components, e.g.: an airbag SoC chipset in an automotive airbag system application.



(a) (b)

Fig. 4. A typical airbag system: (a) overview; (b) airbag SoC chipset top-level implementation architecture

During the operation, the sensors (buckle switches, accelerometers, pressure sensors, etc.) mounted in key locations of the vehicle, continuously measure the positions of impact, the severity of the collision and other variables. This information is provided to the airbag SoC chipset in form of analog signals. The airbag SoC chipset translates the analog sensor signals into digital words. The translated digital sensor data is reported to the main uC via the SPI (Serial Peripheral Interface) communication. Based on this information the airbag main

uC decides if, where (location) and when the airbags (e.g.: actuators) is deployed. Accordingly, this makes the verification (computer-based simulation) and validation (lab evaluation) of the airbag SoC product, especially the sensor interfaces become a challenging task, mainly because of:

- Verification for the airbag SoC and its sensor interfaces has to cover real-time embedded mixed signal domains.
- Failure during the reception, decoding and processing of sensor data in the airbag controller system can originate unexpected or false deployment events of the airbag system putting human safety in danger.
- Most of the functionalities of sensor interfaces can only be verified at the system level of the chip and at the system application level. Only using classical mixed-signal simulation approach becomes a bottle neck.
- Many verification scenarios of the sensor interfaces such as long-term verification run with checking of millions sensor data frames are not suitable using computer-based simulation.

In addition, reducing time-to-market and first time right design in automotive electronics industry, which are key requirements in project to win customer and market share, has posed a great challenge to the design and verification team. With this case study, we are evaluating the assertion-based monitoring methodology on the modern airbag system application with the focus on the new airbag sensor interface using the new DSI3 standard, promoted by the DSI consortium[1]. DSI3 goals are to improve performance, reduce cost and promote open standard but still remains at the lowest cost possible compare to the current widely used PSI5 standard. Higher performance is achieved among others, by increased communication speed from the slave sensor to the master.

4.2 Formalization of DSI3 Discovery Mode Requirements

In this section, we formalize DSI3 *discovery mode* requirements, illustrated in the highlighted section of Figure 5. In the DSI3 discovery mode, μC interacts with the sensor interfaces via the voltage (`v`) and current (`i`) lines. It is the initial phase of the DSI3 standard protocol and it works as follows. First, the power apply is turned on, resulting in the voltage ramp from $0V$ to `V_high` (phase (1) in Figure 5). Then, μC issues commands for probing the presence or absence of sensors. These commands are converted by the SoC to analog pulses carried over the voltage lines. In Figure 5, (2) shows a discovery pulse command. A sensor that is connected to the sensor interface responds by an inverted analog pulse carried over the current line (shown in (3) of Figure 5). Finally, if a sensor is not connected to the sensor interface, the discovery pulse command is not followed by any response on the current line, as illustrated in (4) of Figure 5.

In addition to the correct ordering of events, described in the previous paragraph, the DSI3 Bus Standard also defines a number of timing requirements that must be met by any correct implementation of the protocol:

[1] `http://www.dsiconsortium.org`

1. The minimal time between the moment that the power is applied and the first discovery pulse command is sent, as shown by (a) in Figure 5;
2. The maximal total duration of the discovery mode, measured between the moment that the power is applied and the end of the sensor probing by the μC, as illustrated by (b) in Figure 5;
3. The expected time between any two consecutive discovery pulse commands ((c) in Figure 5); and
4. The expected time between a discovery pulse command and the response by the sensor (or its lack of response if the sensor is not connected), as shown by (d) in Figure 5.

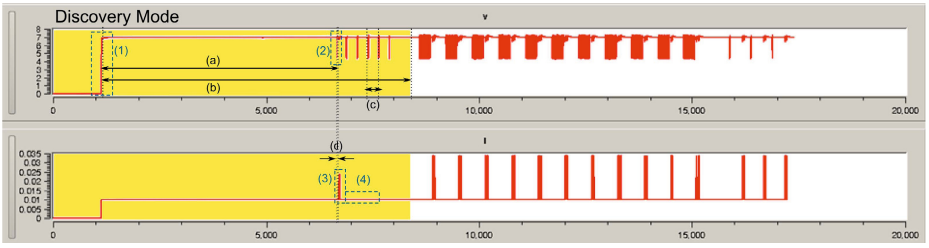


Fig. 5. DSI3 Discovery Mode requirements - overview

Specification of Events of Interest. In order to be able to formalize these timing properties defined by the DSI3 Bus Standard, we first must be able to accurately characterize and recognize the "events" corresponding to power application, discovery pulse commands, the sensor response and its lack of response. The graphical specification of these patterns is shown in Figure 6.

We first consider applying power to the SoC, illustrated in Figure 6 (a), which is characterized by a ramp that goes from $0V$ to V_{high}. We consider that the power is on, characterized by the event `pw_app`, when the voltage signal goes above V_{high}.

```
% Regions of interest
1:  define b:v_zero := a:v == 0;
2:  define b:v_above_high := a:v >= Vhigh;
3:  define b:v_between_high_zero := a:v > 0  and a:v <= Vhigh;

% Power applied
4:  define b:pw_app :=
5:     rise(b:v_above_high)  and
6:     (b:v_between_high_zero  since!  fall(b:v_zero);
```

A *discovery pulse command* is carried on the voltage line and is characterized by its *shape* and *duration*, as shown in Figure 6 (b). The DSI3 standard requires

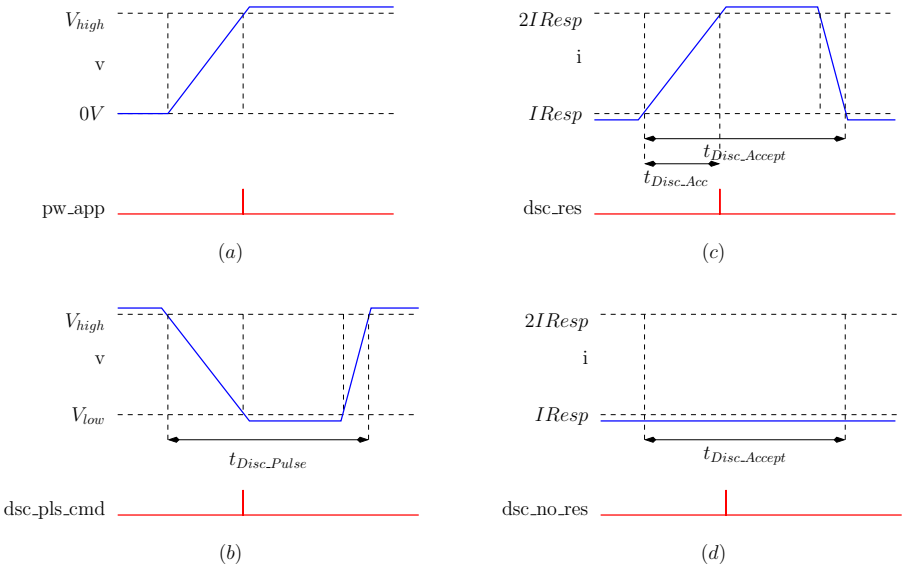


Fig. 6. Graphical specification of events of interest: (a) power applied; (b) discoverly pulse command; (c) sensor response; and (d) sensor no response

that the distance between two consecutive discovery pulse commands is t_{Disc_Per} ($\pm$ tolerance). In order to formalize this requirement in STL (shown in the next paragraph), we first define the regions of interest that are needed to capture a discovery pulse command (lines $1-3$). We then characterize the correct shape of the pulse (lines $4-7$) and its duration (lines $8-11$), resulting in the specification of the discovery pulse command (line 12).

```
% Regions of interest
1:  define b:v_above_high := a:v >= Vhigh;
2:  define b:v_below_low := a:v <= Vlow;
3:  define b:v_between_high_low := a:v >= Vlow  and a:v <= Vhigh;

% Pulse shape
4:  define b:cmd_dp_shape :=
5:     fall(b:v_above_high)  and
6:     (b:v_between_high_low  until! b:v_below_low  until!
7:      b:v_between_high_low  until! b:v_above_high);

% Pulse end-to-end timing
8:  define b:cmd_dp_e2e_timing :=
9:     fall(b:v_above_high)  and
10:    ((not rise(b:v_above_high)) until![tDisc_Pulse-tol:tDisc_Pulse+tol]
11:    rise(b:v_above_high));
```

```
% Pulse = shape + end-to-end timing
12: define b:cmd_dp := b:cmd_dp_e2e_timing and b:cmd_dp_shape;
```

The specification of the sensor response (`dsc_res`) and no response (`dsc_no_res`) patterns (Figures 6 (c) and (d)) is very similar to the specification of the discovery pulse command, and we skip their presentation due to the lack of space.

Assertions for DSI3 Discovery Mode Requirements. After specifying events of interest, we are ready to formalize the requirements that relate these events and define the timing constraints between them, as described in the DSI3 bus protocol, and summarized in Figure 7.

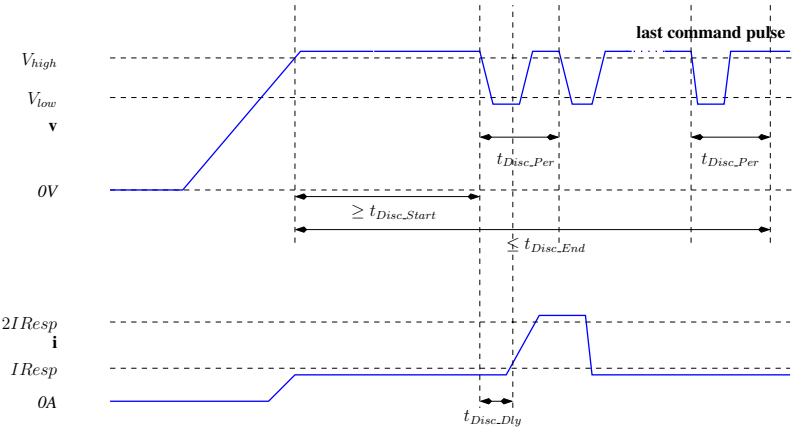


Fig. 7. Graphical specification of DSI3 discovery mode requirements

We start with the requirement saying that between the power applied event and the first discovery pulse command, there must be at least t_{Disc_Start} time elapsed. We formalize this requirement with the following assertion.

```
% Timing between power applied and first discovery pulse commands
1:  first_disc_cmd_dly assert:
2:    always (b:pw_app -> (((not b:cmd_dp ) until![tDisc_Start:inf]
3:    b:cmd_dp);
```

The second requirement says that the discovery mode has a maximum duration of t_{Disc_End}. We consider that the discovery starts when the power is applied, and that it ends t_{Disc_Per} time after the last discovery command is issued. We first define the auxiliary property `end_disc` to characterize the end of the discovery mode, and then formalize the assertion as follows.

```
% Discovery end
1: define b:end_disc :=
2:   ( not b:cmd_dp since![=t_Disc_Pulse] b:cmd_dp ) and
3:    always not b:cmd_dp ;

% Discovery mode maximum duration
4:  disc_duration assert:
5:    always (b:pw_app -> (eventually![0:t_Disc_End] b:end_disc);
```

The third requirement defines the correct timing between a discovery pulse command, and the associated sensor response when the sensor is connected (and its lack of response when it is not connected). This requirement is dependent on the actual configuration of the system, and we formalize the property in which only the first sensor is connected. In order to specify this requirement, we also need to characterize the first discovery pulse command.

```
% First discovery pulse command
1: define b:first_cmd_dp :=
2:   (b:cmd_dp and historically not b:cmd_dp ;

% Discovery pulse command - response delay
3:  cmd_resp_delay assert:
4:    always ((b:cmd_dp and b:first_cmd_dp ->
5:   (eventually![t_Disc_Dly-tol:t_Disc_Dly+tol] b:dsc_res);

% Discovery pulse command - no response delay
6:  cmd_resp_delay assert:
7:    always ((b:cmd_dp and not b:first_cmd_dp ->
8:   (eventually![t_Disc_Dly-tol:t_Disc_Dly+tol] b:dsc_no_res);
```

Finally, the last requirement says that every two consecutive discovery pulse commands must be separated by $t_{Disc\_Pulse} \pm$ some tolerance, which is formalized with the following STL assertion.

```
% Timing between consecutive discovery pulse commands
1:  cmd_disc_pulse_period assert:
2:    always (b:cmd_dp -> (((not b:cmd_dp ) until![tDisc_Per-tol:tDisc_Per+tol]
3:    b:cmd_dp) or (always not b:cmd_dp )));
```

4.3 Case Study Evaluation

The design-under-test used in the case study was implemented by Infineon Technologies in VHDL (RTL) and VHDL with real number behavior. The design-under-test represents a mixed-abstraction of RTL and behavior model, consists of 23 different functional modules which are connected together via a complex logic core. The simulation time for this design takes approximately between 2 and 3 hours per simulation.

The formalization of the DSI3 requirements was lead by AIT, and was done in several iterations, involving feedback from the Infineon's designers and engineers. The tool used for monitoring the simulation traces against the formalized

requirements was AMT [21]. The monitoring was done on a computer with Intel Core i7 processor, 8GB of RAM and the 64-bit Ubuntu 12.04 LTS running on a virtual machine from 64-bit Windows 7 operating system. The simulated traces files had approximately $210MB$ per simulation. The assertions were checked against the simulation traces one by one, and all the monitoring times were lower than $20s$ per assertion. It follows that the monitoring presented a negligible overhead compared to the simulation time.

The monitoring results provided several interesting insights regarding the formalization of requirement documents. The formalization for discovery pulse commands and the event when the voltage is applied proved to be sufficient to catch these events. However, in our first iteration, we were not able to catch the sensor response. In fact, the DSI3 standard does not specify the minimal duration of the pulse falling (see Figure 6 (c)), but our intuition was that due to the physical constraints the duration must be strictly positive. We thus imposed this constraint in our original formulation of the `dsc_res` property. However, after clarification from Infineon's designers, we found out that at this given stage of development, the design is approximated by a simpler model that allows instantaneous ramps between values, as we can see in Figure 8.

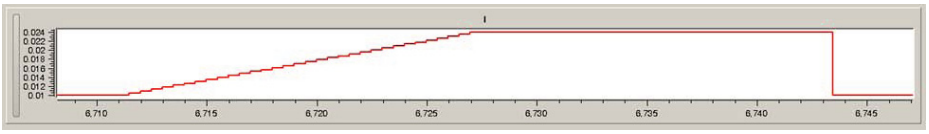


Fig. 8. Zoom in on the simulated sensor response on the current line

The monitoring tool reported a violation of the `cmd_disc_pulse_period` assertion. In the formal assertion, we used the value of $125\mu s$ for t_{Disc_Per}, while the actual distance between consecutive discovery pulse commands in the simulated trace was close to $250\mu s$, as shown in Figure 9. The value $125\mu s$ for t_{Disc_Per} was taken from Table 6-2 in the standard [15]. After discussions with the Infineon's engineers, it turned out that the standard gives only an average value for t_{Disc_Per}, while allowing the designer to choose any other value for t_{Disc_Per} as long as all the other hard timing constraints are met. After reformulating the assertion with the new value for t_{Disc_Per} provided by Infineon's engineers, the simulation traces satisfied the assertion.

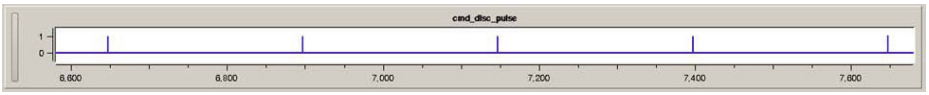


Fig. 9. Zoom in on the detected discovery command pulses extracted from the simulations by the monitoring tool

We conclude that generally requirement documents are not always fully precise regarding parts of the specification, which makes the formalization of requirements a non-trivial task. For some of the properties, interpretation freedom is left, and one must take extreme care to make assumption which match the intended meaning.

5 Lessons Learned and Future Directions

Requirements document often give interpretation freedom to the designer which can result in ambiguous understanding of the desired property. Using STL to monitor the compliance of the airbag SoC to the DSI3 standard protocol, it helps to remove these kinds of ambiguity. In addition, when the STL is implemented as an assertion, it strengthens the communication between different disciplinary teams, ensuring a clear and common understanding between teams on the system properties and requirements. We found that the monitoring itself represents a negligible overhead to the design simulation, while automatically providing useful debugging information to the designer as well as reducing time and error prone due to manual inspection of the simulation results.

We identified a number of features that are still missing in the STL-based monitoring framework and that we will investigate in the near future:

Template Specification Languages for STL: while STL is a rigorous, unambiguous and powerful specification language, it is often not very intuitive to the engineers, and especially to analog designers. Inspired by the graphical specification of properties, as described in the DSI3 bus standard, we will develop a graphical language for specifying common STL patterns, while hiding away low-level STL details from the future.

STL Assertion Libraries: we identified that building libraries of common STL properties for specific applications would greatly facilitate application of this technology and would enhance the reuse of assertions across different phases of design, various actors in the automotive value chain and different project. For instance, an assertion library specifying the full DSI3 bus standard would be reused in every project that requires using this communication protocol. The assertion skeletons would remain the same across the project, and only instantiations of project-specific parameters would need to change. In order to facilitate this goal, we need a more flexible syntax for STL that allows declaration of variables and constants outside of the assertions. We are currently working on adding this feature to the STL language.

Diagnostics for Assertion Violations: when an assertion is violated, it is extremely important to be able to easily extract the reasons of the violation. The AMT tool already provides extensive information about the assertion violation, by computing and making visible to the user the information about the satisfaction/violation in time of all sub-formulas in the violated assertion. However, the causality analysis still needs to be done manually by the engineer in order to gain insight into reasons for assertion violation. We are

planing to further automate this process, by generating reports explaining in human readable language the reasons of assertion violations.

Assertion Language Extensions: in this paper, we focused on the discovery mode of the DSI3 protocol, in which STL can be directly used to accurately specify needed requirements. The expressive power of STL might not be sufficient for later phases of the protocol, when actual data is exchanged between the sensors and the μC over the voltage and current lines. This protocol uses multi-level source and Manchester coding for transferring data. We will look in the future for additional features that STL may need in order to support accurate specification of the full DSI3 bus standard and study the necessary extensions.

Algorithms for Hardware FPGA Monitors for STL: we used in this paper the offline STL monitoring tool AMT for case study evaluation. The offline monitoring has the advantage of being indifferent about the source (simulation, emulation or measurement) of the trace files – their provenance does not affect the monitoring results. However, the trend of implementing mixed-signal designs on FPGA hardware enables much longer design emulations of the design, generating huge amount of data to be processed. It follows that online hardware FPGA implementation of STL monitors, running in parallel with the design emulation, would be beneficial as they would limit the amount of data that needs to be stored at any time and would enable aborting emulation upon assertion violation detection.

6 Conclusions

We have evaluated the mixed-signal assertion-based monitoring methodology by applying it to check correctness of DSI3 sensor interfaces in a modern airbag system-on-chip application. We have demonstrated the usefulness and the potential of the approach, highlighting its benefits but also identifying the features that need to be added to the framework in order to make it mature for industrial use. AIT and Infineon will continue working together to strengthen the assertion-based monitoring technology and tailor it for its effective application in the V&V of automotive applications.

References

1. Althoff, M., Rajhans, A., Krogh, B.H., Yaldiz, S., Li, X., Pileggi, L.: Formal verification of phase-locked loops using reachability analysis and continuization. In: Proceedings of the International Conference on Computer-Aided Design, pp. 659–666. IEEE Press (2010)
2. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1), 3–34 (1995)
3. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011)

4. Bartocci, E., Bortolussi, L., Nenzi, L.: A temporal logic approach to modular design of synthetic biological circuits. In: Gupta, A., Henzinger, T.A. (eds.) CMSB 2013. LNCS, vol. 8130, pp. 164–177. Springer, Heidelberg (2013)
5. Bertrane, J.: Static analysis by abstract interpretation of the quasi-synchronous composition of synchronous programs. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 97–112. Springer, Heidelberg (2005)
6. Dang, T., Donzé, A., Maler, O.: Verification of analog and mixed-signal circuits using hybrid system techniques. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 21–36. Springer, Heidelberg (2004)
7. Donzé, A., Fanchon, E., Gattepaille, L.M., Maler, O., Tracqui, P.: Robustness analysis and behavior discrimination in enzymatic reaction networks. PLoS ONE 6(9), e24246 (2011)
8. Donzé, A.: Breach, A toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010)
9. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013)
10. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010)
11. Donzé, A., Maler, O., Bartocci, E., Nickovic, D., Grosu, R., Smolka, S.: On temporal logic and signal processing. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 92–106. Springer, Heidelberg (2012)
12. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. 410(42), 4262–4291 (2009)
13. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
14. Frehse, G., Krogh, B.H., Rutenbar, R.A.: Verifying analog oscillator circuits using forward/backward abstraction refinement. In: DATE, pp. 257–262. European Design and Automation Association (2006)
15. Distributed System Interface. DSI3 Bus Standard. DSI Consortium
16. Jones, K.D., Konrad, V., Nickovic, D.: Analog property checkers: a ddr2 case study. Formal Methods in System Design 36(2), 114–130 (2010)
17. Little, S., Walter, D., Jones, K., Myers, C.: Analog/Mixed-signal circuit verification using models generated from simulation traces. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 114–128. Springer, Heidelberg (2007)
18. Maler, O., Manna, Z., Pnueli, A.: From timed to hybrid systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 447–484. Springer, Heidelberg (1992)
19. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)
20. Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits. STTT 15(3), 247–268 (2013)

21. Nickovic, D., Maler, O.: AMT: A property-based monitoring tool for analog systems. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 304–319. Springer, Heidelberg (2007)
22. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)
23. Rizk, A., Batt, G., Fages, F., Soliman, S.: On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 251–268. Springer, Heidelberg (2008)
24. Steinhorst, S., Hedrich, L.: Model checking of analog systems using an analog specification language. In: DATE, pp. 324–329. ACM (2008)