Runtime Monitoring with Recovery of the SENT Communication Protocol

Conference Paper *in* Lecture Notes in Computer Science · July 2017

DOI: 10.1007/978-3-319-63387-9_17

8 authors, including:

Konstantin Selyunin
TU Wien
8 PUBLICATIONS **55** CITATIONS
SEE PROFILE

Stefan Jaksic
AIT Austrian Institute of Technology
7 PUBLICATIONS **123** CITATIONS
SEE PROFILE

Thang Nguyen
Infineon Technologies
14 PUBLICATIONS **140** CITATIONS
SEE PROFILE

Ezio Bartocci
TU Wien
161 PUBLICATIONS **2,004** CITATIONS
SEE PROFILE

Runtime Monitoring with Recovery of the SENT Communication Protocol

Konstantin Selyunin[1], Stefan Jaksic[1,3], Thang Nguyen[2], Christian Reidl[2], Udo Hafner[2], Ezio Bartocci[1], Dejan Nickovic[3], and Radu Grosu[1]

[1]Vienna University of Technology, Vienna, Austria
{konstantin.selyunin, ezio.bartocci, radu.grosu}@tuwien.ac.at

[2]Infineon Technologies Austria AG, Villach, Austria
{thang.nguyen, christian.reidl, udo.hafner}@infineon.com

[3]AIT Austrian Institute of Technology, Vienna, Austria
{dejan.nickovic, stefan.jaksic.fl}@ait.ac.at

Abstract. We show how the requirements of the SENT communication protocol between a magnetic sensor and an electronic control unit (ECU) can be monitored in real time, with a monitor capable of processing 70 million samples per second. We elaborate on a complete flow from formalizing electrical and timing requirements using Signal Temporal Logic (STL) and Timed Regular Expressions (TRE), to implementing runtime monitors in FPGA hardware and evaluating the results in the lab. For a class of asynchronous serial protocols, we define a procedure to obtain monitors that are capable to recover after violations. We elaborate on two different approaches to monitor the requirements of interest: (i) temporal testers with SystemC, STL and High-Level Synthesis; (ii) automata-based approach with TRE in HDL. We also present how the results of the monitoring can be used for error logging to provide users with extensive debugging information. Our approach allows to monitor requirements-specification conformance in real time for long-term tests.

Keywords: Case study · Verification in industrial practice · Runtime verification · Lightweight formal methods

1 Introduction

Strict safety standards (e.g. ISO 26262 [1]) force manufacturers in the automotive industry to develop new system-verification methods. Formal verification [2] and model-based design [3], although in principle capable of providing a formal proof of a system correctness, have limitations when applied to real-world industrial problems due to the complexity of the associated systems.

The verification and validation (V & V) phase in automotive electronic development comprises extensive product testing under different scenarios, including stress conditions. Runtime verification [4,5], a light-weight verification technique, treats the system under investigation as a black-box, and reports system's conformance to formal requirements in a current run. Since runtime verification can

be applied non-intrusively to existing systems, it fits in the current V & V setting very well, allowing a rigorous treatment and a traceability of requirements, and enabling automated observation of specification compliance via monitoring.

In this case-study paper we report on the runtime verification of electrical and timing requirements of the Single Edge Nibble Transmission (SENT [6]) protocol. The SENT is mainly used in automotive applications, for instance, in an electronic power steering (EPS), or an electronic braking system (EBS). In these applications sensors transfer data about rotation of a steering wheel or position of a braking pedal, respectively; hence ensuring the correct information transfer and runtime error detection is of utter importance.

The current industry practice relies on hard-crafted checkers, that lack diagnostics information and do not runtime-check the signals on the electrical level. Existing tools for offline trace verification (e.g. the AMT [7]) are not directly applicable in this context, due to the excessive size of the resulting traces: e.g. if one records an analog signal, sampled at 70MHz, for an hour of runtime in an array of 16-bit integers, the trace will result in 504Gb of data. Moreover, it is also often the case that a long-term test takes several days of real-time execution. In order to be able to speed up the checking process and to produce the monitoring results during the execution of the system, we translate high-level specifications into monitors implemented in FPGA and run them in parallel with the system under investigation. We propose an approach that allows to observe the monitoring results in real time, track requirements to implementation, and report violation and debugging information for the higher level analysis.

The contributions of this paper can be summarized as follows:

1. We propose a framework for generating monitors with recovery from a class of high-level specifications;
2. We formalize the electrical and timing requirements of the SENT protocol in STL and TRE specifications;
3. We evaluate our framework on the SENT case study, demonstrating the synergy between formal methods and industrial practice in a real-world setting.

The rest of the paper is organized as follows: Section 2 discusses the related work, and Section 3 provides the preliminaries. Section 4 presents formalization of requirements in two formalisms, the necessary initial step for creating monitors. Section 5 elaborates on runtime monitoring with recovery of asynchronous serial protocols. Section 6 presents in depth the case study and experimental results. Section 7 offers our concluding remarks and directions for future work.

2 Related Work

Runtime verification of formally defined properties is an extremely diverse research area in terms of requirements-specification languages [8–14], approaches to construct the monitors [15–18], and target applications [19–24].

The FoC framework of IBM [8, 25] allows to generate monitors for Property Specification Language (PSL) assertions. Although PSL allows to specify the evolution of a system, the formal semantics is based on the sequence of states and does not include a notion of time explicitly. STL [26] and TRE [27], on the

other hand, were designed to deal with real time, and allow to precisely identify time intervals of interest and bound temporal modalities to these intervals.

As far as hardware implementation is concerned, Schumann et. al. [16] propose an FPGA implementation of runtime monitors for the UAV applications. The authors construct FPGA monitors for security requirements and specify possible attacks that a UAV might undergo. A Bayesian network on top of Metric Temporal Logic (MTL) monitoring allows to estimate system health. The authors do not take into account neither the recovery of monitors after violations nor the electrical characteristics of signals, and define their properties on a higher level of abstraction. On the contrary, in our work we focus on formalizing the electrical and timing requirements of a sequential protocol, with a special emphasis on monitor recovery after capturing specification violations.

In a similar context we refer to the work of Reinbacher et al. [28]. The authors present a framework for monitoring past-time Metric Temporal Logic (MTL) specifications. In order to achieve the reconfigurability of the system, they introduce an over-complex hardware architecture. In our case, we specifically target asynchronous serial protocols, for which we find the TRE formalization with simpler, automaton-based architecture more appropriate.

UPPAAL [29, 30] is a well established tool for the verification of real-time systems which can be modeled with timed automata. This tool provides a description language for modelling, a simulator, and a model checker. In contrast, our goal is to create a standalone monitor in order to verify a discrete time system during runtime. Our monitors are ignorant of the model of the system. In addition, since we are using a formally proven translation from TREs to automata, our monitors are correct by construction.

We are aware of several case studies on monitoring temporal logic specifications - the automotive bus standard [31], the DDR2 memory interface [19], typical automotive functional requirements [32]. All of these works focus on offline monitoring and continuous-time semantics, which covers STL and does not consider specifications based on regular expressions, and omit monitor recovery aspects after capturing a violation. Although the authors in [33] runtime-verify a subset of requirements of the PSI5, the protocol uses different encoding scheme then the SENT; their emphasis is on how to apply runtime verification, and they by and large avoid technical details. In contrast, we compare two formalisms and implementations, to increase integration readiness level for the monitor itself and eliminate the "single source of truth" aspect from the monitoring system.

In [12] the authors also use TREs with events to evaluate the performance of a controller and sensor implementation. Orthogonally to our work, they define measurement specifications over timed patterns.

3 Preliminaries

This section presents the specification languages that we use in this work to state the requirements in a formal way.

3.1 Signal Temporal Logic

Signal Temporal Logic [10] allows specification of properties defined over analog-mixed signals. As the goal of the case study is to produce runtime monitors in digital hardware (FPGA), the monitors operate on a finite representation of originally real-valued signals (ADC is used for quantization and sampling of continuously evolving voltage). For this purpose we interpret STL over discrete time and finite-valued domain. Let w be a multi-dimensional signal of a finite length, $w : [0, d] \mapsto P^n \cup X^m$, where $d \in \mathbb{N}$ is a duration of the signal; $P^n = \{p_1, \cdots, p_n\}$ and $X^m = \{x_1, \cdots, x_m\}$ are boolean (digital) and finite-domain (analog) variables respectively. Analog variables X^m are interpreted over a domain $\mathbb{D} = [0, \gamma] \subseteq \mathbb{N}$, where $\gamma = 2^r - 1, r \in \mathbb{N}$ is defined by a resolution of an ADC. The projection of the signal w to a component $e \in P \cup X$ is denoted by $\pi_e(w)$. The syntax of an STL formula φ with past and future operators is defined by the following grammar [34]:

$$\varphi := p \,|\, x \sim c \,|\, \neg\varphi \,|\, \varphi_1 \vee \varphi_2 \,|\, \varphi_1 \,\mathcal{U}_I \varphi_2 \,|\, \varphi_1 \,\mathcal{S}_I \varphi_2,$$

where $p \in P$, $x \in X$, $c \in \mathbb{D}$, $\sim \in \{<, \leq\}$, I is a time interval $[a, b]$, where $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. For intervals of the type $[a, a]$ we use a notation $\{a\}$. We derive logical and temporal operators from the definition in a standard way: $\top = \varphi \vee \neg\varphi$; $\bot = \neg\top$; eventually $\diamondsuit_I \varphi = \top \,\mathcal{U}_I \varphi$; once $\boxminus_I \varphi = \top \,\mathcal{S}_I \varphi$; next $\bigcirc \varphi = \diamondsuit_{\{1\}} \varphi$; previous $\ominus \varphi = \boxminus_{\{1\}} \varphi$; always $\square_I \varphi = \neg \diamondsuit_I \neg\varphi$; historically $\boxminus_I \varphi = \neg \boxminus_I \neg\varphi$. We introduce two useful macros in our notation, which capture the change in evaluation of a boolean component of w: for $p \in P$, `enter(`p`)` $= \ominus \neg p \wedge p$ and `exit(`p`)` $= \ominus p \wedge \neg p$.

The semantics of an STL formula is defined as follows:

$$\begin{aligned}
(w, i) &\models p && \leftrightarrow \pi_p(w)[i] = \top \\
(w, i) &\models x \sim c && \leftrightarrow \pi_x(w)[i] \sim c \\
(w, i) &\models \neg\varphi && \leftrightarrow (w, i) \not\models \varphi \\
(w, i) &\models \varphi_1 \vee \varphi_2 && \leftrightarrow (w, i) \models \varphi_1 \text{ or } (w, i) \models \varphi_2 \\
(w, i) &\models \varphi_1 \,\mathcal{U}_I \varphi_2 && \leftrightarrow \exists j \in (i + I) \cap \mathbb{T} : (w, j) \models \varphi_2 \\
& && \quad \text{and } \forall k : i < k < j, (w, k) \models \varphi_1 \\
(w, i) &\models \varphi_1 \,\mathcal{S}_I \varphi_2 && \leftrightarrow \exists j \in (i - I) \cap \mathbb{T} : (w, j) \models \varphi_2 \\
& && \quad \text{and } \forall k : j < k < i, (w, k) \models \varphi_1
\end{aligned}$$

The standard semantics of the future operators, i.e. $\varphi_1 \,\mathcal{U}_I \varphi_2$, $\diamondsuit_I \varphi$, $\square_I \varphi$ is defined s.t., the satisfaction of the formulae at the time step i depends on events that happen in the future, namely at $(i + I) \cap \mathbb{T}$, which makes monitoring of these specifications acausal. To overcome such limitation, our hardware monitors comprise only past-temporal operators, and we use a procedure from [35] to convert a formula with future operators to an equi-satisfiable past one.

3.2 Timed Regular Expressions

Timed regular expressions (TRE) [27] allow to pattern-match a specification over a signal. As the authors in [12] mentioned, the fundamental difference between

STL and TREs comes from a fact that the satisfaction of an STL formula is computed for a time point, while the match of a TRE results in a time interval. In this work we adapt the definition of TREs from [12] with an assumption of interpreting TREs over discrete time. We reuse definitions of a signal and its projection from the Section 3.1. To adhere to the definition from [12] and to allow negation in TREs, we make the following assumption: for every boolean variable $p_j \in P^n$ we admit a definition of a complementary variable $p_j^\neg$ with an opposite value of p_j (to which we refer as $\neg p_j$). Every analog variable $x_j \in X^m$ is allowed to be used in TREs only in the form of $x_j \sim c$, where $\sim \in \{<, \leq\}$ and $c \in \mathbb{D}$. With every $x_j \sim c$ we associate the boolean satisfaction variable $p_{x_j \sim c}$; we then analogously define $p^\neg_{x_j \sim c}$ and refer to it as $\neg(x_j \sim c)$.

A timed regular expression ψ is defined according to the following syntax [12]:

$$\psi := \epsilon \mid q \mid \psi_1 \cdot \psi_2 \mid \psi_1 \cup \psi_2 \mid \psi_1 \cap \psi_2 \mid \psi^* \mid \langle \psi \rangle_I$$

where q is of the form p, $\neg p$, $x \sim c$ or $\neg(x \sim c)$; I is a time interval $[a, b] \subseteq \mathbb{N}$.

For improved readability, we will refer to discrete time instance $i \cdot T$, where T is discrete time step, simply as i. The semantics of timed regular expression φ with respect to discrete signal w and time instances $i \leq i'$ is given in terms of satisfaction relation $(w, i, i') \models \varphi$:

$$\begin{array}{ll}
(w,i,i') \models \epsilon & \leftrightarrow i = i' \\
(w,i,i') \models q & \leftrightarrow i \leq i' \text{ and } \forall i'' \text{ s.t. } i \leq i'' < i', \pi_p(w)[i''] = 1 \\
(w,i,i') \models \varphi_1 \cdot \varphi_2 & \leftrightarrow \exists i'' \text{ s.t. } i \leq i'' < i', (w,i,i'') \models \varphi_1 \text{ and } (w,i'',i') \models \varphi_2 \\
(w,i,i') \models \varphi_1 \cup \varphi_2 & \leftrightarrow (w,i,i') \models \varphi_1 \text{ or } (w,i,i') \models \varphi_2 \\
(w,i,i') \models \varphi_1 \cap \varphi_2 & \leftrightarrow (w,i,i') \models \varphi_1 \text{ and } (w,i,i') \models \varphi_2 \\
(w,i,i') \models \varphi^* & \leftrightarrow (w,i,i') \models \epsilon \text{ or } (w,i,i') \models \varphi \cdot \varphi^* \\
(w,i,i') \models \langle \varphi \rangle_I & \leftrightarrow i' - i \in I \text{ and } (w,i,i') \models \varphi
\end{array}$$

We reuse the notation $\{a\}$ for intervals of the form $[a, a]$. We introduce the following macros for describing transitions of a boolean signal: `enter(`p`)` $= \langle \neg p \rangle_{\{1\}} \cdot \langle p \rangle_{\{1\}}$ and `exit(`p`)`$= \langle p \rangle_{\{1\}} \cdot \langle \neg p \rangle_{\{1\}}$. We also use a superscript with a TRE to denote a number of concatenations of this TRE (e.g. if ψ is a TRE, then ψ^3 stands for $\psi \cdot \psi \cdot \psi$). Finally, we use ψ^+ as syntactic sugar for $\psi \cdot \psi^*$.

4 Formalization of the SENT Protocol

In this section we introduce the communication protocol under study: the Single Edge Nibble Transmission Protocol (SENT), and then formalize a subset of its electrical and timing requirements.

4.1 Single Edge Nibble Transmission Protocol

The SENT protocol is an industry standard (SAE J2716 [6]) for transmitting data between a sensor and a controller.

SENT communication is unidirectional from a sensor to a controller; the information is partitioned into frames with the structure shown in Fig. 1. The

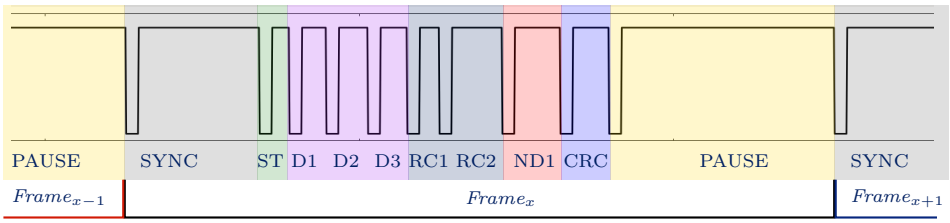

Fig. 1. A SENT frame starts with a mandatory synchronisation pulse (SYNC), followed by a status nibble (ST), data nibbles (D1, D2, D3), rolling counters (RC1, RC2), bit inverse of D1 (ND1), cyclic redundancy check (CRC), and finishes with an optional pause.

transmitted data is split in four-bit data chunks, so-called nibbles, which encode the data in their length. Each nibble has the shape depicted in Fig. 2, where the length of the 'H' region determines the transmitted value (from 0 to 15). In the case study we build runtime monitors for magnetic sensors, which transfer angular information encoded in the three data nibbles D1-D3.

The SAE J2716 standard admits several frame configurations (e.g. the number of data nibbles may vary). SENT devices are configured prior to operation, and the configuration does not change on-the-fly; we take this into account and also assume that the frame structure is static and cannot change at runtime.

To be able to correctly decode sensor data, a controller needs to receive a signal that satisfies electrical and timing requirements of the SENT protocol. We now state these requirements formally, both in STL and TRE and elaborate on checking the frame correctness. Fig. 2 shows a SENT nibble and graphically depicts the requirements to be checked. Table 1 presents in natural language a subset of electrical and timing requirements of the SENT protocol.

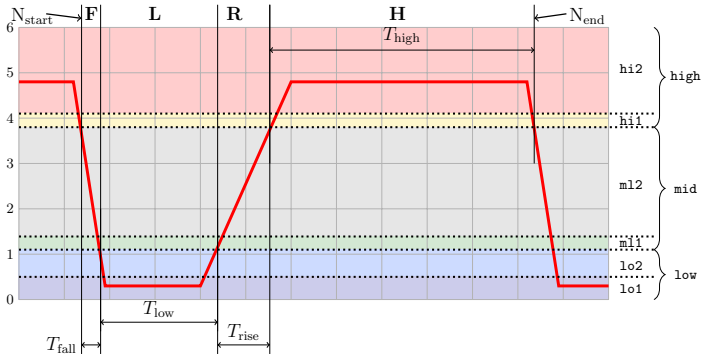

Fig. 2. SENT nibble pulse: A pulse starts (N_{start}) with a falling edge F, followed by a low region L, followed by a rising edge R, followed by a high region H.

4.2 Formalization in STL

Electrical Interface Requirements specify the duration of the slopes, as well as the minimum stable time of the SENT signal. The STL formulae (Eq. 1-4) capture

Table 1. Requirements in natural language

Electrical Interface Requirements		
1	The fall time from V_1 to V_2 must be no longer than T_{fall} μs	F
2	The rise time from V_2 to V_1 must be no longer than T_{rise} μs	R
3	The signal stabilization time below low threshold V_1 or above high threshold V_2 must be at least T_{stable} μs	ST_{low}, L ST_{high}
Transmission Properties of Synchronization & Nibble Pulses		
4	The synchronization pulse shall have a nominal period of 56 clock ticks.	SYNC
5	Five clock ticks of the synchronization pulse shall be driven low.	L
6	All remaining clock ticks of the calibration / synchronization pulse shall be driven high.	SYNC, H_{sync}
7	Five clock ticks of the nibble pulse shall be driven low.	L
8	All remaining clock ticks of the nibble pulse shall be driven high.	NIBBLE, H_{nibble}
9	The minimum pulse period of the nibble pulse shall be 12 clock ticks.	NIBBLE, H_{nibble}
10	The maximum pulse period of the nibble pulse shall be 27 clock ticks.	NIBBLE, H_{nibble}

the temporal order in which the signal should cross voltage regions from Fig. 2. F and R (Eq. 1, 2) are the formal representations of falling and rising time requirements (Tab. 1, Req.1,2). The signal stabilization requirement (Tab. 1,Req.3) is mapped to two STL formulae (Eq. 3, 4) that deal separately with both thresholds. The STL formulae are written using past temporal operators: in this type of formulation a consequent should have happened before an antecedent (i.e. the form "whenever at a time step i φ holds, ψ should have held at $(i - I) \cap \mathbb{T}$").

$$\texttt{F} = \texttt{enter(low)} \rightarrow \texttt{mid}\ \mathcal{S}_{[0,T_{\text{fall}}]}\ \texttt{exit(high)} \tag{1}$$

$$\texttt{R} = \texttt{enter(high)} \rightarrow \texttt{mid}\ \mathcal{S}_{[0,T_{\text{rise}}]}\ \texttt{exit(low)} \tag{2}$$

$$\text{ST}_{\text{low}} = \texttt{exit(low)} \rightarrow \boxminus_{[0,T_{stable}]}\ \texttt{low} \tag{3}$$

$$\text{ST}_{\text{high}} = \texttt{exit(high)} \rightarrow \boxminus_{[0,T_{stable}]}\ \texttt{high} \tag{4}$$

Transmission Properties of Synchronization & Nibble Pulses. The synchronization and the nibble pulse requirements (Tab.1, 4-6 and 7-10 respectively) describe the timing properties these pulses should adhere to. A synchronization pulse has a pre-defined length and is considered as the start of a frame. The shape of synchronization and nibble pulses is to be checked as well (see Fig.2).

To verify the form of the synchronization, nibble, and pause pulses, we split each pulse in regions F, L, R, H (see Fig 2) and check temporal precedence of the regions. The total length of the pulses and the length of the low region L are given in "clock ticks" (Tab. 1, 4-5, 7, 9-10), which are generated by a sensor's internal clock. Let us denote $\delta = (T_{\text{rise}} + T_{\text{fall}})$, then the allowed durations of the H region for the nibble pulse and synchronization pulse are $[7\text{tick} - \delta, 22\text{tick} - \delta]$ and $(51\text{tick} - \delta)$, respectively. Similarly, the length of the H region of the pause pulse is within the following bounds: $[7\text{tick} - \delta, 122\text{tick} - \delta]$.

Requirements for L and H regions can be written directly in past-STL:

$$\texttt{L} = \texttt{exit(low)} \rightarrow \boxminus_{[0,5\texttt{ticks}]}\ \texttt{low} \tag{5}$$

$$\texttt{H}_{\texttt{sync}} = \texttt{exit(high)} \rightarrow \texttt{high}\ \mathcal{S}_{\{51\texttt{tick}-\delta\}}\ \texttt{enter(high)} \tag{6}$$

$$\texttt{H}_{\texttt{nibble}} = \texttt{exit(high)} \rightarrow \texttt{high}\ \mathcal{S}_{[7\texttt{tick}-\delta,\ 22\texttt{tick}-\delta]}\ \texttt{enter(high)} \tag{7}$$

$$\texttt{H}_{\texttt{pause}} = \texttt{exit(high)} \rightarrow \texttt{high}\ \mathcal{S}_{[7\texttt{tick}-\delta,\ 122\texttt{tick}-\delta]}\ \texttt{enter(high)} \tag{8}$$

The general way of capturing precedence relation in STL is by using the bounded until operator $\mathcal{U}_I$. As the authors in [36] show, the hardware implementation of $\mathcal{U}_I$ is not scalable w.r.t. operator time bounds. In order to overcome this issue, we avoid using nested $\mathcal{U}_I$ operators in the formulation, and reformulate the properties. Each SYNC, NIBBLE, and PAUSE patterns of the SENT protocol are the requirements F, L, R, and the corresponding $\texttt{H}_{\{\texttt{sync}|\texttt{nibble}|\texttt{pause}\}}$ requirement put in a sequence. In order to attain efficient hardware implementation, we (i) re-state assertions from $\varphi \rightarrow \psi$ to $\psi \wedge \varphi$, to capture the events when the corresponding requirement has been satisfied; (ii) we then define precedence relation with following macro: $\varphi_1 \texttt{before}_{[t_1,t_2]} \varphi_2 = \varphi_2 \wedge \boxminus_{[0,t_1]} \neg\varphi_1 \wedge \diamondminus_{[t_1,t_2]} \varphi_1$. This allows to use hardware-cheap bounded historically $\boxminus_{[0,t_1]}$ and bounded once $\diamondminus_{[t_1,t_2]}$ operators and significantly reduce hardware resources.

The requirement for NIBBLE is then defined as follows (STL formulae for SYNC and PAUSE are constructed analogously):

$$\texttt{NIBBLE} = (\texttt{F} \wedge \texttt{enter(low)})\ \texttt{before}_{[t_1,t_2]}\ (\texttt{L} \wedge \texttt{exit(low)})\ \texttt{before}_{[t_3,t_4]}\ (\texttt{R} \wedge \texttt{enter(high)})\ \texttt{before}_{[t_5,t_6]}\ (\texttt{H}_{\texttt{nibble}} \wedge \texttt{exit(high)})$$

The top-level FRAME requirement captures precedence relation between SYNC, NIBBLEs, and the PAUSE. The monitor construction is compositional: a frame correctness is reported only when all the lower-level requirements for all the frame components (SYNC, NIBBLEs, PAUSE) are met.

4.3 Formalization in TRE

Although it is possible to formulate TREs in an STL-like style and express the same intent: e.g. the requirements $\texttt{F}^\dagger$ and $\texttt{R}^\dagger$ match falling and rising time intervals of the signal; using the syntax features of the TRE and composing the requirements hierarchically allows to obtain a concise and clear formalization for the requirements of interest. F and R regions (Eq. 9-10) are defined as follows:

$$\texttt{F} = \langle \texttt{mid} \rangle_{[0,T_{\texttt{fall}}]}; \qquad \texttt{F}^\dagger = \texttt{exit(high)} \cdot \langle \texttt{mid} \rangle_{[0,T_{\texttt{fall}}]} \cdot \texttt{enter(low)} \tag{9}$$

$$\texttt{R} = \langle \texttt{mid} \rangle_{[0,T_{\texttt{rise}}]}; \qquad \texttt{R}^\dagger = \texttt{exit(low)} \cdot \langle \texttt{mid} \rangle_{[0,T_{\texttt{rise}}]} \cdot \texttt{enter(high)} \tag{10}$$

The L TRE (Eq. 11) combines the requirements 3 and 5 from Table 1. The H TRE (Eq. 12) will match when the requirement 3 is fulfilled. The two are the necessary building blocks for checking the shape of pulses:

$$\texttt{L} = \langle \texttt{low} \rangle_{[T_{\texttt{stable}},5\texttt{tick}]} \tag{11}$$

$$\texttt{H} = \langle \texttt{high} \rangle_{[T_{\texttt{stable}},127]} \tag{12}$$

We are now able to define the TRE for synchronization, nibble, and pause pulses as a concatenation of regions, restricting the length of the pulses with appropriate time bounds. The SYNC TRE (Eq. 13) will match only when the requirements 1-6 (Tab. 1) are met. The sensor signal will match the NIBBLE TRE (Eq. 14) if the requirements 1-3, 7-10 are fulfilled.

$$\texttt{SYNC} = \langle \texttt{F} \cdot \texttt{L} \cdot \texttt{R} \cdot \texttt{H} \rangle_{\{56\texttt{tick}\}} \tag{13}$$

$$\texttt{NIBBLE} = \langle \texttt{F} \cdot \texttt{L} \cdot \texttt{R} \cdot \texttt{H} \rangle_{[12\texttt{tick},27\texttt{tick}]} \tag{14}$$

$$\texttt{PAUSE} = \langle \texttt{F} \cdot \texttt{L} \cdot \texttt{R} \cdot \texttt{H} \rangle_{[12\texttt{tick},127\texttt{tick}]} \tag{15}$$

The frame and protocol requirements in TRE are formulated as follows:

$$\texttt{SENT\_FRAME} = \texttt{SYNC} \cdot \texttt{NIBBLE}^{8} \cdot \texttt{PAUSE} \tag{16}$$

$$\texttt{SENT\_PROTOCOL} = (\texttt{SENT\_FRAME})^{+} \tag{17}$$

5 Runtime Monitoring with Recovery

A runtime monitor typically partitions the execution traces in those that either satisfy or violate system's specification, possibly providing a quantitative metric of satisfaction (violation). However, for data-driven applications, such as serial protocols, test executions may last for hours and it is required to continue monitoring even after detecting errors. Similarly to compilers, a monitor in such a case must be able to recover after observing a violation, collect the encountered errors, and report them to the user.

For a class of serial protocols, the asynchronous serial protocols (e.g. SENT [6], RS-232 [37], DMX512 [38], etc.), we propose a procedure to construct monitors with *error recovery*. To apply monitoring with recovery, the protocol must fulfil the following requirement: the devices communicate over a single line, where synchronization symbol, control and payload data, respectively, are multiplexed in time. As control signals are absent, the devices rely on the synchronization symbol to successfully capture the beginning of a useful portion of a frame.

By creating runtime monitors with recovery, we are able to: (i) Continue monitoring after detecting violations; (ii) Collect the errors and report them together with their violation type.

5.1 TRE Monitors with Recovery

In the case of asynchronous serial protocols, the devices communicate with sequences that form certain patterns over time; the communication is cyclic, where the data is split in subsequently following frames. These protocols admit a natural formalization in TREs: A frame begins with a unique synchronization pattern (START), followed by n PAYLOAD patterns, and ends with a STOP pattern. The asynchronous serial protocol is then defined as a sequence of frames:

$$\texttt{ASYNC\_SERIAL\_PROTOCOL} = \texttt{FRAME}^{+}, \tag{18}$$

$$\texttt{FRAME} = \texttt{START} \cdot \texttt{PAYLOAD}^{n} \cdot \texttt{STOP}. \tag{19}$$

The above expression exactly generalizes the TRE formalization of the SENT protocol from Section 4.3. It is important to mention that the Kleene star (*) operator should not be used in the specification of `START`, `PAYLOAD` and `STOP` in TREs, as these patterns are finite sequences of symbols; we use the Kleene star operator only at the top TRE (i.e. Eq. 18).

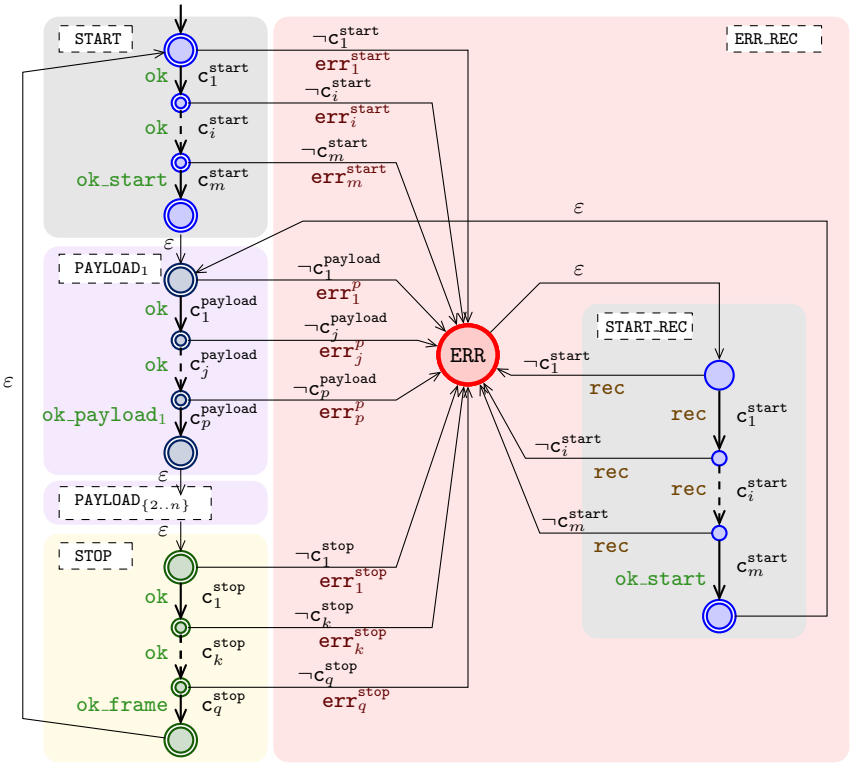

Fig. 3. Monitoring an asynchronous serial protocol with recovery[1]

The sketch of construction procedure for a monitor with recovery is shown in Fig. 3. For each of the `START`, `PAYLOAD`, and `STOP` patterns, we construct the corresponding automata with discrete-time clocks $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, and $\mathcal{A}_{\texttt{stop}}$, respectively. We also create an additional copy of $\mathcal{A}_{\texttt{start}}$, called $\mathcal{A}_{\texttt{rec}}$, which enables the runtime monitor to recover from an error. In this work we take an optimistic approach, and use a weak interpretation of regular expression over finite traces. In case when a trace ends and only a prefix of the regular expression is matched, we decide to accept the input sequence. Therefore all the states in $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, and $\mathcal{A}_{\texttt{stop}}$ are accepting. The automaton-construction procedure from a given TRE, is adopted from [27] to the discrete interpretation of time.

The state transitions are protected by a set C of symbolic transition guards C, where $C = \{c_1^{\texttt{start}}, \ldots, c_m^{\texttt{start}}, c_1^{\texttt{payload}}, \ldots, c_p^{\texttt{payload}}, c_1^{\texttt{stop}}, \ldots, c_q^{\texttt{stop}}\}$.

For each $c_i \in C$ we associate a complementary transition $\neg c_i$ to the global error state. The error state silently transitions to the starting state of the recovery automaton $\mathcal{A}_{\texttt{rec}}$ which consumes garbage symbols until a correct synchronization symbol is observed. The correct `START` pattern is a necessary pre-requisite for a monitor to analyze subsequent frames, and for the decoder to analyze the transferred data: as long as the synchronization symbol of the next frame is not received, the recovery automaton $\mathcal{A}_{\texttt{rec}}$ goes back to the error state.

We introduce a diagnostic variable `out`, defined over a finite set of symbolic values: {`ok`, `ok_start`, $\texttt{ok\_payload}_{1,\ldots,\texttt{N}}$, `ok_frame`, `rec`, $\texttt{err}_{1,\ldots,\texttt{m}}$}. The values have the following meaning: `ok`: the trace has been correct so far; `ok_start`: the starting synchronization symbol has been matched; $\texttt{ok\_payload}_i$: the i^{th} payload symbol has been matched; `ok_frame`: the frame has met all the requirements; `rec`: the monitor is in the recovery state; $\texttt{err}_i$: the specification is violated by an error of type i.

We then transform $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, $\mathcal{A}_{\texttt{stop}}$ and $\mathcal{A}_{\texttt{rec}}$ to transducers $\mathcal{A}'_{\texttt{start}}$, $\mathcal{A}'_{\texttt{payload}}$, $\mathcal{A}'_{\texttt{stop}}$ and $\mathcal{A}'_{\texttt{rec}}$ as follows: (i) For each transition in $\mathcal{A}_{\texttt{i}}$, we output `ok` value; (ii) For each transition leading to a sink state, we output appropriate $\texttt{ok\_}\{\texttt{start|payload|frame}\}$ value; (iii) For each transition guarded by $\neg c_i$ we output $\texttt{err}_i$; (iv) For each recovery automaton transition, except the synchronization symbol matching transition, we associate `rec` value. The transition in $\mathcal{A}'_{\texttt{rec}}$ which matches synchronization symbol outputs `ok_start` (see Fig. 3). For the top-level expression `FRAME`, we create the automaton $\mathcal{A}_{\texttt{frame}}$ by concatenating the $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, and $\mathcal{A}_{\texttt{stop}}$ with ε transitions. This way the user is capable to receive the information about the number of frames that meet the specification, as well as errors and their type.

5.2 STL Monitors with Recovery

The STL monitors are transducers (temporal testers [18]) by construction and are composed hierarchically to output the satisfaction signal of the top-level requirement. The sketch of construction procedure for monitoring with recovery is as follows: (i) we first formalize the `START`, `PAYLOAD`, and `STOP` patterns in STL; (ii) we then change the semantic meaning of STL assertions from (1) $\varphi \rightarrow \psi$ to (2) $\varphi \wedge \psi$: in the first formulation the transducer outputs '1' even if the requirement has never been checked, and '0' when the requirement has been violated (e.g. the F requirement from Sec. 4.2 is fulfilled even the line stays always at '1'); the second case the transducer manifests with the signal the precise time stamp when the requirement has been satisfied (i.e. outputting '1' when the correct falling edge occurred); (iii) for each requirement we identify a set of possible violations and assign an error code $\texttt{err}_i$ to each violation type. Each violation is guarded by an STL assertion $\varphi \wedge \neg\psi \wedge v_i$, where v_i identifies a violation type

[1] For clarity of the presentation, we keep ε-transitions in the Figure 3; these transitions are removed in implementation though keeping the monitor deterministic.

(e.g. `mid` $\mathcal{S}_{[T_{\text{fall}}+1,\infty)}$ `exit(high)` is a v_i clause to capture the violation of the type "too slow falling time" for the STL assertion F from Sec. 4.2).

Finally we check the temporal precedence of the `START`, n `PAYLOAD` sequences and the `STOP` pattern with the `before`$_{[t_1,t_2]}$ macro defined in Sec. 4.2. Using temporal testers allows to monitor all the requirements in parallel, and extending with violation clauses v_i provides the necessary debugging information.

6 Runtime Monitoring of the SENT protocol

This section describes building runtime monitors in FPGA and evaluating the results in industrial environment. A general overview of the framework is followed by implementation and evaluation details.

6.1 From Requirements to Hardware Monitors

Fig. 4 summarizes the process of creating runtime monitors; the proposed framework is not limited to the SENT, and can be applied for other protocols as well.

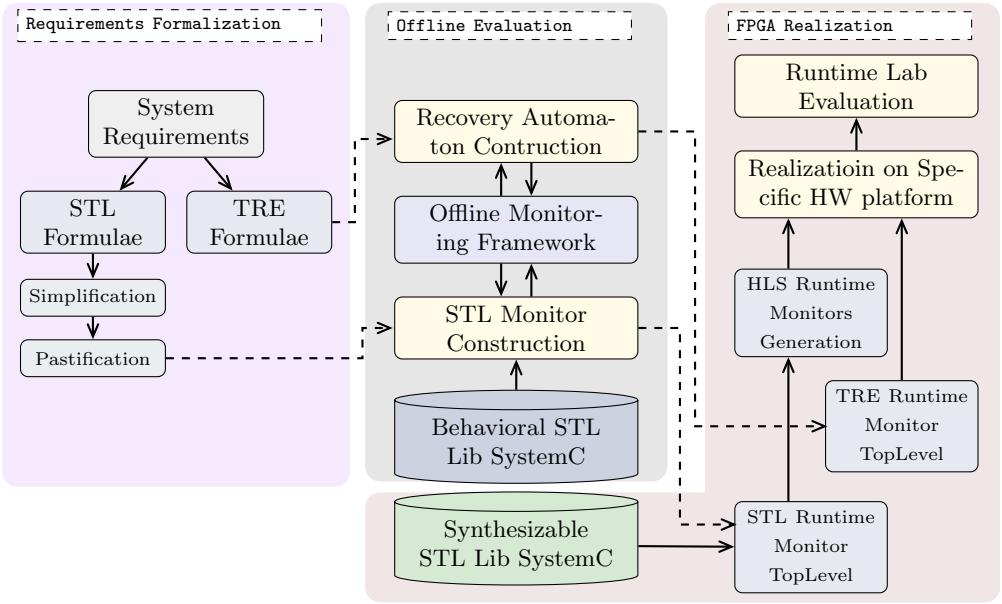

Fig. 4. Monitor Generation

Requirements Formalization. The initial step for creating runtime monitors is to obtain formal representation of the system requirements. Formal semantics allows to eliminate ambiguities in interpretations and precisely define what is to be monitored. In order to evaluate the power of different formalisms, and to eliminate "single source of truth" from the system we use STL and TRE as specification languages. This phase results in a set of formulae (STL & TRE) which describe natural-language requirements.

For STL requirements we admit an automated pre-processing step (see Fig. 4) to obtain formulae that allow efficient hardware realization: on the parse tree of the formula we (i) eliminate duplicate sub-trees (Simplification); (ii) apply a recursive procedure from [35] to convert bounded future STL temporal operators to an equi-satisfiable past operators, resulting in a causal formula with the past temporal operators only (Pastification). The second step is achieved by (i) calculating the temporal depth $\mathcal{D}$ of the formula; (ii) re-writing a formula with past operators which results in postponing a monitoring verdict by $\mathcal{D}$.

Offline Evaluation. In this phase we evaluate monitors offline on short trace fragments, previously recorded from an oscilloscope or an ADC via the Chipscope [39] in order to speed-up debugging and identify implementation bottlenecks.

The monitors for STL formulae are built compositionally from the formula parse tree [18]. With each node of the STL parse tree, which represents either a temporal or a logical operator, we associate a transducer $\mathcal{T}$ which takes as inputs satisfaction signals of its child nodes and outputs the satisfaction signal for the corresponding operator. The satisfaction signal of the root node produces output of the monitor. *Behavioral STL Lib SystemC* (see Fig. 4) is a SystemC implementation of STL transducers, which are used to obtain a monitor. We use SystemC simulation kernel to run the monitor on the pre-recorded traces.

The runtime monitors for the TRE requirements are also implemented in hierarchical fashion: the $\mathcal{A}'_{\texttt{sync}}$, $\mathcal{A}'_{\texttt{nibble}}$, and $\mathcal{A}'_{\texttt{stop}}$ transducers are combined in the top-level recovery automaton described in Section 5.1. We use Vivado Behavioural Simulation to evaluate VHDL code of the top-level $\mathcal{A}'_{\texttt{frame}}$ transducer.

Runtime Monitoring in FPGA is the final phase; the monitors are synthesized in a digital reconfigurable hardware and evaluated in the lab environment. After the off-line phase we obtain the validated monitors for STL and TRE, which follow different paths of hardware implementation.

In case of STL monitoring, we use High-Level Synthesis [40] to generate HDL code for monitors written in SystemC. During the HLS step, the SystemC monitors are transformed to an equivalent synthesizable VHDL or Verilog. We use an alternative implementation of transducers (*Synthesizable STL Lib SystemC*, Fig. 4), which is suitable for HDL code generation. *Behavioral* and *Synthesizable* implementations are functionally equivalent, but HLS imposes constraints on the SystemC code to be hardware-synthesizeable. Keeping *behavioral* and *synthesizable* versions allows quick prototyping using all `C++` features and then produce a hardware-optimized *synthesized* version.

Since transducers $\mathcal{A}'_{\texttt{sync}}$, $\mathcal{A}'_{\texttt{nibble}}$, $\mathcal{A}'_{\texttt{stop}}$, and $\mathcal{A}'_{\texttt{frame}}$ in the TRE approach are implemented in VHDL, we directly use Vivado Synthesis, Logic & Power Optimization, Place & Route tools to obtain a bitstream for FPGA programming.

6.2 FPGA Implementation

We implemented runtime monitors for the SENT protocol in Xilinx Virtex 7 FPGA. The monitors are embedded in the *Line Emulizer* hardware (see Fig. 5), which combines an analog front-end (AFE) capable to interface various sensors

with a high-performance Virtex 7 FPGA. This hardware also models a transmission line with adjustable parameters between a sensor and an ECU.

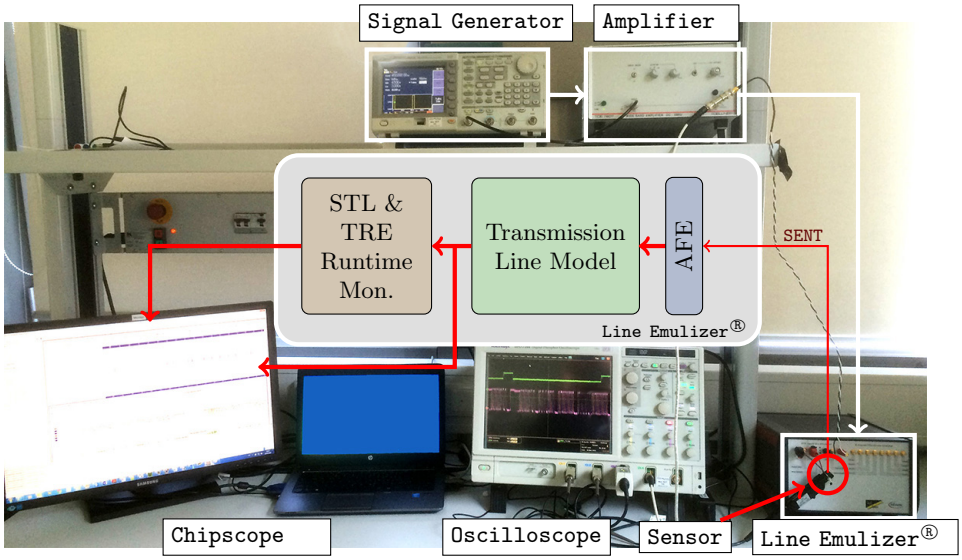

Fig. 5. Runtime Monitoring of the SENT: Hardware Setup

The signal from the SENT sensor (see Fig. 5) comes to the *Line Emulizer*, where it is passed through the AFE and sampled with a high-speed ADC, which results in its finite value representation. During operation in a car, a sensor and an ECU are placed in different locations, hence the sensor signal is affected by a transmission line. To take into account the effects of physical wires, the sensor signal is passed through a digital model of a transmission line. We attach the STL and TRE runtime monitors at the end of the transmission line model (see Fig. 5), to be able to report specification conformance at the receiver side, which is important for proper signal decoding.

The STL and TRE monitors observe at 70 MHz the sensor signal affected by the physical line, calculate verdicts at every clock cycle (i.e. 70 million times per second), and output the result to the user via the Chipscope (Fig. 5). We performed experiments with different models of the line, and conclude that the appropriate line parameters are critical for ensuring the specification compliance. The sensor signal passed through a line with a higher capacitance violates the specification, since the falling and rising times are not met, which can be directly observed from the monitor.

Table 2 reports the estimated FPGA hardware resources (flip-flops, FF & look-up tables, LUT), and the estimated maximum clock period of the runtime monitors. For each HLS-generated monitor we also present its generation time and peak memory usage during HDL-code generation. The monitors in HLS are constructed in a hierarchic fashion, hence the `FRAME` monitor (see Tab. 2) subsumes monitors for other requirements and results the highest hardware footprint. The last row of the Tab. 2 reports the total hardware resources consumed

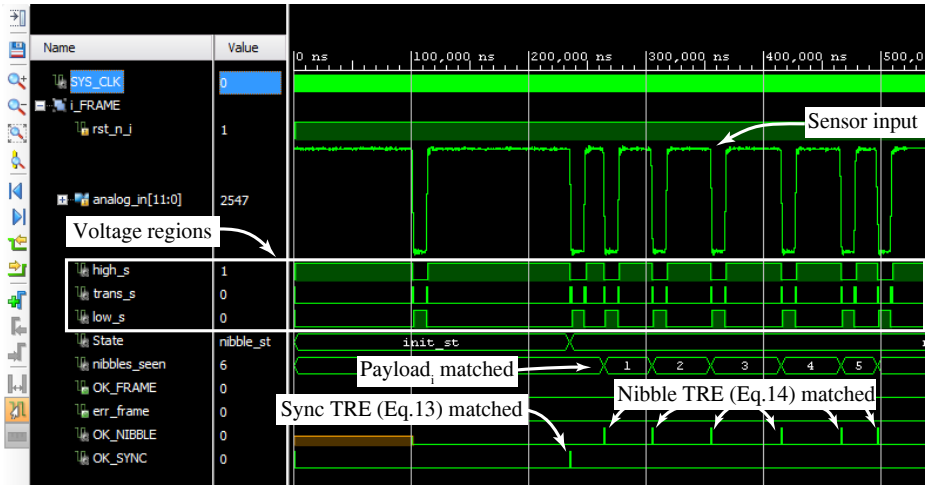

Fig. 6. Runtime TRE monitoring: Vivado functional simulation

by the top-level TRE monitor: the direct hardware implementation results in an order of magnitude lower footprint.

Fig. 6 shows a result of offline evaluation for TRE requirements. The original SENT signal is observed by the monitor, which outputs `OK_NIBBLE`, `OK_SYNC` and the corresponding `ERR` signals. The figure depicts a nominal case, where all the requirements are met.

Runtime monitoring of the SENT signal against STL requirements is shown in the Fig. 7. For this test case the optional pause pulse was deactivated, hence the correct frame is manifested after observing eight correct nibbles (signals `OK_NIBBLE, OK_SYNC, OK_FRAME`). The `OK_NIBBLE` signal is asserted when the corresponding precedence between the requirements `F`, `L`, `R`, and `H` is met. The output of the monitors `F`, `L`, `R`, and `H`, and the corresponding sub-formulae are presented in the lower part of the Fig. 7.

Table 2. STL Monitors Generation: FPGA & HLS resources

Requirement		FF	LUT	Clock	HLS: Time	HLS: Memory
F	HLS	61	118	5.81ns	114.203s	225MB
L		53	85	4.24ns	96.490s	159MB
R		61	113	5.81ns	109.784s	224MB
H_{nibble}		125	249	5.81ns	175.716s	225MB
H_{sync}		28	407	5.81ns	253.507s	224MB
H_{pause}		73	98	4.24ns	162.637s	212MB
NIBBLE		435	1123	7.7ns	394.671s	611MB
SYNC		207	1062	7.7ns	723.690s	605MB
PAUSE		217	710	7.7ns	206.767s	317MB
FRAME		1198	4322	7.7ns	1675.52s	1.39GB
FRAME	TRE	68	350	4.5ns	-	-

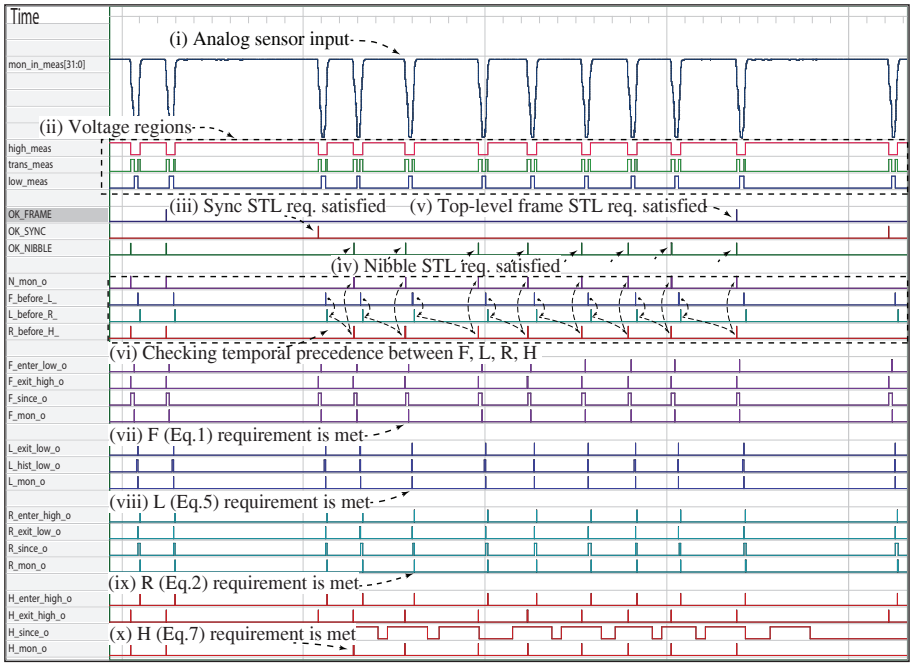

Fig. 7. Runtime monitoring of the STL requirements

7 Conclusion and Outlooks

The case study focuses on assessing STL and TRE for formalizing requirements of the SENT protocol and obtaining hardware monitors for these requirements. We evaluate the two approaches in terms of applicability for formalizing typical protocol requirements, consumption of hardware resources, and monitor reuse.

The hardware resource consumption in Tab. 2 shows that (i) both approaches can be easily mapped to state-of-the-art FPGAs, (ii) STL-based monitors consume an order of magnitude more resources than the TRE monitors. Obtaining hardware monitors based on STL Synthesizable-SystemC library requires an intermediate transformation using HLS, which comes at price of increased hardware footprint. As described in the paper, TREs can be directly translated to automata with recovery which admit efficient hardware realization.

Besides low-level hardware monitoring, which both of the approaches facilitate, SystemC STL monitors can be re-used to check SystemC models. Trace verification in this setting happens during the runtime of the simulation kernel and the monitoring results are obtained at the end of the run. The re-usability of HLS-based monitors though comes at price of FPGA resource consumption.

We found both formalisms applicable for the SENT requirements formalization. TREs allow natural formulation of requirements that are concerned with repetitive sequences of groups of symbols, while formalizing precedence constraints with STL requires in general additional effort to be hardware-efficient.

As it is often the case, specifications comprise both textual and graphical information; we would like to investigate how to combine the information from both representations in a systematic way.

Acknowledgment

This research is supported by the project HARMONIA (845631), funded by a national Austrian grant from FFG (Österreichische Forschungsförderungsgesellschaft) under the program IKT der Zukunft, the EU ICT COST Action IC1402 on Runtime Verification beyond Monitoring (ARVI), the Austrian National Research Network RiSE/SHiNE (S11405-N23 and S11412-N23) project funded by the Austrian Science Fund (FWF) and the Fclose (Federated Cloud Security) project funded by UnivPM. This manuscript benefited from comments of Alena Rodionova, who we kindly acknowledge.

References

1. *ISO 26262: "Road vehicles Functional safety"*. International Organization for Standardization (ISO), 2011.
2. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
3. Manfred Broy, Helmut Krcmar, Sascha Kirstan, and Bernhard Schätz. What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry? In *Emerging Technologies for the Evolution and Maintenance of Software Models*, pages 310–337, 2012.
4. Martin Leucker. Teaching Runtime Verification. In Sarfraz Khurshid and Koushik Sen, editors, *Proc. of Runtime Verification: Second International Conference, RV 2011*, pages 34–48. Springer Berlin Heidelberg, 2012.
5. Ezio Bartocci, Yliès Falcone, Borzoo Bonakdarpour, Christian Colombo, Normann Decker, Klaus Havelund, Yogi Joshi, Felix Klaedtke, Reed Milewicz, Giles Reger, Grigore Rosu, Julien Signoles, Daniel Thoma, Eugen Zalinescu, and Yi Zhang. First international competition on runtime verification: rules, benchmarks, tools, and final results of CRV 2014. *International Journal on Software Tools for Technology Transfer*, pages 1–40, 2017.
6. SAE International. SENT - Single Edge Nibble Transmission for Automotive Applications, J2716, Standard. `http://standards.sae.org/j2716_201001/`, 2016. [Online; Accessed 21-January-2017].
7. Dejan Nickovic and Oded Maler. Amt: A property-based monitoring tool for analog systems. In Jean-Franois Raskin and P.S. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems*, volume 4763 of *Lecture Notes in Computer Science*, pages 304–319. Springer Berlin Heidelberg, 2007.
8. Cindy Eisner. PSL for Runtime Verification: Theory and Practice. In *Runtime Verification, 7th International Workshop, RV 2007, Vancouver, Canada, March 13, 2007, Revised Selected Papers*, pages 1–8, 2007.
9. Srikanth Vijayaraghavan and Meyyappan Ramanathan. *A Practical Guide for SystemVerilog Assertions*. Springer Publishing Company, Incorporated, 2014.
10. Alexandre Donze, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott Smolka. On Temporal Logic and Signal Processing. In *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, pages 92–106. Springer Berlin Heidelberg, 2012.

11. Oded Maler and Dejan Ničković. Monitoring properties of analog and mixed-signal circuits. *International Journal on Software Tools for Technology Transfer*, 15(3):247–268, 2013.
12. Thomas Ferrère, Oded Maler, Dejan Ničković, and Dogan Ulus. *Measuring with Timed Patterns*, pages 322–337. Springer International Publishing, Cham, 2015.
13. Ebru Aydin-Gol, Ezio Bartocci, and Calin Belta. A formal methods approach to pattern synthesis in reaction diffusion systems. In *Proc. of CDC 2014: the 53rd IEEE Conference on Decision and Control*, pages 108–113. IEEE, 2014.
14. Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Grosu, and Calin Belta. SpaTeL: a novel spatial-temporal logic and its applications to networked systems. In *Proc. of HSCC'15: the 18th International Conference on Hybrid Systems: Computation and Control*, pages 189–198. IEEE, 2015.
15. Deian Tabakov, Kristin Y. Rozier, and Moshe Y. Vardi. Optimized temporal monitors for systemc. *Formal Methods in System Design*, 41(3):236–268, 2012.
16. Johann Schumann, Patrick Moosbrugger, and Kristin Y. Rozier. *R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems*, pages 233–249. Springer International Publishing, Cham, 2015.
17. M. Boule and Z. Zilic. Efficient automata-based assertion-checker synthesis of psl properties. In *2006 IEEE International High Level Design Validation and Test Workshop*, pages 69–76, 2006.
18. A. Pnueli and A. Zaks. On the Merits of Temporal Testers. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 172–195. Springer Berlin Heidelberg, 2008.
19. Kevin D. Jones, Victor Konrad, and Dejan Nickovic. Analog property checkers: a DDR2 case study. *Formal Methods in System Design*, 36(2):114–130, 2010.
20. Thang Nguyen and Dejan Nickovic. Assertion-based monitoring in practice - checking correctness of an automotive sensor interface. *Sci. Comput. Program.*, 118:40–59, 2016.
21. Ezio Bartocci and Pietro Liò. Computational modeling, formal analysis, and tools for systems biology. *PLoS Computational Biology*, 12(1), 2016.
22. Ezio Bartocci, Luca Bortolussi, and Laura Nenzi. A temporal logic approach to modular design of synthetic biological circuits. In *Proc. of CMSB 2013: the 11th International Conference on Computational Methods in Systems Biology*, volume 8130 of *LNCS*, pages 164–177. Springer, 2013.
23. Sara Bufo, Ezio Bartocci, Guido Sanguinetti, Massimo Borelli, Umberto Lucangelo, and Luca Bortolussi. Temporal logic based monitoring of assisted ventilation in intensive care patients. In *Proc. of ISoLA 2014: the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Part II*, volume 8803 of *LNCS*, pages 391–403. Springer, 2014.
24. Thomas Reinbacher, Kristin Yvonne Rozier, and Johann Schumann. Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems - 20th Int. Conf., (TACAS), Grenoble, France*, pages 357–372, 2014.
25. Anat Dahan, Daniel Geist, Leonid Gluhovsky, Dmitry Pidan, Gil Shapir, Yaron Wolfsthal, Lyes Benalycherif, Romain Kamdem, and Younes Lahbib. Combining system level modeling with assertion based verification. In *6th International Symposium on Quality of Electronic Design (ISQED) 21-23 March 2005, San Jose, CA, USA*, pages 310–315, 2005.
26. Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking Temporal Properties of Discrete, Timed and Continuous Behaviors. In *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 475–505. Springer Berlin Heidelberg, 2008.

27. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.
28. Thomas Reinbacher, Matthias Függer, and Jörg Brauer. Real-time runtime verification on chip. In *Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, Revised Selected Papers*, pages 110–125, 2012.
29. Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop, Ruttgers University, NJ, USA*, pages 232–243, 1995.
30. Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.
31. Thang Nguyen and Dejan Nickovic. Assertion-Based Monitoring in Practice Checking Correctness of an Automotive Sensor Interface. In *Proc. of Formal Methods for Industrial Critical Systems*, pages 16–32. Springer, 2014.
32. Georgios E. Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using S-TaLiRo. In *American Control Conference, ACC 2012, Montreal, QC, Canada*, pages 3567–3572, 2012.
33. Konstantin Selyunin, Thang Nguyen, Ezio Bartocci, and Radu Grosu. *Applying Runtime Monitoring for Automotive Electronic Development*, pages 462–469. Springer International Publishing, 2016.
34. Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to Timed Automata. In Eugene Asarin and Patricia Bouyer, editors, *Proc. of Formal Modeling and Analysis of Timed Systems*, volume 4202 of *Lecture Notes in Computer Science*, pages 274–289. Springer Berlin Heidelberg, 2006.
35. Oded Maler, Dejan Nickovic, and Amir Pnueli. On Synthesizing Controllers from Bounded-Response Properties. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 95–107. Springer Berlin Heidelberg, 2007.
36. Stefan Jaksic, Ezio Bartocci, Radu Grosu, Reinhard Kloibhofer, Thang Nguyen, and Dejan Nickovic. From Signal Temporal Logic to FPGA Monitors. In *Proc. of 13. ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pages 218–227, 2015.
37. Jan Axelson. *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems; 2nd ed.* Lakeview Research, Madison, WI, 2007.
38. ANSI E1.11-2008 (R2013). Entertainment Technology – USITT DMX512-A – Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories . http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+E1.11-2008+(R2013), 2008. [Online; Accessed 20-January-2017].
39. Xilinx Inc. Vivado Design Suite Tutorial, Programming and Debugging. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug936-vivado-tutorial-programming-debugging.pdf, 2016. [Online; Accessed 12-January-2017].
40. Xilinx Inc. Vivado High-Level Synthesis. http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html *(Accessed 18.01.2017)*.