

# Timestamp Temporal Logic (TTL) for Testing the Timing of Cyber-Physical Systems

MOHAMMADREZA MEHRABIAN, Arizona State University

MOHAMMAD KHAYATIAN, Arizona State University

AVIRAL SHRIVASTAVA, Arizona State University

JOHN C. EIDSON, University of California Berkeley

PATRICIA DERLER, National Instruments corporation

HUGO A. ANDRADE, University of Texas at Austin

YA-SHIAN LI-BABOUD, National Institute of Standard and Technology (NIST)

EDWARD GRIFFOR, National Institute of Standard and Technology (NIST)

MARC WEISS, Marc Weiss Consulting

KEVIN STANTON, Intel corporation

In order to test the performance and verify the correctness of Cyber-Physical Systems (CPS), the timing constraints on the system behavior must be met. Signal Temporal Logic (STL) can efficiently and succinctly capture the timing constraints of a given system model. However, many timing constraints on CPS are more naturally expressed in terms of events on signals. While it is possible to specify event-based timing constraints in STL, such statements can quickly become long and arcane in even simple systems. Timing constraints for CPS, which can be large and complex systems, are often associated with tolerances, the expression of which can make the timing constraints even more cumbersome using STL. This paper proposes a new logic, Timestamp Temporal Logic (TTL), to provide a definitional extension of STL that more intuitively expresses the timing constraints of distributed CPS. TTL also allows for a more natural expression of timing tolerances. Additionally, this paper outlines a methodology to automatically generate logic code and programs to monitor the expressed timing constraints. Since our TTL monitoring logic evaluates the timing constraints using only the timestamps of the required events on the signal, the TTL monitoring logic has significantly less memory footprint when compared to traditional STL monitoring logic, which stores the signal value at the required sampling frequency. The key contribution of this paper is a scalable approach for online monitoring of the timing constraints. We demonstrate the capabilities of TTL and our methodology for online monitoring of TTL constraints on two case studies: 1) Synchronization and phase control of two generators and, 2) Simultaneous image capture using distributed cameras for 3D image reconstruction.

**CCS Concepts:** • Computer systems organization → Embedded software;

**Additional Key Words and Phrases:** Cyber-Physical Systems, CPS, Timing Constraints, Time Testing, Real-Time Systems, Verification, Safety Critical Systems

This work was partially supported by funding from National Science Foundation grants CCF 1055094 (CAREER), CNS 1525855 and NIST Awards 60NANB16D305, and 70NANB16H305.

Authors' addresses: M. Mehrabian, M. Khayatian, A. Shrivastava, Arizona State University, School of Computing, Informatics and Decision Systems Engineering, 699 S Mill Ave, Tempe, AZ 85281, US; J. C. Eidson, University of California Berkeley, Berkeley, CA 94720; P. Derler, National Instruments corporation, 11500 N Mopac Expy, Austin, TX 78759; H. A. Andrade, University of Texas at Austin, 787 Balra Dr, El Cerrito, CA 94530; Y. Li-Baboud, E. Griffor, NIST, 100 Bureau Drive, Gaithersburg, MD 20899; M. Weiss, Marc Weiss Consulting, Aptos, CA 95003; K. Stanton, Intel corporation, 2111 NE 25th Ave., Hillsboro, OR 97124.

© 2017

xxxx-xxxx/2017/10-ART1 \$15.00

<https://doi.org/xx.xxxx/nnnnnnn.nnnnnnn>

This article was presented in the International Conference on Embedded Software 2017 and appears as part of the ESWEEK-TECS special issue

## 1 INTRODUCTION

Cyber-Physical Systems (CPS) integrate physical and computational worlds to form smart, coordinated, efficient and responsive infrastructures. Deploying CPS that increase the efficiency in sectors of health, energy, aviation and freight rail by 1% will save \$186 billion in the U.S. over a 15 year period [2].

TIME is a fundamental concept in CPS which allows the integration of discrete (cyber) and continuous (physical) domains [20]. CPS use sensors whose data needs are often time-tagged for efficient data fusion and knowledge of when the measurement was taken. Computing, communication and control commands in dynamic real-time systems need to be executed within a specified latency. Correct and robust orchestration of different tasks and/or distributed parts requires correct temporal behavior within and among CPS components. Current and future CPS systems such as health-care monitoring and active control devices, intelligent transportation, and electrical power systems are a few safety-critical examples requiring synchronization and latency controls.

In order to be confident about the behavior of a built or designed CPS, its timing behavior must be tested and verified. Prior to performing testing of temporal behavior, timing constraints must be expressed in a formal language. That enables robust analysis of constraint satisfiability and consistency. The formal expression enables the application developer to explicitly specify timing requirements in the design phase and provide the basis to automate the generation of application and associated test code to enable a more systematic, rigorous, and iterative verification process of the SUT (System Under Test). Temporal logic provides the formalism to define time specifications, where evaluation of constraint satisfiability is based on reasoning about the propositions. There are several types of temporal logic which reason about variables on a discrete or continuous time domain. LTL (Linear Temporal Logic) is defined for sequences of boolean predicates, MTL (Metric Temporal Logic) is expressed on real-valued signals in discrete time and STL (Signal Temporal Logic) is utilized for specifying timing constraints on real-valued signals over continuous time. Event-based timing constraints can be expressed in STL by using the *Rise* and *Fall* operators [17]. However, the STL statements become quite complicated and difficult to understand. In particular, using STL expressions to specify simple latency constraints among events become complicated, as they must be expressed in a nested manner. Furthermore, expressing the acceptable tolerance of the timing constraints make the constraint expressions more complicated. Since timing constraints are specified and written manually by humans, they should be readable and intuitive, to bridge specification at the programming language level with synthesis and validation during application compilation and verification on hardware platforms.

In this paper, we introduce Timestamp Temporal Logic (TTL) – in order to more simply and intuitively express the timing constraints of distributed CPS. TTL allows for the specification of the acceptable tolerance of the timing constraints. Further, we also outline a systematic methodology to generate the logic for real-time monitoring of TTL timing constraints. In our methodology, a timing constraint is monitored by extracting timestamps of rising and falling edges of boolean signals and is evaluated by working with timestamps only. In comparison, the monitoring logic for STL constraints will have to store all signal values at the required sampling frequency in the time period of interest, which can require significant memory footprint. Reduced memory requirements of the TTL constraint monitoring logic, allow us to perform online analysis of the timing constraints of a potentially large-scale distributed CPS. Online constraint monitoring can be used at runtime to enhance safety by early termination, and also can be used to reduce the debugging time during prototyping[6].

In order to illustrate the application and verification process of TTL, we developed two distributed CPS case studies: 1) Synchronization and phase control of two direct current (DC) motors to

simulate two generators connected to the same power grid and, 2) Simultaneous image capture using distributed cameras for 3D image reconstruction. We expressed the timing constraints of both these applications in TTL. We synthesized the TTL constraint monitoring logic on a Field Programmable Gate Array (FPGA), and also developed a test application for online evaluation of the TTL statements.

## 2 RELATED WORK

Temporal logic has been used to formally describe, reason and verify the temporal behaviors of the system. Linear Temporal Logic (LTL) was introduced in 1996, to specify timing constraint on sequences of boolean predicates [9]. Timed Linear Temporal Logic (TLTL) was introduced to support real-time properties of the system [4]. Similarly, Metric Temporal Logic (MTL) and Metric Interval Temporal Logic (MITL) are the extensions of LTL to express a real-time timing constraint on boolean predicates [3, 14].

In 2004, Maler and Nickovic proposed Signal Temporal Logic (STL) to express dynamic timing constraints over real-valued signals [15]. STL is constructed based on MTL and is used to reason about continuous signals. STL expresses temporal constraints over both finite and infinite time horizons. However, reasoning about the future of a signal at the evaluation time is not causal. Jakšić et al. solved the causality problem of the STL by introducing *past* STL [13]. In [8], Donzé et al. extended STL by adding frequency constraints on real-valued signals. They proposed Time-Frequency logic (TFL) which expresses the timing requirements based on Short-Time Fourier Transform (STFT) of a signal[8].

Various monitoring tools have been proposed that observe the system behavior for specified initial conditions and inputs [5, 10, 11]. In [7], Donzé et al. created a toolbox in Matlab (Breach) that enables monitoring of STL constraints. AMT is a similar toolbox used for monitoring temporal constraints expressed in Property Specification Language (PSL)/STL [19].

### 2.1 Limitations of STL-based timing specification

Expressing level-based constraints on the real-valued signals is done using STL semantics [15]. *Globally* ( $\square$ ), *Eventually* ( $\diamond$ ), *Until* ( $\mathcal{U}$ ) and *Since* ( $\mathcal{S}$ ) express different level-based constraints in STL. Event-based constraints are expressed with *Rise* ( $\uparrow$ ) and *Fall* ( $\downarrow$ ) operators in STL which are defined as follows [17]:

$$\begin{aligned}\uparrow \psi &= (\psi \wedge (\neg \psi \mathcal{S} \top)) \vee (\neg \psi \wedge (\psi \mathcal{U} \top)) \\ \downarrow \psi &= (\neg \psi \wedge (\psi \mathcal{S} \top)) \vee (\psi \wedge (\neg \psi \mathcal{U} \top))\end{aligned}$$

Although instantaneous events can be captured in STL using the Rise and Fall operators, they often result in convoluted and esoteric specification. Specifically, expressing sequential constraints in STL results in a nested expressions. For instance, consider the following constraint: “Whenever signal  $x_1$  rises above 0.5, signal  $x_2$  should rise above 0.6 within 1 second and after that, signal  $x_3$  should fall below 0.4 within 5 seconds.”. This constraint can be expressed in STL using Rise ( $\uparrow$ ) and Fall ( $\downarrow$ ) operators as:

$$\psi = \square \left( (\uparrow (x_1 > 0.5)) \Rightarrow \left( \diamond_{[0,1]} (\uparrow (x_2 > 0.6)) \Rightarrow \left( \diamond_{[0,5]} (\downarrow (x_3 > 0.4)) \right) \right) \right)$$

By replacing Rise( $\uparrow$ ) and Fall( $\downarrow$ ) operators, we have:

$$\begin{aligned}\psi &= \square \left( [(x_1 > 0.5) \wedge (\neg(x_1 > 0.5) \mathcal{S} \top)] \vee [\neg(x_1 > 0.5) \wedge ((x_1 > 0.5) \mathcal{U} \top)] \right) \Rightarrow \\ &\quad \left( \diamond_{[0,1]} [(x_2 > 0.6) \wedge (\neg(x_2 > 0.6) \mathcal{S} \top)] \vee [\neg(x_2 > 0.6) \wedge ((x_2 > 0.6) \mathcal{U} \top)] \right) \Rightarrow\end{aligned}$$

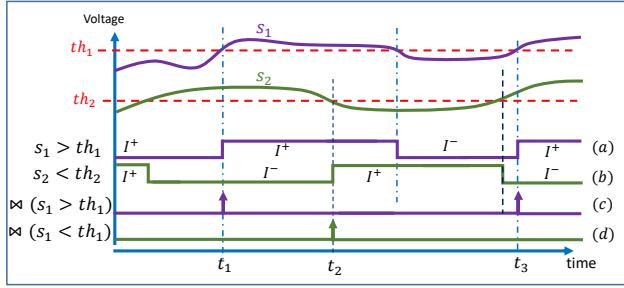


Fig. 1. Comparing two signals ( $s_1$  and  $s_2$ ) with their corresponding thresholds ( $th_1$  and  $th_2$ ) results in boolean signals (a and b) and applying differentiator operator ( $\bowtie$ ) on boolean signals results in instantaneous events (c and d).

$$\left( \diamond_{[0,5]} \left( [(\neg(x_3 > 0.4) \wedge ((x_3 > 0.4) \mathcal{S} \top)) \vee ((x_3 > 0.4) \wedge (\neg(x_3 > 0.4) \mathcal{U} \top))] \right) \right)$$

Expressing this relatively simple and common constraint is complicated in STL.<sup>1</sup>

Since timing constraints are specified and written manually by *humans*, they should be as intuitive as possible. Expressing timing constraints with sequential ordering causes nesting in STL statements and makes the statements difficult to understand. In comparison, the augmented syntax of Timestamp Temporal Logic (TTL) can more intuitively express the timing constraints of distributed CPS. TTL also allows for a more natural expression of timing tolerances.

## 2.2 Limitations of STL-based monitoring

The main limitation of STL-based monitoring techniques is excessive memory consumption of monitoring tools when they either store the signal for offline monitoring or evaluate the timing constraints for online monitoring. In offline monitoring, the monitoring tool stores the entire signal [19]. In online monitoring, a sufficient range of signals is stored to evaluate a constraint with future operators[6]. In both cases, data acquisition rate is selected based on the desired time accuracy. For example, if the desired time accuracy is  $1\ \mu\text{s}$ , the sampling should be at least  $0.5\ \mu\text{s}$ . This requires storing 1 million floating point number (4 bytes) per second. In other words, monitoring a signal for ten minutes requires 4.8 GB of memory.

In [13], Jakšić et al. presented a methodology for monitoring STL constraints using FPGAs. In this work, both the SUT and the monitoring logic are implemented on the same FPGA and the clock used for SUT is the same as the monitoring system. However, in a distributed CPS, having access to SUT clock is not possible and even if it is, it may not be reliable. Local clock references must be synchronized to a global reference in order to measure time intervals over distributed systems where timestamps are captured with monitoring devices in different locations.

In contrast, the monitoring program to evaluate TTL constraints works only on timestamps of the important events in the system, and therefore requires much less memory. Also, the timestamps are taken on the clock from the testbed, which is synchronized, more reliable, and under our control.

<sup>1</sup>The same timing constraint is expressed in TTL simply as:

$$\psi = \left[ \mathcal{L}((x_1, 0.5, \nearrow), (x_2, 0.6, \nearrow)) \leq 1 \right] \wedge \left[ \mathcal{L}((x_2, 0.6, \nearrow), (x_3, 0.4, \searrow)) \leq 5 \right]$$

where the first sentence expresses that the latency ( $L$ ) between the two events – one when signal  $x_1$  increases above 0.5 V, and the second, when signal  $x_2$  increases above 0.6 V – is less than 1s, and the second sentence defines the latency constraint between the events on signals  $x_2$  and  $x_3$ .

### 3 TIMESTAMP TEMPORAL LOGIC (TTL)

In this section, we present our proposed logic, Timestamp Temporal Logic (TTL) which is targeted towards the description of timing constraints for CPS.

#### 3.1 Event Representation

Typically, an event is represented by a single point in the time domain which can be represented by a Kronecker delta function ( $\delta t$ ). Accordingly, a signal event is constructed from a real-valued signal crossing a threshold value. We represent a signal by a triplet,  $\langle s, th, \nearrow \text{ or } \searrow \rangle$ , which is 1 at the time when the signal,  $s$ , crosses a threshold,  $th$  (crossing the threshold from below  $\nearrow$  or from above  $\searrow$ ), and 0 everywhere else. A signal event can be singleton or repetitive. In a singleton signal event, there is only one event ( $e_1$ ) which is represented by a single timestamp while in repetitive signal events, a sequence of events  $\{e_1, e_2, \dots, e_n\}_{(n \in \mathbb{N})}$  is represented by multiple timestamps. A boolean signal can be divided into time intervals during which the value of the signal is true or false, indicated by  $I^+$  and  $I^-$  (Figure 1 signals a and b). The occurrence of an event corresponding to a rising/falling edge is defined as the starting point of each positive interval ( $I_i \in I^+$ ).

**DEFINITION 1.** *Differentiate operator,  $s' = \bowtie(s)$  converts a boolean signal  $s \in \mathbb{B}$  to a signal event where the value of  $s'$  is 1 when  $s(t^+) \oplus s(t) \wedge \neg s(t) = \top$ , and  $\perp$  otherwise.  $\oplus$  is the XOR operator and,  $t^+$  refers to the right neighborhood of signal at time  $t$  in continuous domain.*

Extracting a signal event from a real-valued signal over the continuous time-domain is done by comparing the values of the signal with a threshold,  $th$ , and then, passing the output through the *Differentiate Operator* ( $\bowtie$ ) in discrete time-domain. As depicted in Figure 1, signals  $s_1$  and  $s_2$  are converted into signal events, c and d, after comparing with their corresponding thresholds ( $th_1, th_2$ ) and applying the differentiate operator,  $\bowtie$ .

**DEFINITION 2.** *A projection function  $\pi_p$  is a function that maps a proposition built upon real-valued signals to a boolean-valued signal.*

$$\pi_p(s[t]) : D \rightarrow \mathbb{B}$$

where  $\mathbb{B}$  is the boolean domain.

#### 3.2 TTL Syntax

The TTL syntax is defined based on STL with extensions to enable distributed CPS with respect to absolute time, improved clarity, and signal expression simplifications without substantial loss of meaning. TTL operators are built based on high-level operators that specify timing requirements on both the value of a formula and the occurrence time events. The output of TTL operators are finally a boolean.

**DEFINITION 3.** *The comparison operator  $\triangleright$ , is a mapping function from a real-valued signal to a boolean value, where  $\triangleright \in \{>, ==, <\}$ .*

**DEFINITION 4.** *The maximum tolerance  $\epsilon$ , is defined as the tolerated level of accuracy needed to monitor a temporal requirement where  $\epsilon \in \mathbb{R}^+$  and  $\epsilon > \frac{1}{f_s}$ . ( $f_s$  is the digitizing sampling frequency)*

The grammar of TTL is defined as follows:

$$\begin{aligned} \psi := & p | \phi \triangleright c | \neg \psi | \psi_1 \wedge \psi_2 | \mathcal{L}_{[a,b]}(\varphi_1, \varphi_2) \triangleright c | \mathcal{S}_{[a,b]}(\varphi_1, \varphi_2, \dots, \varphi_n, \epsilon) \\ & | C_{[a,b]}(\varphi_1, \varphi_2, \dots, \varphi_n) | \mathcal{F}_{[a,b]}(\varphi) \triangleright c | \mathcal{P}_{[a,b]}(\varphi_1, \varphi_2) \triangleright c \\ & | S_{p,[a,b]}(\varphi, m) | \mathcal{B}_{[a,b]}(\varphi, N, d_k, m) \end{aligned}$$

where  $p$  is a boolean proposition in  $P = \{p_1, \dots, p_n\}$ ,  $\{a, b, c, d_k, m\} \in \mathbb{R}^+$ ,  $N \in \mathbb{N}^+$ ,  $\phi$  is a formula representing real-valued signals,  $\psi$  is a formula constructed from boolean signals and  $\varphi$  is a formula

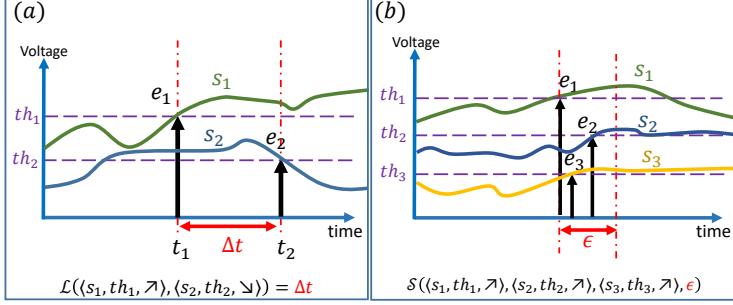


Fig. 2. a. The latency between events extracted from signal  $s_1$  crossing its threshold ( $th_1$ ) from below and signal  $s_2$  crossing its thresholds ( $th_2$ ) from above is  $\Delta t = t_2 - t_1$ . b. Three signals ( $s_1$ ,  $s_2$  and  $s_3$ ) cross their corresponding thresholds ( $th_1$ ,  $th_2$ ,  $th_3$ ) from below simultaneously considering error tolerance,  $\epsilon$ .

constructed from signal events.  $\mathcal{L}$ ,  $\mathcal{S}$ ,  $\mathcal{C}$ ,  $\mathcal{F}$ ,  $\mathcal{P}$ ,  $\mathcal{Sp}$  and  $\mathcal{B}$  are latency, simultaneity, chronological, frequency, sporadic and burst operators. One can verify that  $\psi$  can be converted into  $\varphi$ :

$$\varphi \Rightarrow \psi$$

As depicted in Figure 1, signals c and d, the triplet  $\langle s, th, \nearrow \rangle$  is equivalent to  $\Rightarrow (s > th)$  and the triplet  $\langle s, th, \searrow \rangle$  is equivalent to  $\Rightarrow (s < th)$ .

Latency, simultaneity and chronological constraints are expressed on singleton signal events and frequency, sporadic, burst and phase constraints are expressed for repetitive signal events.

**3.2.1 Latency [ $\mathcal{L}_{[a,b]}(\varphi_1, \varphi_2)$ ].** A latency constraint monitors the time difference between the occurrence of two signal events  $\varphi_1$  and  $\varphi_2$ . For example, in a car, the actuation trigger of the airbag system should be delivered within a known latency when the collision sensor detects an impact. Latency constraints can be used to specify a condition on “maximum, minimum or exact” latency between two signal events when used along with  $<$ ,  $>$  or  $==$  operators, respectively. Figure 2.a shows two signal events having latency of  $\Delta t$ . Each signal event is specified by a real-valued signal, a threshold  $th$  and a rising/falling edge (rising edge for  $s_1$  and falling edge for  $s_2$ ).

**3.2.2 Simultaneity [ $\mathcal{S}_{[a,b]}(\varphi_1, \varphi_2, \dots, \varphi_n, \epsilon)$ ].** The simultaneity concept for two or more events refers to satisfaction of two or more conditions at the same time. Many applications require simultaneous sensing or actuating. For instance, when multiple cameras take the photo of an object with a fast motion from different angles for 3D view reconstruction, all capture actions should be done at the same time in order to perform a successful reconstruction. Usually, a tolerance ( $\epsilon$ ) is set to indicate the maximum acceptable time difference between the occurred events, i.e., the precision. Figure 2.b depicts a simultaneity constraint for three events. Each event is specified by a real-valued signal, a threshold  $th$  and a rising/falling edge.

**3.2.3 Chronological [ $\mathcal{C}_{[a,b]}(\varphi_1, \varphi_2, \dots, \varphi_n)$ ].** A chronological constraint is specified when the occurrence order of events matters. For example, in a car accident, the airbag must actuate after the time the collision sensor detects the impact and the seat belt must be retracted after the time the airbag actuates. Strict adherence to the chronological ordering is necessary to avoid any harm to the passengers.

**3.2.4 Frequency [ $\mathcal{F}_{[a,b]}(\varphi)$ ].** A frequency constraint expresses the time interval between every two consecutive events extracted from a repetitive signal event. The time interval should equal the specified period. Conventionally, frequency is defined as the number of occurrences per second.

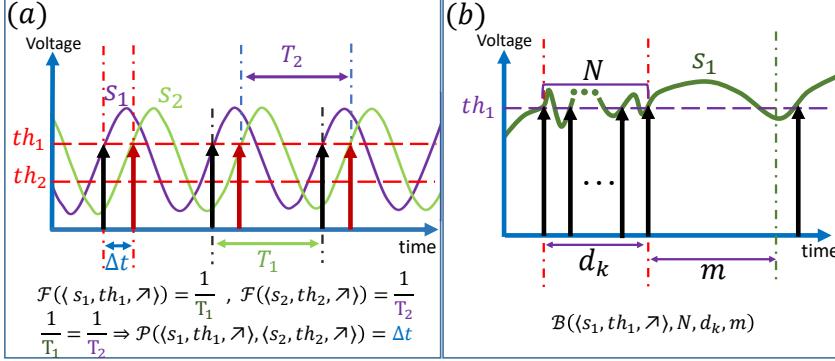


Fig. 3. a. The frequency of event sequences extracted from two signals ( $s_1$  and  $s_2$ ) crossing their corresponding thresholds ( $th_1$  and  $th_2$ ) from below is  $\frac{1}{T_1}$  and  $\frac{1}{T_2}$  respectively, and the phase difference between in each period is  $\Delta t$ . b. Burst constraint, whenever a signal ( $s_1$ ) crosses its threshold ( $th_1$ ) from below exactly  $N$  times in a time interval with  $d_k$  seconds width, it must not cross its threshold from below again for  $m$  seconds.

Table 1. Satisfaction relations and language semantics,  $\mathcal{L}$ : latency,  $\mathcal{F}$ : frequency,  $\mathcal{P}$ : phase,  $\mathcal{S}$ : simultaneity,  $\mathcal{C}$ : chronological,  $\mathcal{Sp}$ : sporadic and  $\mathcal{B}$ : burst constraints

1	$(s, t) \models p$	iff $\pi_p(s)[t] = \text{True}$
2	$(s, t) \models \neg\psi$	iff $(s, t) \not\models \psi$
3	$(s, t) \models \phi \nabla c$	iff $\phi[t] \nabla c$
4	$(s, t) \models \psi_1 \wedge \psi_2$	iff $(s, t) \models \psi_1 \wedge (s, t) \models \psi_2$
5	$(s, t) \models \mathcal{L}_{[a,b]}(\varphi_1, \varphi_2) \nabla c$	iff $\exists t' \in [t+a, t+b]$ s.t. $(s, t') \models \varphi_1$ and $\exists t'' > t'$ s.t. $(s, t'') \models \varphi_2 \wedge (t'' - t') \nabla c$ .
6	$(s, t) \models \mathcal{S}_{[a,b]}(\varphi_1, \dots, \varphi_n, \epsilon)$	iff $\forall t_i \in [t+a, t+b], i = \{1, \dots, n\}$ s.t. $(s, t_i) \models \varphi_i$ , $\max\{t_i\} - \min\{t_i\} \leq \epsilon$
7	$(s, t) \models \mathcal{F}_{[a,b]}(\varphi) \nabla c$	iff $\forall t_i \in [t+a, t+b], i = \{1, \dots, n\}$ s.t. $(s, t_i) \models \varphi, (t_{i+1} - t_i) \nabla \frac{1}{c}$
8	$(s, t) \models \mathcal{P}_{[a,b]}(\varphi_1, \varphi_2) \nabla c$	iff $\mathcal{F}_{[a,b]}(\varphi_1) == \mathcal{F}_{[a,b]}(\varphi_2) == d, \forall t', t'' \in [t+a, t+b]$ s.t. $(s, t') \models \varphi_1, (s, t'') \models \varphi_2, \text{mod}( t'' - t' , \frac{1}{d}) \nabla c$
9	$(s, t) \models \mathcal{C}_{[a,b]}(\varphi_1, \dots, \varphi_n)$	iff $\forall t_i \in [t+a, t+b], i = \{1, \dots, n\}$ s.t. $(s, t_i) \models \varphi_i, t_i < t_{i+1}$
10	$(s, t) \models \mathcal{Sp}_{[a,b]}(\varphi, m)$	iff $\forall t_i \in [t+a, t+b], i = \{1, \dots, n\}$ s.t. $(s, t_i) \models \varphi, (t_{i+1} - t_i) \geq m$
11	$(s, t) \models \mathcal{B}_{[a,b]}(\varphi, N, d_k, m)$	iff $\forall t_i \in [t+a, t+b], i = \{1, \dots, n\}$ s.t. $(s, t_i) \models \varphi, (i == N), (t_i - t_1) \leq d_k, t_{i+1} - t_i \geq m$

Here, we express the frequency constraint as  $f = \frac{1}{T}$  where  $T$  is the measured period between each pair of consecutive events. For instance, the frequency of power lines in North America is 60 Hz. If we pass the power line signal into a zero cross detector, it should yield a sequence of events with half of the period of the sinusoidal signal, i.e.  $T = \frac{1}{2 \times 60}$  s. Figure 3.a shows a frequency constraint with period  $T$  on two repetitive signal events compared with their corresponding thresholds  $th_1$  and  $th_2$ .

**3.2.5 Phase [ $\mathcal{P}_{[a,b]}(\varphi_1, \varphi_2)$ ].** A phase constraint specifies a desired latency between two repetitive signal events with the same frequency (i.e.  $\mathcal{F}_{[a,b]}(\varphi_1) = \mathcal{F}_{[a,b]}(\varphi_2)$ ). For example, in power systems, having a specific phase between two sinusoidal signals at different locations is critical in

maintaining the system stability. Figure 3.a shows the phase constraint between two signal events with the same frequency,  $\frac{1}{T_1} = \frac{1}{T_2}$ , and the phase between two signal events constructed from signals  $s_1$  and  $s_2$  when they cross their corresponding thresholds is  $\Delta t$ .

**3.2.6 Sporadic** [ $Sp_{[a,b]}(\varphi, m)$ ]. Some events in CPS are not strictly periodic but the time interval between occurrences is bounded by a *minimum time* or possibly a *maximum time*. For example, the appearance of cars at an intersection or highway entrance can be expressed as a sporadic constraint. As Figure 4.b (part b.1) shows, after observing event  $e_1$ ,  $e_2$  should not occur for  $m$  seconds and after the occurrence time of  $e_2$ ,  $e_3$  should not occur for  $m$  seconds again.

**3.2.7 Burst** [ $B_{[a,b]}(\varphi, N, d_k, m)$ ]. A burst constraint describes a sequence of repeated non-periodic occurrences of a condition or event in a specified time interval. A burst constraint can be used to specify a minimum inter-occurrence recovery period necessary between consecutive event occurrences. For instance, whenever a condition is met 30 times in a 10 ms interval, the next occurrence should happen 10 s after the occurrence of the last one so that the system can recover. In Figure 3.b, the occurrence time interval is  $d_k$ , the occurrences limit is  $N$  and the recovery time is  $m$ .

### 3.3 TTL Semantics

TTL timing constraints can be combined with STL in order to express different kinds of timing constraints specified for CPS. The definition of *Globally* ( $\square$ ), *Eventually* ( $\diamond$ ), *Until* ( $\mathcal{U}$ ) and *Implication* ( $\Rightarrow$ ) operators are the same as expressed in STL [16] (Table 2), and they are expressed on a specific time interval in the future.

Combining TTL and STL does not always yield a meaningful statement. For example, to express a condition like “*Globally signal  $s_1$  crosses its threshold (3 V)*” (Zeno behavior) but, we can express a constraint as “*Signal  $s$  should be greater than 3 Until latency of two signal event is 3 ms*” or we may have  $\square_{[0,10]}(\mathcal{F}(s_1, 2.5, \nearrow)) > 5$ , which means in the next 10 s, the frequency of the events extracted from signal  $s_1$  crossing 2.5 V from below should be greater than 5Hz.

In Table 1, the satisfaction relation  $(s, t) \models \psi$  means signal  $s$  satisfies  $\psi$  starting from time  $t$ . A formula,  $\phi$ , is evaluated by comparing a signal with a number,  $c$  ( $>$ ,  $<$ ,  $=$ ).

The latency constraint is met if the difference between the time that the conditions  $\varphi_1$  and  $\varphi_2$  are satisfied is greater than, less than or equal to  $c$  which represents an at least, at most or exact latency constraint respectively (Table 1.5). Similarly, the frequency constraint is met if the time difference between every two consecutive instances that the condition  $\varphi$  is satisfied is greater than, less than or equal to  $\frac{1}{c}$  which represents an at least, at most or exact frequency constraint (Table 1.7). A phase constraint is met when the frequencies of satisfying conditions  $\varphi_1$  and  $\varphi_2$  are the same ( $\mathcal{F}(\varphi_1) = \mathcal{F}(\varphi_2) = d$ ) and the time difference between the instances within the period ( $\frac{1}{d}$ ) that the conditions  $\varphi_1$  and  $\varphi_2$  are satisfied is greater than, less than or equal to  $c$  which represents an at least, at most or exact phase constraint, respectively (Table 1.8). As defined in Table 1.9, a chronological constraint is satisfied when an increasing chronological ordering among the signal events  $\varphi_1, \dots, \varphi_n$  is met. In other words, the time that the first condition is met is earlier than the time that the second condition is met and the time that the second condition is met is earlier than the time that the third condition is met and so on i.e.  $t_1 < t_2 < \dots < t_n$ . For simultaneity constraint, the constraint is met if the conditions  $\varphi_1, \dots, \varphi_n$  are satisfied at the same time within a small tolerance window,  $t \pm \epsilon$ . In other words, if the set  $T = \{t_1, t_2, \dots, t_n\}$  contains the instances that each condition is met, the difference between the latest instance ( $\max(T)$ ) and the earliest instance ( $\min(T)$ ) should be smaller than  $\epsilon$ . A sporadic constraint is met if the minimum time interval between two consecutive instances that a condition  $\varphi$  is satisfied be at least  $m$  seconds (Table 1.10). The main difference between the “at least” latency constraint and sporadic constraint is the type of signal event the

Table 2. Semantics for level based constraints

$(s, t) \models \psi_1 \mathcal{U}_{[a, b]} \psi_2$	iff $\exists t' \in [t+a, t+b] (s, t') \models \psi_2 \wedge \forall t'' \in [t, t'], (s, t'') \models \psi_1$
$\diamondsuit_{[a, b]} \psi$	$\top \mathcal{U}_{[a, b]} \psi$
$\square_{[a, b]} \psi$	$\neg \diamondsuit_{[a, b]} \neg \psi$
$p \Rightarrow q$	$\neg p \vee (p \wedge q)$

condition  $\varphi$  is specified on. A latency constraint can only be specified on a singleton signal event while a sporadic constraint is specified on a repetitive signal event (Table 1). In order to evaluate a burst constraint as described in Table 1.11, whenever a condition,  $\varphi$ , is satisfied  $N$  times in the specified time interval,  $d_k$ , it shouldn't meet the condition,  $\varphi$ , again for a specified time interval,  $m$ .

When expressing a timing constraint specified on two or more signal events, there are some time intervals that do not contain any events. For instance, assume a latency constraint specified on two signal events  $s_1$  and  $s_2$ . After observing  $e_1$  on  $s_1$ , the value of latency is unknown until  $e_2$  is observed on  $s_2$ . In order to solve this problem, the output is represented as  $NaN$ , Not a Number, until the second event is observed (Figure 4).

**DEFINITION 5.**  *$NaN$ : Not a Number, a numeric data type that represents an undefined or unrepresentable value, described as  $\forall c \in \mathbb{R}, NaN \nabla c = \perp$  where  $\nabla \in \{>, <, ==\}$ .*

Based on the definition, the comparison between  $NaN$  and the other values is always false.

For instance, in Figure 4, part a.2, before observing the second event, the output of the latency constraint is  $NaN$  and after observing the event on  $s_2$ , the value of latency is equal to  $\Delta t$ .

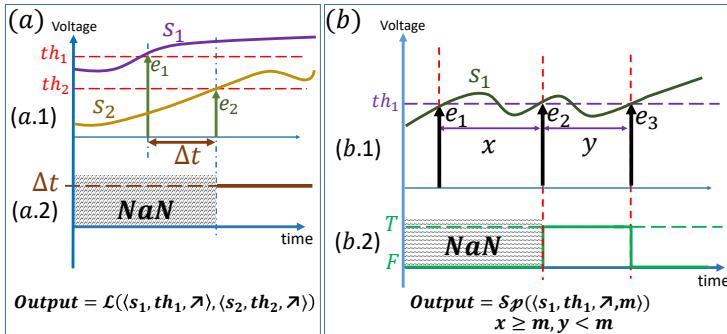


Fig. 4. a. Latency constraint between two events extracted from two signals (a.1) and its output (a.2). The output is  $NaN$  before observing the second event. b. sporadic constraint on a sequence of events (b.1) and its output (b.2).

### 3.4 TTL Capabilities

In this subsection, we outline TTL capabilities in capturing both event-based and value-based timing constraint in a simplified manner.

**3.4.1 Nested Constraints:** Constraints can be expressed in a nested form as illustrated in the following example.

$$\left( \diamondsuit_{[0, 100]} (C((\mathcal{L}(\langle s_1, 2.5, \nearrow \rangle, \langle s_2, 3, \nearrow \rangle) == 10), \langle s_3, 3.5, \searrow \rangle)) \right)$$

This statement specifies a constraint as: "Eventually, within the time interval  $([t, t + 100])$  there should be a chronological ordering between the time that the latency between two signal events

exceed their corresponding threshold is equal to 10s and, the time the signal  $s_3$  falls below its threshold. Note that, similar to STL, time intervals are relative (not absolute).

One concern in the nested TTL constraints is the range in which the statement is evaluated. In nesting statements, the time interval range of the inner constraint is extended based on the time interval range of the outer constraint. For instance, assume the following timing constraint:

$$\diamondsuit_{[a,b]}(\mathcal{S}_{[c,d]}(e_1, e_2, \dots, e_n))$$

It means the constraint is met at time “t“ if “always in the future time interval  $[t + a, t + b]$ , events  $e_1, e_2, \dots$  occur at the same time in the relative future time interval  $[t' + c, t' + d]$  where  $t + a < t' < t + b$ “.

**3.4.2 Combined Constraints:** The following expression shows a combinatorial example containing combined constraints represented in TTL: “*Whenever the value of  $s_1$  is greater than 3 V for 5 s and the value of  $s_2$  drops below 5 V in the next 5 s, then, within 5 s, the frequency of  $s_5$  crossing 0 V from below to above should be greater than 10 Hz for 2 s and  $s_3$  should go above 1 V simultaneously when  $s_4$  goes below 2 V*“. The corresponding TTL constraint is given as:

$$\begin{aligned} ((\square_{[0,5]}(s_1 > 3)) \wedge (\diamondsuit_{[0,5]}(\bowtie(s_2 < 5))) \Rightarrow \\ \diamondsuit_{[0,5]}(\square_{[0,2]}(\mathcal{F}(\langle s_5, 0, \nearrow \rangle) > 10)) \wedge (\mathcal{S}(\langle s_3, 1, \nearrow \rangle, \langle s_4, 2, \searrow \rangle))) \end{aligned}$$

In this example, the monitoring is done based on the **value** of a signal (e.g.  $s_1 > 3$ ), the time an **event** occurred (e.g.  $\langle s_2, 5, \searrow \rangle \Rightarrow (s_2 < 5)$ ) and TTL constraints (e.g. frequency and simultaneity).

Since the outputs of constraints are boolean signals, when they are used in a nested constraint, the time at which the output value becomes *true* is taken at the time of event occurrence.

**3.4.3 Absolute Time:** TTL has the capability to specify a constraint on the absolute occurrence time of an event. A signal event is generated at the desired timestamp ( $D_T$ ). For example, “*signal  $s_1$  should cross 3 V from below at 2017-04-06T19:46:54+00:00 as a UTC time format*”. In order to express an absolute time constraint, we have a signal called *Absolute Time Signal*( $A_T$ ) that keeps the absolute time. The timestamp is a 64-bit unsigned number, typically comprised of a 32-bit seconds field from a defined epoch and 32-bit fractional second field. In order to evaluate a constraint, it is enough to compare  $A_T$  with the desired timestamp ( $D_T$ ) and create a signal event and then, use it in a Simultaneity constraint. The aforementioned example can be represented in TTL as:

$$\mathcal{S}(\langle A_T, 2017 - 04 - 06T19 : 46 : 54 + 00 : 00UTC, \nearrow \rangle, \langle s_1, 3, \nearrow \rangle)$$

**3.4.4 Exclusive Constraint.** Another advantage of TTL is the capability of expressing a timing constraint with a specific profile like Burst with minimal syntax. For instance, the burst timing constraint used to avoid overheating issues can be specified using three parameters.

### 3.5 Tolerance Specification

When expressing a requirement on latency, frequency and phase constraints (including absolute time), the desired level of uncertainty tolerance is given by  $c$  as  $c = c \pm \epsilon$ . For example, assume an exact latency constraint on the occurrence of two events as  $\mathcal{L}(\varphi_{e_1}, \varphi_{e_2}) == c$  and the user-defined level of uncertainty tolerance as  $\epsilon$ , then, this constraint can be written as:

$$(\mathcal{L}(\varphi_{e_1}, \varphi_{e_2}) > c - \epsilon) \wedge (\mathcal{L}(\varphi_{e_1}, \varphi_{e_2}) < c + \epsilon)$$

Similarly, when defining a timing constraint with “ $<$ ” or “ $>$ ” operators, one can account for user-defined tolerance by defining  $c'$  to be:

$$\mathcal{L}(\varphi_{e_1}, \varphi_{e_2}) < c' = c + \epsilon$$

or

$$\mathcal{L}(\varphi_{e_1}, \varphi_{e_2}) > c' = c - \epsilon$$

respectively. The user may specify the desired tolerance for meeting the constraints by replacing the comparison value,  $c$  and cover the delay, jitter, timing error and other timing anomalies.

## 4 TIME TESTING METHODOLOGY

In this section, a methodology is presented to automate the testing process of timing constraint that can be implemented on commercially available platforms. In this process, there are three entities: 1. CPS (system under test), 2. TTL statements that specify the timing requirements, and 3. a measurement system that monitors the timing constraints.

### 4.1 Methodology Steps

Our proposed methodology has five steps to perform the testing on a CPS as follows:

**4.1.1 Making TTL Parse Tree.** In order to produce the parse tree from the TTL statement, all constraints should be written as a string, and all operators should be separated by parentheses. Then, the expression string is converted into a list of tokens. In each step, one token is taken from the array in order, then the rules are applied until all tokens are applied on the tree (a non-binary tree). After taking the last token, the parse tree is created. In parsing a TTL statement, there are three different types of tokens:

(1) Operators

- Temporal operators ( $\mathcal{U}$ ,  $\square$ ,  $\diamond$ ,  $\mathcal{L}$ ,  $\mathcal{S}$ ,  $C$ ,  $\mathcal{F}$ ,  $\mathcal{P}$ ,  $\mathcal{Sp}$ ,  $\mathcal{B}$  and  $\bowtie$ )
- Boolean operators ( $\neg$ ,  $\wedge$  and  $\Rightarrow$ )
- Comparison operators ( $>$ ,  $<$ ,  $\equiv$ ,  $\nearrow$  and  $\searrow$ )

(2) Operands

- Signals
- Real numbers

(3) Separators

- “(”, “)”, “⟨” and “⟩”

We use right child and left child notations to show the ordering between the children of a node. We define the rules to parse a TTL statement in Table 3. In Figure 5, the TTL statement is converted into its corresponding parse tree by the algorithm.

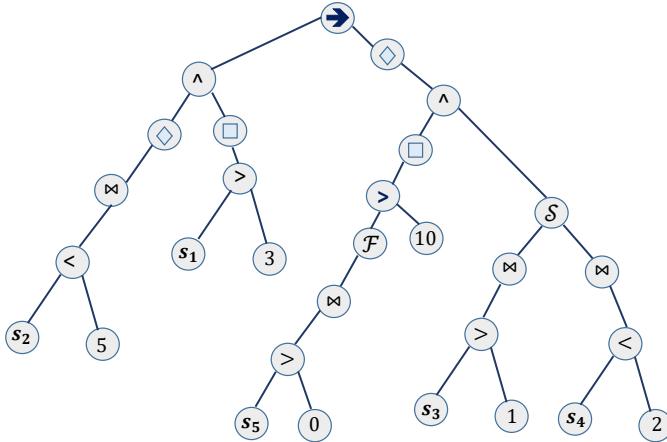
**4.1.2 Creating the Block Diagram.** Each node in the parse tree corresponds to a computing block in the block diagram except for leaves which are monitored signals and thresholds (Figure 6).

The proposed methodology is implemented on National Instruments (NI) equipments using LabVIEW software which has the entire driver for monitoring.

Each block produces the output for its parent based on the inputs it receives. Chronological, simultaneity, sporadic and burst constraint blocks produce a boolean output signal to show the exact time at which signals meet their constraints. For example, in Figure 4.b, there are three events on  $s_1$  and because the second event,  $e_2$ , appears after the first one,  $e_1$ , (later than minimum time interval  $x \geq m$ ), the output is “1” and because the third event,  $e_3$ , occurred less than the minimum time ( $y < m$ ) after  $e_2$ , the output is “0”. It is clear that before observing  $e_2$ , the output value cannot be computed so it will be *Nan*.

Table 3. The rules to create a parse tree from a TTL statement

Current Token	Next Token	IsEmpty(G)	IsNotEmpty(G)
"(" or "/"	X	Create a new node	Create a right child
Operands	X		Create a right node and assign the operand to it
All operators except "/" and "	X	Assign the operator to the current node	Travel upward until reach an empty node and assign the operator to it. If no empty node is found, create a parent node for the root, assign the operator to it and goto the new root
" ; "	is not "("		goto the parent node
" ; "	is "("		goto the parent node and create an empty right child node for it.
" )"	X		goto the parent node
" > "	X		goto the parent node and assign ">" to it
" \wedge "	X		assign ">" to the current node
" \vee "	X		assign "<" to the current node



$$\left( \left( \left( \square_{[0,5]} (s_1 > 3) \right) \wedge \left( \diamond_{[0,5]} (\bowtie (s_2 < 5)) \right) \right) \Rightarrow \left( \diamond_{[0,5]} \left( \left( \square_{[0,2]} ((\mathcal{F}((s_5, 0, \nearrow))) > 10) \right) \wedge (\mathcal{S}((s_3, 1, \nearrow), (s_4, 2, \searrow))) \right) \right)$$

Fig. 5. Generated parse tree for a TTL statement.

The outputs of latency, frequency and phase constraint blocks are real-valued signals and after comparing with a threshold and applying the differentiate operator they are converted into signal events.

**4.1.3 Creating the Physical Connection.** After determining the list of monitored signals, we use an appropriately isolated data acquisition device to measure the signals without changing the functionality of the CPS. For instance, the input impedance of the acquisition device should be high enough so that the system does not experience any voltage drop [21]. Appropriately shielded cables should be used in order to avoid interference between signals, especially in high-frequency applications. Similarly, we need to use interface circuits like optocouplers to improve the isolation when isolation of the acquisition device is not high enough.

**4.1.4 Signal Monitoring.** In order to deal with timing constraints in TTL, real-valued signals should be represented as signal events based on the edge type (rising, falling). Then, using a reliable

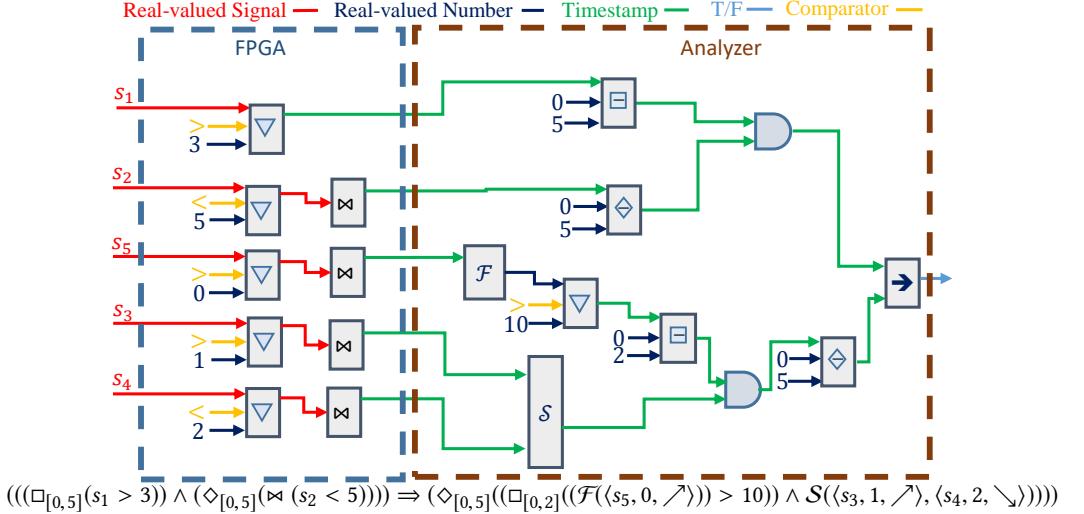


Fig. 6. TTL computing blocks according to the parse tree (Figure 5). Data acquisition and converting signals to timestamps are done on FPGA. The analyzer receives the timestamps, rebuilds the boolean signals and analyzes them.

clock, signal events are converted to timestamps and stored in a database for offline analysis or sent to an online testing application.

Similar to STL, TTL semantics in Table 1 are expressed for continuous signals, where each signal event represents a single point in time. However, implementation on hardware platforms requires discretization of the continuous signal. We assume that all signals are well-behaved such that there are no undetectable transient events between contiguous pairs of samples. This can be achieved by using appropriate Data Acquisition (DAQ) devices that have high sampling rates.

Time synchronization, absolute or relative, is necessary in measuring temporal properties of a distributed SUT. A distributed test device must be synchronized to ensure accurate, reproducible and repeatable measurement of the SUT based upon the temporal constraints evaluated. Providing timestamps based on a global time is an option in a system in which the occurrence of an event in absolute time matters. Otherwise, measuring the relative time between events is sufficient, and time testing can be implemented by the methodology regardless of accuracy to global time.

**4.1.5 Constraint Evaluation.** The result of a TTL statement is evaluated from the right-most block of the analyzer (Figure 6). Past operators are used to specify a timing constraint over past time intervals as defined in [13, 17] since they should wait for future time to evaluate. For evaluating future operators,  $\diamond[a, b]$  and  $\square[a, b]$ , the analyzer waits for  $b$  seconds and then starts evaluating the constraint. Therefore, the response at each instance ( $t$ ) actually is corresponding to  $t - b$ . Obviously, online monitoring of a constraint that contains future operators with infinite time interval is impossible (e.g.  $\square(x > 3)$ ). In order to solve this issue, the compiler modifies the operator to one with time interval,  $[t_s, t_f]$  where  $t_s$  and  $t_f$  are start and finish time of the testing. In our methodology, we use timestamps to represent a signal event. A boolean signal can be captured by two sequences of timestamps (rising and falling edges).

## 4.2 Methodology Capabilities

Since our methodology works based on the timestamps of events, it requires less memory for monitoring and is suitable for implementing future operators in online testing. In STL, an event-based constraint is evaluated for every instance of time while in TTL, an event-based constraint is evaluated by observing the timestamp of events only. For example, a latency constraint as:

$$\mathcal{L}(\langle s_1, 0.5, \nearrow \rangle, \langle s_2, 0.6, \nearrow \rangle) \leq 1$$

is evaluated by comparing only the subtraction of two timestamps extracted from the rising edges of signals  $s_1$  and  $s_2$  crossing their corresponding thresholds (i.e. the result is true if  $t_2 - t_1 \leq 1$  where  $t_1$  and  $t_2$  are extracted timestamps corresponding to rising edges of signals  $s_1$  and  $s_2$  thresholds 0.5 and 0.6, respectively). For evaluating frequency, latency and phase constraints, their outputs (a real-valued number) are compared with another number and provides a True/False signal. This True/False signal is represented by a set of rising and falling timestamps. For instance, as Figure 6 shows, once the frequency of the signal  $s_5$  is greater than 10 Hz, the output will be true. This boolean signal is represented as two sets of timestamps containing rising and falling edge. Then, the output timestamps are passed to the globally operator( $\square$ ). The globally operator subtracts 2 s from every signal's falling edge timestamp. If the result is less than the previous signal's rising edge timestamp, the timestamp is fully removed. Moreover, our methodology has the capability to be implemented on FPGA boards which are fast, reliable and low-cost. The acquired timestamps can be analyzed on the FPGA itself, transferred to a machine with a higher performance to process, or stored in a database when the CPS is distributed for offline analysis.

## 5 TESTBED IMPLEMENTATION AND CASE STUDIES

We implemented two case studies and defined their timing constraints in TTL. Then, upon the TTL statements, the testbed application is prepared in LabVIEW 2015. Validation of the results from the testbed application is compared with measured signal events on an oscilloscope.

### 5.1 Power Grid Synchronization

In order to reconnect a generator to the power grid for distribution of Alternating Current (AC) power, the generator should be synchronized to the system parameters to ensure voltage and frequency stability. When power components providers are connecting to the AC grid, their voltage (amplitude), frequency and phase must match. The frequency must be within  $\pm 0.067$  Hz in 60 Hz and the maximum phase deviation allowed is  $\pm 10$  degrees [1]. Since frequency and phase parameters are time sensitive, we can specify timing constraints on these. We modeled a power grid synchronization using two DC motors where the first one (master motor) represents the grid reference for frequency and phase, and the second one (slave motor) demonstrates the generator which should be controlled by the master motor. Figure 7 shows the schematic of controlling two DC motors.

Two dials labeled from 0 to 360 degrees are installed on the motors' shafts to illustrate two sinusoidal signals. The angular speed of reference motor is set to 60 revolutions per second which indicates the power grid frequency (60 Hz). A small hole is drilled on both dials at zero degrees in order to detect a revolution using photomicrosensor. The goal is to synchronize the speed (frequency) and phase of the slave motor with the master after an activation event rises. The setup for our two DC motors is depicted in Figure 8. In this setup, an Arduino Mega 2560 board is used for each motor, and they are implemented as a distributed synchronization system. Two Arduino boards are connected by two wireless modules (NRF24L01+, 2.4 GHz) by which the master motor controller sends required data to the slave controller. Moreover, the master controller has an

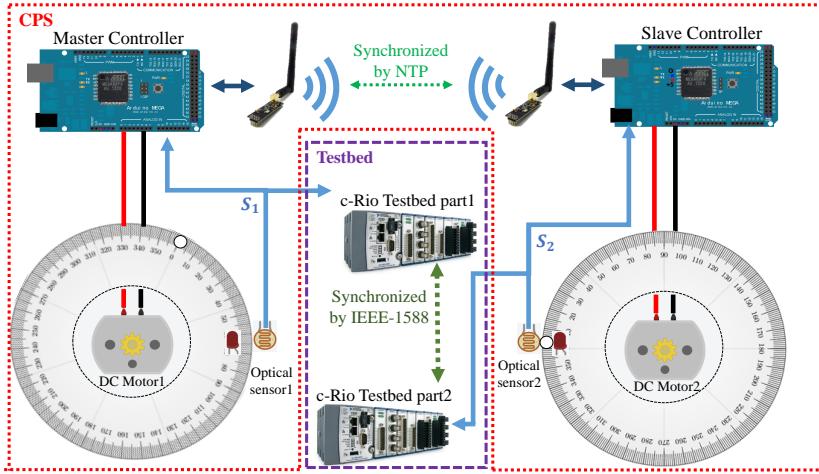


Fig. 7. Two motors are controlled by two Arduino Mega 2560 boards synchronized by wireless modules (NRF24L01). The phase of motors are monitored by two cRIO platforms.

additional role for the slave and the slave uses it as the reference for clock synchronization by NTP [18]. Using NTP, the two devices can be synchronized to a precision of about 2 ms through the exchange of NTP messages.



Fig. 8. The left motor is the master and the right one is the slave. The frequency of both motors should be 60 Hz and their phase difference should be less than 4 ms.

Since the CPS implementation is in a distributed manner, the testbed should be distributed as well (Testbed part 1 and 2 in Figure 7). One cRIO controller is dedicated for each Arduino board to monitor the sensors' signals. In a distributed system, having a common understanding of time is a critical feature of the testbed. In order to achieve this capability, cRIO controllers use the NI-TimeSync plug-in that utilizes the IEEE 1588 Precision Time Protocol (PTP)[12] as the synchronization protocol with 100 ns precision utilizing a Local Area Network (LAN).

In order to monitor the device timing, an NI-9381 module is used on each cRIO 9067 which contains 37 pins including eight analog input/output and four digital input/output pins. We use

two analog input pins on NI-9381 and connect them to the sensor output pins on each motor. Two installed sensors, Omron EESX970C1, have 5 V as their output when they detect the hole. Once the sensor output crosses 2.5 V from below, a hole is detected (the threshold is 2.5 V).

The cRIO controller is managed by the LabVIEW tool containing a front panel interface. The front panel is the user interface, which includes controls and indicators to send commands and monitor the parameters. In the testbed implementation, using the LabVIEW front panel shows the result of meeting the time constraint of the motor synchronization scenario online. Moreover, in order to verify the phase constraint, it is also validated using an oscilloscope (Figure 9). The testing methodology has the capability to test the constraints by logging signal timestamps then applying the testing approach in an offline manner, but here we implemented the testbed online.

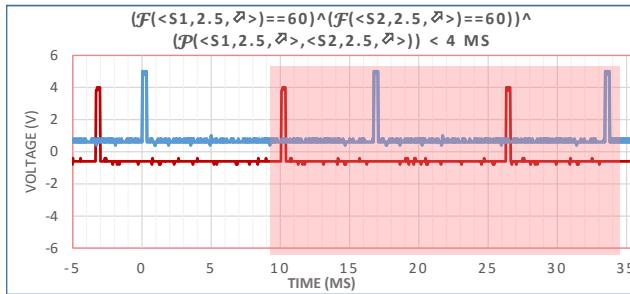


Fig. 9. Results from the power systems case study. The blue and red signals are acquired from optical sensors on master and slave motors, respectively. The transparent red area shows the time interval that the constraint is not met (phase is not less than 4ms).

According to the testing framework, sensors on both motors are monitored, and event timestamps are sent to the LabVIEW 2015 application. The application is executed on a 64-bit Windows 7 with Intel(R) Core TM i7 2.93 GHz and 8 GB of RAM.

In order to test the timing of this application, its timing specifications should be defined clearly according to the timing constraints listed in section 3. Then, they can be represented by TTL statements and ready to be tested by the testing methodology. We defined the timing constraint for this case study as “*The frequency of the rising edges of two signals  $s_1$  and  $s_2$  from the master and the slave sensors crossing threshold, 2.5 V, must be 60 Hz and the time at which two sensors detect the drilled hole should be exactly the same in each period with at most 4 ms error*”.

In this case study, timing constraints are considered as: 1) the frequency of both motors should be the same at 60 Hz and, 2) the phase between the motors should be less than 4 ms. So, the TTL statement for the master controller is:

$$\mathcal{F}(\langle s_1, 2.5, \nearrow \rangle) == 60 \text{ Hz}$$

Similarly, we can write the TTL statement for the slave controller as:

$$\mathcal{F}(\langle s_2, 2.5, \nearrow \rangle) == 60 \text{ Hz}$$

and

$$\mathcal{P}(\langle s_1, 2.5, \nearrow \rangle, \langle s_2, 2.5, \nearrow \rangle) < 0.004 \text{ s}$$

We rewrite these three TTL expressions with a single TTL statement separated with logic AND( $\wedge$ ). Our monitoring program evaluates results online. Figure 13 shows two snapshots of the analyzer evaluating the timing constraint of the case studies.

In this case study, two 1024 data size Direct Memory Access (DMA) First-In-First Out (FIFO) data transfer with *64-bit quad signed integer* are used with time measurement precision on the order of  $25\text{ ns}$ . This is because the cRIO NI-9067 works with a  $40\text{ MHz}$  clock. The FPGA synthesis report indicated the Total Used Slices as: 16.9 % (2,257 out of 13,300). Here, we allocated a large size buffer for the DMA FIFO on the FPGA in order to reduce the communication and processing on the desktop computer because smaller buffer sizes can cause loss of data. The TTL analysis is done on a desktop computer by gathering all timestamps from the cRIO devices.

## 5.2 Simultaneous Image Capturing for 3D Reconstruction

For the second case study, we implemented a distributed system with two cameras. Each camera takes a picture of a rolling ball from different angles. We used two ArduCAM UNO boards whose ESP8266 wireless modules are used for communication and 2 MP Complementary Metal-Oxide-Semiconductor (CMOS) cameras to take images and save on SD cards. Figure 10 depicts the experiment testbed as well as the monitoring platform.

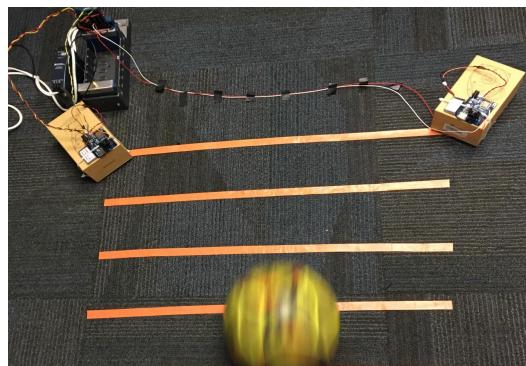


Fig. 10. Two ArduCAM boards taking simultaneous images from a rolling ball. The cRIO 9067 platform is used to monitor the trigger signals on both cameras.

As Figure 11 shows, there is a server (a desktop computer or a laptop) with a wireless communication device to send the command to camera boards. Once each camera receives the message from the server, it should take a picture after no more than  $0.2\text{ s}$  and send it back to the server. Then, these images can be used for 3D image reconstruction. In order to avoid blurring, all images should be taken simultaneously. ArduCAM1 (the bottom camera in Figure 11) raises a flag on its digital pin #2 once it receives the command. This pin is connected to the testbed as signal  $s_1$ . Each ArduCAM board is set to raise a flag on one of digital pin #3 when they take the photo. The pin #3 of each camera is connected to the corresponding cRIO device as signals  $s_2$  and  $s_3$ . Applying the configuration from the first case study, each camera is attached to one of the cRIO devices. Since we defined that the cameras must take the pictures synchronously, the events detected on  $s_2$  and  $s_3$  should occur simultaneously within  $0.2\text{ s}$  after they receive the command message. Thus, the time interval between the event detected on signals  $s_1$  and  $s_2$  should be less than  $0.2\text{ s}$ .

In specifying the timing constraints, we can define the requirement as a latency constraint less than a specified value and a simultaneity constraint within a time error tolerance. For this application, we defined the timing constraints as: *The signal  $s_2$  should go above 2.5 V at the same time with signal  $s_3$  when it goes above 2.5 V with 0.01 s tolerance and the latency between the time that the activation signal,  $s_1$ , goes above 2.5 V and the time that the capture signal,  $s_2$ , goes above 2.5 V should be less than 0.2 s.*

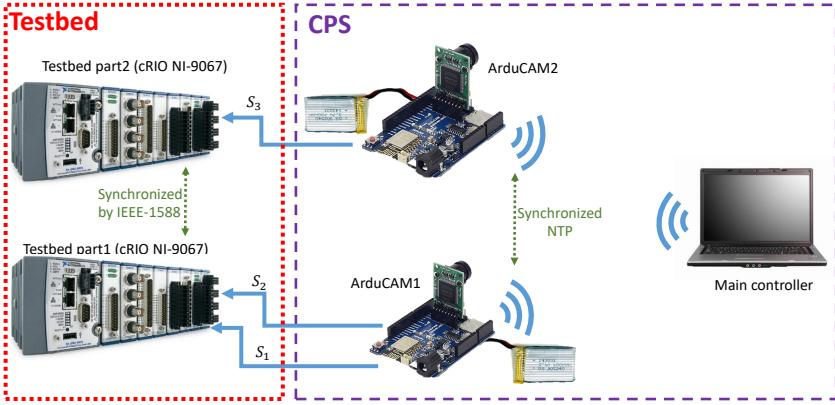


Fig. 11. Two ArduCAM UNO camera boards are connected to a server by wireless ESP8266 wireless modules. The server sends the command to take a picture simultaneously.

In TTL, we can write these timing constraints as:

$$(\mathcal{S}(\langle s_2, 2.5, \nearrow \rangle, \langle s_3, 2.5, \nearrow \rangle, 0.01) \wedge (\mathcal{L}(\langle s_1, 2.5, \nearrow \rangle, \langle s_2, 2.5, \nearrow \rangle) < 0.2))$$

The automated monitoring of the TTL constraints showed that the latency difference in the actuation times of cameras was 0.01 s, as well as the data monitored on the oscilloscope (Figure 12.a). Similarly, the interval between  $s_1$  and  $s_2$  is less than 0.2 s that is validated by the oscilloscope results depicted in Figure 12.b. Therefore the constraint is met on the deployed platforms and the boolean indicator shows the constraint has been satisfied. (Figure 13.b).

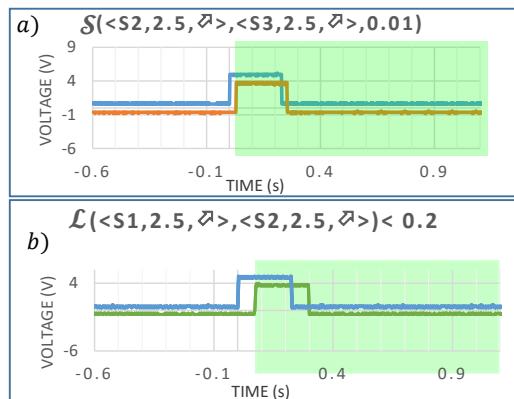


Fig. 12. Results from distributed camera case study. a) Images are taken simultaneously. b) The delay between the time that the images are taken is less than 0.2 s.

In this case study, three DMA FIFO memory buffers are used on FPGA for processing and Total Slices: 28.9 % (3849 out of 13300) of cRIO-9067 FPGA was used. Figure 12 shows the two trigger signals monitored on the oscilloscope.

## 6 CONCLUSIONS AND FUTURE WORK

In this paper, we augmented a framework to express temporal constraints based on the physical and application requirements of CPS. A systematic methodology for automatic test generation and timing constraints monitoring based on global time is proposed. The feasibility of our approach is demonstrated on experiments using two case studies of voltage stability in the power grid as a measurement and control application as well as a 3D image reconstruction application related to synchronous actuating.

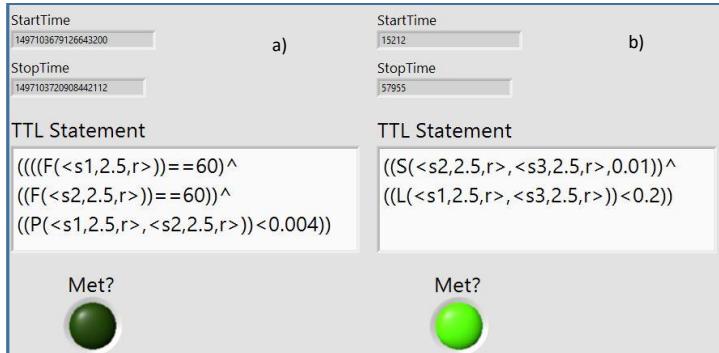


Fig. 13. Snapshots of our analyzer written in LabVIEW for case studies. Timing constraints are not met for power grid simulation (a) and met for simultaneous image capturing (b)

As future work, we will use TTL and our testing methodology on a system with distributed measuring devices in order to extend our work to large-scale systems.

## 7 ACKNOWLEDGEMENTS

The authors would like to express appreciation for the NIST technical reviewers, Spencer Breiner, Dhananjay Anand, and Martin Burns, who provided technical feedback and additional ideas for future work.

*The mention of commercial products in this paper is for illustration purposes only and does not imply recommendation or endorsement by the National Institute of Standards and Technology (NIST).*

## REFERENCES

- [1] The Grid Code (February 2017) Issue 5, Revision 20.
- [2] 2016. NIST Cyber Physical Systems Program. <https://www.nist.gov/programs-projects/cyber-physical-systems-program>. (2016). [Online; accessed 12-September-2016].
- [3] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. 1996. The benefits of relaxing punctuality. *Journal of the ACM (JACM)* 43, 1 (1996), 116–146.
- [4] Andreas Bauer *et al.* 2006. Monitoring of real-time properties. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer.
- [5] Johan Bengtsson *et al.* 1996. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems III*.
- [6] Jyotirmoy V Deshmukh *et al.* 2015. Robust online monitoring of signal temporal logic. In *Runtime Verification*. Springer.
- [7] Alexandre Donzé. 2010. Breach, a Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *CAV*.
- [8] Alexandre Donzé *et al.* 2012. On temporal logic and signal processing. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 92–106.
- [9] Georgios Fainekos *et al.* 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410 (2009).
- [10] Goran Frehse *et al.* 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *CAV*.

- [11] Thomas A Henzinger *et al.* 1997. HyTech: A Model Checker for Hybrid Systems. In CAV.
- [12] IEEE Instrumentation and Measurement Society. 2002. IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (IEEE Std 1588-2002). (2002).
- [13] Stefan Jakšić *et al.* 2015. From signal temporal logic to FPGA monitors. In *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*. IEEE.
- [14] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-time systems* 2, 4 (1990), 255–299.
- [15] Oded Maler and Dejan Nickovic. Monitoring Temporal Properties of Continuous Signals. In *FTRTFT 2004*. Springer.
- [16] Oded Maler *et al.* 2008. Checking Temporal Properties of Discrete, Timed and Continuous Behaviors. In *Pillars of computer science*.
- [17] Oded Maler *et al.* 2013. Monitoring properties of analog and mixed-signal circuits. *International Journal on Software Tools for Technology Transfer* 15 (2013).
- [18] D Mills. 1989. Network Time Protocol (Version 2) Specification and Implementation; RFC-1119. *Internet Requests for Comments* 1119 (1989).
- [19] Dejan Nickovic and Oded Maler. 2007. AMT: A Property-based Monitoring Tool for Analog Systems. In FORMATS.
- [20] Aviral Shrivastava *et al.* 2016. Time in Cyber-Physical Systems. In *Proc. of CODES+ISSS*.
- [21] Aviral Shrivastava *et al.* 2017. A Testbed to Verify the Timing Behavior of Cyber-Physical Systems. In *Proceedings of The 54th Annual Design Automation Conference*.