An Efficient Timestamp-Based Monitoring Approach to Test Timing Constraints of Cyber-Physical Systems

Mohammadreza Mehrabian
Arizona State University
Mohammadreza.Mehrabian@asu.edu

Mohammad Khayatian
Arizona State University
mkhayati@asu.edu

Ahmed Mousa
Arizona State University
aomoussa@asu.edu

Aviral Shrivastava
Arizona State University
Aviral.Shrivastava@asu.edu

Ya-Shian Li-Baboud
National Institute of Standards and Technology
ya-shian.li-baboud@nist.gov

Patricia Derler
National Instruments
patricia.derler@ni.com

Edward Griffor
National Institute of Standards and Technology
edward.griffor@nist.gov

Hugo A. Andrade
Xilinx
hugoa@xilinx.com

Marc Wiess
Marc Weiss Consulting
marcweissconsulting@gmail.com

John C. Eidson
University of California Berkeley
eidson@eecs.berkeley.edu

Anand Dhananjay
National Institute of Standards and Technology
dhananjay.anand@nist.gov

ABSTRACT

Formal specifications on temporal behavior of Cyber-Physical Systems (CPS) is essential for verification of performance and safety. Existing solutions for verifying the satisfaction of temporal constraints on a CPS are compute and resource intensive since they require buffering signals from the CPS prior to constraint checking. We present an online approach, based on Timestamp Temporal Logic (TTL), for monitoring the timing constraints in CPS. The approach reduces the computation and memory requirements by processing the timestamps of pertinent events reducing the need to capture the full data set from the signal sampling. The signal buffer size bears a geometric relationship to the dimension of the signal vector, the time interval being considered, and the sampling resolution. Since monitoring logic is typically implemented on Field Programmable Gate Arrays (FPGAs) for efficient monitoring of multiple signals simultaneously, the space required to store the buffered data becomes the limiting resource. The monitoring logic, for the timing constraints on the Flying Paster (a printing application requiring synchronization between two motors), is illustrated in this paper to demonstrate a geometric reduction in memory and computational resources in the realization of an online monitor.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; **Embedded software**;

1 INTRODUCTION

Cyber-Physical Systems (CPS) are becoming an integral part of human life. While it is desirable to build systems with guarantees of correct behavior, it is becoming increasingly difficult, due to the increasing scale, complexity, and non-deterministic nature of applications, networks, processing platforms, and unpredictable interactions with the physical world [1]. One promising approach to ensure that the system is executing in a safe manner is to monitor the system at runtime [2]. In online monitoring, application constraints are continuously monitored during runtime. Online monitoring can be used to analyze the system behavior in the field and check for bugs in the design. In contrast, offline monitoring in real systems utilizes forensic analysis and therefore does not offer the ability for timely correction of system deviations. Although offline monitoring can be useful, online monitoring is desirable since it may be possible to detect early violations and prevent a system from reaching an unsafe state [3].

Since the correct operation of many CPS applications relies upon the correct timing of the system, both functional and temporal requirements of a CPS must be monitored [4]. This paper focuses on the monitoring of timing constraints in CPS. Many existing monitoring systems define system timing constraints using Signal Temporal Logic (STL) [5]. STL allows users to define timing constraints on real-valued signals relative to current time. For example, in the *Globally* constraint in STL, a user could specify a timing constraint $\phi := \Box_{[2,6]}(|x[t]| < 2)$, which means that a property ϕ will be true at time $t = \tau$, *iff* the real-valued signal $x[t] \in [-2, 2]\ \forall \tau \in [2, 6]$. To compute whether the timing constraint ϕ was met at time $t = \tau$, the conventional approaches [5–10] record the value of the signal $x[t]$ at all times in the interval $t \in [\tau + 2, \tau + 6]$. The signal values are compared against the constraint, $(|x[t]| < 2)$, to determine if the requirements are satisfied within the time interval $[\tau + 2, \tau + 6]$. The constraint evaluation is repeated for each sampling period.

Often, existing monitoring systems are implemented in a simulation. To test real-time systems, FPGA (Field Programmable Gate Array) implementation has the potential to minimize computational latencies and allows for simultaneous monitoring of multiple signals, while supporting the flexibility for modifications and upgrades. The scheme by Jakšić *et al.* [11] was implemented on FPGAs. However, for FPGAs, the available memory to store the signal histories and perform the computation becomes the main bottleneck.

To evaluate how practical the existing state-of-the-art monitoring schemes are, we built a model of a Flying Paster application. We specified seven timing constraints to minimize the amount of

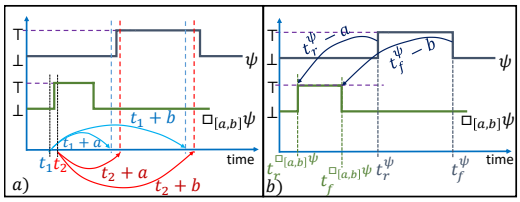

Figure 1: *Globally*: a) Conventional monitoring calculation at each time-step, b) TMA uses two subtractions per pulse.

unused paper while ensuring sufficient time to paste and splice the paper of the new roll before the first roll expires. The implemented test code to evaluate against the timing constraints using the (latest) *Counters* approach by Jakšić *et al.* [11], could not be compiled due to insufficient memory on the commercial off-the-shelf (COTS) FPGA board with 82, 000 flip-flops (FFs) and 41, 000 look-up tables (LUTs), at a sampling rate of 20 kHz. In CPS, examples of systems using high sampling rates include power systems where IEC 61869-9 specifies sampling rates at 4.8 kHz for Alternating Current (AC) and up to 96 kHz for Direct Current (DC) measurements [12].

In this paper, we propose a more efficient online approach for monitoring the timing constraints of CPS, **T**imestamp-based **M**onitoring **A**pproach. The key improvement is rather than evaluating a constraint at each sampling period, TMA only computes the constraint satisfaction at the occurrence of relevant events extracted from monitored signals. For constraint $\phi := \square_{[2,6]}(|x[t]| < 2)$, TMA identifies $x[t]$ as the signal-of-interest when $x[t]$ goes above or below 2 V. Accordingly, ϕ is re-computed only at the occurrence of the next event-of-interest. TMA can monitor all seven timing constraints of Flying Paster model application at a sampling rate of 20 kHz, using only 11% of the FFs and 11.5% of LUTs on the same FPGA.

In general, for a constraint that is defined over a time interval of T , and must be monitored for the duration of experiment d, with a sampling frequency of f, the conventional approach requires $O(Tfd)$ computation time. The requirements of both, computation time and memory, depending on the interval size and the sampling rate. In contrast, our approach has a complexity of $O(k)$, where k is the maximum number of events during time d. Case in point, both the computation and memory requirement of the monitoring logic for implementing a *Globally* constraint using the Jakšić *et al.* approach [11] increases with the interval size of the *Globally* operator, while the monitoring logic of TMA is independent of the time interval, and scales well. Another important point to note is that although event-based constraints (e.g, whenever signal s_1 *rises* above 2.5 V, the signal s_2 should *fall* below 1 V in less than 2 s.) can be specified in STL the logic that is generated can be complex and resource intensive since an event-based constraint is composed of several STL temporal operators. On the other hand, event-based temporal constraints are specified using TTL, *Simultaneity*, *Chronological*, *Phase*, *Frequency*, *Latency*, among others [13]. TTL provides for a more intuitive and simple specification (e.g., $\mathcal{L}(\langle s_1, 2.5, \nearrow\rangle, \langle s_2, 1, \nearrow\rangle) < 2$). In this paper, we apply two of the primitives, namely *Latency* and *Simultaneity*, to illustrate the online monitoring approach.

2 RELATED WORK

Conventional monitoring methods have high memory usage and processing time requirements since they evaluate timing constraints at every time-step. Offline tools for analyzing the timing requirements in CPS have been implemented in Breach[14], and S-Taliro [15]. Both tools record simulation data and evaluate timing constraints after the simulation has finished. Figure 1.a depicts the conventional monitoring approach. It plots the value of Boolean signal ψ along the time axis at the top. To evaluate the constraint $\square_{[a,b]}\psi$ at time t_1, the existing techniques look at the entire interval $[t_1 + a, t_1 + b]$. If the signal is *true* for the entire duration, the constraint is met at t_1. Since the signal ψ was *false* for some time (just after $t_1 + a$), the constraint $\square_{[a,b]}\psi$ is not met at time t_1. However, the constraint is met at t_2. The computation required to evaluate this constraint is $O(Tf^2)$, where T is the time interval over which the temporal operator is defined, which in this case, is $T = b-a$, and f is the sampling frequency. The memory buffer required for this computation will be $O(Tf)$. If there is a constraint with P temporal operators, and w signals, then the amount of computations is $O(TPwf^2)$, while the amount of buffer needed will be $O(TPwf)$. To monitor one timing constraint with four temporal operators defined over an interval of 100 s, with a system sampling rate and analog to digital converter (ADC) resolution of 20 kHz ($t_s = 50\ \mu s$) and 12-bit, we need 12 MB of memory ($M = 4 \times \frac{100}{50\ \mu s} \times \frac{12}{8}$). Primarily, because of the computational complexity, evaluating temporal constraints in real-time is not scalable. Practical CPS applications, such as power generation and distribution have numerous constraints, each containing multiple, high-frequency data signals to be monitored simultaneously, and may have evaluation time intervals over extended durations.

Recognizing the high overhead, AMT [16] proposed an incremental approach to compute the constraints at a segment granularity. However, they can reduce the complexity only by the factor of the granularity. An incremental method was proposed by Deshmukh *et al.* [10] where timing constraints are evaluated by traversing the parse tree generated for STL formulas. They optimize their calculation by eliminating repetitive computations.

While all the previous approaches were implemented in simulation, Jakšić *et al.* [11] 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. Although Jakšić *et al.* showed a way to reduce memory usage, the storage remains a concern (even for bounded constraints). This technique converts future STL operators into past ones and translates all constraints such that their interval starts from zero. Then, a counter is dedicated for 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). In each time-step, the active counter is incremented to measure the duration of positive pulses. The maximum number of counters is $\lceil \frac{2a}{b-a+1} \rceil$ where each counter has $\lceil \log_2 a \rceil$ bits. For example, $\square_{[5000,5001]}\psi$ needs 5000 counters, each with $\log_2 5000 = 13$ bits. Therefore, we need around 8 kB to monitor just one signal. In contrast, to monitor the same constraint using TMA, only the last two timestamps of the events-of-interest and the last two timestamps of the result are needed. Therefore, four $32 - bit$ variables for each operator is needed, which is independent of interval length and sampling rate, and a small memory footprint for the state machine. We need one

state machine per operator and the size of each state machine is very small since it needs only 2 bits to store the state.

STL expressions are often combined and/or nested and must be evaluated recursively. Additionally, although STL has the capability to express event-based timing constraints, they are constructed out of a variety of level-based timing constraints. In order to represent only one event (rising or falling) in STL, we should use past and future operators together in one expression[1]. In contrast, TTL can easily express the event timing constraint so that the implementation test code can be succinct as well.

3 TIMESTAMP MONITORING APPROACH

We use TTL to specify the application timing constraints, since TTL succinctly expresses both event-based and level-based timing constraints commonly used in CPS. TTL considers temporal behavior of analog signals upon a given threshold function in level-based timing constraints. Also, this logic can express event-based timing constraints where the time at which a signal value changes (e.g. $\mathcal{L}(\langle s_1, 2.5, \nearrow\rangle, \langle s_2, 1, \nearrow\rangle) > 2$, whenever a rising s_1 signal *crosses* 2.5 V, a rising s_2 shall not *cross* 1 V earlier than 2 s). Hence, we first convert analog signals to discrete event boolean signals by the method in [13, 17]. Therefore, we have $\mathbb{R} \rightarrow \mathbb{B}$ to transform the analog to boolean signals. Then, timestamps corresponding to rising and falling edges are extracted. We define finite sets of rising edges Γ_r and falling edges Γ_f for a boolean signal, ψ, as: $\Gamma_r = \{t^{\psi}_{r_1}, ..., t^{\psi}_{r_n}\}$ and $\Gamma_f = \{t^{\psi}_{f_1}, ..., t^{\psi}_{f_n}\}$ where $t^{\psi}_{r_i}$ and $t^{\psi}_{f_i}$ are the timestamps for the i^{th} rising and falling edge on ψ. Figure 2.a depicts a boolean signal ψ, which is created when the analog signal $s(t)$ crosses a threshold, $f(t)$ that shows after threshold crossing, the boolean signal is described by $t^{\psi}_{r_i}$ and $t^{\psi}_{f_i}$, $(i = 1, ..., n)$. Now, we present a boolean signal as a tuple consisting of an initial state (ψ_{init}), a set of rising edges (Γ_r) and a set of falling edges (Γ_f): $\psi = (\psi_{init}, \Gamma_r, \Gamma_f)$

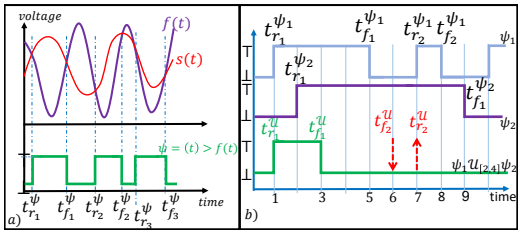

Figure 2: a) Crossing signal $s(t)$ with function $f(t)$, b) Illustration for *Until* computation by TMA.

The *differentiate* operator ($\bowtie$), $\varphi = \bowtie(\psi)$ extracts the rising edge of a boolean signal $\psi \in \mathbb{B}$ where the value of φ is 1 if $(\psi(t^+) \oplus \psi(t)) \wedge \neg\psi(t) = \top$, and $\bot$ otherwise. $\oplus$ is the XOR operator and, t^+ refers to the right neighborhood (the next time-step) of signal at time t. By applying differentiate operator on a boolean signal ψ ($\varphi = \bowtie \psi$), we provide another boolean signal, φ, which is *true* for a short period (sampling time) and *false* otherwise. Since this operator provides the event set, Θ^{φ}, it contains just the timestamps which show the time of ***events*** (not rising and falling) as: $\Theta^{\varphi} = \{\theta^{\varphi}_1, ..., \theta^{\varphi}_n\}$.

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

3.1 Level-based Approach

In this section, we introduce three algorithms executed at each rising and falling edge to define the set of timestamps for online constraint evaluation of level-based TTL operators.

3.1.1 Globally Rules. Given a boolean signal (ψ) expressed with a set of rising and falling edges Γ^{ψ}_r and Γ^{ψ}_f respectively, for every new pair of timestamps, generated from the signal threshold crossings, we update the set of rising and falling edges for $\Box_{[a,b]}\psi$ ($\Gamma^{\Box}_r$ and $\Gamma^{\Box}_f$) by applying Algorithm 1 on the received timestamps. The new $\Box_{[a,b]}\psi$ rising and falling edges are computed based on the most recent t^{ψ}_r (expressed as the current rising edge timestamp on ψ) , t^{ψ}_f (expressed as the current falling edge timestamp on ψ) as well as the values of a and b. The computed rising and falling edges are only added to $\Gamma^{\Box}_r$ and $\Gamma^{\Box}_f$ if its timestamp for the rising edge is less than that of the falling edge.

Algorithm 1 Globally (t^{ψ}_r , t^{ψ}_f , a, b)

1: $t^{\Box}_{r_i} = t^{\psi}_r - a$
2: $t^{\Box}_{f_i} = t^{\psi}_f - b$
3: **if** $t^{\Box}_{r_i} < 0$ **then**
4: $\quad t^{\Box}_{r_i} = 0$
5: **end if**
6: **if** $t^{\Box}_{r_i} <= t^{\Box}_{f_i}$ **then**
7: $\quad \Gamma^{\Box}_r = \Gamma^{\Box}_r + \{t^{\Box}_{r_i}\}$
8: $\quad \Gamma^{\Box}_f = \Gamma^{\Box}_f + \{t^{\Box}_{f_i}\}$
9: **end if**

3.1.2 Eventually Rules. A boolean signal (ψ) expressed with, Γ^{ψ}_r and Γ^{ψ}_f, for every new pair of timestamps, we update the set of rising and falling edges by applying Algorithm 2. The calculated timestamps are only added to the set under the constraint; a rising edge must occur after the last falling edge. Also, if the last computed falling is in the range of new pulse, the last falling should be replaced with the new falling edge to append the last pulse on the result.

3.1.3 Until Rules. Given two boolean signals, ψ_1 and ψ_2, with new rising and falling edges $t^{\psi_1}_r$, $t^{\psi_2}_r$, $t^{\psi_1}_f$ and $t^{\psi_2}_f$, we update the set of rising and falling edges for $\psi_1 \mathcal{U}_{[a,b]} \psi_2$ ($\Gamma^{\mathcal{U}}_r$ and $\Gamma^{\mathcal{U}}_f$) using Algorithm 3 with the incoming pairs of timestamps. The new rising and falling edges for *Until* are computed in the first 2 lines. Starting at line 3, new edges are either appended or discarded, depending on whether or not they comply with the signals. For example, any negative time value and any set of edges with a falling happening before a corresponding rising edge indicate the constraint is not satisfied. Similarly, any edge with rising that comes before the falling edge of the previous set is discarded and the previous falling is replaced with the new falling since the last positive pulse should be extended to the new falling edge. A pair of timestamps appended to $\Gamma^{\mathcal{U}}_r$ and $\Gamma^{\mathcal{U}}_f$ signifies that there is a new valid interval where the constraint, $\psi_1 \mathcal{U}_{[a,b]} \psi_2$, was met. As depicted in the *Until* example in Figure 2.b, we have $t^{\mathcal{U}}_{r_1} = max(1, 2-4) = 1$ and $t^{\mathcal{U}}_{f_1} = min(5, 9) - 2 = 3$. Since $t^{\mathcal{U}}_{r_1} < t^{\mathcal{U}}_{f_1}$ they can be used to update $\psi_1 \mathcal{U}_{[2,4]} \psi_2$ by being

Algorithm 2 Eventually (t_r^{ψ}, t_f^{ψ}, a, b)

$t_{r_i}^{\diamond} = t_r^{\psi} - b$
$t_{f_i}^{\diamond} = t_f^{\psi} - a$
if $t_{r_i}^{\diamond} < 0$ **then**
  $t_{r_i}^{\diamond} = 0$
end if
if $t_{f_{i-1}}^{\diamond} < t_{r_i}^{\diamond}$ **then**
  $\Gamma_r^{\diamond} = \Gamma_r^{\diamond} + \{t_{r_i}^{\diamond}\}$
  $\Gamma_f^{\diamond} = \Gamma_f^{\diamond} + \{t_{f_i}^{\diamond}\}$
end if
if $t_{r_i}^{\diamond} <= t_{f_{i-1}}^{\diamond}$ and $t_{f_{i-1}}^{\diamond} < t_{f_i}^{\diamond}$ **then**
  $\Gamma_f^{\diamond} = \Gamma_f^{\diamond} - \{t_{f_{i-1}}^{\diamond}\}$
  $\Gamma_f^{\diamond} = \Gamma_f^{\diamond} + \{t_{f_i}^{\diamond}\}$
end if

appended to $\Gamma_r^{\mathcal{U}}$ and $\Gamma_f^{\mathcal{U}}$. The potential $\psi_1 \mathcal{U}_{[2,4]} \psi_2$ rising and falling edges obtained from the second pulse of ψ_1 are then computed as follows: $t_{r_2}^{\mathcal{U}} = max(7, 2-4) = 7$ and $t_{f_2}^{\mathcal{U}} = min(8, 9) - 2 = 6$. Since $t_{f_2}^{\mathcal{U}} \leq t_{r_2}^{\mathcal{U}}$ they must be disregarded rather than appended to $\Gamma_r^{\mathcal{U}}$ and $\Gamma_f^{\mathcal{U}}$. This concludes that $\mathcal{U}_{[2,4]}$, were met in the interval from time $t = 1$ to $t = 3$, when the first pulse of ψ_1 must hold until the rising event on ψ_2 is true at some time step between a and b[2]. The Finite State Machine (FSM) in Figure 3, calculates the result of *Until* operator with just four states (two bits)[3] .

Algorithm 3 Until ($t_r^{\psi_1}$, $t_r^{\psi_2}$, $t_f^{\psi_1}$, $t_f^{\psi_2}$, a, b)

$t_{r_i}^{\mathcal{U}} = max(t_r^{\psi_1}, t_r^{\psi_2} - b)$
$t_{f_i}^{\mathcal{U}} = min(t_f^{\psi_1}, t_f^{\psi_2}) - a$
if $t_{r_i}^{\mathcal{U}} < 0$ **then**
  $t_{r_i}^{\mathcal{U}} = 0$
end if
if $t_{f_{i-1}}^{\mathcal{U}} < t_{r_i}^{\mathcal{U}}$ and $t_{r_i}^{\mathcal{U}} < t_{f_i}^{\mathcal{U}}$ **then**
  $\Gamma_r^{\mathcal{U}} = \Gamma_r^{\mathcal{U}} + \{t_{r_i}^{\mathcal{U}}\}$
  $\Gamma_f^{\mathcal{U}} = \Gamma_f^{\mathcal{U}} + \{t_{f_i}^{\mathcal{U}}\}$
end if
if $t_{r_i}^{\mathcal{U}} <= t_{f_{i-1}}^{\mathcal{U}}$ and $t_{f_{i-1}}^{\mathcal{U}} < t_{f_i}^{\mathcal{U}}$ **then**
  $\Gamma_f^{\mathcal{U}} = \Gamma_f^{\mathcal{U}} - \{t_{f_{i-1}}^{\mathcal{U}}\}$
  $\Gamma_f^{\mathcal{U}} = \Gamma_f^{\mathcal{U}} + \{t_{f_i}^{\mathcal{U}}\}$
end if

3.2 Event-based Approach

The second category of operators in TTL is event-based. They deal with timestamps of events (Θ set) and produce boolean signals represented by rising and falling sets (Γ_r and Γ_f).

[2]In the calculations for $\psi_1 \mathcal{U}_{[a,b]} \psi_2$ operator, we just consider the overlapped pulses on ψ_1 and ψ_2.

[3]The reader can find all proofs in https://github.com/cmlasu/tma. This link also contain a simulation software, *TMA_Testing.zip*, to evaluate TTL timing constraints.

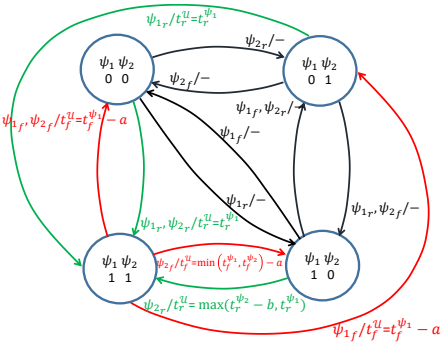

Figure 3: FSM to implement an *Until* operator. FSM captures rise and fall time of boolean signals ψ_1, ψ_2 and computes $\mathcal{U}$.

3.2.1 Simultaneity Constraint. To determine the satisfiability of the *Simultaneity* constraint, the point in time where a set of events have occurred within a time tolerance of ϵ is evaluated. Figure 4.a shows the example of three events occurring within ϵ so that the constraint is met between $\theta_{min} - b$ and $\theta_{max} - a$. We use timed-automata to evaluate this timing constraint. As Figure 4.b, if the timed-automata detects n events and ϵ duration passed after observing the first event the constraint can be calculated.

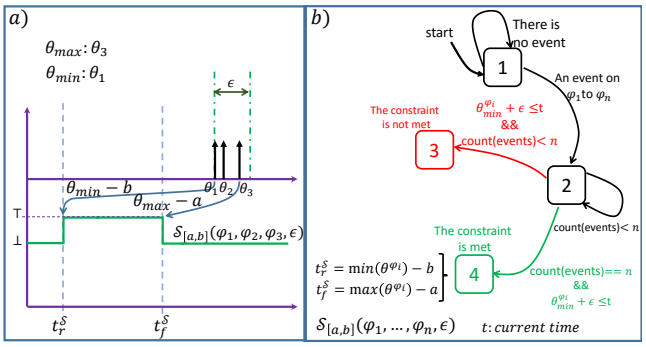

Figure 4: a) Calculation of *Simultaneity* constraint b) The timed-automata to calculate Simultaneity constraint.

3.2.2 Latency Constraint. A latency constraint specifies the time difference between the occurrence of two events. A simple example of a latency constraint is the minimum, maximum or exact time interval between two events, denoted as follows: $\mathcal{L}(\varphi_1, \varphi_2) \triangledown c$ where $\triangledown \in \{>, <, ==\}$. The test code generation takes as input two events (φ_1 and φ_2) and compares the difference between the event timestamps with a real number c. Since the signals are singletons, the sets of Θ^{φ_1} and Θ^{φ_2} each contain one element. Hence, whenever event Θ^{φ_2} is received, the latency can be calculated. The latency constraint evaluation is comprised of two steps: (1) calculating the delay Δt between two timestamps, $\theta_1^{\varphi_1}$, and $\theta_1^{\varphi_2}$, and (2) comparing Δt with c. *Latency* block, $\Delta t = \theta_1^{\varphi_2} - \theta_1^{\varphi_1}$ in comparison block if $((\Delta t \triangledown c) == \top)$ then the rising and falling edges of result are: $t_r^{\mathcal{L}} = \theta_1^{\varphi_2} - b$, $t_f^{\mathcal{L}} = \theta_1^{\varphi_1} - a$.

4 EMPIRICAL EVALUATION

In this section, we applied TMA method to monitor timing constraints in Flying Paster application and compared the required number of FFs and LUTs with the *Counters* algorithm in [11]. Also, we implemented *Globally* operator with different time intervals to show the required memory for an FPGA implementation (Table 1).

Table 1: Memory requirement on FPGA for *Globally*.

		#FFs		#LUTs	
		Jakšić [11]	TMA	Jakšić [11]	TMA
1	$\Box_{[0,100]}$	1902	1820	2981	2696
2	$\Box_{[0,200]}$	3935		5895	
3	$\Box_{[0,300]}$	7821		9314	
4	$\Box_{[100,200]}$	1891		2875	
5	$\Box_{[200,400]}$	3702		5431	
6	$\Box_{[300,600]}$	6312		9612	

4.1 Case Study: Flying Paster Application

The schematic diagram of Flying Paster application [18, 19] is shown in Figure 5. The active roll B feeds the paper and should make contact with the reserve roll A before B runs out of paper, for continuous operations. C and E are idler wheels. Sensor H measures the radius of the paper on B and whenever the radius is less than a threshold, it generates an Approaching Out of Paper (AOP) event. Then, roll A starts to rotate. On the outer side of the reserve roll paper, there is a double-sided adhesive *tape*, which can be detected by sensors F and G. To calculate the angular velocity of roll A, sensors F and J are utilized. When the velocity of the paper at the edge of A becomes the same as roll B, *match* signal is generated. Once *match* is observed, after detecting two rotations, G can detect the tape. When G detects it, idler wheel E pushes the belt to the spare roll A, such that after *tapeToContactAngle*, papers on A and B attach together by the adhesive tape and then the paper on roll A follows the path. Cutter D should cut the paper on roll B after *tapeToCutAngle*. In order to ensure that the new paper is attached properly, we should have $tapeToContactAngle < tapeToCutAngle$.

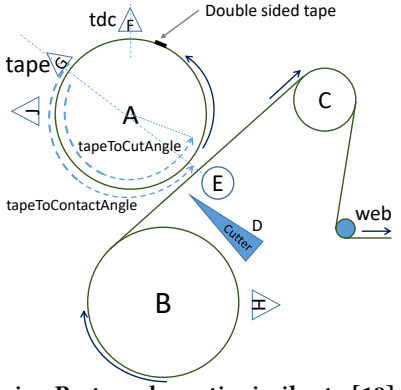

Figure 5: Flying Paster schematic similar to [18]. Active roll, B, is replace by reserve one, A, to feed the web.

To implement this application, we used two Hansen DC motors as rolls A and B. Motors are driven by an Arduino Mega2560 board to control the speed and also to generate *AOP*, *match*, *contact* and *cut* signals. On each motor, we installed a dialed disk with a drilled hole at zero degree (Figure 6). An Omron EESX970C1 sensor was installed close to each disk to detect the drilled hole and hence, measure the rotation speed of each motor. We utilized an NI-cRIO 9035 with an on-board FPGA, Xilinx Kintex-7 7K70T, containing 82,000 FFs and 41,000 LUTs with a 40 MHz clock frequency. For signal monitoring, we used a 20 kHz NI-9381 I/O module and it uses a 12-bit ADC. The pins of NI-9381 were directly connected to *photomicrosensors*, *AOP*, *match*, *cut* and *contact* signals.

Next, we express timing constraints of flying paster application based on STL. The notations for the case study variables are as follows: linear velocity (V), radius (r) and angular velocity (ω). 1) The velocity of the paper on active roll should be constant:

$$V_{active} = (r_{active} \times \omega_{active}) \pm 1\% \text{ m/s}$$

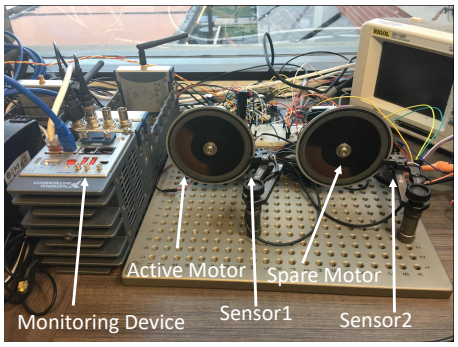

Figure 6: Implemented Flying paster comprises 2 motors and is monitored by reconfigurable data acquisition system.

$$\Box_{[t_i,t_s]}(V_{active} = r_{active} \times \omega_{active} \pm 1\%)$$

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

3) After *match*, the paper speed of the spare should remain the same as active: $V_{active} = r_{active} \times \omega_{active}$ and $V_{spare} = V_{active} \pm 1\%$

$$\Box_{[t_{match},t_{cut}]}(V_{active} = r_{active} \times \omega_{active})$$
$$\Box_{[t_{match},t_{cut}]}(V_{spare} = r_{spare} \times \omega_{spare})$$
$$\Box_{[t_{match},t_{cut}]}(V_{spare} = V_{active} \pm 1\%)$$

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

$$t_{spareTDC} - t_{matchOnSpare} < \frac{4\pi}{\omega_{spare}}$$
$$\Diamond_{[t_{match},t_{match}+\frac{4\pi}{\omega_{spare}}]}(\uparrow spareTDC)$$

5) When tape is 225 degrees after G, *contact* signal must fire.

$$t_{contact} - (t_{spareTDC} + \frac{225\ degrees}{\omega_{spare}}) < \pm 1\ ms.$$
$$\Box_{[t_{spareTDC}+\frac{225\ degrees}{\omega_{spare}+1\ ms},t_{spareTDC}+\frac{225\ degrees}{\omega_{spare}-1ms}]}(\uparrow contact)$$

6) When tape is 270 degrees after G, *cut* signal must fire.

$$t_{cut} - (t_{spareTDC} + \frac{270\ degrees}{\omega_{spare}}) < \pm 1\ ms$$
$$\Box_{[t_{spareTDC}+\frac{270\ degrees}{\omega_{spare}\pm 1\ ms},t_{spareTDC}+\frac{290\ degrees}{w_{spare}\pm 1\ ms}]}(\uparrow cut)$$

7) AOP to cut should not be more than $t_{termination}$.

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

We implemented the timing constraints of Flying Paster with three approaches (conventional, Jakšić [11] and TMA) on FPGA.

4.1.1 Temporal Logic Expression. We began with the conventional method describing the constraints in STL. We changed the future STL formulas to the past ones [11], and we represented the same timing constraint in TTL for TMA. For example, in $\Box(\uparrow AOP \Rightarrow \Diamond_{[0,t_{action}]}(\uparrow match))$, we have:

1) Conventional Method (which is pointed out as *Register Buffer* in [11]): Since rising and falling edges ($\uparrow$ and $\downarrow$) cannot be represented in STL, we express them as the way in [17]:

$$\uparrow \psi = (\psi \wedge (\neg\psi\ \mathcal{S}\ T)) \vee (\neg\psi \wedge (\psi\ \mathcal{U}\ T))$$
$$\downarrow \psi = (\neg\psi \wedge (\psi\ \mathcal{S}\ T)) \vee (\psi \wedge (\neg\psi\ \mathcal{U}\ T))$$

Therefore, the example is converted to:

$$\Box((AOP \wedge (\neg AOP\ \mathcal{S}\ T)) \vee (\neg AOP \wedge (AOP\ \mathcal{U}\ T)) \Rightarrow$$
$$\Diamond_{[0,t_{action}]}(match \wedge (\neg match\ \mathcal{S}\ T) \vee (\neg match \wedge (match\ \mathcal{U}\ T)))$$

2) Jakšić in [11] method: Future STL should be converted into past:
$\Box(\Diamond_{\{t_{action}\}} \uparrow AOP \Rightarrow \Diamond_{[0,t_{action}]}(\uparrow match))$
The edge ($\uparrow$) operator should be replaced by the equivalent constraint like the conventional method.

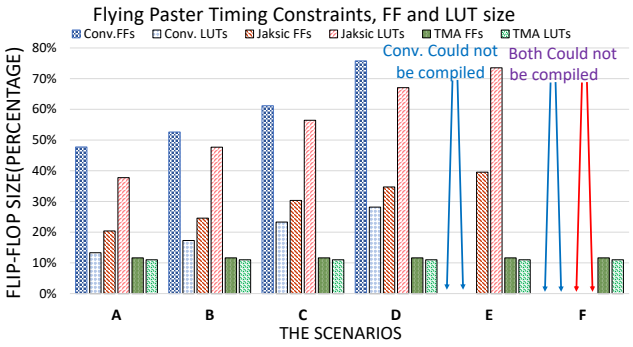

Figure 7: Comparison of FF and LUT numbers in 3 methods.

3) TMA method: Since the constraint is a latency between *AOP* and *match*, it can be easily written in TTL as:

$\mathcal{L}(\langle AOP, 2.5\ V, \nearrow\rangle, \langle match, 2.5\ V, \nearrow\rangle) \leq t_{action}$

The level threshold, 2.5 V, is the threshold to detect *true* or *false* on the boolean signal (0 V and 5 V correspond to $false$ and $true$, respectively). Next, we implemented the constraint specifications on the FPGA using the three monitoring methods.

Table 2: Six different scenarios in which the linear speed of active roll, *AOP* to *match* and time to *contact* time are varied.

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

As Figure 7 depicts, conventional and Jakšić methods required more FFs and LUTs in the case study. With increasing intervals, the FF and LUT utilization increases linearly for the Jakšić method. In $\diamondsuit_{[a,b]}\psi = \diamondsuit_{\{a\}}\diamondsuit_{[a,b-a]}$, the variability is $b - a + 1$. In contrast, TMA takes a constant amount of memory in all scenarios because it does not require retention of signal history. When a signal event is observed, the result can be deduced. Moreover, the computation part – that affects the LUT size – is minimal by reducing operators (either event-based or level-based) to simple computations.

4.2 Low variability signals

We evaluate the last timing constraint of flying paster application ($\diamondsuit_{[t_{AOP}, t_{AOP}+t_{termination}]}(\uparrow cut)$), using all three methods to see the efficiency of TMA in monitoring low variability signals for different values of $t_{termination}$ as shown in the third row of Table 2. Figure 8 compares the FF utilization based upon the conventional, Jakšić, and TMA approaches for constraint evaluation in the case study application, where TMA used the least amount of memory.

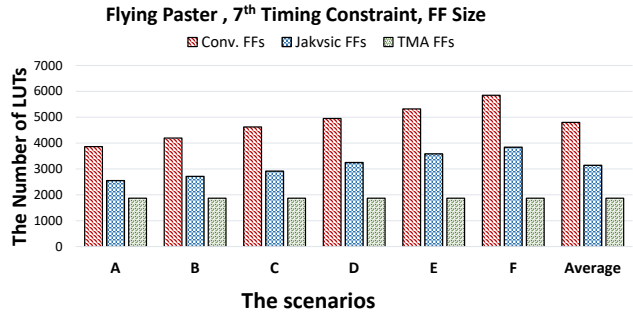

Figure 8: #FFs utilization in three methods.

5 CONCLUSION

We propose a lightweight monitoring methodology, TMA, for CPS timing constraints and demonstrated the efficiencies gained based on an initial case study. The approach utilizes signal timestamps to compute the range for a constraint, rather than processing the levels of signals, requiring data at each sample. The proposed method minimizes computation overhead compared to existing monitoring approaches. The implementation is independent of the constraint interval, allowing the memory usage to be constant for any interval. TMA is particularly geared towards monitoring constraints in TTL, which allows for the succinct description of common timing constraints in CPS, thus simplifying the description and the constraint evaluation algorithms. Future research in this area includes expansion of constraint primitives, such as duration, to fully capture and express temporal constraints in CPS.

***Disclaimer**: Certain commercial entities, equipment, or materials are identified in this document in order to describe the experimental design or to illustrate concepts. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology or the institutions of the other authors, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.*

REFERENCES

[1] Aviral Shrivastava *et al.* A Testbed to Verify the Timing Behavior of Cyber-Physical Systems. In *DAC*. ACM, 2017.
[2] Oded Maler et al. Checking Temporal Properties of Discrete, Timed and Continuous Behaviors. *LNCS*, 2008.
[3] Dejan Nickovic. *Checking Timed and Hybrid Properties: Theory and Applications*. PhD thesis, Université Joseph-Fourier-Grenoble I, 2008.
[4] Thomas Reinbacher, Matthias Függer, and Jörg Brauer. Runtime Verification of Embedded Real-time Systems. *Formal methods in system design*, 2014.
[5] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient Robust Monitoring for STL. In *CAV*. Springer, 2013.
[6] Georgios E Fainekos and George J Pappas. Robustness of Temporal Logic Specifications. In *FATES/RV*. Springer, 2006.
[7] Georgios E Fainekos and George J Pappas. Robustness of Temporal Logic Specifications for Continuous-time Signals. *Theoretical Computer Science*, 2009.
[8] Georgios E Fainekos *et al.* Verification of Automotive Control Applications using S-Taliro. In *American Control Conference (ACC)*. IEEE, 2012.
[9] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Program Monitoring with LTL in EAGLE. In *IPDPS. 18th*. IEEE, 2004.
[10] Jyotirmoy V Deshmukh et al. Robust online monitoring of signal temporal logic. In *Runtime Verification*, pages 55–70. Springer, 2015.
[11] Stefan Jakšić*et al.* From Signal Temporal Logic to FPGA Monitors. In *MEMOCODE*, 2015.
[12] Wang Mian*et al.* A Review on AC and DC Protection Equipment and Technologies: Towards Multivendor Solution. In *CIGRE INTERNATIONAL COLLOQUIUM*, 2017.
[13] Mohammadreza Mehrabian *et al.* Timestamp Temporal Logic (TTL) for Testing the Timing of Cyber-Physical Systems. In *ESWEEK*. ACM, 2017.
[14] Alexandre Donzé. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *CAV*, volume 10, pages 167–170. Springer, 2010.
[15] Yashwanth Annpureddy *et al.* S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *TACAS*. Springer, 2011.
[16] Dejan Nickovic and Oded Maler. AMT: A Property-based Monitoring Tool for Analog Systems. *Formal Modeling and Analysis of Timed Systems*, 2007.
[17] Oded Maler and Dejan Ničković. Monitoring Properties of Analog and Mixed-signal Circuits. *STTT*, 2013.
[18] Patricia Derler *et al.* Using PTIDES and Synchronized Clocks to Design Distributed Systems with Deterministic System Wide Timing. In *ISPCS*. IEEE, 2013.
[19] Drupaloge. PrintIP - Lithoman IV flying splice, last accessed nov. 2017. URL https://www.youtube.com/watch?NR=1%5C&v=wYRGiXMUzA4.