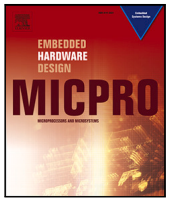
Microprocessors and Microsystems 101 (2023) 104890

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro



A run-time verification method with consideration of uncertainties for cyber–physical systems

Mohammadreza Mehrabian [a,*], Mohammad Khayatian [b], Aviral Shrivastava [c], Patricia Derler [d], Hugo Andrade [e]

[a] South Dakota School of Mines and Technology, USA
[b] Vecna Robotics, USA
[c] Arizona State University, USA
[d] PARC, a Xerox Company, USA
[e] AMD Inc., USA

ARTICLE INFO

Keywords:
Cyber–physical systems
IoT
Run-time verification
Real-time systems
Robotics
Temporal logic

ABSTRACT

Since many Cyber–Physical Systems (CPS) interact with the real world, they are safety- or mission- critical. Temporal specification languages like STL (Signal Temporal Logic) have been developed to capture the properties that built CPS must meet. However, the existing temporal logics/languages do not provide a natural way to express the tolerance with which the timing properties must be met. As a consequence of this, the specified properties may be vague, the ensuing CPS design may end up being over- or under-provisioned, and the validation of whether the built CPS meets the specified CPS properties may turn out to be erroneous. To address these issues, a run-time verification methodology is proposed, that allows users to explicitly specify the tolerance with which timing properties must be met. To ensure the correctness of measurement-based validation of a built CPS, this article: (i) proposes a test to determine if a given measurement system can validate the properties specified in TTL, and (ii) proposes a measurement-based testing methodology to provide one-sided guarantee that the built CPS meets the specified CPS properties. The guarantees are one-sided in the sense that when the measurement-based testing concludes that the properties are met, then they are guaranteed to be met (so not false positive). However, when the measurement-based testing concludes that the properties were not met, then they may have met (there can be false negative). In order to validate our claims, we built a model of flying paster (part of the printing press that swaps in a new roll of paper when the current roll is about to finish) using Arduino Mega 2560 and two Hansen brushed DC motors and specified the timing constraints among the various events in this system, along with the tolerances with which they should be met in TTL. We generated the testing logic and validated that we get no false positive, even though we encounter 4.04% false negative. The rate of false negatives can be reduced to be less than any arbitrary value by using more accurate measurement equipment.

1. Introduction

Cyber–Physical Systems (CPS) are systems that integrate the interaction of computational and physical worlds and enable intelligent and automated sensing and control. Many current and envisioned CPS, like intelligent traffic management systems [3,4], smart grids [5], drones [6], etc., have safety–critical requirements that must always be met. Many specifications[1] concerning the safety and performance of CPS are related to the timing of the system where the correctness is not only tied to providing the correct response but also at the right time [7,8]. Since specification of a timing constraint using our natural language can be ambiguous, several types of temporal logic have been proposed to formally specify the timing behavior of a system. LTL (Linear Temporal Logic) [9] is used to capture properties of Boolean predicates, MTL (Metric Temporal Logic) [10] can capture

* Corresponding author.
E-mail addresses: mohammadreza.mehrabian@sdsmt.edu (M. Mehrabian), mkhayati@asu.edu (M. Khayatian), aviral.shrivastava@asu.edu (A. Shrivastava), patricia.derler@parc.com (P. Derler), hugo.andrade@amd.com (H. Andrade).

[1] Terminology note: In this paper, we use the term "constraint" to imply what the system must do/achieve, and "specification" to imply the limits of the system components. For example, while building an autonomous vehicle, the timing constraint may be that the vehicle should be able to drive at 80 kmph, but engine specification may be that it can only achieve 60 kmph.

https://doi.org/10.1016/j.micpro.2023.104890
Received 2 May 2022; Received in revised form 27 April 2023; Accepted 5 June 2023
Available online 22 June 2023
0141-9331/© 2023 Elsevier B.V. All rights reserved.

system properties using Boolean predicates in continuous time and STL (Signal Temporal Logic) [11] can specify system properties in terms of real-valued signals over continuous time.

The main limitation of the existing temporal logics/languages that we identify in this article is that they do not explicitly consider the *tolerance* with which the specified constraint must be met.[2] For example, suppose there is a timing constraint that a signal $y(t)$ must become high 200 ms after the signal $x(t)$ becomes high. This timing constraint can be represented in STL as $\square\ \big(\uparrow x(t)^{3} \rightarrow \diamond_{[200\ \mathrm{ms}]}(\uparrow y(t))\ \big)$.[4] However, this timing constraint is vague — since it does not specify the tolerance with which the constraint must be met. Time is inherently continuous and therefore the delay between two events cannot be exactly equal to a real-value. Moreover, every measurement device has non-zero uncertainty in measurement and the exact (real) time of the occurrence of an event cannot be reestablished from a record of the timestamps at which they were measured to occur [12], and therefore the given property can only be evaluated to a certain finite uncertainty. The right constraint should have been that — signal $y(t)$ must become high 200 ms ± 10 ms after the signal $x(t)$ becomes high.

Another example is distributed synchronized cameras where 3 cameras must take a picture at the same time from different angles to combine and perform 3-D scene reconstruction. The constraint for this system can be written in STL as $\square(\uparrow t_A \rightarrow \diamond(\uparrow t_B) \rightarrow \diamond(\uparrow t_C))$. Here, t_A, t_B, and t_C are the times of the shutter events on the 3 different cameras. Note that this timing specification also does not specify the required precision (millisecond, microsecond, or nanosecond precision) with which the timing constraint must be met. If these constraints are interpreted strictly, then it is impossible to build such a system. In the absence of tolerance specification, it is impossible to figure out if a built CPS is over or under-designed.

Furthermore, if the tolerances are not specified, it is hard for the verification teams to determine if a built CPS meets the specified properties. For example, if the timing constraints must be met with a tolerance of nanoseconds, but the measurement system can only measure with millisecond precision, then it is clearly not possible to use the measurement system to validate the CPS properties. By having tolerance in the specification, it is clear in the verification step to choose the right systems. If the acceptable uncertainty is not formally determined in the system specifications, verification teams may need to study the system or use informal methods to figure it out.

Finally, we show that even if have a precise-enough measurement system, because of the inherent uncertainty in measurements it is impossible to guarantee that a built CPS meets the specified timing properties if the tolerance with which the properties must be met are not specified. In fact, type I (false positive), and type II (false negative) observations are possible. This means that even if the measurement system says that a property is being met, it may actually not be met, and even if the measurement system says that a property is not being met, it may actually be met. For safety–critical properties, the type I misjudgement (false positive) can be especially dangerous.

In summary, the contributions of this article are:

- The article argues that the tolerance with which timing properties must be met by a CPS, must be specified along with the CPS properties, otherwise the property specification may be vague, and the validation of whether the built CPS meets the specified CPS properties may be erroneous, and the ensuing CPS design may end up being over-provisioned or sub-optimal. The method in this paper uses a specification language, TTL (Timestamp Temporal Logic) [13], proposed in 2017, that allows the users to explicitly specify the tolerance with which timing constraints must be met.
- The proposed approach considers several factors of the measurement system, including the precision of digital-to-analog and analog-to-digital converters and clock synchronization accuracy that affect the precision of timestamps, to determine if the measurement system can be used to validate if a built CPS meets its timing properties specified in TTL or not.
- We propose a verification methodology that given the precision of measurement equipment, will ensure that type I misjudgements cannot happen. This is achieved at the cost of more type II misjudgements (false negatives). However, the rate of type II misjudgements (false negatives) can be reduced to arbitrary levels by improving the precision of the measurement equipment.

To demonstrate the usefulness of our approach we developed a Flying Paster system — the part of the printing press that swaps the active roll of paper with a new one when the active roll is about to run out of paper and breaker tripping in power systems. We implemented a model of flying paster using Arduino Atmega 2560 and two DC motors, and specified the safety and performance-related timing specifications without tolerances in STL and with tolerances in TTL.

By using our approach, we determine that the original monitoring system (using Arduino Atmega 2560s) was not accurate enough to validate the required properties. The uncertainty in the monitoring system (i.e., 106 μs) was more than the precision required by the properties (100 μs) of the flying paster application. In fact, the data acquisition system could only sample at 10 kS/s, while to validate the timing constraints of flying paster with tolerance of 100 μs, we needed at least a sampling rate of 20 kS/s. We switched to National Instruments NI-cRio that allows more than 20 Kilo samples per second sampling rate, and we were able to validate the properties over the built CPS.

We developed the logic for validating whether the built flying paster system meets the timing properties. Experimental results show that there were no false positives (the number of cases where the built CPS does not meet the specification, but the monitoring system could not catch it) when we used logic derived from TTL specification that contains tolerances. However, the logic derived from the STL specification that does not contain tolerances showed about 2.61% rate of false positives. These false positives can be very dangerous for safety–critical properties.

We performed similar experiments on a MATLAB model of power breaker circuit. Power breakers are required for cutting off the electricity power in case of over-current. In order to prevent failure, it should satisfy some temporal properties like "*the breaker should trip if the duration at which the voltage is above 1.2 p.u. (per unit) is greater than 160 ms.*". In this experiment, we could demonstrate that without considering tolerance as a part of statements in temporal logic, three violations were not detected by the monitoring system (i.e., around 13%). However, since TTL considers uncertainties in the statements, there were no undetected violations, even though there were about 4% false negatives (cases when the monitoring system reported that the timing constraint was not met, but actually they were met). However, as mentioned before, the rate of false negatives can be reduced to below any arbitrary value by choosing a better measurement system. In the last experiment, we used tolerable errors in STL expressions and showed that to implement the required circuit on FPGA, TTL implementation needs fewer resources.

2. Related work

One approach to verify the temporal specifications of CPS is runtime verification where the system's behavior is monitored during its operation [14]. In order to verify the timing specifications of real-time systems, it is firstly required to express the system temporal behavior in a formal language like Temporal Logic (TL). Temporal

[2] We use the term *tolerance* to specify how much deviation from the ideal/nominal behavior is acceptable and we use the term *uncertainty* to capture the variation in the measured values.

[3] $\uparrow x(t)$ shows the rising edge when $x(t)$ becomes high (i.e., there is an event on $x(t)$) [1].

[4] $\diamond_{[a]}$ or $\diamond_{=a}$ is the *Punctual* form of $\diamond_{[a,b]}$ when $b = a$ [2].

Logic, proposed in [15], is a system of rules to represent the system behavior and reason about propositions qualified in terms of time. Pnueli [16,17] and Owicki et al. [18] have proposed Temporal Logic for the specification and verification of reactive systems. Linear Temporal Logic (LTL) [19], Computation Tree Logic (CTL) [20,21], Metric Temporal Logic (MTL) [10], Metric interval Temporal Logic (MITL) [22], Timed Propositional Temporal Logic (TPTL) [23], and Signal Temporal Logic (STL) [11] have been proposed to define the timing specifications of real-time systems. Although, the introduced temporal logic formalisms are very useful and expressive for representing the temporal specifications of real-time systems, representing uncertainties is still challenging.

Some of the common languages to express temporal specifications of CPS are LTL, MTL, and STL where they comprise three major timing operators (*Globally, Eventually*, and *Until*) for expressing any level-based timing constraint. However, the expressions are often combined and/or nested and must be evaluated recursively. Additionally, although those logic languages have the capability to express event-based timing constraints, combined temporal expressions are constructed out of a variety of level-based timing constraints. In order to represent only one event (rising or falling), we should use past and future operators together in one expression.[5]

In 2006, Fainekos proposed the robust interpretation of MTL over continuous-time signals taking values in metric spaces [24]. Since the classical methods to test temporal logic just involve boolean abstraction, when a specification is either satisfied or violated, it is not possible to know its degree. In essence, the robustness degree function gives a real value that indicates how far is a signal from violating or satisfying a specification using a metric ρ [25]. This technique is a significant improvement in falsification approaches [26,27]. Using Booleanziner rules, it is practical to transform STL formulas to MITL and define STL robustness in a similar approach [28]. In the domain of uncertainty, the robustness techniques can consider ρ as the maximum robust value for a specification and then, by considering ρ value, monitor the signal to know if it has satisfied the specification. In this regard, there are some algorithms and tools [22,29,29] for online monitoring of CPS. The algorithms basically make a parse tree for the formula and give a range to show the robustness of the monitored signal value. Although temporal logic robustness can be used to express uncertainty on values (spatial robustness) in MTL/STL, it does not have a direct definition to express the existing uncertain values (robustness) in *time* domain. In order to consider uncertainties in time domain, we can consider *time robustness* to know how robustly the formula is satisfied or violated with respect to time. In the other words, if the time robustness of a monitored signal is known, it is intuitively possible to consider a part of that (or its entire) as uncertainty. The approach proposed in 2010 [30] maps every spatial robustness to temporal. It has two definitions for left/right robustness[6] when a specification is either satisfied or violated. To calculate the robustness value, the algorithm receives two sort of different values, (i) a sequence of variable step-size time-tamped values of the monitored signal and, (ii) a time function with a finite sequence of points where its derivative is changed. Then, based on the signal values and the time derivatives, the time robustness can be calculated incrementally. Although the method calculates time robustness in an explicit way, it is computationally expensive for online monitoring since its complexity is of the order of the sum of the signal's sampling frequency. Furthermore, because the methods based on MTL/STL use both past and future operands to express signal events, representing events needs long and cumbersome expressions.

In Propositional Linear Temporal Logic (pLTL) [31], it is required to find an optimal solution for temporal specification in the system behavior to judge its correctness. As a fact, the time complexity of finding an optimal solution for a pLTL specification is doubly exponential in the number of prepositions in a single statement. There is a similar issue in using STL. One solution is to use Mixed Integer Linear Program (MIPL). However, MIPL is an NP-hard problem to solve. [32] proposed a method as a probabilistic extension to STL to evaluate complex specifications. Indeed, for uncertain and changing environments, a probabilistic variant of STL is proposed to express safety constraints on random variables. This approach presents an efficient receding horizon algorithm to maximize the probability of satisfaction of a temporal specification. In PrSTL, a time-bounded specification ψ is assigned to a system (e.g., CPS). PrSTL allows for computing the probability of satisfaction given a sequence of states over the target system. The ideal in this logic is estimating the true states of targets and because the estimates over the target states are given in the form of probability distributions, the signal evaluation is done in terms of probability. Nevertheless, these works only evaluate the signal in the form of probability distributions instead of the robustness or traditional Boolean evaluation based on stochastic methods with solid judgements. Moreover, they assume the state of the system is always fully known [33].

In the domain of analog mixed signals, AMS-LTL [34] is an extension of STL that uses the notion of events as atomic properties for the predicates in which an event is to express a change in the truth of the propositions. This logic proposes auxiliary state machines and auxiliary functions, two types of formalisms, for Analog and Mixed-Signal (AMS) assertions (e.g., a user can create properties or asserted behavior for AMS). It has been shown that the complexity of satisfaction/violation using the monitoring algorithm for AMS-LTL is EXPSPACE-complete.[7]

One popular solution for verifying the behavior of CPS is simulating the CPS and running the monitoring system beside it and see if the requirements are met at run-time. In this domain, the conventional monitoring methods generally use the same formalisms (i.e. STL/MTL/MITL) to express timing requirements in the monitoring processes. Since those languages do not explicitly/intrinsically express the tolerable error as a part of language, the developed monitoring system might be overestimated, large, or heavy in most of cases and may leave the violation of some monitored timing specifications undetected. Some tools for analyzing the timing requirements in CPS have been implemented in Breach [35], and S-Taliro [36]. Both tools record simulation data and evaluate timing constraints considering the simulation.

Recognizing the high overhead, AMT [37] proposed an incremental approach to compute the constraints at a segment granularity. An incremental method was proposed by Deshmukh et al. [38] where timing constraints are evaluated by traversing the parse tree generated for STL formulas. They optimize calculations by eliminating repetitive computations. Since all of these examples are implemented in simulation, they have the access to the real values and hence, do not consider the uncertainties in their computations.

Beside the previous approaches that work in simulation, there are some examples implemented on FPGA. *Selyunin* et al. [39] proposed a framework for generating monitors with recovery from a class of high-level STL specifications. This method firstly simplifies the STL formula, converts them into equi-satisfiable past operators, and then using an offline evaluation. The code is synthesized in a digital reconfigurable hardware. It uses SystemC simulation kernel to run the monitor on pre-recorded traces. R2U2 is another method implemented on FPGA for a security threat detection [40]. *Schumann* et al. proposed a technique receiving the properties of an Unmanned Aerial Vehicle (UAV) using MTL/LTL statements and then diagnosis security attacks by a Bayesian Network model. Indeed, the authors construct FPGA monitors for security requirements and specify possible attacks that a UAV might undergo.

[5] $\uparrow \psi = (\psi \wedge (\neg\psi \mathcal{S} \top)) \vee (\neg\psi \wedge (\psi \mathcal{U}^{\cdot} \top))$ for rising edges and $\downarrow \psi = (\neg\psi \wedge (\psi \mathcal{S} \top)) \vee (\psi \wedge (\neg\psi \mathcal{U}^{\cdot} \top))$ for falling edge.

[6] Similar to *Latency* constraint we propose in this paper.

[7] In complexity theory, EXPSPACE is the set of all decision problems solvable by a deterministic Turing machine in O(2p(n)) space, where p(n) is a polynomial function of n.

While the aforementioned methods do not discuss about the efficiency of the FPGA implementation, *Reinbacher* et al. in 2013 presented an algorithmic framework, for monitoring temporal specifications expressed in past-time MTL/LTL. They showed that their work optimized the required memory space since the needed storage for the signal history is bounded by $\lceil \log_2(n) \rceil$ where n is elapsed time from when the monitoring is started. Jakšić et al. [41] implemented a monitoring method called *Counters* algorithm on FPGA. The *Counters* algorithm reduces the computation complexity from $O(n^2)$ to $O(n \log(n))$, where n is the size of time interval of the temporal constraints. This technique converts future STL operators into past expressions and translates all constraints such that their interval starts from zero. Then, a counter is dedicated to measuring the duration of a positive pulse in each interval. The number of needed counters depends on the variability of the monitored signal and the length of the interval bound (a).

Although these methods showed a way to reduce memory usage, the storage remains a concern (even for bounded constraints). Since it is not possible to express the maximum allowed deviation for measured and computed values, the designer/verification developers implement their monitoring methods as precise as possible while it is not always required. Intuitively, such implementation needs more resources in comparison with the methods that adjust their precision degree considering use-defined allowed tolerance. Moreover, by relaxing the conditions with considering tolerance, it becomes possible to take uncertainties into account and cover the corner cases and provide a guarantee to make sure the monitoring approach is able to catch all timing violations.

3. Problems in run-time verification using existing temporal logics

Several temporal logics like CTL, LTL, MTL, and STL have been developed to specify the properties that a built CPS must meet. STL is most applicable to express the properties of CPS since it allows the specification of properties of real-valued signals in continuous time. In this section, we discuss about one of the common challenges in using STL expressions in the online monitoring of temporal specifications. In order to illustrate that, there is an example in the following section.

3.1. An example for time uncertainty in event detection

In STL, a CPS designer can express specifications like (examples taken from [42]):

$$\psi = \square(x(t) < 3.5).$$

This STL statement specifies that the value of the signal $x(t)$ should never go above 3.5 V.

Another example is:

$$\psi = \square_{[2,6]}(|x(t)| < 2)$$

This STL statement specifies that in the next 2s to 6s, the value of the signal $x(t)$ remains between −2 V and 2 V.

The issue that we want to draw attention to is that STL specifications do not provide a mechanism for the CPS designer to explicitly specify the tolerance with which these specifications should be met. This is the case, even though STL claims to model continuous time. Without the tolerance specification, it is impossible to correctly validate if a built CPS meets its timing constraints or not.

Consider the example shown in Fig. 1. The figure depicts a signal ψ that should be monitored by a verification system to see if it satisfies the STL statement, $\square_{[1,5]}\psi$. The STL statement expresses a property that is considered to be met at time t iff, in the next 1 to 5 s (i.e. $t+1$ to $t+5$), the signal (*psi*) is true continuously.

The diagram (Fig. 1) depicts that ψ becomes true at real-time 0.91 s and becomes false again at 4.81 s. If we do the calculation to determine the satisfaction of $\square_{[1,5]}\psi$ at $t = 0$, we will conclude that the temporal

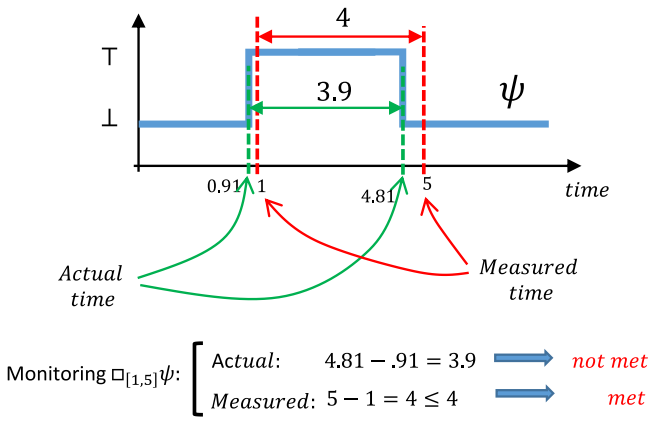


Fig. 1. The problem of monitoring a STL timing statement. Since in the discretization process an interval is just converted into a single value, the final evaluation can be wrong if the uncertainty in the measurement device is not taken into account.

specification **has not been satisfied**, since ψ was not true from 4.81 s to 5 s.

However, if we use a digital monitoring circuit with a sampling period ($\delta = 0.2$ s), then we reach a different conclusion. Suppose the sampling times (time steps) are $\{0, 0.2, 0.4, \ldots\}$ s. The sampling system will first record that the signal ψ is true will be at the 1 s mark. It will last record that the signal was true will be at the 5 s mark. So $\square_{[1,5]}\psi$ is found to be met.

False positive misjudgement (type I error) can cause serious problems, particularly in safety–critical applications because the measurement system concludes that the timing constraint is met, while in reality it is not met! This problem is severe because the probability of type I errors (or false positive evaluations) only decreases with the precision of the measurement equipment, but is not eliminated. Thus it is hard to guarantee the safety of the CPS.

One possible solution to fix this issue, for existing temporal logics e.g. STL/MTL/MITL, is to account for the measurement error by manually modifying the time intervals of the timing operators. However, this approach will be hard for nested/complex temporal statements, there is no specific method to consider tolerance in those languages, and if the tolerance is considered case-by-case, for each case there is a need to have a proof for the correctness. Since TTL semantics converts temporal specifications into mathematical conditions, it is possible to have general proofs for complex and nested specifications. Furthermore, since representing temporal specifications in STL needs more temporal operators, it needs more space on FPGA to design and verify in comparison with using TTL.

4. Tolerance in timestamp temporal logic (TTL)

All digital devices have a level of measurement uncertainty (say δ)and STL statements cannot be monitored correctly without considering the uncertainty (δ) in the statements. For example, if we want to check if a signal falls from high to low with a measurement system that has a sampling period of 0.2 s, then if the measurement system registers that the signal went from high to low at say the 5 second mark, then it is impossible to say when between 4.8 s and 5 s the signal went from low to high. The best we can say is that the signal went from high to low somewhere between 4.8 s and 5 s. In this case, we say that the uncertainty in the measurement of the time at which the signal went from high to low is $\delta = 0.2$ s. Note that the uncertainty can be low for a system, but it cannot be eliminated for any real measurement system. Just assuming the left timestamp (4.8s in this case) or the right timestamp (5 s in this case) – both approaches are not right and can lead to the wrong evaluation of the safety condition.

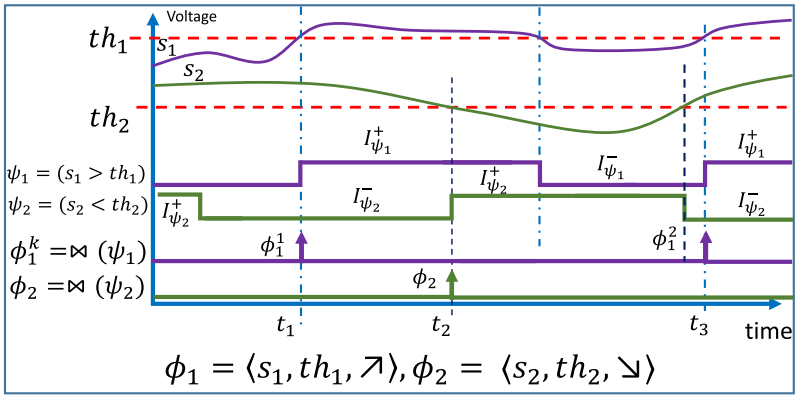


Fig. 2. The process to events from analog signals by threshold crossing. Each signal is compared with a threshold and the satisfaction is shown by a Boolean signal.

The only way to avoid wrong evaluations is to add conservativeness to the condition calculations. This can be done, if we know an upper-bound on δ. However, specifying the error of the measurement devices does not make sense in the temporal constraints of an application. δ is a property of a measurement system. From the application perspective, it makes more sense to define a **tolerance** or ε – with which the timing constraints must be met. Then a measurement device can be deemed to be capable or not of being able to evaluate the satisfaction of a timing constraint (with a given tolerance), if the measurement uncertainty is within the tolerance of the timing constraint.

Timestamp Temporal Logic (TTL) provides a definitional extension of STL that more intuitively expresses the timing specifications of distributed CPS and allows for a more natural expression of timing tolerance. It provides room for considering tolerance as a part of language and enables accurate monitoring.

4.1. Event representation in TTL

Simple events in TTL are defined as signal transitions from one state to another. In order to extract events for a real-time system, we utilize the classical discretization/quantization process. In such a process, there are three major parts (i) sampling, discretizing in time using regular sampling with an interval δ_{ADC}, (ii) quantization, uniform scalar quantization with a step q, and (iii) interpolation, detecting the threshold crossing to extract events [43]. Since system Ω is hybrid, we assume that the monitored signal is band-limited and does not produce infinite events occurring in a finite time (Zeno behavior [44]). Moreover, the conversion is done using uniform sampling in Analog to Digital Converters (ADC). Considering such signals and the quantization process, in the interpolation, we are able to calculate the maximum error in space (d) and time (δ_{ADC}) [45]. Section 6.4 discuss the way to accumulate the total uncertainty by considering ε_{ADC}.

For example, a signal event generated by threshold crossing on analog signals is presented by a triplet, $\langle s, th, \nearrow \text{ or } \searrow \rangle$, which is 1 ($\top$) at the time when the signal, s, crosses a threshold, th (crossing from below $\nearrow$ or from above $\searrow$), and 0 ($\bot$) everywhere else. The time of the event is called a timestamp and is represented by a real number. A signal event can be a singleton or repetitive. In a singleton signal event, there is only one event (ϕ) which is represented by a single timestamp while repetitive signal events are expressed by a sequence of timestamps $\{\phi^1, \phi^2, \ldots, \phi^n\}_{(n\in\mathbb{N})}$.

Finding out the timestamps of simple events, like $\langle s, th, \nearrow \text{ or } \searrow \rangle$ is a three-step process. Fig. 2 outlines the process. The top 2 curves in the graph depict comparing two signals (s_1 and s_2) with their corresponding thresholds (th_1 and th_2), and that results in the middle 2 Boolean signals (ψ_1 and ψ_2). The Boolean signals (ψ_1 and ψ_2) can be divided into time intervals during which the value of the signal is true or false, indicated by I^+ and I^-. The time of the rising event on the Boolean signals (ψ_1 and ψ_2) then represents the occurrence of the specified event, and can be obtained by differentiator. In fact, the Boolean signal ψ_1, gives the last 2 curves in the figure. Thus, $\hat{\phi} = \bowtie(\psi)$, where the value of $\phi \in \hat{\phi}$ is $\top$ when $\psi(t^+) \oplus \psi(t) \wedge \neg\psi(t) = \top$, and $\bot$ otherwise. $\oplus$ is XOR and t^+ refers to the right neighborhood of signal at time t in continuous domain. Then, using the function $\tau(\phi)$[8] we can extract the event's timestamp.

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 the discrete time-domain. As it is depicted in Fig. 2, signals s_1 and s_2 are firstly converted into Boolean signals after comparing with their corresponding thresholds (th_1, th_2), and then by applying the differentiate operator, $\bowtie$ sequence of timestamps ϕ_1 and ϕ_2 are generated.

Table 1
Grammar of TTL.

$\psi :=$	υ
	$\mid \neg\psi$
	$\mid \psi_1 \wedge \psi_2$
	$\mid \Box\psi$
	$\mid \psi_1 \mathcal{U} \psi_2$
	$\mid \mathcal{L}(\phi_1, \phi_2, \varepsilon)\nabla l$
	$\mid \mathcal{C}(\phi_1, \phi_2, \ldots \phi_n, \varepsilon)$
	$\mid \mathcal{S}(\phi_1, \phi_2, \ldots, \phi_n, \varepsilon)$
	$\mid \mathcal{F}(\phi, \varepsilon)\nabla f$
	$\mid \mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p)\nabla p$

4.2. Expressing tolerance in TTL syntax

The TTL syntax is defined based on STL with extensions to enable expressing events, natural specification, and considering tolerance. TTL operators are built based on high-level operators that specify timing requirements on both the value of a formula and the occurrence time of events. The output of TTL operators is finally a Boolean value.

Definition 4.1. The satisfaction relation $(s, t) \vDash \psi$, indicating that signal s satisfies ψ starting from position t.

Definition 4.2. Given the sets χ of events and the set $\mathbf{V}$ of atomic propositions, the set $TTL_\chi(V)$ of TTL formulas (event-based) is inductively defined using the following grammar shown in Table 1: where $\upsilon \in \mathbf{V}$, $\phi_1, \phi_2, \ldots, \phi_n \in \chi, \nabla \in \{\langle, \rangle, =\}$ and $l, f, p, \varepsilon, \varepsilon_f, \varepsilon_p \in \mathbb{R}^+ + \{0\}$.

In the rest of this section, we describe the meaning of each operator existing in the grammar.

[8] We will explain it in Section 5

- The atomic υ proposition is true if there is a time at which a threshold crossing exists.
- $\neg\psi$ is true iff $\psi = \bot$.
- The output of each TTL operator is a Boolean continuous signal. Hence the STL operators can be applied to them.[9] Therefore, $\square$ and $\mathcal{U}$ receive Boolean signals and return Boolean as well.
- If the evaluated two TTL formulas, ψ_1 and ψ_2 are both true, $\psi_1 \wedge \psi_2 = \top$.
- $\mathcal{L}(\phi_1, \phi_2, \varepsilon)$ calculates the latency between two events ϕ_1 and ϕ_2. (i) ϕ_1 and ϕ_2 are singleton,[10] (ii) ϕ_1 occurs before ϕ_2, and (iii) the difference between the actual occurrence of two events, ϕ_1 and ϕ_2 should be less/greater than or equal to l with the user-defined tolerance value uncertainty of ε.
- $\mathcal{C}(\phi_1, \phi_2, \ldots, \phi_n, \varepsilon)$ specifies that the event ϕ_i occurs before event ϕ_{i+1} $(1 \leq i \leq n)$, with a tolerance of ε. ε determines the acceptable minimum distance between the events.
- $\mathcal{S}(\phi_1, \phi_2, \ldots, \phi_n, \varepsilon)$ specifies that the events ϕ_1 to ϕ_n occur at the same time within a time duration of ε.
- $\mathcal{F}(\phi, \varepsilon_f)$ specifies that the occurrence frequency of event ϕ[11] should be less/greater than or equal to f within an tolerance value of ε_f (in Hz).
- $\mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p)$ specifies that the phase difference between two repeating events ϕ_1, and ϕ_2 on two different signals is less/greater than or equal to p, where, ε_f is tolerance in the frequency domain and ε_p is the tolerance in phase.

5. Conditions for guaranteed verification by monitoring

In this section, we explain the mathematical equations and proof to have the right expressions with considering Tolerance and Uncertainty in this section. The equations for Latency operator are explained here. At the end of this section, we demonstrate how TTL fixes the problem in monitoring by using ε.

However, before starting to explain the details of the language, there are some parameters that should be defined.

5.1. Problem definition

In this work, we take a very general approach to monitor a traced continuous signal u of the system Ω, with considering a temporal specification, ψ. t is the dense time, and $u(t)$ represents the signal value at time t in voltage.

5.1.1. The type of signals and discretization process

Let $\mathbb{D}$ be a value domain, Boolean ($\mathbb{B}$), or Real-value ($\mathbb{R}$) signal.

Continuous-Time Signals: A continuous signal s maps the continuous time domain (dense time) to a real-valued domain. Since CPS monitoring is usually based upon finite traces [46], the signal length is r expressed by $|s| = r$.

Definition 5.1. Signal s is a map. $s : \mathbb{T} \to \mathbb{R}$, $\mathbb{T}$ is a set of non-negative real numbers as time, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}$ is the value of continuous signals.

The interval for the entire signal is $I = [0, r)$. Its value at time t where $t \in \mathbb{R}_{\geq 0}$ is $s(t)$.

A sequence of disjoint non-empty intervals $I : \{I_0, I_1, I_2, \ldots, I_k\}, I_i \cap I_j = \varnothing$ is a time partition compatible with a finitely-varying continuous-time Boolean signal converted into discrete using Γ function, $\Gamma : \mathbb{R} \to \{\bot, \top\}$ that makes a discrete signal using a threshold value, th:

$$\Gamma_s = \begin{cases} \top, & \text{if } s \geq th \\ \bot, & \text{otherwise} \end{cases} \tag{1}$$

If x is a continuous Boolean signal, and if $\bigcup_{j=0}^{k} I_j = [0, r)$ and $\forall I_j$ in the form of (t_j, t_{j+1}), $[t_j, t_{j+1})$, $(t_j, t_{j+1}]$, or $[t_j, t_{j+1}]$ such that $t \leq t_{j+1}$, for all t and t', $x(t) = x(t')$.

Discrete-Time Signals: A discrete-time signal σ is a sequence of samples of a continuous-time Boolean signal x and made by the function $g : \mathbb{R} \to \mathbb{B}$. The value of σ at position i for continuous signal x is $\sigma_x[i]$ and equals a Boolean value where $i \in \mathbb{N}$.

There are different discretization methods and we discuss periodic or uniform sampling here, which is the type of sampling used most often in practice. It can be expressed by $\Gamma_s(\theta) = \sigma_s(\theta.\delta_{ADC})$ s.t. $\theta \in \mathbb{N}$. σ_s is the discrete time signal of continuous signal s. We name θ as *Timestamp*. Moreover, the time interval δ_{ADC} between successive samples is called the sampling period or sample interval and it is $\frac{1}{f_s}$. f_s is the sampling rate (samples per second) or the sampling frequency (hertz). As we will explain in Section 6.4, it is a part of total uncertainty which is δ.

Therefore, we have function $\Gamma : s(t) \xrightarrow{\delta_{ADC}, th} \sigma$.

Moreover, as a requirement to provide the one-way guarantee for run-time verification, we assume that the value of a monitored signal changes at a *constant rate* between every two captured samples.

Definition 5.2. The temporal logic formula, ψ_s, is the system specification for signal $s(t)$ behavior.

In the monitoring process, there is a real signal (ground truth) continuous signal, $s(t)$, and a discrete signal, $\sigma_s[\theta]$. In monitoring, since the real value of the signal is not known (because of quantization/discretization) there is an error value. This error may cause uncertainty in the system implementations and also monitoring.

The informal definition of each part is explained below:

- s is a continuous signal over continuous time t.
- ψ is a Boolean signal over continuous time. It can be *true* (positive pulse) or *false* (negative pulse) for a duration of time.
- δ_{ADC} is the sampling time to convert a continuous signal to discrete. It can be taken as measurement error as well because it is the maximum error in the discretization process.
- $T(\phi)$ is a function that receives the event ϕ and returns its actual time of occurrence in the continuous domain. It is a real number.
- $\tau(\phi)$ receives the event ϕ and returns the event's timestamp in discrete domain. Its return value is an integer number.
- ε: is the user-defined tolerance value to cover the existing measurement errors in Cyber–Physical Systems. By this value, we make the temporal property more conservative since the measurement error can cause uncertainties. Its value in all operators is $\varepsilon > 0$ and a part of syntax because no measurement system is perfect.
- $\bigwedge\!\!\!\!\!\!\diagup$ is differentiator operator and converts a Boolean signal into events ($\hat{\varphi} = \bigwedge\!\!\!\!\!\!\diagup(\sigma)$).[12]

5.2. TTL operators

In this section we have formal definitions for each operator in TTL. The maximum latency constraint is expressed by the following statement in TTL:

Maximum Latency: $\mathcal{L}(\phi_1, \phi_2, \varepsilon) < l, 0 < \varepsilon < l$

If $T(\phi)$ represents the actual occurrence time of event ϕ in ***real numbers***, we have $0 < T(\phi_2) - T(\phi_1)$ and $T(\phi_2) - T(\phi_1) < l - \varepsilon$. $T(\phi_2) - T(\phi_1)$ is required to be less than $l - \varepsilon$ so as to guarantee that the latency between the two events ϕ_1, and ϕ_2 is less than l. Fig. 3.a shows how the latency constraint is calculated in the continuous time. Now, the question remains, what are the conditions under which we can guarantee that $T(\phi_2) - T(\phi_1) < l - \varepsilon$ is satisfied in a discrete system.

[9] As mentioned before, ψs are Boolean signals whereas ϕs are events

[10] They occur once in a specific duration.

[11] ϕ is repetitive here and ϕ^i corresponds to the ith occurrence of the event ϕ.

[12] $\hat{\phi}$ is a sequence of events.

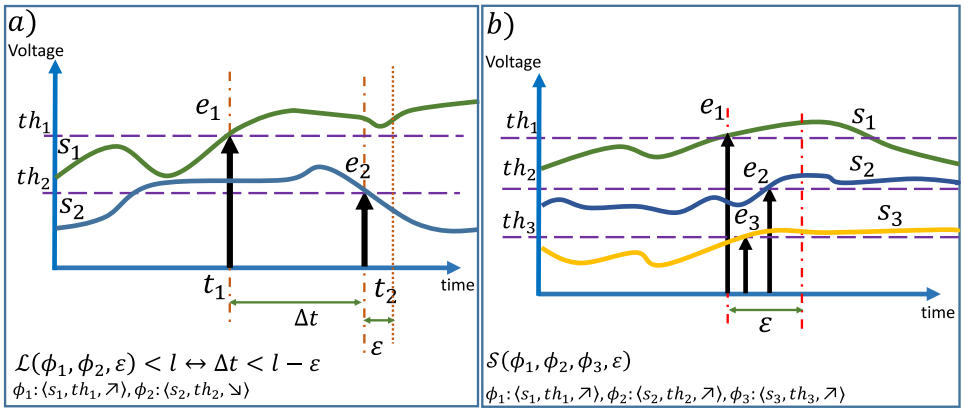


Fig. 3. The effect of tolerance in *Latency* and *Simultaneity* calculations. Figure (a) shows a Latency constraint and (b) is the demonstration for Simultaneity operator in TTL.

In formal, continuous signal s at time t should satisfy this relation: $(s,t) \models \mathcal{L}(\phi_1, \phi_2, \varepsilon)$ iff $T(\phi_2) - T(\phi_1) < l - \varepsilon$.

In order to answer this question, we should look at CPS again. Most CPS implementations and measurement systems sample signals (with a fixed sampling time of δ) and, therefore, capture a timestamp as the occurrence of the event. The actual time of an event is inferred from the timestamp within an error δ. The measurement error has several sources such as quantization, sampling time, analog to digital converter (ADC) resolution [47]. If $\tau(\phi)$ represents the ***integer*** timestamp at which the event ϕ is captured, and both $T(\phi)$ and $\tau(\phi)$ are initiated to zero when the system starts to operate, then $\tau(\phi) = \left\lceil \frac{T(\phi)}{\delta} \right\rceil$ is the relation between $T(\phi)$ and $\tau(\phi)$.

Therefore, we know that $T(\phi_1) = (\delta\tau(\phi_1) - \delta, \delta\tau(\phi_1))$ and similarly $T(\phi_2) = (\delta\tau(\phi_2) - \delta, \delta\tau(\phi_2))$. Hence, their subtraction to find the latency between two events ϕ_1 and ϕ_2 will be bounded between $\delta(\tau(\phi_2) - \tau(\phi_1)) - \delta$ and $\delta(\tau(\phi_2) - \tau(\phi_1)) + \delta$.[13] In fact:

$$\delta(\tau(\phi_2) - \tau(\phi_1)) - \delta < T(\phi_2) - T(\phi_1) < \delta(\tau(\phi_2) - \tau(\phi_1)) + \delta \tag{2}$$

Also, we already know that in order to be conservative, it is enough if we have:

$$T(\phi_2) - T(\phi_1) < l - \varepsilon \tag{3}$$

to guarantee the time difference between events is certainly less than l. Therefore, based on Eqs. (2) and (3):

$$\delta\tau(\phi_2) - \delta\tau(\phi_1) - \delta < l - \varepsilon \tag{4}$$

Since we know that $0 < \delta\tau(\phi_2) - \delta\tau(\phi_1)$, $0 < l - \varepsilon + \delta$. Hence, to have the guarantee for Eq. (3), we should test:

$$\tau(\phi_2) - \tau(\phi_1) < \frac{l - \varepsilon}{\delta} + 1 \tag{5}$$

and the condition is (by considering Eqs. (2) and (4)) is $\varepsilon - \delta < l$.

On the other hand, in inequality (4), if $\varepsilon < \delta$, since we are adding a positive number to l ($\delta\tau(\phi_2) - \delta\tau(\phi_1) < l - (\varepsilon - \delta)$), it does not guarantee (4). Therefore, we should have $\delta < \varepsilon$. As a fact, if inequality (5) is not true, inequality (3) might still be true. However, by considering $\varepsilon > \delta$ we make sure if (5) is true, (3) is definitely true as well. This relation between ε and δ makes sense because in Eq. (5), the added value (δ) due to the discretization process is compensated by ε. In the above equations, (5) is a property of **measurement** and $\delta < \varepsilon$ is the property of **measurement system**.

Therefore, we have:

$$(s,t) \models \mathcal{L}(\phi_1, \phi_2, \varepsilon) \tag{6}$$

Regarding inequity(5), if we consider $\delta < \varepsilon$ since the accepted error is greater than the actual error, we can ensure that if there is a violation in the Latency constraint, we can catch it. In fact, without considering ε, there is a gray area in which it is not clear to know whether the Latency constraint is violated or met.

Minimum Latency: $l < \mathcal{L}(\phi_1, \phi_2, \varepsilon)$

By considering Eq. (2), we have $l + \varepsilon - \delta < \delta(\tau(\phi_2) - \tau(\phi_1))$. If $\varepsilon < \delta$, we cannot guarantee that the time difference between two events is greater than l. Therefore, $\delta < \varepsilon$. Similar to the maximum latency specification, there will be an error of δ between the actual time and the captured timestamp of events. Hence, we have:

$$\frac{l + \varepsilon}{\delta} - 1 < \tau(\phi_2) - \tau(\phi_1), \delta < \varepsilon \tag{7}$$

Exact Latency: $\mathcal{L}(\phi_1, \phi_2, \varepsilon) = l$

The constraint means that $T(\phi_2) - T(\phi_1) = l \pm \varepsilon$ or $l - \varepsilon < T(\phi_2) - T(\phi_1) < l + \varepsilon$. A monitoring system can ensure this specification by checking if $\frac{l-\varepsilon}{\delta} + 1 \leq \tau(\phi_2) - \tau(\phi_1) \leq \frac{l+\varepsilon}{\delta} - 1$. By ε, we defined a specific duration to shrink the time for satisfaction regarding δ. Again, if $\varepsilon < \delta$ we cannot guarantee to catch all violations. Hence, the measurement system is considered to be able to evaluate the specification correctly if $\delta < \varepsilon$.

5.3. TTL-based run-time verification approach detects all timing violations

As an example to demonstrate the advantage of TTL over STL in monitoring approaches, let us take a look at Fig. 1 and its example again. In Section 3, we explained the issue in which the monitoring device cannot detect the violation. However, since TTL considers the measurement error, it can solve the false positive problem using ε. As a fact, TTL is able to express the width of a signal pulse by considering rising and falling edges. In Fig. 1, we can replace $\square_{[1,5]}\psi$ by minimum latency, $\mathcal{L}(\langle s_1(t), 2.5, \nearrow \rangle, \langle s_1(t), 2.5, \searrow \rangle) > 4s$. From the example (in Section 3), we know $\delta = 0.2$ and can take $\varepsilon = 0.4$. Based on the numbers on Fig. 1 and δ and ε, we have $\tau(\langle s_1(t), 2.5, \nearrow \rangle) = \left\lceil \frac{2}{0.2} \right\rceil = 10$ and $\tau(\langle s_1(t), 2.5, \searrow \rangle) = \left\lceil \frac{6}{0.2} \right\rceil = 30$.

$$\mathcal{L}(\langle s_1(t), 2.5, \nearrow \rangle, \langle s_1(t), 2.5, \searrow \rangle, 0.4) > 4s$$
$$\iff \frac{l + \varepsilon}{\delta} - 1 < \tau(\langle s_1(t), 2.5, \searrow \rangle) - \tau(\langle s_1(t), 2.5, \nearrow \rangle)$$
$$\left\lceil \frac{4 + 0.4}{0.2} \right\rceil - 1 < 30 - 10 \Longrightarrow 21 \not< 20$$

Accordingly, the monitoring system shows the timing constraint is violated when it is really violated. Therefore, while STL mistakenly shows the temporal requirement is met, TTL is able to correctly catch the violation.

This conversion, *Globally* to *Latency* can be generalized for the other two operators in STL. In fact, in order to utilize the capabilities of TTL, it is possible to have corresponding operators for *Eventually* and *Until* as well. This conversion can be done by considering the rising and falling edges on the Boolean signals.

[13] Since the subtraction is calculated, the maximum error is δ.

To summarize, we can express the STL (or MITL) operators in TTL as below. If we consider ϕ_r and ϕ_f as the rising and falling edges, $\Diamond_{[a,b]}\psi$, $\Box_{[a,b]}\psi$, and $\psi_1 \mathcal{U}_{[a,b]}\psi_2$ are defined as below:

$$\Diamond_{[a,b]}\psi \longrightarrow \mathcal{L}_{[a,b]}(\phi_r, \phi_f, \varepsilon) > 0, \varepsilon > \delta$$

$$\Box_{[a,b]}\psi \longrightarrow \mathcal{L}_{[a,b]}(\phi_r, \phi_f, \varepsilon) > b - a, \varepsilon > \delta$$

$$\mathcal{L}_{[a,b]}(\phi_{1_r}, \phi_{1_f}, \varepsilon) > 0 \wedge \mathcal{L}_{[a,b]}(\phi_{2_r}, \phi_{2_f}, \varepsilon) > 0 \wedge \mathcal{L}_{[a,b]}(\phi_{2_r}, \phi_{1_f}, \varepsilon) > 0, \varepsilon > \delta$$

where $(s,t) \vDash \mathcal{L}(\phi_1, \phi_2, \varepsilon) > l$ then $\frac{l+\varepsilon}{\delta} - 1 < \tau(\phi_2) - \tau(\phi_1)$ iff $\theta_1 = \tau(\phi_1), \theta_2 = \tau(\phi_2)$ s.t. $\theta_1, \theta_2 \in [t+a, t+b]$

5.4. TTL provides a one-way guarantee in CPS monitoring

Since the TTL operators can be used in monitoring systems to verify the operation of safety–critical applications, they must be able to catch all timing violations. On the other side, we know TTL specifies the timing constraints within a tolerance value (ε). Accordingly, some satisfactions might fall in the tolerable duration category which we refer to as the *gray area*. If a timing constraint is met within the gray area, our rule detects them as violated since it may be evaluated as met just because of uncertainty in the measurement system. Based on such rule, some detected timing violations will be false negative cases (when a violation is reported while it is met) but there will be no false positive. Therefore, from this point of view, TTL provides a one-way guarantee for run-time verification. The rate of false negative is directly dependant on the size of the gray area.

Providing the one-way guarantee mean that if the evaluation system says a timing constraint is met, we are certain that it is actually met given the specified tolerance by the designer and the existing uncertainty of the measurement devices. Having zero false positive is valuable for safety–critical systems. It is worth noting that providing a two-way guarantee is not possible due to the existence of uncertainty in the monitoring system. Our approach provides the one-way guarantee at the cost of being more conservative and having more false negatives.

5.5. TTL rules

In the rest of this section, we show the rules for TTL temporal operators to guarantee the monitoring accuracy (proofs for Latency are in Section 5) and summarize all in Table 2.

Maximum Latency: $\mathcal{L}(\phi_1, \phi_2, \varepsilon) < l, 0 < \varepsilon < l$ we already showed that if we have $\tau(\phi_2) - \tau(\phi_1) < \frac{l-\varepsilon}{\delta} + 1$, $\delta < \varepsilon$ the maximum latency is guaranteed.

Minimum Latency: $l < \mathcal{L}(\phi_1, \phi_2, \varepsilon)$ if we have $\frac{l+\varepsilon}{\delta} - 1 < \tau(\phi_2) - \tau(\phi_1), \delta < \varepsilon$, the minimum latency is guaranteed.

Exact Latency: $\mathcal{L}(\phi_1, \phi_2, \varepsilon) = l$ is guaranteed if we have $\frac{l-\varepsilon}{\delta} + 1 \leq \tau(\phi_2) - \tau(\phi_1) \leq \frac{l+\varepsilon}{\delta} - 1$, $\delta < \varepsilon$.

Chronological: $\mathcal{C}(\phi_1, \phi_2, \ldots, \phi_n, \varepsilon)$ means that $\varepsilon < T(\phi_{i+1}) - T(\phi_i)$. A measurement system with accuracy of δ will be able to ensure the specification by monitoring $\frac{\varepsilon}{\delta} - 1 < \tau(\phi_{i+1}) - \tau(\phi_i)$ only if $\delta < \varepsilon$.

Simultaneity: $\mathcal{S}(\phi_1, \phi_2, \ldots, \phi_n, \varepsilon)$ means that the time difference between each pair of events is less than ε, or by the other words

$$max\big(T(\phi_1), T(\phi_2), \ldots, T(\phi_n)\big) - min\big(T(\phi_1), T(\phi_2), \ldots, T(\phi_n)\big) < \varepsilon$$

A measurement system with accuracy of δ will be able to ensure the specification by monitoring

$$max\Big(\tau(\phi_1), \tau(\phi_2), \ldots, \tau(\phi_n)\Big) - min\Big(\tau(\phi_1), \tau(\phi_2), \ldots, ts(\phi_2)\Big) < \frac{\varepsilon}{\delta} + 1$$

The added δ value (everything is normalized by δ) is just to consider the measurement error value compensated by ε. Additionally, as before, the measurement is valid only if $\delta < \varepsilon$. Fig. 3.b demonstrate the *Simultaneity* calculation.

Minimum Frequency: $f < \mathcal{F}(\phi, \varepsilon_f)$

This temporal specification defines the minimum frequency for a repetitive event on a signal and can be converted into the time domain. In fact in the time domain, it means $T(\phi^n) - T(\phi^{n-1}) < \frac{1}{f \pm \varepsilon_f}$ where ϕ^i corresponds to the ith occurrence of the event ϕ on the same signal. To simplify, we have: $T(\phi^n) - T(\phi^{n-1}) < \frac{1}{f+\varepsilon_f}$. Similar to latency, $\tau(\phi^n) - \tau(\phi^{n-1}) < \frac{1}{\delta(f+\varepsilon_f)} + 1$, $\delta < \frac{1}{\varepsilon_f}$.

Maximum Frequency: $\mathcal{F}(\phi, \varepsilon_f) < f$

In time domain, this operator defines the minimum period for a repetitive event. Indeed, we should have $\frac{1}{f-\varepsilon_f} < T(\phi^n) - T(\phi^{n-1})$.

Therefore, $\frac{1}{\delta(f-\varepsilon_f)} - 1 < \tau(\phi^n) - \tau(\phi^{n-1})$ where $\delta < \frac{1}{\varepsilon_f}$, $\varepsilon_f < f$.

Exact Frequency: $\mathcal{F}(\phi, \varepsilon_f) = f$

Exact frequency means $T(\phi^n) - T(\phi^{n-1}) = \frac{1}{f \pm \varepsilon_f}$ in time domain. By simplifying the equations, we have $\frac{1}{f+\varepsilon_f} < T(\phi^n) - T(\phi^{n-1}) < \frac{1}{f-\varepsilon_f}$, $\varepsilon_f < f$.

Considering the measurement error of δ, the system must monitor $\frac{1}{\delta(f+\varepsilon_f)} + 1 < \tau(\phi^n) - \tau(\phi^{n-1}) < \frac{1}{\delta(f-\varepsilon_f)} - 1$, and the monitoring is valid only if $\delta < \frac{1}{\varepsilon_f}$, $\varepsilon_f < f$.

Minimum Phase: $p < \mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p)$

If events ϕ_1 and ϕ_2 occur at the **same frequency** (exact frequency) then *Phase* can be defined.[14] This constrain defines the desired latency between consequent events on two different event sources (ϕ_1 and ϕ_2).

Hence, based on this concern, we must satisfy two conditions: (i) $\mathcal{F}(\phi_1, \varepsilon_f) = \mathcal{F}(\phi_2, \varepsilon_f)$, and (ii) $p - \varepsilon_p < T(\phi_2^n) - T(\phi_1^n)$.

From condition (i), we have $|(T(\phi_1^n) - T(\phi_1^{n-1})) - (T(\phi_2^n) - T(\phi_2^{n-1}))| < \frac{1}{\varepsilon_f}$. If we assume $A : T(\phi_1^n) - T(\phi_1^{n-1})$ and $B : T(\phi_2^n) - T(\phi_2^{n-1})$, we have: $-\frac{1}{\varepsilon_f} < A - B < \frac{1}{\varepsilon_f}$. Hence, there are two cases, (a) $A < \frac{1}{\varepsilon_f} + B$, and (b) $B - \frac{1}{\varepsilon_f} < A$.

From Eq. (2), we know that

$$\delta(\tau(\phi_1^n) - \tau(\phi_1^{n-1})) - \delta < A < \delta(\tau(\phi_1^n) - \tau(\phi_1^{n-1})) + \delta$$

$$\delta(\tau(\phi_2^n) - \tau(\phi_2^{n-1})) - \delta < B < \delta(\tau(\phi_2^n) - \tau(\phi_2^{n-1})) + \delta$$

To be conservative, in (a), A should be in its maximum value and B should be in its Minimum value. Therefore, $\delta(\tau(\phi_1^n) - \tau(\phi_1^{n-1})) + \delta < \frac{1}{\varepsilon_f} + \delta(\tau(\phi_2^n) - \tau(\phi_2^{n-1})) - \delta$, and in (b), $\delta(\tau(\phi_2^n) - \tau(\phi_2^{n-1})) + \delta < \frac{1}{\varepsilon_f} + \delta(\tau(\phi_1^n) - \tau(\phi_1^{n-1})) - \delta$.

Thus, the pre-conditions for satisfaction of *Phase* constraint are:

$$\tau(\phi_1^n) - \tau(\phi_1^{n-1}) - (\tau(\phi_2^n) - \tau(\phi_2^{n-1})) < \frac{1}{\delta\varepsilon_f} - 2 \tag{8}$$

$$\tau(\phi_2^n) - \tau(\phi_2^{n-1}) - (\tau(\phi_1^n) - \tau(\phi_1^{n-1})) < \frac{1}{\delta\varepsilon_f} - 2 \tag{9}$$

From condition (ii), the specification implies that $\frac{p+\varepsilon_p}{\delta} - 1 < \tau(\phi_2^n) - \tau(\phi_1^n)$. Since the monitoring system must ensure that $p + \varepsilon_p < T(\phi_2^n) - T(\phi_1^n)$ is monitored correctly, the monitoring system will be implemented as $p' + \varepsilon_p' - 1 < \tau(\phi_2^n) - \tau(\phi_1^n)$ where $p' = \frac{p}{\delta}$ and $\varepsilon_p' = \frac{\varepsilon_p}{\delta}$. To check if the measurement system can evaluate the timing specification, $\delta < \frac{1}{\varepsilon_f}$ and $\delta < \varepsilon_p$ statements should hold.

Maximum Phase: $\mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p) < p$

For this specification, the two frequencies of ϕ_1 and ϕ_2 should be equal within a tolerance. $T(\phi_2^n) - T(\phi_1^n) < p - \varepsilon$ where ϕ_1^i and ϕ_2^i correspond to the ith occurrence of the events ϕ_1 and ϕ_2 respectively. Similar to the minimum phase, the monitoring system should monitor $\tau(\phi_2^n) - \tau(\phi_1^n) < p' - \varepsilon_p' + 1$.

Exact Phase: $\mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p) = p$

This TTL operator means $T(\phi_2^n) - T(\phi_1^n) = p \pm \varepsilon_p$ or $p - \varepsilon_p < T(\phi_2^n) - T(\phi_1^n) < p + \varepsilon_p$. Considering the error in the measurement system, we have $p' - \varepsilon_p' + 1 < \tau(\phi_2) - \tau(\phi_1) < p' + \varepsilon_p' - 1$.

[14] Otherwise, having Phase constraint is meaningless.

Table 2
The conditions that must be met in the monitoring system to guarantee the meeting of TTL constraints.

TTL Temporal Operators	Monitoring condition when the constraint is met
$v \vDash \psi$	iff $v = \top \quad \forall t \geq 0 \quad (s,t) \vDash \psi$
$\mathcal{L}(\phi_1, \phi_2, \varepsilon) < l$	$\tau(\phi_2) - \tau(\phi_1) < \frac{l-\varepsilon}{\delta} + 1,\ \delta < \varepsilon,\ 0 < \varepsilon < l$
$l < \mathcal{L}(\phi_1, \phi_2, \varepsilon)$	$\frac{l+\varepsilon}{\delta} - 1 < \tau(\phi_2) - \tau(\phi_1), \delta < \varepsilon,\ 0 < \varepsilon < l$
$\mathcal{L}(\phi_1, \phi_2, \varepsilon) = l$	$\frac{l-\varepsilon}{\delta} + 1 < \tau(\phi_2) - \tau(\phi_1) < \frac{l+\varepsilon}{\delta} - 1,\ \delta < \varepsilon,\ 0 < \varepsilon < l,$
$\mathcal{C}(\phi_1, \phi_2, \ldots, \phi_n, \varepsilon)$	$\frac{\varepsilon}{\delta} - 1 < \tau(\phi_{i+1}) - \tau(\phi_i),\ \delta < \varepsilon$
$\mathcal{S}(\phi_1, \phi_2, \ldots, \phi_n, \varepsilon)$	$max\Big(\tau(\phi_1), \tau(\phi_2), \ldots, \tau(\phi_n)\Big) - min\Big(\tau(\phi_1), \tau(\phi_2), \ldots, ts(\phi_2)\Big) < \frac{\varepsilon}{\delta} + 1,\ \delta < \varepsilon$
$f < \mathcal{F}(\phi, \varepsilon_f)$	$\tau(\phi^n) - \tau(\phi^{n-1}) < \frac{1}{\delta(f+\varepsilon_f)} + 1,\ \delta < \frac{1}{\varepsilon_f},\ \varepsilon_f < f$
$\mathcal{F}(\phi, \varepsilon_f) < f$	$\frac{1}{\delta(f-\varepsilon_f)} - 1 < \tau(\phi^n) - \tau(\phi^{n-1}),\ \delta < \frac{1}{\varepsilon_f},\ \varepsilon_f < f$
$\mathcal{F}(\phi, \varepsilon_f) = f$	$\frac{1}{\delta(f+\varepsilon_f)} + 1 < \tau(\phi^n) - \tau(\phi^{n-1}) < \frac{1}{\delta(f-\varepsilon_f)} - 1,\ \delta < \frac{1}{\varepsilon_f},\ \varepsilon_f < f$
$p < \mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p)$	$p' + \varepsilon'_p - 1 < \tau(\phi_2^n) - \tau(\phi_1^n),\ \delta < \frac{1}{\varepsilon_f}$ and $\delta < \varepsilon_p,\ \varepsilon'_p = \frac{\varepsilon_p}{\delta},\ p' = \frac{p}{\delta}$
$\mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p) < p$	$\tau(\phi_2^n) - \tau(\phi_1^n) < p' - \varepsilon'_p + 1,\ \delta < \frac{1}{\varepsilon_f}$ and $\delta < \varepsilon_p,\ \varepsilon'_p = \frac{\varepsilon_p}{\delta},\ p' = \frac{p}{\delta}$
$\mathcal{P}(\phi_1, \phi_2, \varepsilon_f, \varepsilon_p) = p$	$p' - \varepsilon'_p + 1 < \tau(\phi_2) - \tau(\phi_1) < p' + \varepsilon'_p - 1,\ \delta < \frac{1}{\varepsilon_f}$ and $\delta < \varepsilon_p,\ \varepsilon'_p = \frac{\varepsilon_p}{\delta},\ p' = \frac{p}{\delta}$

Note that the events in *Frequency* and *Phase* formulas are necessarily periodic, whereas in the other timing specifications, they should be singleton.

All aforementioned operators and their meanings are summarized in Table 2.

6. How to know a monitoring equipment is good enough to verify a specific CPS

This section is an effort towards standardizing the process of testing the timing properties of CPS where a design of a testbed is outlined. The testbed can be used to test the CPS to check if all the timing constraints are being met or not in a systematic and correct manner to enable correct-by-construction (CbC) synthesis of the testbed. The testbed – like the distributed CPS it is trying to test – is also a distributed CPS, with each node (of the testbed) monitoring the required signals from the CPS node. Hardware timestamping and IEEE 1588 Precision Time Protocol (PTP) synchronization of the clocks among the CPS components provides observations at the same timescale through vast geographies, without losing accuracy with time. In this section, there is also a discussion about the key timing parameters of the testbed that will affect the time testing capability. In this regard, it studies the specifications that must be met by the testbed, such that the testbed can validate the timing constraints.

The most important design parameters of the distributed testbed that affect the errors in the timing measurements are described below.

6.1. Analog to digital converter (ADC) parameters

The testbed monitors all signals by sampling them because they should be digitalized to use on the cyber side. This is done by Analog to Digital Converters (ADCs) on the probes. The sampling rate of an ADC, f_s, is expressed as samples per second, or Hertz (Hz). In order to be able to monitor a signal correctly, the sampling rate must be sufficiently high to capture the fastest observable dynamics of interest in the signal. Suppose we intend to find out the time at which a signal rises above 3.4 V. Fig. 4 shows this signal, monitored with two different sampling rates. On the left with sampling rate of $f_s = 1$ kHz, the threshold crossing time of the signal is detected as $t = 1$ ms. However, on the right, with sampling rate of $f_s = 0.5$ kHz, the threshold crossing time of the signal is detected as $t = 2$ ms.

Since an ADC converts the voltage signal into digitized sampled events, the accuracy of measurement is also limited by the number of bits used to express the sampled value, $nbits_{ADC}$, and the voltage range of the ADC, VR_{ADC}. An n-bit ADC can represent 2^n values. A 12-bit ADC that measures the range of 0 V to 5 V has steps of ≈ 1 mV. The precision of the ADC is defined in terms of resolution of the ADC, or V_{ADC} can be calculated as: $V_{ADC} = \frac{VR_{ADC}}{2^{nbits}{}_{ADC}}$. The resolution of the ADC can affect the time at which the monitoring device detects an event on a signal. Fig. 5 illustrates the conversion of an analog signal to digital samples with various resolutions, VR_{ADC} of 5 V. If the user wants to detect the time when a signal rises above 4 V, then in the left diagram, with $nbits_{ADC} = 12$, the time at which the threshold crossing is detected is $t = 3$ ms, while in the right diagram, with $nbits_{ADC} = 11$, the time at which the threshold crossing is detected is $t = 4$ ms.

6.2. Input impedance

Wiring a signal to a DAQ device adds a load to the CPS circuit under test, which causes a change in the shape of the monitored signal. For pure resistive loads, this change is a simple voltage drop while for general loads, the shape of the monitored signal is changed based on the equivalent resistance and reactance of the measuring device (including capacitance effect of the cables) and the SUT. As a result, based on the rate of change in the value of the signal, the measurements of the signal may be delayed or its amplitude may be attenuated. Z_{in} is defined as the CPS equivalent circuit from the terminal connected to the testbed. The test and measurement device must have a sufficiently high input impedance to minimize perturbation of the measurement process on the signal.

Fig. 6.b shows the monitored signal perturbed by the loading effect of wiring the measurement device to the SUT. The threshold detection time of the original signal is before the threshold detection time of the monitored signal.

6.3. Clock fractional frequency offset

A clock's fractional frequency offset is defined as $f_{clock} = \frac{f_{inst} - f_0}{f_0}$, where f_{inst} is the instantaneous clock frequency, and f_0 is the nominal clock frequency. Thus, this is the unitless instantaneous fractional offset from the nominal frequency of an oscillator [48]. Environmental conditions such as voltage and temperature variations or mechanical vibrations, can affect the rate at which an oscillator runs. Typically, the fractional frequency offset of a clock, f_{clock}, is expressed in Parts Per Million (PPM), indicating the maximum amount of error in one million time units. Thus, the uncertainty after an elapsed time $t_{elapsed}$ due to a fractional frequency offset of f_{clock} is $t_{elapsed} \times f_{clock}$. For instance, a clock with 5 PPM error, has 5 μs error after 1 s, an uncertainty of about 0.5 s after a day, or about 2.5 min after a year. 7 depicts this issue.

Since all clocks deviate from each other, distributed clocks must be synchronized to a reference to have an agreement on time and have a unique and time notion. Synchronization protocols match the clock of a device to a reference clock. However, no synchronization protocol is perfect, and there is a synchronization uncertainty t_{sync}, that depends on several factors, including the number of bits used to represent the time, when the time stamping is done (e.g., in the hardware or in software), network jitter, network asymmetries delays, etc. [49]. The Network Time Protocol or NTP [50] can usually keep time synchronized to within tens of milliseconds over the public Internet ($t_{sync} \approx$

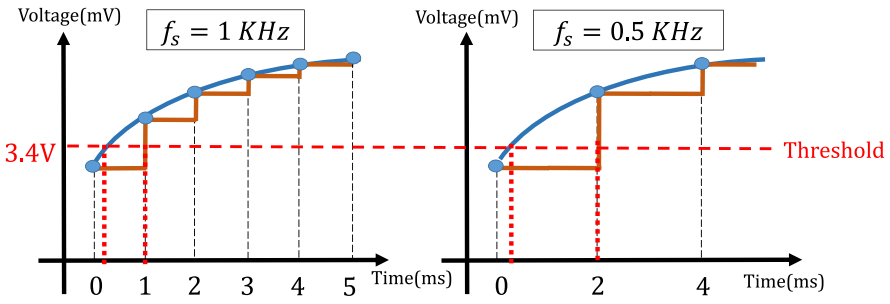


Fig. 4. A digitized analog signal at two different sampling rates. Given a threshold of 3.4 V, the threshold crossing time is detected at different times depending on the sampling rate.

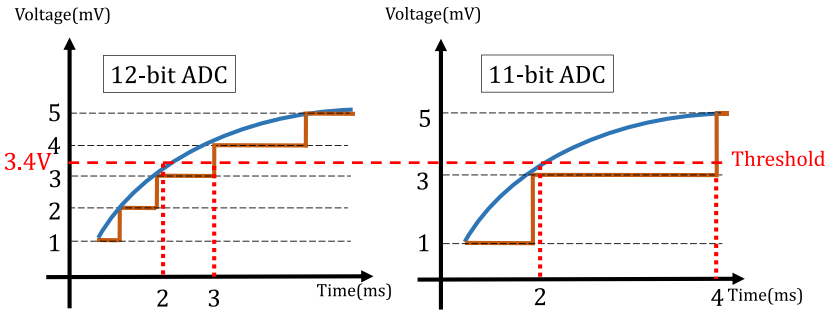


Fig. 5. An analog signal sampled using two ADCs that have the same range (0 V to 5 V) and different resolutions. (a) a 12-bit ADC is used. (b) a 11-bit ADC is used. The threshold crossing time in the left figure is at $t = 3$ ms while it is different in the right figure for the same signal (the threshold crossing is at $t = 4$ ms).

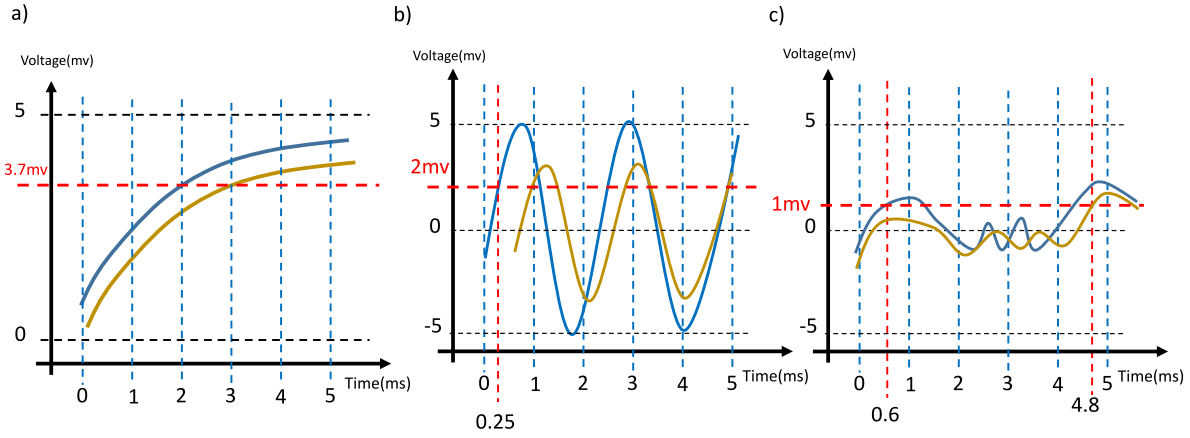


Fig. 6. (a) Voltage drop on a DC signal connected to a resistive load. (b) Voltage drop and shift on an AC signal connected to a load that has both reactive and resistive components. (c) Change in the shape of an arbitrary signal due to the loading effect.

10 ms). The Precision Time Protocol, PTP [51], can provide time synchronization over a LAN with sub-microsecond accuracy. PTP with the White Rabbit [52] extension used for the CERN Large Hadron Collider, can synchronize to sub-nanosecond accuracy. For CPS distributed over a wide area with high precision and accuracy needs, GNSS (Global Navigation Satellite Systems) can provide 100 ns accuracy.

Another important parameter is the rate of synchronization, r_{sync}, which is the number of times per second (e.g., in units of Hz) that synchronization is performed. Every time we perform synchronization, the time offsets are within t_{sync} of each other. But from thereon, until the next synchronization, the clock times will move apart at the rate of f_{clock}, if the local clock uses the protocol to adjust its time but not its frequency. The worst-case clock offset, ϵ_{wcco}, while the system clock is synchronized via the time synchronization protocol in steady-state and while all other environmental conditions are stable, can be calculated as: $\epsilon_{wcco} = t_{sync} + \frac{f_{clock}}{r_{sync}}$. Note that the units are in time, since f_{clock} is unitless and the reciprocal of r_{sync} is in units of time (see Fig. 7).

6.4. Analysis to calculate the total uncertainty

In order to determine whether timing behavior is verifiable by a given testbed, it is important to understand the sources of timing measurement uncertainty, described as δ in the timing constraint specification.

Consider a distributed CPS, with an exact latency constraint $\mathcal{L}(e_1, e_2, \epsilon) = l$ for events e_1 and e_2, where e_1 occurs on signal s_1 and e_2 on s_2 respectively. These events are detected at different nodes of the CPS. The latency constraint states that, given t_1 as the occurrence of e_1, the time at which e_2 occurs should be equal to $t_1 + l \pm \epsilon$. The testbed must capture the time at which an event occurs. However, the measured time will be erroneous. This can due to the several factors, including the sampling frequency f_s, the ADC resolution V_{ADC}, and the clock error, δ_{wcco}.

Consider an event described by the tuple $\langle s_1, v_t, rising \rangle$, marking the threshold v_t crossing of signal s_1 on a rising edge. Since the ADC output

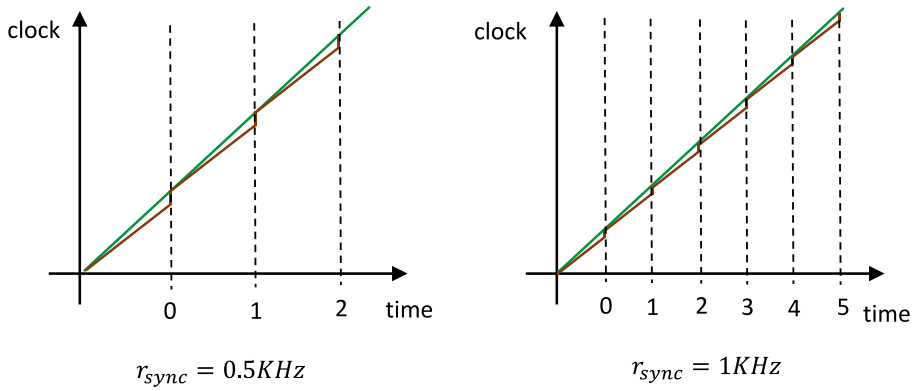


Fig. 7. The effect of synchronization frequency (r_{sync}) on the clock uncertainty..

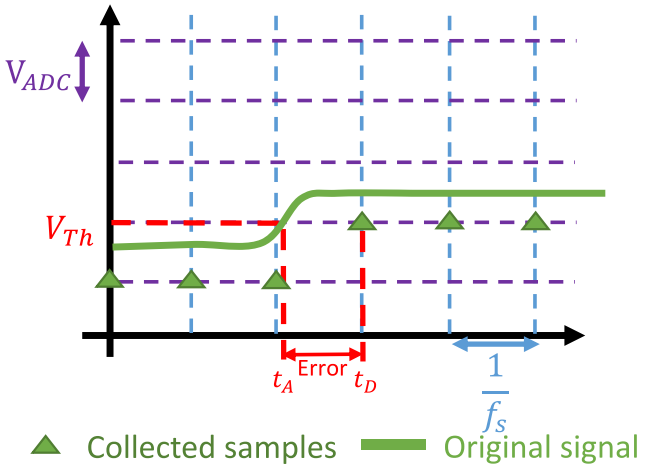


Fig. 8. Worst-case error between actual occurrence time and detection time for an ADC with sampling frequency f_s and fixed threshold detection based on an integer multiple of the ADC resolution.

is a multiple of the supported resolution, the testbed may not be able to detect the exact point of the threshold crossing. Thus, the threshold value must be mapped to the nearest upper bound of the value. Since all sampled data are collected at known points in time (integer multiples of $\frac{1}{f_s}$), a threshold crossing is detected with a maximum error $\delta_{ADC} = \frac{1}{f_s}$. Fig. 8 illustrates the worst-case error $\frac{1}{f_s}$ in an example.

Since all samples are timestamped using the local clock of the measurement system, clock synchronization error (δ_{wcco}) must be taken into account. Thus, the maximum time error between the actual event occurrence and the detected event occurrence is the sum of the ADC error and the clock synchronization error: $\delta_{total} \leq \delta_{wcco} + \delta_{ADC}$.

Since there will be at most δ_{total} error in both the measurements of e_1 and e_2, then the testbed can confidently verify whether the exact latency constraint is being met or not. Other types of constraints (e.g., simultaneity, frequency, phase, etc.) are also expressed with a temporal error tolerance and one can similarly reason and verify the temporal behavior.

7. Empirical evaluation

In order to demonstrate the usefulness of the proposed monitoring approach, three CPS applications have been used. Flying paster is a part of a printing press that swaps a full paper roll when the current paper roll is running out of paper. Flying paster is a distributed CPS with a variety important timing constraints that should be met. The other application is the breaker tripping in a power system when a fault occurs. For coordination of breakers in a Distributed Energy System (DES), they should meet a set of timing constraints. The last application is a quadcopter in which the time specifications for its motors have been monitored.

7.1. Flying paster application

A flying paster is part of a printing press, a distributed system enabling continuity of operation through the automatic exchange of an expiring paper roll with a new roll. Fig. 9.a shows a schematic of the flying paster with the active roll A, which feeds the web. When the radius of paper in roll A is less than a given threshold, the roll is replaced by the spare roll S. The radius of the paper around roll A, (r_A), is measured by the sensor H. When this radius falls below a given threshold, the *Approaching Out of Paper* (**AOP**) event (ϕ_{AOP}) is generated, which initiates the paper roll replacement process by starting the rotation of roll S. A strip of adhesive tape on the paper roll S is used to attach the paper from roll S to roll A. The location of the tape is detected by sensor F, which creates the γ event (ϕ_γ). The frequency of the γ event is used to calculate the angular velocity, ω_S, of S. Once the linear velocities of roll S and A are equal, a **Match** event (ϕ_{Match}) is generated. Then, sensor F generates the event *Top Dead Center* (TDC) to indicate the detection of the tape. Two complete rotations of S after event **TDC**, the idler wheel E pushes the paper from roll A towards roll S, at which point the paper from roll S adheres to the outgoing paper from roll A. This event, ($\phi_{Contact}$), occurs after roll S performed two rotations plus 255 degrees, $tapeToContactAngle = 225°$. Immediately after it, the Cutter D cuts the paper from A. This is called the **Cut** event (ϕ_{Cut}), and occurs when roll S has two rotations plus $tapeToCutAngle = 270°$ after TDC.

7.1.1. Flying paster implementation

A picture of the implementation of a scaled model of the flying paster is shown in Fig. 9.b. Rolls A and S in Fig. 9.a are implemented using two Hansen DC motors dialed (0–360 degree) disks, driven by two Arduino Mega2560 boards. Disks have a hole at *zero* degrees, which is detected by a photo-micro sensor. Photo-micro sensors implement the sensor H, and F and are installed next to the disks. The paper is modeled in software with the initial length of 125 m and 0.05 mm of thickness so that the initial diameter for both rolls is 9 cm (radius of 4.5 cm). The **AoP** event is generated when the radius of A becomes less than 2.5 cm.

7.1.2. Flying paster specification in STL

Based on the desired operation of the flying paster, its timing specifications are expressed in STL as follows. Noted that:

- A: Active roll.
- S: Spare roll.

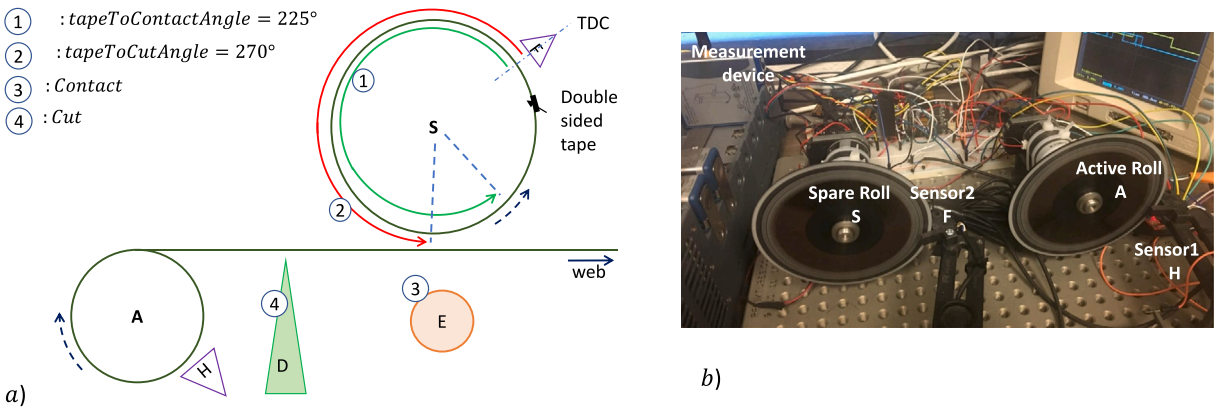


Fig. 9. The schematic of flying pater and its modeling implementation. (a) Flying Paster. Active roll A, is replaced by the spare roll, S to feed the web, (b) A scale model implemented by two DC motors, Arduino boards and two optical sensors.

- v: linear velocity.
- r: radius.
- ω: angular velocity.
- t_{action}: the duration from t_{AoP} to t_{Match}.
- $t_{termination}$: the duration between t_{AoP} to t_{Cut}.

(1) The velocity of the paper on active roll should be constant:

$$v_A = (r_A \times \omega_A) \text{ m/s}, \quad \square_{[t_i,t_s]}(v_A = r_A \times \omega_A),$$

(2) The time interval between AOP rising to $Match$ rising edge must be no more than t_{action}: $\square(\uparrow AOP \Rightarrow \Diamond_{[0,t_{action}]}(\uparrow Match))$

(3) After *match*, the paper speed of the spare should remain the same as active: $v_A = r_A \times \omega_A$ and $v_S = v_A$

$$\square_{[t_{Match},t_{Cut}]}(v_A = r_A \times \omega_A), \quad \square_{[t_{Match},t_{Cut}]}(v_S = r_S \times \omega_S),$$
$$\square_{[t_{Match},t_{Cut}]}(v_S = v_A)$$

(4) Catch the TDC (2 rotations of A after *Match*).

$$t_{TDC} - t_{Match} < \frac{4\pi}{\omega_S}, \quad \Diamond_{[t_{Match},t_{Match}+\frac{4\pi}{\omega_S}]}(\uparrow TDC),$$

(5) When tape is 225 degrees after TDC, *Contact* signal must fire.

$$t_{Contact} - (t_{TDC} + \frac{225\ degrees}{\omega_S}) < 1 \text{ ms},$$
$$\square_{[t_{TDC}+\frac{225\ degrees}{\omega_S}, t_{TDC}+\frac{225\ degrees}{\omega_S}+0.001]}(\uparrow Contact)$$

(6) When tape is 270 degrees after TDC, *Cut* signal must fire.

$$t_{cut} - (t_{spareTDC} + \frac{270\ degrees}{\omega_S}) < 1 \text{ ms},$$
$$\square_{[t_{spareTDC}+\frac{270\ degrees}{\omega_S}, t_{spareTDC}+\frac{290\ degrees}{\omega_S}+0.001]}(\uparrow cut)$$

(7) *AOP* to *Cut* should not be more than $t_{termination}$ (The user defines $t_{termination}$ and it is the maximum duration in which the roll changing should be done. In this scenario it is 6s).

$$\Diamond_{[t_{AOP},t_{AOP}+t_{termination}]}(\uparrow Cut)$$

7.1.3. Temporal specifications of flying paster in TTL

The timing specifications in TTL are:

- $\mathcal{F}(\phi_\gamma, 0.005 \times \frac{v_A}{2\pi r_S}) = \frac{v_A}{2\pi r_A}$: The linear velocity of the paper (v_A) (measured by sensor F) of the active roll should be constant at 20 m/s$\pm$ 10^{-3} m/s, otherwise, it cannot be fed to the printing press. It is known that $v_A = r_A \times \omega_A$ where r_A is the current paper radius and ω_A is the angular velocity of the roll A. The tolerable error for the velocity is 0.5 percent of the velocity on the active roll ($0.005 \times \frac{v_A}{2\pi r_S}$).
- $\mathcal{L}(\phi_{AOP}, \phi_{Match}, 10^{-3}\text{ s}) < 6$ s: The time interval from **AoP** to **Match** should be no more than 6 s with the maximum of 1 ms (10^{-3} s) of uncertainty. Otherwise, the paper on the old roll will run out before the new paper can be attached.
- $\mathcal{C}(\phi_{Match}, \phi_{TDC}, 10^{-4}\text{ s})$ and $\mathcal{L}(\phi_{Match}, \phi_{TDC}, 10^{-4}\text{ s}) < \frac{2\pi}{\omega_S}$: This captures the specification that **Match** happens before **TDC**, and that **TDC** happens before completing one full rotation from **Match**. Otherwise **Contact** and **Cut** do not work correctly.
- $\mathcal{C}(\phi_{Contact}, \phi_{Cut}, 10^{-4}\text{ s})$: *Cut* event must occur after *Contact* event, otherwise, the paper from the active roll is cut before attaching the new paper. The allowed tolerance to detect this chronology is 100 µs (10^{-4} s).
- $\mathcal{L}(\phi_{Contact}, \phi_{cut}, 10^{-4}\text{ s}) > 3$ ms: The delay between **Contact** and **Cut** should be less than 1.5 ms not to have a late cut. The acceptable uncertainty is 100 µs (10^{-4} s).
- $\mathcal{L}(\phi_{TDC}, \phi_{contact}, 10^{-4}\text{ s}) < \frac{4\pi}{\omega_S} + \frac{225}{\omega_S}$: This captures the specification that **Contact** must occur 2 rotations plus 225 degrees from the **TDC** event. Otherwise the two papers are not connected.
- $\mathcal{L}(\phi_{TDC}, \phi_{cut}, 10^{-4}\text{ s}) > \frac{4\pi}{\omega_S} + \frac{270}{\omega_S}$, when tape is in 2 rotations plus 270° of TDC, **Cut** must fire. Otherwise, old paper is not cut in time.

7.1.4. Testing the accuracy of the monitoring approach using STL and TTL

In order to verify the capabilities of TTL in run-time monitoring, we run online monitoring for flying paster in 7 different scenarios. The scenarios are listed in Table 3 where the active motor rotates in 10 m/s to 22 m/s as linear velocity. As the result, changing the velocity affects the other timing specifications. For example, t_{action} and $t_{termination}$ are increased when the linear velocity is decreased. In this experiment, we took three timing specifications for the application. The latency between (i) AoP and match, (ii) Match and TDC, and (iii) Contact and Cut. We used the profiles from FPGA implementation on Ni cRio 9067/9035.

To know the real values for signals we used the sampling rates of the DAQ devices which are two Ni-9402 with the accuracy of 55 ns. The sampling rate of 100 µs is used to discretize the values. We expressed the timing requirements of flaying paster in both STL and TTL, implement them using TMA, and run each scenario for 100 times. We collected data for violation and satisfaction of timing constraints. As Fig. 10 depicts, we divided the results into four categories:

- True Positive: When a timing requirements is **satisfied** and the monitoring system shows **it is met** as well.
- True Negative: When a timing requirements is **violated** and the monitoring system shows **it is not met** as well.
- False Positive: When a timing requirements is **violated** and the monitoring system shows **it is met**.
- False Negative: When a timing requirements is **satisfied** and the monitoring system shows it is not met.

Table 3
Seven different scenarios for flying paster applications.

	A	B	C	D	E	F	G
v_A	22 m/s	20 m/s	18 m/s	16 m/s	14 m/s	12 m/s	10 m/s
t_{action}	2 s	3 s	4 s	5 s	6 s	7 s	8 s
$t_{termination}$	3 s	4 s	5 s	6 s	7 s	8 s	9 s

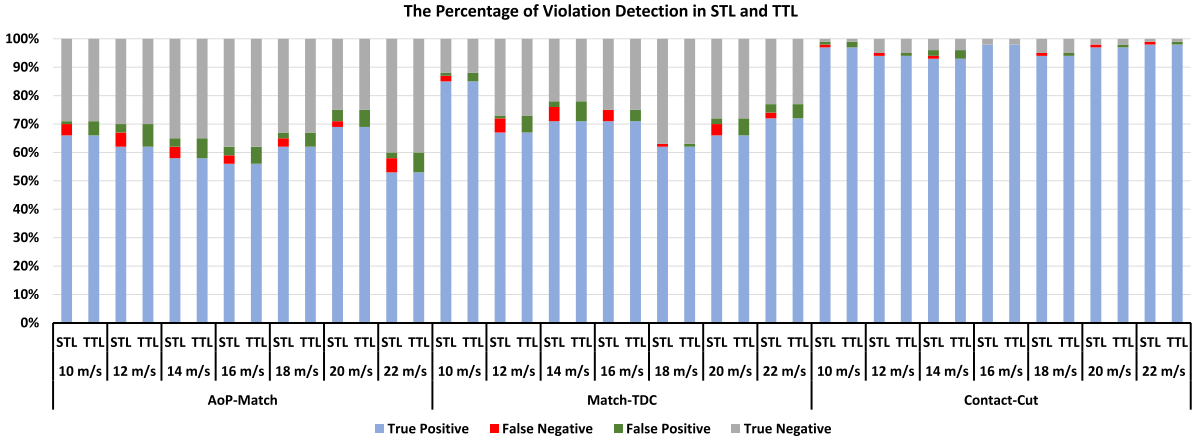


Fig. 10. Monitoring the time between AoP to Match, Match to TDC, and Contact to Cut events for 7 different velocity for the active roll. The timing constraints are expressed in STL and TTL and then monitored in order. The false positive is zero when we use TTL. However, since it is a one-side guarantee for accuracy of the monitoring, it has around 4% of false negative rate.

Table 4
The average of violation detection using STL and TTL.

	True Positive	True Negative	False Positive	False Negative
STL	75.76%	20.19%	2.61%	1.42%
TTL	75.76%	20.19%	0%	4.04%

The results are depicted in Fig. 10. The Red bar is for the false positive which is the most important result for this application. With higher false positive number, the probability of system crashing is higher as well since the monitoring device could not detect the timing violation. The red bar does not exist in TTL version since we could guarantee its correctness but using STL causes having false positive in almost all experiments.

As Table 4 shows, the rate of true positive and true negative, when we use either STL or TTL to express timing requirements, are the same. However, the big difference is their false negative and false positive coverage. Using STL as the logic language causes 2.61% false positives on average which is high for safety–critical applications [53]. On the other side, since the guarantee is one-sided, we inevitably have a false negative rate in the TTL-used verification method. Fig. 10 shows this at the green bars in TTL parts. The average of false negative rate for online monitoring methods using TTL is about 4.04%. It is a little bit higher than false negative in STL and it is the cost to have an accurate monitoring. In fact, in 4.04% of the experiments, we have false alarms for timing violations but we can correctly detect all violations by the methods utilizing TTL.

7.2. The satisfaction of the testbed requirements

In order to test and verify the timing constraints of the experimental setups, testbed timing specifications (synchronization accuracy, ADC sampling rate, ADC resolution, etc.) must exceed the CPS specifications as mentioned in Section 6.4. In this section, the specifications of the monitoring equipment are scrutinized to know whether they are qualified for monitoring of Flying Paster.

7.2.1. NI-cRIO setup

In a CompactRIO system (NI-cRIO), a controller with a processor and user-programmable FPGA is populated with one or more conditioned I/O modules from NI or third-party vendors. These modules provide direct sensor connectivity and specialty functions. cRIO is available in both a rugged industrial form factor and board-level design and it provides high-performance processing capabilities, sensor-specific conditioned I/O, and a closely integrated software toolchain that make them ideal for Industrial Internet of Things (IIoT), monitoring, and control applications.

As one of the testbeds for monitoring the time sensitive applications, two cRIO devices, NI-9067 and NI-9035, have been used as chassis while two signal acquisition modules, NI-9381 and NI-9232, installed on them for data acquisitions. Indeed, the modules collected the data to be processed on the FGPA board on the chassis. The specifications of the testbed equipment for NI-cRIO setup are summarized in the first two rows of Table 5.

The cRIO FPGA board has a clock with 40 MHz frequency and 5×10^{-6} clock drift. FPGA clocks are synchronized once a second using NI-TimeSync [54] that supports PTP. Measurement devices are connected via the dedicated Ethernet network. Implementation of the IEEE 802.1AS includes a very specific profile of IEEE 1588 (PTP) (part of IEEE 802.1 Time Sensitive Networking (TSN) standards) and uses hardware timestamping and compensation both in network elements and endpoints to minimize time synchronization errors. TSN generally provides both synchronization and also small and deterministic packet latency between testbed devices.

7.2.2. Clock specifications

The clock drift of the measurement nodes is 5×10^{-6}, where each node synchronizes every second via PTP with a precision of 1 µs to the grandmaster. The worst-case clock time offset of each cRIO is $\frac{5\ \mu s}{1\ s} + 1\ \mu s = 6\ \mu s$.

Table 5
The specifications of monitoring devices.

Monitoring Setup	DAQ card	Clk Drift	r_{sync}	Sampling Rate	ADC	Z_{in}	ϵ_{total}
cRIO-9067	NI-9381	5 PPM	1 Hz	10 kS/s	12 Bits	1 MΩ	106 μs
cRIO-9035	NI-9232	5 PPM	1 Hz	102.4 kS/s	24 Bits	305 kΩ	15.7 μs

7.2.3. ADC and sampling time

Since the voltage range of digital module is from 0 V to 5 V and it uses a 12-bit and 24-bit ADC for NI-9381 and 9232 respectively, the ADC resolution V_{ADC} can be calculated as $\frac{5-0}{2^{12}} \approx 1$ mV and $\frac{5-0}{2^{24}} \approx 300$ nV in order. As Table 5 shows, the sampling rates for NI-9381 and NI-9232 modules are 20 and 102.4 kilo samples per second meaning the sampling time is around 50 μs and 9.7 μs respectively. Based on the calculated errors in clock and ADC, the total error cRIO setup is:
NI-9381:

$$\epsilon_{total} = 6\ \mu s + 100\ \mu s = 106\ \mu s$$

NI-9232:

$$\epsilon_{total} = 6\ \mu s + 9.7\ \mu s = 15.7\ \mu s$$

ϵ_{total} is defined in Flying Paster specifications that is 100 μs (the minimum tolerable uncertainty), the precision of NI-9381 is not enough (it is 106 μs) while NI-9232 (its precision is 15.7 μs) is good enough for testing the applications ($\epsilon > \epsilon_{total}$).

7.3. Providing on-way guarantee in run-time verification for breaker tripping in power systems

In order to achieve optimum electrical distribution system protection in power systems, a set of rules are defined on when should a breaker trip – in case of an overcurrent – so that **only** the closest breaker to the fault trip. We also simulated the input signal of a breaker to verify if it trips correctly according to the specified rule (IEEE 1547). One of the timing constraint states that the breaker should trip if the duration at which the voltage is above 1.2 p.u. (per unit) is greater than 160 ms. The generated signals and monitoring system are simulated in Matlab.

7.3.1. Specification in STL

This timing constraint can be written in STL as:

$$\square(\square_{[0,0.16]} s(t) > 1.2 \implies trip)$$

Since this timing constraint does not include tolerance, its monitoring will not account for uncertainties in the measurement and therefore, we can have false positive.

7.3.2. Specification in TTL

The same timing constraint can be specified in TTL as:

$$\mathcal{L}(\langle s(t), 1.2, \nearrow\rangle, \langle s(t), 1.2, \searrow\rangle, 0.02) < 0.16$$

which is specified based on two events, when the voltage becomes greater than 1.2 p.u. and when it becomes less than 1.2 p.u. We simulated a signal to showcase that existing monitoring approaches fail to detect a timing violation when STL is used for specification while our approach can detect them. Fig. 11 shows two simulation scenarios and cases where the breaker shall trip. In the top rows, two arbitrary signals are generated where the frequency is increasing. The threshold is a yellow dashed line drawn at 1.2 p.u. The second rows from top show the actual time where the breaker shall trip, the third rows show the monitoring without considering uncertainty and tolerance and the bottom row show our monitoring approach. The red boxes highlight cases where the breaker shall trip (according to the second row) but without considering the uncertainty and tolerance, it is not detected as shall trip (third row), while our approach successfully detects them as shall trip (bottom row). Blue boxes (one case in the left figure and three cases in the right figure) highlight false negative cases where the it is falsely detected as shall trip in our approach.

In these simulations, $\delta = 0.01s$ and $\varepsilon = 0.02$ s. We can see that by considering the tolerance (ε) and uncertainty (δ), our approach can detect all shall trip cases, however, the rate of False negative will be higher than a monitoring system where uncertainty and tolerance are not accounted for. This acknowledges that our approach provides a one-way guarantee that other approaches do not.

7.4. The impacts of having tolerance on the required resources in run-time verification

After knowing the impact of considering tolerance on the monitoring of safety temporal specifications, we studied its effects on required area in synthesizing on FPGA. Therefore, we implemented run-time verification method, TMA, using three different precision models.

1. Tightly accurate implementation. We used the highest-precision numbers for clock values, variables, constants, and operations. For instance, the numbers are Extended precision, Long Integer, Double precision. Mathematical operations are compatible with the numeric values. Hence, they are also in their highest precision.
2. Moderately accurate implementation. We used the precision for the numeric parameters based on the defined tolerance in the TTL statements. For example, most of the timing constraints in Section 7.1.3 has 10^{-4}, some has 10^{-3}, and the velocity of the active roll should be at the target speed within 0.5% of tolerance. Based on the tolerance value, we decreased the required memory from the data types in Tightly Accurate version to integer and fixed-point, and the corresponding operations also updated.
3. Loosely accurate implementation. We increased the tolerances in Section 7.1.3 from 10^{-4} to 10^{-3}, from 10^{-3} to 10^{-2}, and the tolerance of velocity to 2%. According to the modifications on the tolerance, some timing specifications also has been changed. For instance, the constraint for velocity has been changed to 8 m/s and time between Contact and Cut has been increased to 50 ms.

Based on the above scenarios, we implemented the monitoring system for all timing requirements of flying paster on NI-cRIO 9067 and compared the required space in terms of the number of Flip-flops and Lookup Tables.

As Fig. 12 demonstrates, changing the CPS requirements by designers to have relaxed specifications (increasing the level of tolerance), the needed space on FPGA board to monitor the timing constraints is reduced. By defining the maximum tolerable uncertainty, the developers are able to reduce the required calculation resources. In this experiment, we could reduce the required number of FFs and LUTs by 1.23% and 0.6%. This simplification for implementing the CPS itself is also applicable since the designer develop real-time systems as precise as possible while it is not needed based on the tolerable error.

7.5. Monitoring temporal properties of quadcopter

A multirotor helicopter, known as Quadrotor or Quadcopter, has four rotors to lift and fly. In fact, a lift force is created by narrow-cord horizontally rotating airfoils [55]. The quadcopter's flight controller sends information to the motors via their electronic speed control circuits (ESC) information on thrust, RPM, (Revolutions Per Minute),

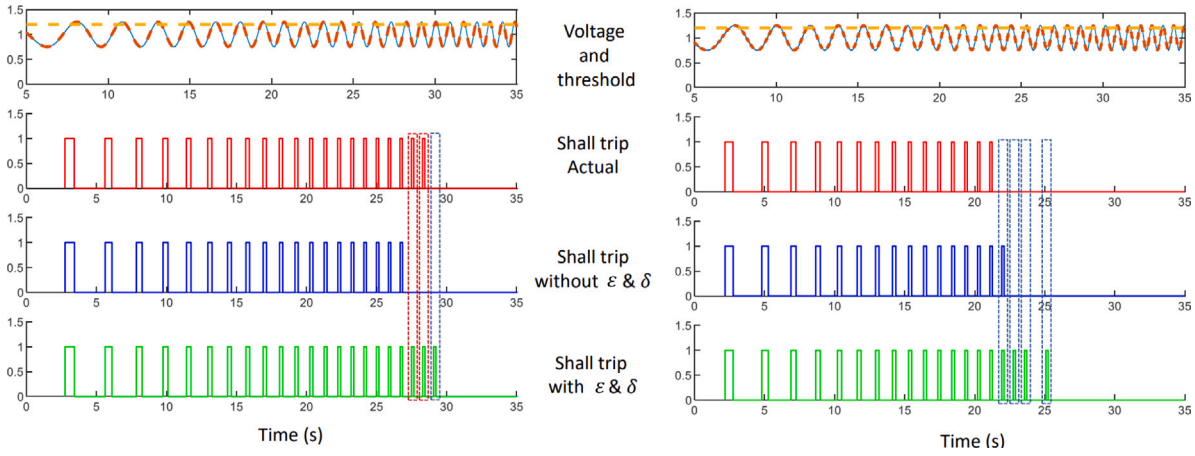


Fig. 11. (Left) One false positive case is detected (red box) where the monitoring without considering the uncertainty and tolerance fail to detect it while our approach successfully detect it. (Right) Three false negative cases are shown (blue boxes) where our approach detects it as shall trip. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

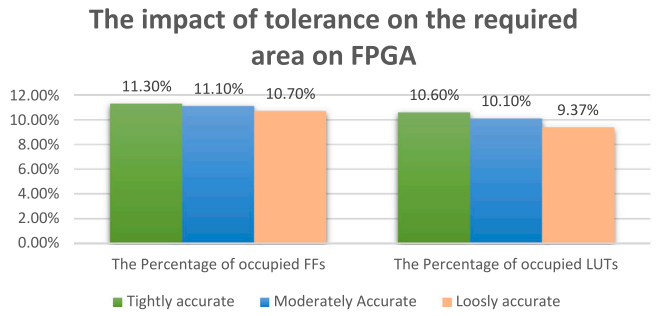


Fig. 12. The percentages of occupied Flip-flops and LUTs for the Flying Paster application on cRio device when the tolerance values are tightly, moderately and loosely accurate selected.

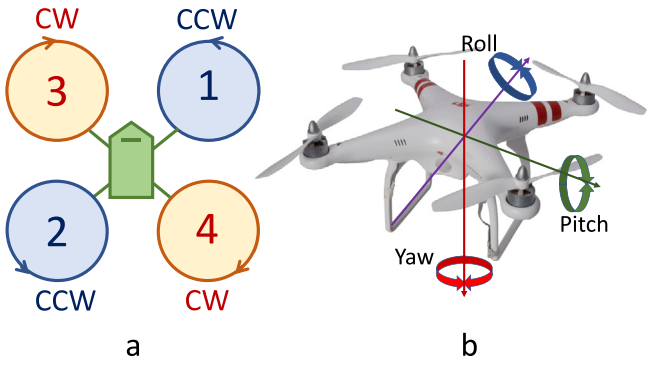


Fig. 13. Motor rotations, yaw, pitch, and roll in a quadcopter. (a) two motors should rotate Clockwise (CW), and two Counterclockwise (CCW). (b) Changing thrust on some motors changes roll, yaw, and pitch and finally moves the vehicle.

and direction. The flight controller will also combine IMU, Gyro, and GPS data before signaling to the quadcopter motors on thrust and rotor speed. As Fig. 13.a depicts, there are four motors in a Quad X (because it shapes an X). In order to implement a flight scenario, it is required that some motors rotate Clockwise (CW) and the other Counterclockwise (CCW). This way, the four propellers can generate lift and thrust simultaneously. The rotation of the drone along the x-axis is called roll, along the y-axis is called pitch, and along the z-axis is called yaw (Fig. 13.b). To control the motion of the quadcopter, we need to control the speed of each motor. For instance, the drone's left–right motion can be controlled by changing its roll. If the drone needs to move towards left/right, the thrust on the right/left motors is increased. Similarly, the drone's front–back motion can be controlled by changing the pitch and changing the thrust on the front or back motors. Sometimes it is required to reverse the rotation of two or more motors to have dynamic braking and prevent a crash [56]. Thanks to brushless outrunner motors [57] which have more than enough power to create reverse direction, we can do the reversion using hardware equipment or software codes at the millisecond level. As the last case study, we use a temporal property of a flying quadcopter to reverse motors simultaneously considering ϵ as the acceptable error in which all motors should be inverted within that duration. We have 4 signals in this scenario: ϕ_{m1}, ϕ_{m2}, ϕ_{m3}, and ϕ_{m4} are four signals showing that motor1, motor2, motor3, and motor4 has been reversed in order.

To ensure that the vehicle works well in a dangerous situation, we need to test all motors to see their responses in obstacle avoidance algorithms. Hence, in the worst case, it is required to monitor ϕ_{m1} – ϕ_{m4} and see if they are roughly received at the same time (within 100 ms) [58].

Table 6
The number of required Flip-Flops and Lookup Tables on FPGA (PYNQ) using STL and TTL with considering tolerance.

	#FF	#LUTs
Specification in STL [60]	22415	28836
Specification in TTL	614	894

STL:

$$\square((\uparrow \phi_{m1} \rightarrow \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m2} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m3} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m4})$$
$$\vee (\uparrow \phi_{m2} \rightarrow \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m1} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m3} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m4})$$
$$\vee (\uparrow \phi_{m3} \rightarrow \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m1} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m2} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m4})$$
$$\vee (\uparrow \phi_{m4} \rightarrow \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m1} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m2} \wedge \Diamond_{[0,100\ \text{ms})} \uparrow \phi_{m3}))$$

TTL:

$$S(\uparrow \phi_{m1}, \uparrow \phi_{m2}, \uparrow \phi_{m3}, \uparrow \phi_{m4}, 100\ \text{ms})$$

We implemented the monitoring system on FPGA, Xilinx PYNQ board. PYNQ [59] is an open-source project from Xilinx that makes it easy to design embedded systems with Zynq Systems on Chips (SoCs). This framework enables embedded programmers to exploit the capabilities of Xilinx Zynq. As a result, for the implementation, we consider the size of Flip-Flops and LUTs in both methods. For implementing specification in STL style we utilized the method proposed in [60,61] for TTL. The observation is summarized in Table 6.

As Table 6 shows, the required resources to implement the TTL monitoring algorithm on FPGA is less than implemented STL monitoring algorithm. That is because, there are several STL temporal operators

in this example, and also the available methods using STL need to store the time interval for each statement. In this experiment, we considered tolerance in STL as well as TTL, but since STL required more operators for the same functionality, the implementation needs more computation power and space on hardware.

8. Conclusion and future works

We proposed a formalism to consider the allowed tolerance by designers. We represent the conditions to ensure a run-time verification system is accurate enough to monitor a sort of timing specifications for safety–critical applications. TTL provides the required proofs to guarantee the accuracy of online monitoring process. In fact, by considering the maximum allowable error value, TTL can cover the existing uncertainties in the environment and system itself. Moreover, since the system/verification designers are aware of the tolerable error, they do not need to implement the monitoring device as precise as possible. It is enough the monitoring accuracy satisfies the constraints within their tolerance values. This reduces the required space and electricity power to synthesis run-time verification methods on hardware.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

References

[3] M. Khayatian, M. Mehrabian, A. Shrivastava, RIM: Robust intersection management for connected autonomous vehicles, in: Proceedings - Real-Time Systems Symposium, Vol. 2018-Decem, IEEE, 2019, pp. 35–44, http://dx.doi.org/10.1109/RTSS.2018.00014.

[4] M. Khayatian, R. Dedinsky, S. Choudhary, M. Mehrabian, A. Shrivastava, R 2 im-robust and resilient intersection management of connected autonomous vehicles, in: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems, ITSC, IEEE, 2020, pp. 1–6.

[5] S. Karnouskos, Cyber-physical systems in the SmartGrid, IEEE Int. Conf. Ind. Inform. (INDIN) 1 (10) (2011) 20–23, http://dx.doi.org/10.1109/INDIN.2011.6034829.

[6] M. Hassanalian, A. Abdelkefi, Classifications, applications, and design challenges of drones: A review, Prog. Aerosp. Sci. 91 (2017) 99–131.

[7] E.a. Lee, Cyber-physical systems - Are computing foundations adequate? 1, 2006, pp. 1–9, October. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.8011&rep=rep1&type=pdf.

[8] M. Khayatian, M. Mehrabian, E. Andert, R. Grimsley, K. Liang, Y. Hu, I. McCormack, C. Joe-Wong, J. Aldrich, B. Iannucci, et al., Plan B: Design methodology for cyber-physical systems robust to timing failures, ACM Trans. Cyber-Phys. Syst. (TCPS) 6 (3) (2022) 1–39.

[9] A. Pnueli, The temporal logic of programs, Proc. - Annual IEEE Symp. Found. Comput. Sci. FOCS 1977-Octob (1977) 46–57, http://dx.doi.org/10.1109/sfcs.1977.32.

[10] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (4) (1990) 225–299.

[11] O. Maler, D. Ničković, Monitoring properties of analog and mixed-signal circuits, Int. J. Softw. Tools Technol. Transf. 15 (3) (2013) 247–268, http://dx.doi.org/10.1007/s10009-012-0247-9.

[1] O. Maler, D. Ničković, Monitoring properties of analog and mixed-signal circuits, Int. J. Softw. Tools Technol. Transf. 15 (3) (2013) 247–268.

[2] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146.

[12] H. Kopetz, Sparse time versus dense time in distributed real-time systems, in: ICDCS, 1992, pp. 460–467.

[13] M. Mehrabian, M. Khayatian, r. Shrivastava, J.C. Eidson, P. Derler, H.A. Andrade, Y.-S. Li-Baboud, E. Griffor, M. Weiss, K. Stanton, Timestamp temporal logic (TTL) for testing the timing of cyber-physical systems, ACM Trans. Embed. Comput. Syst. (TECS) 16 (5s) (2017) 1–20.

[14] X. Zheng, C. Julien, M. Kim, S. Khurshid, X. Zheng, C. Julien, M. Kim, S. Khurshid, On the state of the art in verification and validation in cyber physical systems TR-ARiSE-2014-001 in cyber physical systems, ARiSE 1 (1) (2014) 1–13.

[15] A.N. Prior, Time and Modality, OUP Oxford, USA, 2003.

[16] A. Pnueli, The temporal logic of programs, Proc. - Annual IEEE Symp. Found. Comput. Sci. FOCS 1977-Octob (1977) 46–57, http://dx.doi.org/10.1109/sfcs.1977.32.

[17] A.P. Zohar Manna, the Temporal Logic of Reactive and Concurrent Systems, Springer, New York, 1991, p. 1.

[18] S. Owicki, L. Lamport, Proving liveness properties of concurrent programs, ACM Trans. Program. Lang. Syst. (TOPLAS) 4 (3) (1982) 455–495, http://dx.doi.org/10.1145/357172.357178.

[19] D.M. Gabbay, et al., On the temporal analysis of fairness, in: ACM SIGPLAN-SIGACT Symposium, POPL, ACM, USA, 1980, pp. 163–173.

[20] E.A. Emerson, E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, Sci. Comput. Program. 2 (3) (1982) 241–266, http://dx.doi.org/10.1016/0167-6423(83)90017-5.

[21] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Program. Lang. Syst. (TOPLAS) 8 (2) (1986) 244–263, http://dx.doi.org/10.1145/5397.5399.

[22] A. Dokhanchi, B. Hoxha, G. Fainekos, Metric interval temporal logic specification elicitation and debugging, in: 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE, IEEE, 2015, pp. 70–79.

[23] R. Alur, T.A. Henzinger, A really temporal logic, J. ACM 41 (1) (1994) 181–203, http://dx.doi.org/10.1145/174644.174651, URL https://dl.acm.org/doi/abs/10.1145/174644.174651.

[24] G.E. Fainekos, G.J. Pappas, Robustness of temporal logic specifications, in: Formal Approaches to Software Testing and Runtime Verification, Springer, 2006, pp. 178–192.

[25] G.E. Fainekos, G.J. Pappas, Robustness of temporal logic specifications for continuous-time signals, Theoret. Comput. Sci. 410 (42) (2009) 4262–4291.

[26] Y. Annpureddy, C. Liu, G. Fainekos, S. Sankaranarayanan, S-taliro: A tool for temporal logic falsification for hybrid systems, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2011, pp. 254–257.

[27] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, A. Gupta, Probabilistic temporal logic falsification of cyber-physical systems, ACM Trans. Embed. Comput. Syst. (TECS) 12 (2s) (2013) 1–30.

[28] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, S. Sankaranarayanan, Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications, in: Lectures on Runtime Verification, Springer, 2018, pp. 135–175.

[29] J.V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, S.A. Seshia, Robust online monitoring of signal temporal logic, Form. Methods Syst. Des. 51 (1) (2017) 5–30.

[30] A. Donzé, O. Maler, Robust satisfaction of temporal logic over real-valued signals, in: International Conference on Formal Modeling and Analysis of Timed Systems, Springer, 2010, pp. 92–106.

[31] A.P. Sistla, E.M. Clarke, The complexity of propositional linear temporal logics, J. ACM 32 (3) (1985) 733–749.

[32] C. Yoo, C. Belta, Control with probabilistic signal temporal logic, 2015, arXiv preprint arXiv:1510.08474.

[33] Y. Lin, H. Li, M. Althoff, Model predictive robustness of signal temporal logic predicates, 2022, arXiv preprint arXiv:2209.07881.

[34] S. Mukherjee, P. Dasgupta, S. Mukhopadhyay, Auxiliary specifications for context-sensitive monitoring of AMS assertions, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 30 (10) (2011) 1446–1457.

[35] A. Donzé, Breach, a toolbox for verification and parameter synthesis of hybrid systems, in: Lecture Notes in Computer Science, in: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 6174 LNCS, 2010, pp. 167–170, http://dx.doi.org/10.1007/978-3-642-14295-6_17.

[36] Y. Annpureddy, C. Liu, G. Fainekos, S. Sankaranarayanan, S-TaLiRo: A tool for temporal logic falsification for hybrid systems, in: Lecture Notes in Computer Science, in: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 6605 LNCS, 2011, pp. 254–257, http://dx.doi.org/10.1007/978-3-642-19835-9_21.

[37] D. Nickovic, O. Maler, AMT: A property-based monitoring tool for analog systems, in: International Conference on Formal Modeling and Analysis of Timed Systems, Springer, USA, 2007, pp. 304–319.

[38] J.V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, S.A. Seshia, Robust online monitoring of signal temporal logic, Form. Methods Syst. Des. 51 (1) (2017) 5–30, http://dx.doi.org/10.1007/s10703-017-0286-7.

[39] K. Selyunin, S. Jaksic, T. Nguyen, C. Reidl, U. Hafner, E. Bartocci, D. Nickovic, R. Grosu, Runtime monitoring with recovery of the sent communication protocol, in: Lecture Notes in Computer Science, in: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 10426 LNCS, 2017, pp. 337–355, http://dx.doi.org/10.1007/978-3-319-63387-9_17.

[40] P. Moosbrugger, K.Y. Rozier, J. Schumann, R2U2: monitoring and diagnosis of security threats for unmanned aerial systems, Form. Methods Syst. Des. 51 (1) (2017) 31–61, http://dx.doi.org/10.1007/s10703-017-0275-x.

[41] S. Jaksic, E. Bartocci, R. Grosu, R. Kloibhofer, T. Nguyen, D. Nickovie, From signal temporal logic to FPGA monitors, in: 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign, Vol. 15, No. 15, MEMOCODE 2015, 2015, pp. 218–227, http://dx.doi.org/10.1109/MEMCOD.2015.7340489.
[42] On signal temporal logic, 2021, https://people.eecs.berkeley.edu/~sseshia/fmee/lectures/EECS294-98_Spring2014_STL_Lecture.pdf. (Accessed on 13 October 2021).
[43] Z. Cvetkovic, I. Daubechies, B.F. Logan, Single-bit oversampled A/D conversion with exponential accuracy in the bit rate, IEEE Trans. Inform. Theory 53 (11) (2007) 3979–3989.
[44] E.A. Lee, Constructive models of discrete and continuous physical phenomena, IEEE Access 2 (2014) 797–821.
[45] Z. Cvetkovic, M. Vetterli, On simple oversampled A/D conversion in L/sup 2/(R), IEEE Trans. Inform. Theory 47 (1) (2001) 146–154.
[46] O. Maler, D. Nickovic, Monitoring Temporal Properties of Continuous Signals, Vol. 2, Springer, 2004, pp. 152–166, http://dx.doi.org/10.1007/978-3-540-30206-3_12, URL http://link.springer.com/10.1007/978-3-540-30206-3_12.
[47] A. Shrivastava, M. Mehrabian, M. Khayatian, P. Derler, H. Andrade, K. Stanton, Y.S. Li-Baboud, E. Griffor, M. Weiss, J. Eidson, INVITED: A testbed to verify the timing behavior of cyber-physical systems: Invited, Proc. - Des. Autom. Conf. Part 12828 (69) (2017) 1–6, http://dx.doi.org/10.1145/3061639.3072955.
[48] M. Lombardi, Time and Frequency from A to Z, A to Z Notes, Vol. 1, Springer, 2010, pp. 1–100.
[49] J.C. Eidson, K.B. Stanton, Timing in cyber-physical systems: The last inch problem, in: IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication, Vol. 2015-Novem, ISPCS, IEEE, 2015, pp. 19–24, http://dx.doi.org/10.1109/ISPCS.2015.7324674.
[50] D. Mills, RFC1305: Network time protocol (Version 3) specification, implementation, RFC Editor, USA, 1992.
[51] K.B. Lee, J.C. Eidson, H. Weibel, D. Mohl, NISTIR 7302 Proceedings of the 2005 Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems Proceedings of the 2005 Conference on IEEE 1588 Standard for a Precision Clock Synchronizatio, Vol. 1, No. December, NIST, 2005, pp. 1–533.
[52] M. Lipiński, T. Włostowski, J. Serrano, P. Alvarez, White rabbit: A PTP application for robust sub-nanosecond synchronization, in: IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication, Vol. 1, ISPCS, IEEE, 2011, pp. 25–30, http://dx.doi.org/10.1109/ISPCS.2011.6070148.
[53] J.H. Lala, R.E. Harper, Architectural principles for safety-critical real-time applications, Proc. IEEE 82 (1) (1994) 25–40, http://dx.doi.org/10.1109/5.259424.
[54] NI time sync, version 1.1, 2010, [Online; 2010 National Instruments Corporation], http://www.ni.com/pdf/manuals/373185a.pdf.
[55] D. Norris, Build Your Own Quadcopter: Power Up Your Designs with the Parallax Elev-8, McGraw-Hill Education, 2014.
[56] G. Naduvilakandy, Dynamics and Control of a Quadcopter (Ph.D. thesis), Texas A&M University-Kingsville, 2016.
[57] V. Mach, S. Kovář, J. Valouch, M. Adamek, Brushless DC motor control on arduino platform, Prz. Elektrotech. (2018).
[58] ardupilot/ArduCopter at master ArduPilot/ardupilot, , 2023, https://github.com/ArduPilot/ardupilot/tree/master/ArduCopter. (Accessed on 09 January 2023).
[59] PYNQ - Python productivity for Zynq - Home, 2023, http://www.pynq.io/. (Accessed on 09 January 2023).
[60] S. Jakšić, E. Bartocci, R. Grosu, R. Kloibhofer, T. Nguyen, D. Ničković, From signal temporal logic to FPGA monitors, in: 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE, IEEE, 2015, pp. 218–227.
[61] M. Mehrabian, M. Khayatian, A. Mousa, r. Shrivastava, Y.-S. Li-Baboud, P. Derler, E. Griffor, H.A. Andrade, M. Wiess, J.C. Eidson, et al., An efficient timestamp-based monitoring approach to test timing constraints of cyber-physical systems, in: Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.

Mohammadreza Mehrabian received his B.E. degree in Computer Hardware Engineering from Shahid Bahonar University of Kerman, Kerman, Iran 2004. He received his Master degree in Computer Architecture from Amirkabir University of Tehran, Tehran, Iran in 2011. In 2015, he joined Arizona State University as a Ph.D. student in Make Programming Simple Lab. His research interests currently include Cyber-Physical Systems (CPS), Internet of Things (IoT), Runtime Verification, Temporal Logic, Embedded System Design, Hardware/Software Co-Design, FPGA Design, bare-metal programming, Machine Learning, and Virtual/Augmented Reality. He recieved Richard Newton Young Fellow award, 53rd DAC in 2016, CIDSE Doctrol Fellowship in 2020, Ferdinand A. Stanchi Fellowship in 2019, and University Graduate Fellowship in 2018 from ASU. Currently, he is an Assistant Professor in the department of Electrical Engineering and Computer Science South Dakota School of Mines and Technology and manages Depandable Cyber-Physical Systems lab in this department.

Mohammad Khayatian is a Senior Robotics Software Engineer at Vecna Robotics. Prior to that, he was an Assistant Professor of Computer Engineering at San Jose State University. Mohammad received his Ph.D. in Computer Engineering from Arizona State University. and his Bachelor and Master from Shiraz University in Electrical Engineering with a focus on Control systems. He has served as an internal/external reviewer many conferences including DAC, MEMOCODE, EMSOFT, CODES+ISSS, CASES, FCCM, VLSID, DATE, ICCPS, and RTSS. Mohammad's research interests include Timing of Cyber–Physical Systems (CPS), Connected Autonomous Vehicles, Control System Theory and Time-sensitive Programming of Embedded Systems. Mohammad and his colleague's paper was the nominee for the best paper award at DAC 2017. He received the Richard Newton Young Student Fellowship in 53rd Design Automation Conference 2017 and has received Best Ph.D. Student in Computer Engineering Award from School of Computing, Informatics, and Decision Systems in 2018 and 2019. Mohammad received the Dean's Dissertation Award from ASU in 2021.

Aviral Shrivastava is a full Professor in the School of Computing and Augmented Intelligence (SCAI) at the Arizona State University, where he established and heads the Make Programming Simple Lab (https://labs.engineering.asu.edu/mps-lab/). He completed his Ph.D. in Information and Computer Science and from the University of California, Irvine, and Bachelors in Computer Science and Engineering from IIT Delhi. Prof. Shrivastava's main theme of research in on making programming simple for embedded and cyber–physical systems. Prof. Shrivastava and his students have proposed novel computer architectures and compiler transformations for hardware errortolerant computing, multicore computing, accelerated computing. They have also proposed languages, code generation and runtime for expressing and efficiently executing time-sensitive distributed intelligent applications. Prof. Shrivastava has co-authored 1 book, and has contributed chapters in 4 books. He has more than 120 articles and conference papers in top embedded system journals and conferences, like DAC, ESWEEK, ACM TECS, and ACM TCPS. His papers have received several awards, including nomination for best paper at DAC 2017, best student paper award at VLSI 2016, second highest ranked paper at LCTES 2010, and best paper candidate ASPDAC 2008. He published at least one paper every year at DAC (the top conference in the field) in the last decade (2011 to 2019). Overall, his works have received more than 3000 citations, growing at the rate of over 200 citations every year. His i50-index is 14, i10-index is 84, and hindex is 31 (reference Google Scholar). His inventions have been granted 5 patents, and 5 more applications are pending. Prof. Shrivastava is the recipient of the prestigious 2010 NSF CAREER award. His students thesis were awarded CIDSE outstanding Ph.D. thesis award in 2021 and 2017 and outstanding Master's thesis awards in 2011 and 2014. Prof. Shrivastava's research efforts have been supported by federal agencies (NSF, DOE, NIST), state funding agencies (SFAZ), as well as industry. Prof. Shrivastava has mentored 2 postdocs, 9 Ph.D. students, and over 20 Masters students. His students are very well placed, including a full Professor at UNIST, South Korea, Assistant Professor at SJSU, ARM research lab, Google, Synopsys, Apple (x2), Qualcomm, Cadence etc. Prof. Shrivastava is currently supervising 3 Ph.D., and 5 Masters students. Prof. Shrivastava teaches undergraduate and graduate level courses on computer organization, computer architecture, and embedded systems, and has student evaluations averaging over 4/5. He revamped the computer organization and computer Architecture courses at ASU to shift the focus towards processor design instead of assembly language programming and included modules about modern multicore architectures. He has redesigned the embedded systems course around projects in which

students build an autonomously driving car, culminating in an autonomous car race! (https://www.youtube.com/channel/UCDfyzk7HFqeXCb5BK02SemQ)Prof. Shrivastava is currently the General Chair of Embedded Systems Week (ESWEEK), which is the top event in the field of Embedded Systems, comprising of several conferences, symposia and workshops. He also serves in the Steering committee of the Languages Compilers, Theory and tools for Embedded Systems (LCTES). Currently, he is the deputy Editor-in-Chief of IEEE Embedded Systems Letters (IEEE ESL), and associate editor for ACM Transactions of Cyber-Physical Systems (ACM TCPS), ACM Transactions Embedded Computing Systems (ACM TECS), and the IEEE Transactions on Computer Aided Design (IEEE TCAD). Previously he has served as the program chair of CODES+ISSS 2017 and 2018, LCTES 2019, and chair of the Design and Applications track of RTSS 2020.

Dr. Patricia Derler is a computer scientist, researcher, and software engineer with extensive background in modeling, design, simulation, verification, and testing of complex, distributed, heterogeneous systems, as is currently a member of the research staff at the Palo Alto Research Center (PARC). Prior to joining PARC, she held positions in industry and academia, including a postdoctoral research engagement at UC Berkeley, a research scientist role at National Instruments, and the appointment of director of engineering at Kontrol, a startup developing technology towards enabling certification of autonomous vehicles. Dr. Derler received her Ph.D. in Computer Science from the University of Salzburg, Austria, where she graduated with presidential honors (Promotio sub auspiciis Praesidentis rei publicae).

Hugo A. Andrade works in the Adaptive and Embedded Computing Group at AMD, where he leads the University Program and focuses on enabling the use of Adaptive Compute technologies for academic teaching, research, and entrepreneurial activities. Before joining Xilinx in San Jose, CA, he was most recently Principal Product Manager, Advanced Software Technologies, at NI in Berkeley, where he focused on researching technologies for system level development for heterogeneous CPS/IoT platforms. He served as liaison to academic and industrial research labs in the area, was a visiting industrial fellow at the University of California, Berkeley, and was founding manager and technical lead of the NI Berkeley LabVIEW Advanced R&D site. Hugo has authored or co-authored over 65 patents and 25 academic research articles in the areas of virtual instrumentation, hardware/software interfacing, reconfigurable computing, graphical programming, models of computation, and system level design.