From Signal Temporal Logic to FPGA Monitors

Stefan Jakšić*†, Ezio Bartocci†, Radu Grosu†, Reinhard Kloibhofer*, Thang Nguyen‡ and Dejan Ničković*
*AIT Austrian Institute of Technology, Austria
†Faculty of Informatics, Vienna University of Technology, Austria
‡Infineon Technologies AG, Austria

***Abstract*—**

Due to the heterogeneity and complexity of systems-of-systems (SoS), their simulation is becoming very time consuming, expensive and hence impractical. As a result, design simulation is increasingly being complemented with more efficient design emulation. Runtime monitoring of emulated designs would provide a precious support in the verification activities of such complex systems.

We propose novel algorithms for translating signal temporal logic (STL) assertions to hardware runtime monitors implemented in field programmable gate array (FPGA). In order to accommodate to this hardware specific setting, we restrict ourselves to past and bounded future temporal operators interpreted over discrete time. We evaluate our approach on two examples: the mixed signal bounded stabilization property and the serial peripheral interface (SPI) communication protocol. These case studies demonstrate the suitability of our approach for runtime monitoring of both digital and mixed signal systems.

I. Introduction

Modern system-of-systems (SoS) merge a number of embedded elements that are often developed independently. Such components are heterogeneous, and combine digital controllers with analogue sensors and actuators. They interact with their physical environment and are interconnected via communication protocols. This results in complex interactions generating emergent behaviors that are not predictable in advance. Correct system integration is crucial to achieve high standards with respect to safety and security. For instance, the ISO 26262 [15] standard from the automotive domain obliges suppliers to provide sufficient correctness evidence about their systems to the regulatory bodies.

Due to the heterogeneity and the complexity of modern SoS, verification and validation (V&V) poses a major challenge and represents today the main bottleneck in the design process. The industrial V & V process includes the following workflow of activities. The pre-silicon verification covers all types of simulation-based tasks at different levels of design abstraction using techniques such as the mixed-signal and mixed-abstraction simulation. The post-silicon verification refers to the verification of the real integrated circuit (IC) in the lab. It extends the pre-silicon verification by covering the full design functionality and admitting much longer test scenarios which are impractical or impossible to simulate.

Despite the fact that verification methodologies are well-established in the industry, they still involve many manual tasks. Verification engineers need to create input stimuli, execute simulation models and observe the correctness of the output waveforms. Consequently, this process is repetitive, tedious and time-consuming. It is widely accepted that for complex mixed-signal designs, the verification activities account for 60%-70% of the total project development. Hence, any approach which would boost up the verification and improve time-to-market and product quality is of extreme interest.

We propose to improve this situation by combining *assertion-based runtime verification* with design *emulation* on field programmable gate array (FPGA), an integrated circuit that can be configured by the designer. Assertion-based runtime verification is a rigorous, yet practical method for checking design correctness.

Design emulation uses FPGA with mixed-signal test chip as an early prototype for stress and data transmission test scenarios executed over a long period of time [19]. It complements the pre-silicon verification phase by enabling design exploration beyond the limits introduced by simulation-based methods. Design emulation is used both to explore the behavior of digital and analog components. In the latter case, the (possibly mixed signal) component can be approximated with its discretized behavioral model. By combining these two approaches, we provide a rigorous method for runtime verification of long executions resulting from mixed signal design emulations. In addition to design emulations, our proposed solution can be used to monitor real mixed-signal devices in post-silicon validation in real-time.

We choose Signal Temporal Logic (STL) [16] as our specification language. STL allows describing complex timing relations between digital and analog "events", where the latter are specified via numerical predicates. In this paper, we restrict ourselves to the rich subset of STL that contains *past* and *bounded future* operators. Due to the FPGA hardware context, we interpret STL formulas over the discrete time.

We propose a compositional construction for STL monitors implemented on FPGA. An FPGA operates at a given maximum frequency and contains *look-up tables* (LUT) that can be used to implement complex combinatorial functions and flip-flops (FF) that are used as memory elements. Given its limited speed and computational resources, it is imperative to implement STL monitors efficiently. In order to achieve this goal, we use a simple architecture for our STL monitors, ensuring that we minimize the usage of the resources. We base the STL monitor generation on *temporal testers* [20]. Intuitively, a tester for an STL formula φ is a *transducer* that observes an execution trace w and outputs true at time t if and only if w satisfies φ at time t. In order to decide the satisfaction of the formula at runtime, we restrict ourselves to *deterministic* testers, which have a natural translation to sequential circuits.

We use analog-to-digital converters (ADC) to handle nu-

978-1-5090-0237-5/15/$31.00 ©2015 IEEE

merical predicates in STL. An ADC is a device that periodically transforms real-valued quantities to their digital number approximations. We then apply the predicate operations on the quantized values of the input signal. The bounded future STL formulas do not directly admit deterministic testers - the satisfaction of φ at t depends on inputs at some future $t' > t$. Inspired by [18], we propose a procedure to transform the bounded future STL formula φ to an equisatisfiable past STL formula ψ. The two formulas are related as follows - φ is satisfied at time t if and only if ψ is satisfied at time $t+b$, where b is the bounded future horizon of φ. It follows that instead of monitoring φ, it is sufficient to monitor ψ and delay the verdict by b time units. We adapt the procedure in [17] to the discrete time setting and directly translate the past fragment of STL to deterministic testers.

We implement the entire STL monitor generation and deployment flow - from formal specifications to the lab environment. We demonstrate our approach on two case studies coming from both the digital and the mixed signal domain: (1) the bounded stabilization: and (2) the serial peripheral interface (SPI) communication protocol. We summarize our main contributions as follows:

- We focus on monitoring both digital and mixed signal designs;
- We provide the hardware monitoring procedure which handles bounded future temporal logic formulas by transforming them into their past counterparts evaluated with a fixed delay;
- We implement and present the entire flow from specifications to hardware monitors and evaluate our approach in a real lab environment.

II. From Signal Temporal Logic to Hardware Monitors

A. Signals, Signal Temporal Logic and Temporal Testers

In this paper, we study Signal Temporal Logic (STL) with both *past* and *future* operators in the context of the runtime verification problem, in which the monitors are implemented on FPGA hardware. STL is a specification language that allows to express real time requirements of mixed-signal behaviors, such as the following mixed-signal stabilization property.

Example 1: Consider the mixed-signal bounded stabilization property, depicted in Figure 1, with the following requirements:

- The absolute value of the continuous signal x is always smaller than 5;
- When the Boolean *trigger* signal rises, within 600 time units the absolute value of x has to drop below 1 and remain below this threshold for at least 300 time units.

In order to accommodate the specification language to this particular domain of application, we interpret STL over discrete time and finite valued domains. We define a *signal* w as a total function $w : [0,\delta] \to \mathbb{B}^m \times \mathbb{D}^n$, where $[0,\delta] \subseteq \mathbb{N}$ denotes the discrete time domain and $\mathbb{D}$ is a finite valued domain. We denote by $|w| = \delta$ the length of the signal w.

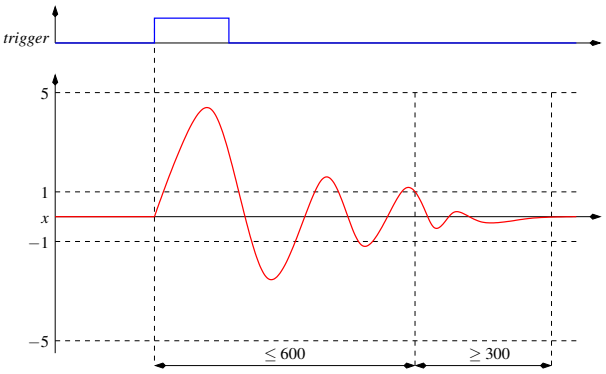

Fig. 1. Example: bounded stabilization property.

Let $P = \{p_1, \ldots, p_m\}$ be the set of boolean variables and $X = \{x_1, \ldots, x_n\}$ the set of variables defined over $\mathbb{D}$. We denote by $\pi_e(w)$ the projection of w to $e \in P \cup X$. Given a signal w and its projection $\pi_p(w)$ to some $p \in P$, we say that $\pi_p(w)$ has (Δ, l)-variability if within every interval $[i, i+\Delta-1]$, the value of p changes at most l times.

The syntax of a STL formula φ over $P \cup X$ is defined by the grammar

$$\varphi \quad := \quad p \mid x \sim u \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathcal{U}_I \varphi_2 \mid \varphi_1 \, \mathcal{S}_I \varphi_2$$

where $p \in P$, $x \in X$, $\sim \in \{<, \leq\}$, $u \in \mathbb{D}$, I is of the form $[a,b]$ or $[a,\infty)$ such that $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. For intervals of the form $[a,a]$, we will use the notation $\{a\}$ instead.

The semantics of a STL formula with respect to a signal w is described via the satisfiability relation $(w,i) \models \varphi$, indicating that the signal w satisfies φ at the time index i, according to the following definition where $\mathbb{T} = [0, |w|]$.

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

Example 2: We now formalize in STL the requirements from Example 1 with the following formula.

$$\square\,(|x| < 5 \wedge (\mathit{trigger} \rightarrow \diamondsuit_{[0,600]} \square_{[0,300]} |x| < 1))$$

From the basic definition of STL, we can derive the following standard operators.

$$\begin{array}{lcl}
\mathsf{true} & = & p \vee \neg p \\
\mathsf{false} & = & \neg\mathsf{true} \\
\varphi_1 \wedge \varphi_2 & = & \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\diamondsuit_I \varphi & = & \mathsf{true} \, \mathcal{U}_I \varphi \\
\square_I \varphi & = & \neg \diamondsuit_I \neg\varphi \\
\boxminus_I \varphi & = & \mathsf{true} \, \mathcal{S}_I \varphi \\
\boxminus_I \varphi & = & \neg \boxminus_I \neg\varphi
\end{array}$$

We give strict semantics to the $\mathcal{U}_I$ and $\mathcal{S}_I$ operators from which we can derive classical future and past LTL operators[1], using the following rules:

$$\begin{aligned}
\varphi_1 \,\mathcal{U}\, \varphi_2 &= \varphi_2 \vee (\varphi_1 \wedge \varphi_1 \,\mathcal{U}_{[1,\infty)} \varphi_2) \\
\varphi_1 \,\mathcal{S}\, \varphi_2 &= \varphi_2 \vee (\varphi_1 \wedge \varphi_1 \,\mathcal{S}_{[1,\infty)} \varphi_2) \\
\bigcirc \varphi &= \mathsf{false}\, \mathcal{U}_{\{1\}} \varphi \\
\ominus \varphi &= \mathsf{false}\, \mathcal{S}_{\{1\}} \varphi
\end{aligned}$$

Given a STL formula φ defined over P, we denote by P_φ the set of all subformulas of φ that also includes P and that assigns to every sub-formula ψ of φ a boolean variable[2]. Given a signal w over $P \cup X$ and a formula φ, we define the *satisfaction signal* w_φ over $P_\varphi \cup X$ such that for all $\psi \in P_\varphi$ and $i \in [0, \delta]$, $\pi_{w_\varphi}(\psi)(i) = \mathsf{true}$ iff $(w,i) \models \psi$.

We also define two useful subsets of STL; (1) *past* STL which forbids the usage of the $\mathcal{U}_I$ operator; and (2) *bounded future* STL which restricts the usage of the $\mathcal{U}_I$ operator to the case where $I = [a,b]$ for some $0 \leq a \leq b < \infty$. From now on, we will use STL of the form $\square\, \varphi$, where $\varphi := \varphi_p \mid \varphi_f \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$, where φ_p contains only past and φ_f only bounded future temporal operators. We simply refer to this fragment as STL in the rest of the paper when clear from the context.

B. Temporal Testers

We now present the concept of *temporal testers* we will use in the paper to monitor signals. Pnueli et al. introduced temporal testers in [20] showing how to build monitors in a modular way starting from the syntax tree structure of the temporal logic formula. A temporal tester is a transition system that behaves as a transducer to compute the satisfaction signal of a STL formula.

A *transition system* (TS) T is the tuple $(Q, \hat{q}, V, \mathcal{T}, \lambda)$, where

- Q is a finite set of *locations*,
- $\hat{q} \in Q$ is an *initial* location,
- V is a finite set of variables, where each variable $v \in V$ is defined over some finite domain $\mathbb{D}$,
- $\mathcal{T} \subseteq Q \times Q$ is the *transition relation*, and
- $\lambda : \mathcal{T} \to \mathcal{B}(V \cup V')$ is a labeling function that assigns to each transition a boolean constraint over the current and next state variables, where $V' = \{v' \mid v \in V\}$.

We denote by $s : V \to \mathbb{D}$ the *state* over V. Given a set of variables $V \cup V'$, a state function s and a boolean constraint $\theta \in \mathcal{B}(V \cup V')$, we say that (s, s') satisfies θ, denoted by $(s, s') \models \theta$ if by replacing all $v \in V$ and $v' \in V'$ that appear in θ by $s(v)$ and $s'(v')$ respectively, the constraint evaluates to true. A finite *run* r of T is the sequence of locations $q_0, q_1, \ldots, q_n$ that satisfies the following requirements:

1) q_0 is the *initial location*, and
2) for all $0 \leq i \leq n-1$, $(q_i, q_{i+1}) \in \mathcal{T}$.

A finite *trace* σ over V is a sequence $\sigma = s_0, s_1, \ldots, s_n$ of its states. We say that σ is a trace of T if there exists a run $r = q_0, q_1, \ldots, q_{n+1}$ of T such that for all $0 \leq i \leq n$, $(s_i, s_{i+1}) \models \lambda(q_i, q_{i+1})$. We say that T is *deterministic* if $(q,q'), (q,q'') \in \mathcal{T}$ such that $q' \neq q''$ implies that $\lambda(q,q') \wedge \lambda(q,q'')$ is unsatisfiable.

Given two TS's T_1 and T_2, where $T_i = (Q_i, \hat{q}_i, V_i, \mathcal{T}_i, \lambda_i)$, we define their *parallel composition* $T = T_1 \mid\mid T_2$ as the TS $(Q, \hat{q}, V, \mathcal{T}, \lambda)$, where

- $Q = Q_1 \times Q_2$,
- $\hat{q} = (\hat{q}_1, \hat{q}_2)$,
- $V = V_1 \cup V_2$,
- $\mathcal{T} = \{((q_1,q_2),(q_1',q_2')) \mid (q_1,q_1') \in \mathcal{T}_1 \text{ and } (q_2,q_2') \in \mathcal{T}_2\}$ and $\lambda((q_1,q_2),(q_1',q_2')) = \lambda_1(q_1,q_1') \wedge \lambda_2(q_2,q_2')$.

The *basic temporal tester* for a formula φ is a TS T_φ with $P_\varphi \subseteq V_\varphi$ that satisfies the following condition - for every trace $s_0, s_1, \ldots, s_n$ of T_φ, and w such that for all $p \in P$ and $i \in [0,n]$ $s_i(p) = \pi_p(w)(i)$, $s_i(\varphi) = \mathsf{true}$ iff $(w,i) \models \varphi$. The *full temporal tester* (or simply temporal tester) $T_{\{\varphi\}}$ for an arbitrary temporal formula φ is the parallel composition $||_{\psi \in cl(\varphi)} T_\psi$ of the basic temporal testers for its sub-formulas.

C. Hardware Monitor Synthesis

In this section, we present the algorithms for translating STL specifications into deterministic temporal testers that can be synthesized on FPGA hardware. In order to solve this problem, we need to address the following challenges: (1) implement numerical predicates over real-valued signal; (2) provide a translation into memory-efficient monitors from timed properties; and (3) find an appropriate approach for evaluating properties with bounded future operators.

Numerical predicates over real-valued signals are rather an implementation issue - hence we will discuss them in more detail in Section III.

We achieve the translation from real-time STL specifications into memory-efficient monitors by exploiting the *bounded variability* property of STL timed operators. Similarly to (Δ, l)-variable signals, we say that a temporal operator has (Δ, l)-variability if within any interval of size Δ its satisfaction value can change l times at most. Instead of remembering the satisfaction status of the formula at every cycle, this property allows us to record only the changes in the satisfaction of the formula.

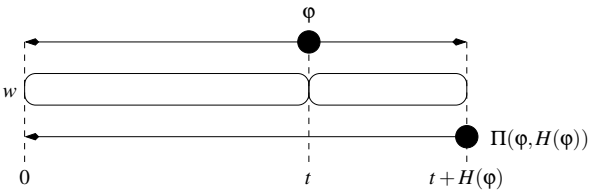

Fig. 2. From bounded future φ to past $\Pi(\varphi, b)$.

Finally, we handle monitoring of STL specifications with bounded future operators by transforming them into *equi-satisfiable* STL formulas that contain only past operators.

[1]Note that in discrete time, STL and LTL with boolean predicates over $\mathbb{D}$ have the same expressive power, however treating temporal operator with bounds directly leads in general to more efficient monitoring algorithms.

[2]We abuse the notation and use the same symbol to denote the formula and its associated boolean variable.

This transformation eliminates the "predictive" aspect of the original formula by changing the time direction from future to past. This is possible because the formula refers only to a statically pre-computable bounded future horizon. Hence, its evaluation can be delayed until the horizon is reached. Figure 2 illustrates the transformation Π from a bounded future STL formula φ into its equisatisfiable past STL counterpart $\Pi(\varphi, H(\varphi))$, where $H(\varphi)$ is the time horizon of φ.

1) Monitoring Past STL *Specifications:* In order to handle arbitrary STL formulas, we will implement them as deterministic temporal testers. For the past STL, we first observe that directly treating $\mathcal{S}_I$ and $\diamondsuit_I$ may not be straightforward - we use instead the following equivalences to first simplify the formulas.

$$\begin{aligned}
\varphi_1 \,\mathcal{S}_{[0,b]}\, \varphi_2 &= (\varphi_1 \,\mathcal{S}\, \varphi_2) \wedge \diamondsuit_{[0,b]}\, \varphi_2 \\
\varphi_1 \,\mathcal{S}_{[a,b]}\, \varphi_2 &= \boxminus_{[0,a-1]} (\varphi_1 \wedge \ominus (\varphi_1 \,\mathcal{S}\, \varphi_2)) \wedge \diamondsuit_{[a,b]}\, \varphi_2 \\
\diamondsuit_{[a,b]}\, \varphi &= \diamondsuit_{\{a\}}\, \diamondsuit_{[0,b-a]}\, \varphi
\end{aligned}$$

As a result, we need to only build testers for the $\ominus$, $\mathcal{S}$, $\diamondsuit_{[0,b]}$ and $\diamondsuit_{\{a\}}$ temporal operators. In what follows, we describe the algorithms for building basic temporal testers for each of these operators.

a) $\ominus$ *and* $\mathcal{S}$ *operators [14]:* the temporal tester for $\psi = \ominus \varphi$ must satisfy $\neg\psi'$ in the first step, and in every following step executes the transition labeled by $\psi' \leftrightarrow \varphi$. We notice that $\psi = \varphi_1 \,\mathcal{S}\, \varphi_2$ is equivalent to the formula $\varphi_2 \vee (\varphi_1 \wedge \ominus (\varphi_1 \,\mathcal{S}\, \varphi_2))$. It follows that the temporal tester for ψ must satisfy $\varphi_2' \leftrightarrow \psi'$ is the first step and $\psi' \leftrightarrow (\varphi_2' \vee (\varphi_1' \wedge \psi))$ in every following step. Both temporal testers require a single memory element.

b) $\diamondsuit_{[0,a]}$ *operator:* the temporal tester for $\psi = \diamondsuit_{[0,a]}\, \varphi$ uses a single counter c bounded by $a+1$ to implement a discrete time clock[3] and works as follows. The tester observes the satisfaction of φ over time and moves through its locations, generating the output that follows the satisfaction relation of the operator. Whenever the tester observes the satisfaction of φ, the output of the tester must trivially satisfy ψ. Whenever the tester detects a falling edge in the satisfaction of φ, the counter c is reset to 0. As long as φ is violated and the counter c is smaller or equal to a, the counter is incremented and the output must still satisfy ψ - the property is satisfied since the last observation where φ was true lies within the previous $[0,a]$ interval. If the tester still observes violation of φ while the counter c reaches $a+1$, it must satisfy the output $\neg\psi$, indicating the violation of the formula. The temporal tester for $\diamondsuit_{[0,a]}$ requires a single $\lceil \log_2(a) \rceil$-bit counter.

c) $\diamondsuit_{\{a\}}$ *operator:* we first note that the operator $\diamondsuit_{\{a\}}\, \varphi$ simply shifts the satisfaction of φ by a time steps, i.e. $(w,i) \models \varphi$ if and only if $(w,i+a) \models \diamondsuit_{\{a\}}\, \varphi$. There is a very simple implementation of this formula by observing the following equivalence

$$\diamondsuit_{\{a\}}\, \varphi = \underbrace{\ominus \ominus \ldots \ominus}_{a} \varphi$$

[3]From now on, we refer to the clocks from timed automata terminology as counters, in order to avoid confusion with system clock signals in hardware.

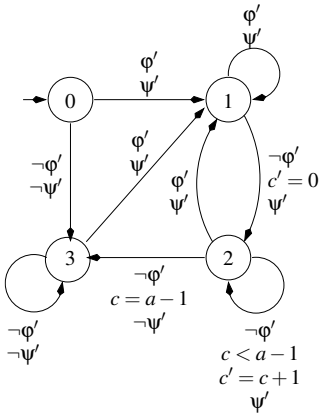

Fig. 3. Temporal tester for $\psi = \diamondsuit_{[0,a]}\, \varphi$.

Despite its simplicity, this direct implementation for the $\diamondsuit_{\{a\}}\, \varphi$ requires a memory registers and may not be optimal if a is large and φ has bounded variability. Hence, we sketch an alternative algorithm, illustrated in Figure 4, for building a temporal tester for $\psi = \diamondsuit_{\{a\}}\, \varphi$ when φ has (Δ, l)-variability. Instead of recording the last a values of φ, we instead memorize only the relative times of the last l changes in φ by using discrete counters. In the first a steps, the tester must satisfy the output $\neg\psi$. Whenever a change is observed in φ, a fresh discrete counter c is reset to 0 and incremented in the next a steps. When the counter reaches a, the tester enforces the same change in ψ. The number of active counters required at any point in time is dependent on the variability of the input signal and the bound a - whenever an active counter reaches a, it is deactivated and can be reused. The implementation of this algorithm with discrete counters requires at most $\lceil \frac{a \cdot l}{\Delta} \rceil \cdot \lceil \log_2 a \rceil$ bits.

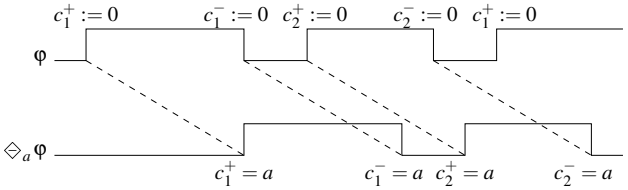

Fig. 4. Computing $\diamondsuit_a\, \varphi$ with discrete counters.

We remind the reader that we use $\diamondsuit_{[a,b]}\, \varphi = \diamondsuit_{\{a\}}\, \diamondsuit_{[0,b-a]}\, \varphi$ to decompose the monitoring of arbitrary once operators. The satisfaction signal $\psi = \diamondsuit_{[0,b-a]}\, \varphi$ has the $(b-a+1, 2)$-variability. Hence, the algorithm that uses counters needs $\lceil \frac{2 \cdot a}{b-a+1} \rceil \cdot \lceil \log_2 a \rceil$ bits to implement the operator. It follows that the direct implementation with the shift registers is more optimal when the input signal has high variability, while the algorithm with the counters works better with low-varying signals. For instance, the direct implementation of $\diamondsuit_{\{500\}}\, p$ requires 500 registers, while the algorithm with counters needs 500 counters, each having $\lceil \log_2 500 \rceil = 9$ bits. On the other hand, the direct implementation of $\diamondsuit_{[500,1000]}\, \varphi = \diamondsuit_{\{500\}}\, \diamondsuit_{[0,500]}\, \varphi$ requires 500 bits, while the algorithm using counters needs only $2 \cdot \lceil \log_2 500 \rceil = 18$ bits. The optimal implementation choice

for the $\diamondsuit_{\{a\}}$ operator can be easily automated by doing a syntactic analysis of the formula.

2) From Bounded Future to Past STL *Specifications:* We now address the problem of developing monitors for bounded future STL formulas. The challenge for monitoring such formulas comes from the fact that the satisfaction at time index i depends on the inputs that will become available only in some future (but bounded) horizon $[i, i+h]$. The bound of the future horizon can be syntactically computed from the specification, as shown in [11], [18]. In order to solve this problem, we adapt the *pastification* procedure from [18], which transforms the bounded future specification φ into an *equisatisfiable* past specification ψ that is evaluated with the delay h. Formally, the two formulas are related as follows: $(w,i) \models \varphi$ iff $(w,i+h) \models \psi$, where h is the *temporal depth* of φ. The temporal depth h is the maximum size of the input w suffix $[i,i+h]$ needed to determine the satisfaction of φ at the time index i. Formally, the temporal depth $h = H(\varphi)$ of φ is the syntax-dependent upper bound on the actual depth of the formula and is computed using the following recursive definition:

$$\begin{array}{lcl} H(p) & = & 0 \\ H(\neg\varphi) & = & H(\varphi) \\ H(\varphi_1 \vee \varphi_2) & = & \max\{H(\varphi_1), H(\varphi_2)\} \\ H(\bigcirc \varphi) & = & H(\varphi)+1 \\ H(\varphi_1 \ \mathcal{U}_{[a,b]} \varphi_2) & = & b + \max\{H(\varphi_1)-1, H(\varphi_2)\} \end{array}$$

Consider the formula $\varphi_1 \ \mathcal{U}_{[a,b]} \varphi_2$ and let us interpret φ_1 as the satisfaction signal of its first argument. An arbitrary interval $I = [i, i+b]$ admits a minimal partition $I_1, \ldots, I_n$ such that in every partition I_i the value of φ_1 is constant and in every two adjacent partitions the value of φ_1 differs. It is simple to see that the maximum number of partitions for such arbitrary I is bounded by $z = \lceil \frac{b}{2} \rceil$. We can then decompose φ_1 into z signals $\varphi_1^1, \ldots, \varphi_1^z$ such that $\varphi_1 = \bigcup_{i \in [1,z]} \varphi_1^i$, $\varphi_1^i \wedge \varphi_1^j$ is false for every $i \neq j$ and each φ_1^i has $(b,2)$-variability with at most one uniform subinterval of $[i, i+b]$ where φ_1^i is true. This decomposition is achieved in practice by letting φ_1^i rise and fall only on the j^{th} rising and falling of φ_1, where $j = i \bmod b$. After the decomposition, we have the following simple equivalence:

$$\begin{array}{lcl} (w,i) \models \varphi_1 \ \mathcal{U}_{[a,b]} \varphi_2 & \leftrightarrow & (w,i) \models \bigvee_{i=1}^{z} \varphi_1^i \ \mathcal{U}_{[a,b]} \varphi_2 \\ & \leftrightarrow & (w,i) \models \bigvee_{i=1}^{z} (\bigcirc \varphi_1^i \wedge \\ & & \diamondsuit_{[a-1,b-1]} (\varphi_1^i \wedge \bigcirc \varphi_2)) \end{array}$$

The *pastification* operation Π on the STL formula φ with past and bounded future and its bounded horizon $d = H(\varphi)$ is defined recursively as follows:

$$\begin{array}{lcl} \Pi(p,d) & = & \diamondsuit_{\{d\}} p \\ \Pi(\neg\varphi,d) & = & \neg\Pi(\varphi,d) \\ \Pi(\varphi_1 \vee \varphi_2,d) & = & \Pi(\varphi_1,d) \vee \Pi(\varphi_2,d) \\ \Pi(\bigcirc \varphi,d) & = & \Pi(\varphi,d-1) \\ \Pi(\diamondsuit_{[a,b]} \varphi,d) & = & \diamondsuit_{[0,b-a]} \Pi(\varphi,d-b) \\ \Pi(\varphi_1 \ \mathcal{U}_{[a,b]} \varphi_2,d) & \leftrightarrow & \bigvee_{i=1}^{z} \Pi(\varphi_1^i \ \mathcal{U}_{[a,b]} \varphi_2,d) \\ & \leftrightarrow & \bigvee_{i=1}^{z} \Pi(\bigcirc \varphi_1^i \wedge \\ & & \diamondsuit_{[a-1,b-1]} (\varphi_1^i \wedge \bigcirc \varphi_2),d) \end{array}$$

We note that for monitoring $\Pi(p \ \mathcal{U}_I q,d)$ we first need to decompose p into $\frac{d}{2}$ signals, each having $(d,2)$-variability. Hence, for every decomposed signal, we must use at most 2 active counters of size $\lceil \log_2 d \rceil$. As a result, the monitor for $\Pi(p \ \mathcal{U}_I q,d)$ requires $d \cdot \lceil \log_2 d \rceil$ registers. It follows that the monitor for an arbitrary bounded future formula requires at most $d \cdot \lceil \log_2 d \rceil \cdot |\varphi|$ registers.

III. Implementation

We implement our monitors on Zynq7020 All Programmable SoC, a configurable hardware platform. Its main parts are the Processing System and Programmable Logic. The Programmable Logic primarily consists of Configurable Logic Blocks (CLB) that contain lookup tables (LUT) and flip-flops (FF). Such blocks can be arbitrarily connected by programming desired connections in Switch Matrix. Each CLB can implement combinatorial net in LUTs and sequential circuits using FFs. The Zync7020 device contains in total 53200 LUTs and 106400 FFs. It also contains dedicated physical components for specific purpose such as block RAM (bRAM), specific SLICEM blocks for efficient implementation of shift registers and an internal Analog-to-Digital Converter (ADC). We now provide a high-level overview of our implementation flow showing the details of the monitor generation.

A. Implementation Overview

The implementation, illustrated in Figure 5, consists of three phases: (1) pre-processing; (2) code generation; and (3) FPGA flow.

In the pre-processing phase, we first translate the bounded-future STL formula to its equisatisfiable past counterpart. We then simplify the resulting past formula into an equivalent one which uses only basic $\ominus$, $\mathcal{S}$, $\diamondsuit_{[0,a]}$ and $\diamondsuit_{\{a\}}$ operators. This phase follows the algorithms and rewriting rules from Section II-C.

For the code generation phase, we deploy a parser in Java with a specific algorithm to extract information about input signals, operators and composition of the formula. Then, the algorithm uses a hash map to convert the parse tree of the formula into a directed acyclic graph (DAG) and eliminate duplicate sub-formulas. From this DAG we generate a deterministic monitor in a compositional way resulting in synthesizable Verilog code.

In the final phase, we follow the classical FPGA development flow in order to map the synthesizable monitor to the actual hardware. We use PlanAhead 14.7 and Vivado 2014.4 tools to perform the following steps: (1) synthesis; (2) implementation; and (3) bit-stream generation. We program the Zync7020 device with generated bit-stream and connect it in the lab environment to an oscilloscope to probe the signals of interest. In the digital case, we route the internal signals out from the Zync7020 - it is thus sufficient to monitor only external pins in the lab. In the analog case, the lab evaluation is more difficult. We created an *internal logic analyzer* block by deploying Xilinx IP core able to record quantized analog signals at runtime. In addition, we dedicate a specific external pin to alert the user on property violation. In Figure 6 we show a running FPGA monitor with different signals displayed on an oscilloscope.

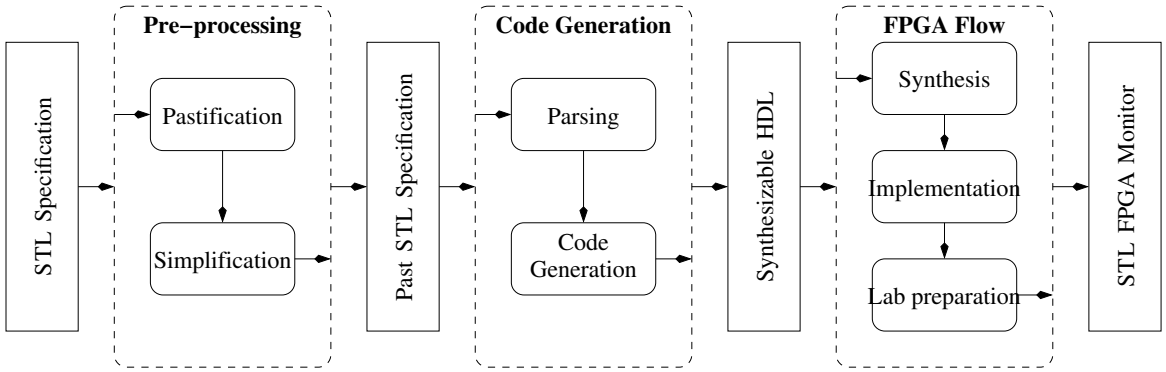

Fig. 5. Overview of the implementation.

Fig. 6. STL monitor running in real time in a lab environment.

B. Implementation Details

1) Numerical predicates: In our implementation, we consider the case where the analog device is an external component connected to the STL hardware monitor. We use the XADC block that is integrated to the Xilinx IP to convert the analog signal to a digital (quantized) one. We then implement synthesizable Verilog code that compares the quantized value of the signal to the threshold from the STL formula and outputs the Boolean signal for further use. We note that the precision of this approach is mainly limited by the characteristics of the ADC: (1) the frequency at which it operates; (2) its resolution; and (3) the maximum voltage that it is able to process. The XADC block from the Xilinx IP operates at a maximal frequency of 1MHz, has a resolution of 12 bits and is able to process signals that have the maximal amplitude of 1V.

2) Monitor Integration: Monitors implemented on FPGA are self-contained hardware units. There are different ways to integrate such monitors with the SUT. We first consider the case when both the SUT and monitors are implemented on the same FPGA programmable logic. This architecture is depicted in Figure 7. In this situation, both SUT and the monitor are purely digital blocks. The monitor non-intrusively observes relevant SUT signals by connecting to the SUT interface. Based on the observations, the monitor generates a verdict. Both the signals of interest and the verdict can be routed out of the FPGA by making the appropriate connections to the FPGA pinouts and displayed on an oscilloscope. The monitor and the SUT operate at the same frequency, limited by the maximum achievable frequency of the FPGA. The architecture allows usage of either internal or external clock generator. This architecture can be used to connect the monitor to digital design emulations. In this case, the SUT is either an emulation of a purely digital design or a digital approximation of an analog design.

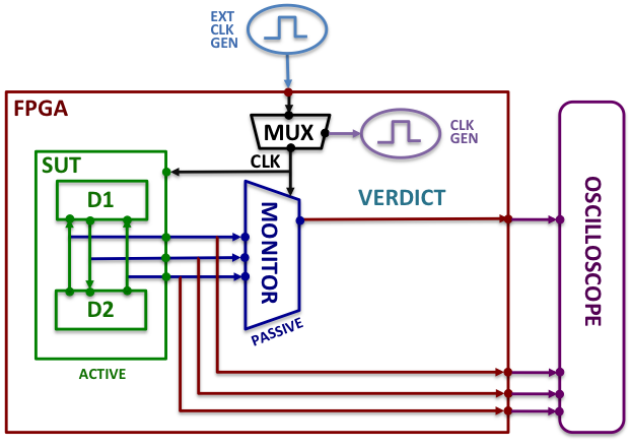

Fig. 7. Self-contained architecture - monitor and SUT on the same FPGA.

We also present an alternative architecture, in which the monitor implemented on FPGA is connected to an external device. This architecture is presented in Figure 8. In this case, the SUT is an external digital, analog or mixed-signal device. The monitor connects to external digital signals via FPGA pinouts. The analog signals cannot be directly connected to the monitor - we use the ADC block to quantify the input signal and convert it to the digital domain, as explained in Section III-B1. The external visualization of signals of interest and the verdict is done via the FPGA pinouts, as in the previous case. The typical use case for this architecture is the real-time monitoring of the real mixed-signal devices in the post-silicon verification phase. The main limitation of this architecture is

STL Formula	# FF	# LUT	MHz
$\square(p\,\mathcal{S}(q \wedge \boxminus r))$	2	3	346
$\square(p\,\mathcal{S}(q \wedge \ominus(r\,\mathcal{S}q)))$	2	3	346
$\square(p \rightarrow (q\,\mathcal{S}r))$	1	2	345
$\square(\diamondsuit p \wedge \diamondsuit q \wedge \diamondsuit r \wedge \diamondsuit s \wedge \diamondsuit t)$	5	7	339
$\square(p \rightarrow (q\,\mathcal{S}(\boxminus r \vee \boxminus s)))$	3	4	346

TABLE I. RESOURCE BENCHMARKS OF UNTIMED PAST STL FORMULAS.

the performance of the AD converter and the inaccuracies that it introduces.

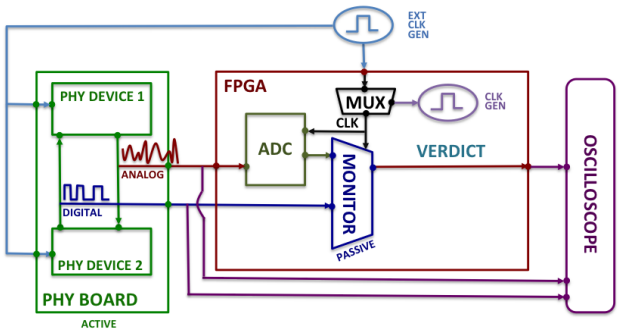

Fig. 8. Integration of a FPGA monitor to an external device.

IV. EVALUATION

A. Experimental Results

We now present the experimental evaluation of our framework. We translate several classes of STL formulas and collect the data on hardware resources allocated to the resulting monitors. For each formula we generate a hardware monitor and report the number of flip-flops (FF) and lookup tables (LUT) it consumes and the maximal achievable clock frequency in MHz. We note that the maximal achievable clock frequency depends in general on the size of the device implemented on the FPGA, since large resource usage requires more complex routing and internal signal propagation.

In our first experiment we consider only the untimed past fragment of STL. The results of this experiment, shown in Table I, clearly indicate that this fragment of the logic admits monitors with a very small footprint.

In our second experiment (see Table II) we evaluate the scalability of our approach w.r.t. a class of formulas from real time past fragment of STL. In particular we explore formulas containing the $\mathcal{S}_{[a,b]}$ and study the impact of bound variations to the resulting size of the monitor. In this experiment all our monitors use the counters algorithm to record real time data. We can first observe that for formulas with lower bound equal to zero, the resulting monitors have a small resource consumption and are insignificantly affected by the size of the upper bound b. This is due to the fact that these monitors require a single counter of size that is logarithmic in b. We can also see that the size of the monitors for this class of formulas are mostly affected by the ratio between the lower and the upper time bounds. The variability of the $\mathcal{S}_{[a,b]}$ operator, which is a function of the aforementioned ratio, directly affects the resource consumption of the monitor.

The third experiment has two purposes. On one hand we evaluate the size of monitors for bounded future STL. On the other hand we compare the resource requirements of two time event recording algorithms: straightforward register buffering and the counters algorithm. Table III shows the results. We first observe that handling future operators can be very costly. This result is as expected - checking online future formulas requires the determinization of the underlying monitors. In our approach we do this step at the syntactic level using the pastification procedure. We observe that the monitors for future formulas are in particular sensitive to the size of their future horizon. Regarding the two event recording algorithms we can notice that counters algorithm prevents the explosion of memory demands. On the other hands it uses more LUTs than the register buffer. This is due to arithmetical operations performed on the counters. We can also see that the maximum achievable frequency for the register buffering approach is in general higher than the one for the counters approach. We suspect that this results from more combinatorial operations done in the counters algorithm. As a consequence, the monitor computes more operations during a single clock cycle, thus prolonging its minimal duration.

B. Mixed Signal Bounded Stabilization

In the first case study, we adapt the mixed signal bounded stabilization property from Example 1. We monitor a system generating a boolean *trigger* and an analog signal x. Upon the rising edge of the trigger, the analog signal is allowed to get unstable, but is required to stabilize within a specified finite time horizon. The informal specification in natural language is stated as follows: "whenever the trigger is on its rising edge, the analog signal x is allowed to take an arbitrary amplitude, but within $1ms$, the signal must take an amplitude smaller or equal than $0.5V$ and continuously remain below that threshold for at least $500\mu s$".

We first model the analog signal x with perturbation by using an external pulse generator. We generate a $5kHz$ sinusoidal signal in the range of $250-800mV$ modulated with a $250Hz$ down ramp with 33% maximal decline, in order to obtain damped sine oscillations. This analog signal operates at lower frequency than our ADC. We down-clock the monitor to $200kHz$ and thus we avoid unnecessary computations.

The formalized requirement in STL and its past counterpart are as follows:

$$\square(\uparrow trigger \rightarrow \diamondsuit_{[0,200]} \square_{[0,100]}(x \leq 0.5))$$
$$\square(\diamondsuit_{\{300\}} \uparrow trigger \rightarrow \diamondsuit_{[0,200]} \boxminus_{[0,100]}(x \leq 0.5)).$$

Figure 9 shows the screenshot from the oscilloscope on which we observe the monitored results, delayed by 300 clock cycles due to formula pastification. The yellow waveform represents the analog input signal x, while the blue signal is the predicate $x \leq 0.5$. The red signal is the boolean *trigger* and finally the green waveform is the assertion verdict, where the peak represents the violation of the property. We can see from Figure 9 that the property is violated because the signal

STL Formula	b=50			b=500			b=5000		
	# FF	# LUT	MHz	# FF	# LUT	MHz	# FF	# LUT	MHz
$\square(p \leftrightarrow q\,\mathcal{S}_{[0,b]}r))$	13	18	346	16	26	346	20	36	346
$\square(p \leftrightarrow q\,\mathcal{S}_{[\frac{b}{2},b]}r))$	42	136	329	61	193	321	84	321	317
$\square(p \leftrightarrow q\,\mathcal{S}_{[\frac{9b}{10},b]}r))$	121	423	286	213	727	250	309	1446	213

TABLE II. RESOURCE BENCHMARKS FOR BOUNDED PAST STL FORMULAS.

STL Formula	Register Buffer			Counters Algorithm		
	# FF	# LUT	MHz	# FF	# LUT	MHz
$\square(p \leftrightarrow q\,\mathcal{U}_{[0,50]}r))$	1614	584	178	811	1807	155
$\square(p \leftrightarrow q\,\mathcal{U}_{[0,100]}r))$	5915	1548	202	1751	4069	110
$\square(p \leftrightarrow q\,\mathcal{U}_{[0,200]}r))$	22006	3476	177	3500	7630	103
$\square(p \leftrightarrow q\,\mathcal{U}_{[25,50]}r))$	1588	532	208	779	1747	169
$\square(p \leftrightarrow q\,\mathcal{U}_{[50,100]}r))$	5880	1463	191	1666	3912	105
$\square(p \leftrightarrow q\,\mathcal{U}_{[100,200]}r))$	22001	3484	156	3620	7916	120

TABLE III. RESOURCE BENCHMARKS FOR BOUNDED FUTURE STL FORMULAS.

x stabilizes too slowly and does not remain below the 0.5 threshold for sufficient time.

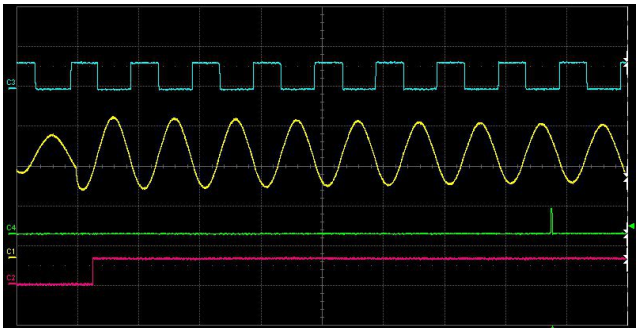

Fig. 9. Stabilization property signals observed in our lab environment.

C. Serial Peripheral Interface

Serial Peripheral Interface (SPI) bus is a de facto standard serial communication interface specification used for short distance communication, primarily in embedded systems. SPI provides full duplex communication with a master-slave architecture.

Regular SPI interface consists of the following signals: *SPI clock* (*sck*), *slave select* (*ss*), *master input slave output* (*miso*) and *master output slave input* (*mosi*). In this case study, we use a digital master device that has additional control signals such as *data available* (*dav*) and *FIFO read enable* (*fre*).

We create a synthesizable test bench (TB) in Verilog to stimulate the SPI device with several scenarios. In addition to regular scenarios we inject errors to trigger property violations. We synthesize both the TB and the SPI master on a single Zync 7020 device. The user controls the emulation with a programmed physical interrupt.

We consider two SPI requirements: *nested pulse* (NP) and *clock division* (CD). NP property specifies mutual interaction of control signals *dev* and *fre* - it states that once the FIFO read is done, there should be no more data available for the transfer. CD property requests that *sck* is slower than the system clock by a rate determined by the *rate* signal value. We fix *rate* to 0, meaning that *sck* must be twice slower than the system clock. In addition, it is required that the *sck* clock is toggling only when *ss* is asserted. We specify these two requirements in natural language as follows:

1) *fre* pulse must be on its rising edge in the last cycle of the *dav* pulse;
2) Every rising edge of *sck* is (1) the first one within the current asserted interval of *ss* or (2) follows a falling edge of *sck* in the previous cycle. The symmetric property must hold for every falling edge of *sck*.

We formalize these requirements in past STL as follows:

$$\square(\downarrow dav \rightarrow (\downarrow fre \wedge \ominus \uparrow fre)$$
$$\square((\uparrow sck) \rightarrow (\neg sck\,\mathcal{S} \uparrow ss \vee \ominus \downarrow sck))$$
$$\square((\downarrow sck) \rightarrow (sck\,\mathcal{S} \uparrow ss \vee \ominus \uparrow sck))$$

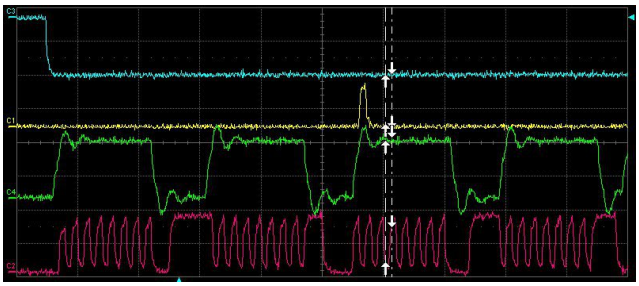

Fig. 10. SPI clock division property observed in the lab environment.

The resulting monitors for these properties operate at the Zynq system clock frequency of $100MHz$ and do not additionally constrain the speed of the SUT. Figure 10 shows the relevant CD property signals on an oscilloscope in our lab environment. The blue signal represents the SPI master device reset signal and it marks the start of the emulation. The green signal denote the SPI slave select *ss* and the red one represents the SPI clock *sck*. Finally, the yellow signal shows the verdict for the property, where the peak in the signal

denotes its violation. As we can see from the Figure 10, the property $\square((\uparrow sck) \rightarrow (\neg sck\, \mathcal{S} \uparrow ss \vee \ominus \downarrow sck))$ fails because at the moment of its violation (peak in the yellow signal), the first rising edge of *sck* arrives slightly before the rising edge of the new *ss* window, thus falsifying both disjuncts $\neg sck\, \mathcal{S} \uparrow ss$ and $\ominus \downarrow sck$ of the right-hand side of the implication.

V. Related Work

In the last decade assertion based hardware monitoring has received an increasing attention. Generating synthesizable monitors from Property Specification Language (PSL) has been proposed by several research groups and it has been implemented first in the tools FoCs [10] developed by IBM and MBAC [5], [6], [7] developed by Zilic and Boulé. On the same line of research is also the work of Borrione et. al. in [4] and Backasch et. al. in [2]. In contrast to our work, all of them focus on untimed digital specifications. FoCs generates monitors for SystemC [1] simulations. MBAC adopts an automata-oriented approach which conceptually differs from our transducer-based compositional construction.

Finkbeiner et al. in [13] present a technique to synthesise monitor circuits from LTL formulas with bounded and unbounded future operators. In contrast to our approach they do not allow past formulas. Moreover, they evaluate their approach only with formulas with the lower time bound equal to zero.

Claessen et al. [9] propose some efficient techniques to synthetize a LTL safety and liveness property checkers as circuits with sequential elements rather than using the classical automata-based approach with transitions relations. The authors focus more on model checking of hardware system design rather than runtime monitoring of analog and mixed signal systems (AMS). Finally, they do not report any experiments on real hardware.

Reinbacher et. al. propose in [21], [22] synthesizable hardware monitors from different fragments of Metric Temporal Logic (MTL). In [21] the authors tackle only the past fragment of MTL using a transducer-based approach similar to ours. In contrast to relative clocks (counters) that we adopt to record events, the authors use an approach in which absolute time stamps are memorized. Hence, the resources needed for implementing their monitors depend on the duration of the emulation runtime. The authors develop a sophisticated architecture that targets reconfigurability of monitors. However, it also introduces an overhead that requires their monitors to operate at a considerable higher frequency than the SUT. In other words, this imposes additional constraints on the maximum speed of the SUT that such monitors can observe. In our approach, the monitor and the SUT run at the same clock frequency. Finally, the authors evaluate their approach using pre-collected data.

In [22] the authors address the future fragment of MTL. They take a radically different approach to ours motivated by the problem of embedded system health estimation. They adopt a three-valued interpretation of the logic and produce a "maybe" output delaying a definite verdict until the formula can be really evaluated. This approach is suitable for estimating system health using a Bayesian network on top of the observers. On the other hand, we are motivated by the verification of AMS along their physical interface (time, voltage, signal dependencies in time) and we choose to follow a different approach. We treat formulas with bounded future temporal operators, rewriting them using past temporal operators which gives us more uniform and simple approach in terms of formula evaluation and system architecture. Similarly to their previous work the authors evaluate their framework only on pre-collected data.

VI. Conclusions and Future Work

In this paper, we have proposed an effective procedure for online checking of digital and mixed signal systems with STL monitors implemented on FPGA hardware. We have evaluated our approach in the lab environment, demonstrating its feasibility and benefits for runtime monitoring of real time systems.

This work is our first step in porting the formal methods theory to the practical rigorous checking of complex, possibly mixed signal designs. We are planning to explore several new research directions on the topic. We will apply our technique to a large automotive airbag SoC mixed signal case study. As part of that effort, we will identify possible extensions to STL that will increase its expressiveness and enable specification of a richer set of requirements, including for instance data integrity and frequency temporal properties [12]. We will study the effect of inaccuracies due to noise and measurement errors in the evaluation of numerical predicates over analog signals. We will also study the applicability of our FPGA-based monitoring framework to the medical domain, following the work presented in [3], [8]. Finally, we will study the problem of diagnosing violations of STL specifications during monitoring and identifying the causes of property falsification.

VII. Acknowledgements.

We would like to thank Oded Maler, Thomas Ferrère and the anonymous reviewers for their comments on the earlier drafts of the paper.

We acknowledge the support of the IKT der Zukunft of Austrian FFG project HARMONIA (nr. 845631), the ICT COST Action IC1402 Runtime Verification beyond Monitoring (ARVI), the Austrian National Research Network S 11405-N23 and S 11412-N23 (RiSE/SHiNE) of the Austrian Science Fund (FWF) and the Doctoral Program Logical Methods in Computer Science of the Austrian Science Fund (FWF).

References

[1] SystemC, http://www.systemc.org/

[2] Backasch, R., Hochberger, C., Weiss, A., Leucker, M., Lasslop, R.: Runtime verification for multicore soc with high-quality trace data. ACM Transactions on Design Automation of Electronic Systems 18(2) (2013)

[3] Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Formal Modeling and Analysis of Timed Systems - 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings. pp. 23–37 (2014)

[4] Borrione, D., Liu, M., Morin-Allory, K., Ostier, P., Fesquet, L.: On-line assertion-based verification with proven correct monitors. In: Proc. of ITI 2005: the 3rd International Conference on Information and Communications Technology. pp. 125–143 (2005)

[5] Boulé, M., Zilic, Z.: Incorporating efficient assertion checkers into hardware emulation. In: Proc. of ICCD. pp. 221–228. IEEE Computer Society Press (2005)

[6] Boulé, M., Zilic, Z.: Efficient automata-based assertion-checker synthesis of PSL properties. In: Proc. of HLDVT. pp. 69–76. IEEE (2006)

[7] Boulé, M., Zilic, Z.: Automata-based assertion-checker synthesis of PSL properties. ACM Transactions on Design Automation of Electronic Systems 13(1) (2008)

[8] Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., Bortolussi, L.: 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. Lecture Notes in Computer Science, vol. 8803, pp. 391–403. Springer (2014)

[9] Claessen, K., Een, N., Sterin, B.: A circuit approach to ltl model checking. In: Formal Methods in Computer-Aided Design (FMCAD), 2013. pp. 53–60 (Oct 2013)

[10] Dahan, A., Geist, D., Gluhovsky, L., Pidan, D., Shapir, G., Wolfsthal, Y., Benalycherif, L., Kamidem, R., Lahbib, Y.: Combining system level modeling with assertion based verification. In: Proc. of ISQED 2005: Sixth International Symposium on Quality of Electronic Design. pp. 310–315. IEEE (2005)

[11] D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: Runtime monitoring of synchronous systems. In: Proceedings of the 12th International Symposium of Temporal Representation and Reasoning (TIME 2005). pp. 166–174. IEEE Computer Society Press (2005)

[12] Donzé, A., Maler, O., Bartocci, E., Nickovic, D., Grosu, R., Smolka, S.A.: On Temporal Logic and Signal Processing. In: Proc. of ATVA 2012: 10th International Symposium on Automated Technology for Verification and Analysis, Thiruvananthapuram, India, October 3-6. LNCS, vol. 7561, pp. 92–106 (2012)

[13] Finkbeiner, B., Kuhtz, L.: Monitor circuits for ltl with bounded and unbounded future. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5779 LNCS, 60–75 (2009)

[14] Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Proc. of TACAS 2002: the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 2280, pp. 342–356. Springer (2002)

[15] ISO 26262:2011: Road Vehicles – Functional Safety. ISO, Geneva, Switzerland

[16] Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits. STTT 15(3), 247–268 (2013)

[17] Maler, O., Nickovic, D., Pnueli, A.: Real time temporal logic: Past, present, future. In: Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings. pp. 2–16 (2005)

[18] Maler, O., Nickovic, D., Pnueli, A.: On synthesizing controllers from bounded-response properties. In: Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings. pp. 95–107 (2007)

[19] Nguyen, T., Wooters, S.: FPGA-based development for sophisticated automotive embedded safety critical system. SAE International Journal of Passenger Cars - Electronic and Electrical Systems 7(1), 125–132 (2014)

[20] Pnueli, A., Zaks, A.: On the merits of temporal testers. In: 25 Years of Model Checking - History, Achievements, Perspectives. pp. 172–195 (2008)

[21] Reinbacher, T., Függer, M., Brauer, J.: Runtime verification of embedded real-time systems. Formal Methods in System Design 44(3), 230–239 (2014)

[22] Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proc. of TACAS 2014. LNCS, vol. 8413, pp. 357–372. Springer-Verlag (2014)