Verifying Timing Synchronization Constraints in Distributed Embedded Architectures

A. C. Rajeev   Swarup Mohalik   S. Ramesh
Global General Motors R&D, India Science Lab, Bangalore, India.
{rajeev.c, swarup.mohalik, ramesh.s}@gm.com

***Abstract*—Correct functioning of automotive embedded controllers requires hard real-time constraints on a number of system parameters. To avoid costly design iterations, these timing constraints should be verified during the design stage itself. In this paper, we describe a formal verification technique for a class of timing constraints called *timing synchronization constraints* in the recent adaptation of AUTOSAR standard (WPII-1.2 Timing Subgroup, Release 4.0). These constraints require, unlike the well studied end-to-end latency constraint, *simultaneous* analysis of multiple task/message chains or multiple data items traversing through a task/message chain. We show that they can be analyzed by model-checking with finite-state monitors. We also demonstrate this method on a case-study from the automotive domain.**

I. Introduction

Distributed embedded control systems in automobiles are extremely complex with multiple Electronic Control Units (ECUs) connected via communication buses. These systems are usually safety-critical, and for such systems, *correctness* is defined not only in terms of *functionality* but also in terms of *timeliness*. Hence, analysis of timing constraints is a crucial step in the design of distributed embedded control systems. Such timing constraints can be divided into two categories: *local* and *global*. Local constraints are restricted to the tasks/messages on a single ECU/bus. An example of a local constraint is the task/message response time constraint (*i.e.*, schedulability). On the other hand, global constraints are system level constraints. An example is the end-to-end latency constraint whose validation requires the analysis of individual chains (*i.e.*, data paths) made of tasks/messages allocated to multiple ECUs/buses.

In addition to the well studied constraints on schedulability and end-to-end latency, there are some constraints that require simultaneous analysis over multiple chains or over multiple data items traversing a chain. These constraints are referred to as *timing synchronization constraints*. Examples include difference between the sampling times of multiple sensor inputs used to produce an actuator output, and difference between the end-to-end latencies of multiple actuator outputs produced using a sensor input. Such constraints have been included in the component description template extensions in the recent adaptation of the AUTOSAR standard (WPII-1.2 Timing Subgroup, Release 4.0). These constraints are also supported by formal notations in TADL [4] and ARText [1], enabling the timing analysis of systems during the design phase itself. In this paper, we present a model-checking [6] based method to analyze such constraints. We model the system using Calendar Automaton [7], [12] (a discrete time modeling formalism) and specify the given timing constraints as properties to be satisfied by the model. SPIN [3] model-checker is used to verify if the model satisfies the properties.

Related Work: Customized analysis modules using simulation based [2] and stochastic [15] techniques can be employed to obtain average-case estimates of various timing synchronization parameters. However, as our goal is to analyze safety-critical systems, we are more interested in computing the worst-case estimates. [9], [13] present constraint solving based approaches to synthesize task parameters like periods and deadlines from a given task graph and a set of timing constraints. [10], [14] model the given system as a Timed Automaton [5] and use model-checking to estimate the worst-case values of timing parameters. Model-checking based methods can compute good worst-case estimates of various timing parameters, provided the system is modeled accurately. This is because model-checking explores all possible configurations of the model. However, the more the accuracy of the model, the less the scalability of model-checking. Hence, a challenge in designing model-checking based methods is to have a proper balance between accuracy and scalability. In this context, [14] considers non-preemptively scheduled tasks only, and [10] does the estimation using the worst-case response times of tasks. In comparison, we consider both preemptively and non-preemptively scheduled tasks, and use a Calendar Automaton model of the system for increased scalability. In addition, our method is more accurate as we use the worst-case execution times of the tasks for estimating the parameters.

The rest of this paper is organized as follows. In Section II, we present the system model and some timing synchronization constraints. We then explain our analysis techniques in Section III. Implementation details and case-studies are discussed in Section IV. We conclude the paper in Section V.

II. System Model and Timing Synchronization Constraints

We consider a distributed system S consisting of a set of sensor, controller and actuator tasks allocated to multiple ECUs and communicating through messages on multiple buses. For ease of presentation, we assume the tasks and messages to be periodic. Each ECU/bus can have its own scheduling algorithm. For ease of presentation, we assume fixed-priority preemptive scheduling for tasks and fixed-priority nonpreemptive scheduling (as in CAN bus) for messages.

We use the term *object* to denote a task/message, and the term *resource* to denote an ECU/bus. An object o_i is represented as $\langle r_i, \pi_i, O_i, P_i, E_i, In_i, Out_i \rangle$ where r_i is the resource on which o_i is scheduled, π_i is the priority, O_i is the initial offset, P_i is the period, E_i is the worst-case execution/transmission time, In_i is the set of input buffers, and Out_i is the set of output buffers. Thus, o_i triggers periodically and executes/transmits according to the scheduler of r_i. Every activation of o_i is called an *instance* of o_i. Each instance of o_i reads data items from In_i, processes/transmits them, and finally writes to Out_i. The buffers are unit-sized *registers* that are sticky and overwritable [8], [12]. We assume that i) a lower π denotes a higher priority, and ii) $E_i \leq P_i$.

The data items from the sensor tasks are processed by several controller tasks and are transmitted in several messages before the final output items are produced by the actuator tasks. This computation/transmission sequence is usually modeled as a *chain* of objects. A chain Ch of length ℓ is a sequence of objects $o_{i_1} \rightarrow \cdots \rightarrow o_{i_\ell}$ such that $\forall j \in \{1, \ldots, \ell\text{-}1\}$: $(\exists b_j : (b_j \in Out_{i_j} \wedge b_j \in In_{i_{j+1}}))$. Thus, b_j can be seen as a *data dependency edge* between o_{i_j} and $o_{i_{j+1}}$. o_{i_1} is called the source object, o_{i_ℓ} the sink object, and $o_{i_2}, \ldots, o_{i_{\ell\text{-}1}}$ the intermediate objects. A data item is said to *enter* Ch when an instance of o_{i_1} triggers and is said to *exit* Ch when an instance of o_{i_ℓ} finishes. Due to the overwritable nature of buffers, some data items read by o_{i_1} may not have corresponding output items from o_{i_ℓ}. If a data item is processed/transmitted in sequence by instances of the objects in Ch and finally results in an output, the sequence of object instances is called a *live path*. Note that there may be multiple outputs corresponding to an input, through multiple live paths; the path corresponding to the first output is called the *LIFO (Last-In-First-Out) path*. The duration of a live path (from the trigger of o_{i_1} to the finish of o_{i_ℓ}) is called the *end-to-end latency* of the path. Though there are several notions of end-to-end latency [8], we consider only *LIFO latency* in this paper.

We are interested in analyzing the following timing constraints defined on a data dependency graph:

a) LIFO Latency Constraint: A LIFO latency constraint is given by the pair $\langle Ch, \Delta_{Ch} \rangle$ where Ch is a chain and Δ_{Ch} is a non-negative integer. It specifies that the maximum end-to-end latency of any LIFO path corresponding to chain Ch should not be greater than Δ_{Ch}.

b) Input/Output Separation Constraints: The time interval between two instances of o_{i_1} that result in consecutive different output items from o_{i_ℓ} is called the input separation between those outputs. If t_1 and t_2 are the sampling times of the inputs leading to consecutive outputs, the input separation between these outputs is $IS = t_2 - t_1$. This gives a measure of the number of data items lost due to overwriting, and can be used to optimize object parameters. To illustrate this, we consider a task $\tau_1 = \langle ECU1, 0, 0, 10, 5, \emptyset, \{b1\} \rangle$ communicating with task $\tau_2 = \langle ECU2, 0, 12, 10, 5, \{b2\}, \emptyset \rangle$ through a message $m = \langle B, 0, 17, 20, 5, \{b1\}, \{b2\} \rangle$. Thus, the chain in this case is $\tau_1 \rightarrow m \rightarrow \tau_2$. For ease, we assume that τ_1, m and τ_2 are the only objects in the system.[1] Fig. 1 shows that the input separation between the consecutive output items shown in green and blue is $20ms$. Since the period of the source task τ_1 is $10ms$, this denotes that the input item read by τ_1 at $20ms$ was overwritten. As the input separation between any pair of consecutive output items from this chain is $20ms$, we may consider changing the period of τ_1 to $20ms$.

The (output) separation between two consecutive different output items is the difference between the times at which the outputs are produced. It gives an estimate of the jitter between consecutive output items [9], [10]. If t_1 and t_2 are the sampling times of the inputs leading to consecutive outputs, the output separation between these outputs is $OS = (t_2 + LIFO_2) - (t_1 + LIFO_1)$, where $LIFO_1$ and $LIFO_2$ are the LIFO latencies of the first and second live paths. Fig. 1 shows that the separation between the outputs shown in green and blue is $20ms$. For this chain, the maximum output separation is $20ms$. An input separation (resp. output separation) constraint is a pair $\langle Ch, \Delta_{in} \rangle$ (resp. $\langle Ch, \Delta_{out} \rangle$) specifying that the maximum separation between consecutive live inputs (resp. consecutive different outputs) is not greater than Δ_{in} (resp. Δ_{out}).

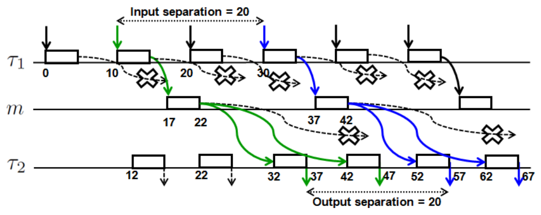

Fig. 1: Input/Output separation.

c) Actuation Constraint: When an input is used to produce outputs from multiple chains in the system, there is a constraint that the LIFO outputs should be within a prescribed time interval. In Fig. 2, an input is processed and sent to three actuators. The LIFO latency should be within 100ms, whereas the actuators should synchronize within 20ms. This is crucial in features like braking, synchronous door opening and hazard warning lights where almost simultaneous operations of multiple actuators in response to a single sensor input has to be ensured. An actuation constraint is specified as $\langle CH, \Delta_{CH}, \Delta_{act} \rangle$ where CH is a set of chains with a common source object, Δ_{CH} is the bound on the maximum LIFO latency of any chain in CH, and Δ_{act} is the bound on the

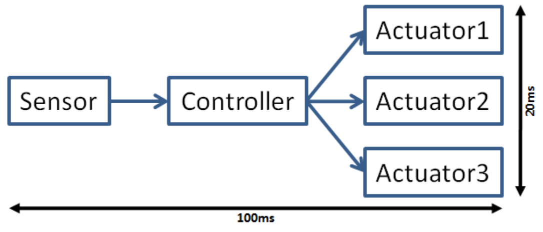

Fig. 2: Actuation.

[1]This example looks simple as we do not consider the effect of scheduling. Analyzing a real-life system with multiple tasks/messages allocated to each ECU/bus can be very complex.

maximum difference between the LIFO latencies of the chains in CH for the same input data item.

d) Correlation Constraint: When an output is dependent upon multiple inputs processed through different chains, the inputs need to be sampled within a small time interval so that the controller will have a realistic and consistent view of the environment [10], [13]. In Fig. 3, three sensors sample the inputs and a controller uses this data to compute the actuator command. Along with a LIFO latency constraint

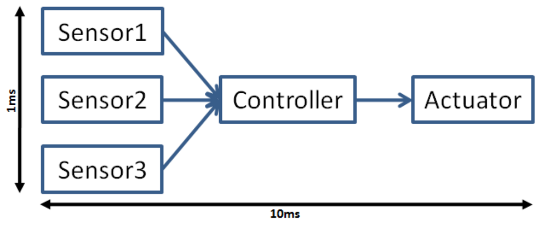

Fig. 3: Correlation.

of 10ms for each chain, there is a correlation constraint of 1ms among the sensors. This constraint is relevant in vehicle speed computation, where four wheel speed sensors sample the RPM of wheels to compute vehicle speed. A correlation constraint is specified as $\langle CH, \Delta_{CH}, \Delta_{cor} \rangle$ where CH is a set of chains with a common sink object, Δ_{CH} is the bound on the maximum LIFO latency of any chain in CH, and Δ_{cor} is the bound on the maximum difference between the sampling times of the inputs leading to an output.

III. Formal Models and Analysis Technique

In this section, we model the behaviour of a system as timed event sequences and show that the timing constraints presented in Section II can be defined as properties of finite segments of these sequences. Hence, these constraints can be verified using monitors encoding the corresponding properties.

A. System as a Calendar Automaton Model

In this section, we first recall the definition of Calendar Automaton (CA) [7] and then briefly discuss the modeling of objects using CA. We represent a CA as (Σ, V, s_0, E) where [12]:

- Σ is a finite set of events.
- V is a finite set of state-variables. An important state-variable is C, storing a calendar with events from Σ. A calendar Cal is defined as a finite set of events e_i and their relative occurrence times (called time-steps) $t_i \in \mathbb{Q}^{\geq 0}$. Two operators on Cal are: $\min(Cal)$ returning the minimum among the time-steps in Cal, and $Cal - \delta$ returning a calendar obtained by subtracting δ from all the time-steps in Cal. $\min(Cal)$ is valid only if $Cal \neq \emptyset$, and $Cal - \delta$ is valid only if $Cal \neq \emptyset$ and $\delta \leq \min(Cal)$.
- s_0 is the initial state. A state assigns appropriate values to the state-variables in V.
- E is a finite set of transitions between the states. They are of two types: (i) discrete transitions that are instantaneous. Such a transition from s to s' is enabled iff $\min(s(C)) = 0$, *i.e.*, at least one event has 0 as time-step. $s'(C)$ is obtained from $s(C)$ by deleting the processed event, adding new events and/or updating the time-steps of existing events. The remaining state-variables are updated as required. (ii) timed transitions that represent time elapse. Such a transition from s to s' is enabled iff $\min(s(C)) > 0$, *i.e.*, no event has 0 as time-step. $s'(C)$ is obtained as $s(C) - \min(s(C))$. Note that this is the maximum time elapse possible without missing any event in the calendar. As a result, the state-space of a CA model is smaller than that of a similar model based on continuous time formalisms such as Timed Automaton [5] or Hybrid Automaton [11]. The remaining state-variables are updated as required.

Next, we recall the behaviour of preemptively scheduled tasks and non-preemptively scheduled messages [12]. The state diagram of a preemptively scheduled task τ_i is shown in Fig. 4. On a period expiry (shown as $trigger_i$), it moves to

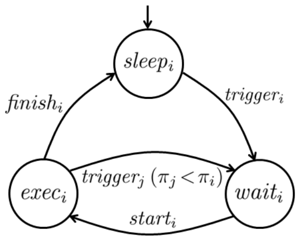
$$\begin{tikzcd} & sleep_i \arrow[dr, bend left, "trigger_i"] & \\ exec_i \arrow[ur, bend left, "finish_i"] \arrow[rr, bend left, "trigger_j\ (\pi_j < \pi_i)"] & & wait_i \arrow[ll, bend left, "start_i"] \end{tikzcd}$$

Fig. 4: State diagram of τ_i.

$wait_i$ denoting that it is waiting in the ready queue. When it is selected for execution (shown as $start_i$), it reads the input buffers and moves to $exec_i$. The time interval from $trigger_i$ to $start_i$ is called the *start time* of this instance of τ_i. From $exec_i$, it may be preempted (shown as $trigger_j$ $(\pi_j < \pi_i)$) to $wait_i$ by a higher priority task τ_j; when it restarts after preemption, it again takes the $start_i$ edge to $exec_i$ but does not read any buffer. Once it finishes execution (shown as $finish_i$), it writes to the output buffers and moves to $sleep_i$. The interval from $trigger_i$ to $finish_i$ is called the *response time* of this instance of τ_i. Note that the event sequence from τ_i is of the form $(trigger_i\ start_i{}^+\ finish_i)^\omega$. The event sequence from a non-preemptively scheduled message m_i is of the form $(trigger_i\ start_i\ finish_i)^\omega$, as its behaviour is similar but with no preemptions.

The CA model of a system S is obtained as the composition of the CA models of the tasks and messages in S. These models can be easily derived from the ones in [12]; we omit them here due to space limitation. As the offsets, periods and execution times of the tasks/messages in S are finite, time-steps in the calendar are finite. Also, the maximum size of the calendar is 2 times the number of tasks/messages in S [12]. Hence, we have (on the lines of [12]):

Proposition 1: The CA model of a system S is finite-state.

B. Constraints as Properties of Finite Timed-Event Sequences

In this section, we show the formulation of various timing constraints as properties of a finite sequence of events and time-steps. Let $E = \bigcup_m \{trigger_m, start_m, finish_m\}$ be the set of events from the objects o_ms in a system S. The behaviour of S can be seen as a set of runs; each run ρ being a pair $\langle \sigma, \alpha \rangle$ where i) $\sigma \in E^{\omega}$ is a sequence of events whose projection onto the set of events from a task τ_k is of the form $(\mathit{trigger}_k\ \mathit{start}_k{}^{+}\ \mathit{finish}_k)^{\omega}$ and onto the set of events from a message m_k is of the form $(\mathit{trigger}_k\ \mathit{start}_k\ \mathit{finish}_k)^{\omega}$, and ii) $\alpha \in \mathbb{Q}^{\omega}$ is a sequence of time-steps. The interpretation of ρ is that $\sigma[i+1]$ occurs $\alpha[i]$ time units after $\sigma[i]$. We use the notation $\rho(i,j)$ to denote the finite segment $\langle \sigma(i,j), \alpha(i,j) \rangle$. We define the duration of $\rho(i,j)$ as $\mathit{duration}(\rho(i,j)) \triangleq \sum_{h=i}^{j-1} \alpha[h]$.

Given a run ρ, we define three types of causal order relations between the indices as follows:

1) $i <_k^{in} j$ if $\sigma[i] = \mathit{trigger}_k$ and $\sigma[j] = \mathit{start}_k$ and NO_FINISH(k,i,j).
2) $i <_k^{k} j$ if $\sigma[i] = \mathit{start}_k$ and $\sigma[j] = \mathit{finish}_k$ and NO_FINISH(k,i,j).
3) $i <_l^{k} j$ if $\sigma[i] = \mathit{finish}_k$ and $\sigma[j] = \mathit{start}_l$ and NO_FINISH(k,i,j).

where NO_FINISH$(k,i,j) \equiv ((i<j) \wedge (\forall h \in \{i+1,\ldots,j-1\} : \sigma[h] \neq \mathit{finish}_k))$. Given a chain $Ch = o_{i_1} \rightarrow \cdots \rightarrow o_{i_\ell}$ in S, we define $\ll_{Ch}$ as $<_{i_1}^{in} \circ <_{i_1}^{i_1} \circ <_{i_2}^{i_1} \circ <_{i_2}^{i_2} \circ \cdots \circ <_{i_\ell}^{i_\ell}$, where $\circ$ denotes the standard composition operator. An illustration of $\ll_{Ch}$ and the corresponding causal order relations is given in Fig. 5. Here, the chain being considered is $Ch = \tau_1 \rightarrow \tau_2 \rightarrow \tau_3$. The shaded portion in each task instance shows the start time of that instance.

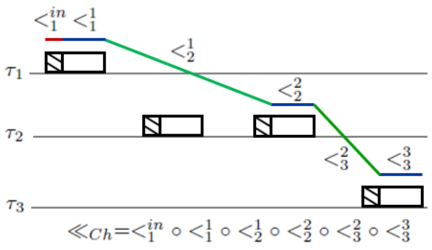
$<_1^{in}$ $<_1^1$
$<_2^1$
$<_2^2$
$<_3^2$
$<_3^3$
τ_1
τ_2
τ_3
$\ll_{Ch} = <_1^{in} \circ <_1^1 \circ <_2^1 \circ <_2^2 \circ <_3^2 \circ <_3^3$

Fig. 5: Relation $\ll_{Ch}$.

a) LIFO Latency: A segment $\rho(i,j)$ is Ch-live if $\sigma[i] = \mathit{trigger}_{i_1}$ and $\sigma[j] = \mathit{finish}_{i_\ell}$ and $i \ll_{Ch} j$. A Ch-live segment $\rho(i,j)$ is said to be Ch-LIFO if $\forall i < j' < j : \rho(i,j')$ is not Ch-live. The latency of a Ch-live segment $\rho(i,j)$ is given by $\mathit{duration}(\rho(i,j))$. System S satisfies a LIFO latency constraint $\langle Ch, \Delta_{Ch} \rangle$ if for every Ch-LIFO segment $\rho(i,j)$ in the behaviour of S, $\mathit{duration}(\rho(i,j)) \leq \Delta_{Ch}$.

b) Input/Output Separation: A segment $\rho(i,j)$ is Ch-separation-checkable if $\exists i', j' : ((i < i', j' < j)$ and $(\rho(i,j')$ is a Ch-LIFO segment) and $(\rho(i',j)$ is a Ch-LIFO segment) and $(\forall j'' \in \{j'+1,\ldots,j-1\} : ((\sigma[j''] = \mathit{finish}_{i_\ell}) \implies (\rho(i,j'')$ is a Ch-live segment)))). We define $\delta_{in}(\rho(i,j))$ as $\mathit{duration}(\rho(i,i'))$ and $\delta_{out}(\rho(i,j))$ as $\mathit{duration}(\rho(j',j))$. System S satisfies an input (resp. output) separation constraint $\langle Ch, \Delta_{in} \rangle$ (resp. $\langle Ch, \Delta_{out} \rangle$) if for every Ch-separation-checkable segment $\rho(i,j)$ in the behaviour of S, $\delta_{in}(\rho(i,j)) \leq \Delta_{in}$ (resp. $\delta_{out}(\rho(i,j)) \leq \Delta_{out}$).

c) Actuation: Given a set $CH = \{Ch_1, \ldots, Ch_n\}$ of n chains with a common source object, a segment $\rho(i,j)$ is CH-actuation-checkable if $\exists j_1, \ldots, j_n : ((i < j_1, \ldots, j_n \leq j)$ and $(\exists m \in \{1,\ldots,n\} : j_m = j)$ and $(\forall m \in \{1,\ldots,n\} : \rho(i, j_m)$ is a Ch_m-LIFO segment)). The end-to-end latency of the longest among these Ch-LIFO segments is $\delta_{CH}(\rho(i,j)) \triangleq \mathit{duration}(\rho(i,j))$, and the actuation interval is $\delta_{act}(\rho(i,j)) \triangleq \mathit{duration}(\rho(\min(j_1,\ldots,j_n),j))$. System S satisfies an actuation constraint $\langle CH, \Delta_{CH}, \Delta_{act} \rangle$ if for every CH-actuation-checkable segment $\rho(i,j)$ in the behaviour of S, $\delta_{CH}(\rho(i,j)) \leq \Delta_{CH}$ and $\delta_{act}(\rho(i,j)) \leq \Delta_{act}$. Refer Fig. 6 for an illustration of a CH-actuation-checkable segment.

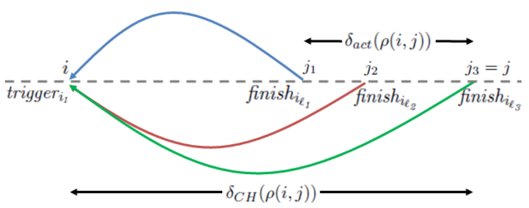
$\delta_{act}(\rho(i,j))$
i
j_1
j_2
$j_3 = j$
$\mathit{trigger}_{i_1}$
$\mathit{finish}_{i_{\ell_1}}$
$\mathit{finish}_{i_{\ell_2}}$
$\mathit{finish}_{i_{\ell_3}}$
$\delta_{CH}(\rho(i,j))$

Fig. 6: A CH-actuation-checkable segment.

d) Correlation: Given a set $CH = \{Ch_1, \ldots, Ch_n\}$ of n chains with a common sink object, a segment $\rho(i,j)$ is CH-correlation-checkable if $\exists i_1, \ldots, i_n : ((i \leq i_1, \ldots, i_n < j)$ and $(\exists m \in \{1,\ldots,n\} : i_m = i)$ and $(\forall m \in \{1,\ldots,n\} : \rho(i_m, j)$ is a Ch_m-LIFO segment)). The end-to-end latency of the longest among these Ch-LIFO segments is $\delta_{CH}(\rho(i,j)) \triangleq \mathit{duration}(\rho(i,j))$, and the correlation interval is $\delta_{cor}(\rho(i,j)) \triangleq \mathit{duration}(\rho(i,\max(i_1,\ldots,i_n)))$. System S satisfies a correlation constraint $\langle CH, \Delta_{CH}, \Delta_{cor} \rangle$ if for every CH-correlation-checkable segment $\rho(i,j)$ in the behaviour of S, $\delta_{CH}(\rho(i,j)) \leq \Delta_{CH}$ and $\delta_{cor}(\rho(i,j)) \leq \Delta_{cor}$.

C. Monitors for Checking the Properties

To check whether a system S satisfies a timing constraint, we have to check whether the constraint is satisfied by all relevant segments in all the runs of S. In this section, we discuss about the monitors used to detect the relevant segments in a run and to verify the constraints.

Monitor 1 detects all CH-actuation-checkable segments in a run and identifies whether any of those segments violates the given actuation constraint. Monitors for other constraints can be obtained similarly; we omit their descriptions due to space limitation. Monitor 1 is based on the actuation constraint $\langle CH = \{Ch_1, \ldots, Ch_n\}, \Delta_{CH}, \Delta_{act} \rangle$. It monitors a run ρ of the CA model of the system, and processes the events (corresponding to discrete transitions) and time-steps (corresponding to timed transitions) in ρ. For this, it stores a finite history $h = \langle \sigma_h, \alpha_h \rangle$ of ρ. If h is non-empty, $\sigma_h[0]$ will be a $\mathit{trigger}_{i_1}$ event from the common source object o_{i_1}. The monitor also stores the indices of $\mathit{finish}_{i_{\ell_j}}$s and the indices of the corresponding $\mathit{trigger}_{i_1}$s – these are stored in $TF = \{(t,f), (t',f'), \ldots\}$ as ($\mathit{trigger}$ index, finish index) pairs. When a time-step e arrives, the monitor updates the last entry in α_h (lines 4-6). Note that all time-steps before the first event are ignored. When an event e arrives, the monitor updates σ_h and α_h. It then checks whether e is the finish event from the sink object of any chain (line 10). If yes, it identifies the index of the corresponding $\mathit{trigger}$ event from the source object (line 12). The check on line 13 confirms that the t computed on line 12 corresponds to a LIFO segment. After this, the pair (t, f) is inserted into TF (line 14). If TF contains indices of $\mathit{finish}_{i_{\ell_m}}$ events from all the sink objects $o_{i_{\ell_m}}$s such that they have a common corresponding $\mathit{trigger}_{i_1}$ event at index k, $h(k, \mathit{length}(h)-1)$ is a CH-actuation-checkable segment (line 15). So h is resized (line 16), and k is subtracted from all the ts and fs in TF so that TF is updated with respect to the resized h (line 17). If any t or f in the updated TF is less than 0, that pair is deleted (line 18). If the duration of the resized h is greater than Δ_{CH}, 'LIFO Latency Constraint Violated' error is raised. f_1 computed on line 22 is the minimum among the f''s satisfying the condition on line 15. If $\mathit{duration}(h(f_1, \mathit{length}(h)-1))$ is greater than Δ_{act}, 'Actuation Constraint Violated' error is raised.

Monitor 1: Monitor for Actuation Constraint

```
1  $h \leftarrow \langle\rangle$
2  $TF \leftarrow \emptyset$
3  while $\mathtt{true}$ do
4      if $e$ is a time-step then
5          if $\mathit{length}(h) > 0$ then
6              $\alpha_h[\mathit{length}(h)-1] \leftarrow e$
7      if $e$ is an event then
8          $\sigma_h \leftarrow \sigma_h \cdot e$
9          $\alpha_h \leftarrow \alpha_h \cdot \infty$
10         if $e$ is $\mathit{finish}_{i_{\ell_j}}$ then
11             $f \leftarrow \mathit{length}(h)-1$
12             $t \leftarrow \mathit{source}(Ch_j, h, f)$ /* Algorithm 1 */
13             if $(t \neq -1) \wedge (\nexists(t'', f'') \in TF : \sigma_h[f''] = \mathit{finish}_{i_{\ell_j}} \wedge t'' = t)$ then
14                 $TF \leftarrow TF \cup \{(t, f)\}$
15                 if $\exists k : (\forall m \in \{1, \ldots, n\} : (\exists(t'', f'') \in TF : (\sigma_h[f''] = \mathit{finish}_{i_{\ell_m}} \wedge t'' = k)))$ then
                       /* $h(k, \mathit{length}(h)-1)$ is CH-actuation-checkable */
16                     $h \leftarrow h(k, \mathit{length}(h)-1)$
17                     $TF \leftarrow TF - k$
18                     $TF \leftarrow TF \setminus \{(t'', f'') \mid t'' < 0 \vee f'' < 0\}$
19                     if $\mathit{duration}(h) > \Delta_{CH}$ then
20                         Print 'LIFO Latency Constraint Violated'
21                         return
22                     $f_1 \leftarrow \min\{f'' \mid \forall m \in \{1, \ldots, n\} : (\exists(t'', f'') \in TF : (\sigma_h[f''] = \mathit{finish}_{i_{\ell_m}} \wedge t'' = 0))\}$
23                     if $\mathit{duration}(h(f_1, \mathit{length}(h)-1)) > \Delta_{act}$ then
24                         Print 'Actuation Constraint Violated'
25                         return
```

We also present Algorithm 1 that is used by the monitors to identify Ch-live segments. It takes a chain $Ch = o_{i_1} \to \cdots \to o_{i_\ell}$, a finite segment h of a run, and the index k of a finish_{i_ℓ} event in h as inputs, and returns the index of the corresponding $\mathit{trigger}_{i_1}$ event in h. If there is no such index, it returns -1.

As Monitor 1 stores only a finite segment of a run, h and TF are of finite size. Hence,

Proposition 2: Monitor 1 is finite-state.

Due to Proposition 1, a model-checker can be used to generate all the runs of the CA model of a system S. Thus, by model-checking the composition of Monitor 1 and the CA model of S, we can identify whether S violates the actuation constraint:

Proposition 3: Given a system S and an actuation constraint AC, Monitor 1 reports violation iff S does not satisfy AC.

Algorithm 1: source(Ch, h, k)

```
1  $idx \leftarrow k$
2  for $j \in \{\ell, \ldots, 1\}$ do
3      while $idx \geq 0 \wedge \sigma_h[idx] \neq \mathit{start}_{i_j}$ do
4          $idx \leftarrow idx - 1$
5      while $idx \geq 0 \wedge ((j > 1 \wedge \sigma_h[idx] \neq \mathit{finish}_{i_{j-1}}) \vee (j = 1 \wedge \sigma_h[idx] \neq \mathit{trigger}_{i_j}))$ do
6          $idx \leftarrow idx - 1$
7      if $idx < 0$ then
8          return $-1$
9  return $idx$
```

IV. Implementation and Case-studies

The analysis techniques discussed in Section III have been implemented as a tool that takes as input an excel sheet containing the system specification and the chains/graphs to be analyzed, and outputs an excel sheet containing the values of the required timing parameters.

The case-studies are based on a dual-core ECU (core C-1 with 13 tasks and core C-2 with 12 tasks) implementing a Collision Preparation System [12]. Table I shows the task parameters. It also shows the input/output buffers of the relevant tasks. T_3 and T_{22} read the input data items delivered on CAN1 and CAN2 buses, and T_6 generates the output data items transmitted on CAN3 bus. T_8, T_{25}, T_{24} and T_4 are the IPC tasks handling the data transfer between C-1 and C-2. The remaining tasks process intermediate data [12]. We have computed the input/output separations of 40 chains between T_3 and T_6, and of 12 chains between T_{22} and T_6. 5 of these chains are shown in Table II. Our method took ≈ 0.13s time and ≈ 330Mb memory for each chain, on a Windows XP laptop with Intel Centrino Duo processor (2.2 GHz) and 2GB RAM.

TABLE II: 5 chains from Table I.

No.	Chain	IS (ms)	OS (ms)
1	$T_3 \to T_8 \to T_2 \to T_5 \to T_1 \to T_{13} \to T_6$	100	100
2	$T_3 \to T_{11} \to T_{12} \to T_8 \to T_{25} \to T_{15} \to T_{14} \to T_{17} \to T_{24} \to T_4 \to T_6$	50	50
3	$T_3 \to T_{11} \to T_{12} \to T_2 \to T_5 \to T_8 \to T_{25} \to T_{15} \to T_{21} \to T_{23} \to T_{18} \to T_{24} \to T_4 \to T_6$	50	50
4	$T_{22} \to T_{14} \to T_{17} \to T_{24} \to T_4 \to T_6$	50	50
5	$T_{22} \to T_{14} \to T_{17} \to T_{20} \to T_{24} \to T_4 \to T_7 \to T_8 \to T_2 \to T_5 \to T_1 \to T_{13} \to T_6$	100	100

We also carried out a variant of actuation constraint analysis between chain $T_2 \to T_5 \to T_{13} \to T_6$ (generating a command signal) and chain $T_{14} \to T_{17} \to T_{20} \to T_{24} \to T_4 \to T_6$ (generating the confirmation signal). The constraint in this case is that the confirmation signal should arrive within $10ms$ of

TABLE I: Tasks on two cores of an ECU.

Core	τ_i	π_i	O_i (ms)	P_i (ms)	E_i (ms)	In_i	Out_i
C-1	T_1	1	28	100	0.013	{b5_1}	{b1_3, b1_13}
	T_2	2	1	50	3.644	{b8_2, b12_2}	{b2_5}
	T_3	3	3	10	0.01	{b1_3, b6_3, CAN1}	{b3_8, b3_11}
	T_4	4	2	10	0.01	{b24_4}	{b4_6, b4_7}
	T_5	5	1	50	2.54	{b2_5}	{b5_1, b5_8, b5_13}
	T_6	6	0	10	0.02	{b4_6, b13_6}	{b6_3, CAN3}
	T_7	7	1	10	0.01	{b4_7}	{b7_8}
	T_8	8	3	10	0.01	{b3_8, b5_8, b7_8, b12_8}	{b8_2, b8_25}
	T_9	9	1	100	0.025	-	-
	T_{10}	10	2	50	0.025	-	-
	T_{11}	11	3	10	0.115	{b3_11}	{b11_12}
	T_{12}	12	2	10	0.258	{b11_12}	{b12_2, b12_8, b12_13}
	T_{13}	13	1	50	0.167	{b1_13, b5_13, b12_13}	{b13_6}
C-2	T_{14}	1	1	50	0.167	{b15_14, b22_14}	{b14_17}
	T_{15}	2	1	10	0.115	{b25_15}	{b15_14, b15_21}
	T_{16}	3	1	50	0.148	-	-
	T_{17}	4	1	50	0.11	{b14_17, b25_17}	{b17_18, b17_20, b17_24}
	T_{18}	5	1	10	0.01	{b17_18, b23_18}	{b18_24}
	T_{19}	6	0	100	0.017	-	-
	T_{20}	7	1	50	0.167	{b17_20, b21_20}	{b20_24}
	T_{21}	8	2	10	0.377	{b15_21}	{b21_20, b21_23}
	T_{22}	9	1	10	0.01	{CAN2}	{b22_14, b22_24}
	T_{23}	10	2	10	0.11	{b21_23, b25_23}	{b23_18, b23_24}
	T_{24}	11	1	10	0.01	{b17_24, b18_24, b20_24, b22_24, b23_24}	{b24_4}
	T_{25}	12	3	10	0.01	{b8_25}	{b25_15, b25_17, b25_23}

the corresponding command signal. Note that this constraint is stronger than the vanilla actuation constraint, as it additionally specifies an order among the outputs from different chains. This extra check was easily incorporated into Monitor 1. Thus, we could verify that the following issues do not occur:

- Command signal is lost due to overwriting, but confirmation signal is not.
- Confirmation signal is lost due to overwriting, but command signal is not.
- Confirmation signal arrives before the corresponding command signal.
- Confirmation signal arrives $10ms$ after the corresponding command signal.
- An old confirmation signal arrives after a new (unrelated) command signal.

V. Conclusion

Schedulability analysis and end-to-end latency analysis are well-studied in the literature. However, timing synchronization problems like input/output separation, actuation and correlation have not gained as much attention. In this paper, we have presented the details on formalizing these constraints and verifying them via model-checking a Calendar Automaton model of the system. Due to the complex interactions among the tasks and messages in a system, one can ask additional questions like ‘Does every input lead to outputs satisfying the actuation constraint?’, ‘Are the inputs at every intermediate task correlated?’, ‘What is the synchronization restriction on duplicate outputs from different chains?’, *etc.* We believe that most of these questions can be formulated as described in this paper and can be verified via model-checking. As future work, scalability and accuracy of our method are to be compared with constraint programming based methods. Such methods are usually less expressive than model-checking based methods, but for specific timing synchronization problems, one may be able to find suitable abstractions so that the resulting formulations can verify the system within a tolerable approximation.

References

[1] ARText. https://www.artop.org/artext/

[2] MLDesigner. http://www.mldesigner.com.

[3] SPIN Model-checker. http://www.spinroot.com.

[4] TIMMO-2-USE. http://www.timmo-2-use.org/timmo/pdf/TIMMO_Brochure.pdf

[5] R. Alur, and D. L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[7] B. Dutertre, and M. Sorea. Modeling and Verification of a Fault-Tolerant Real-Time Startup Protocol using Calendar Automata. In *FORMATS/FTRTFT*, pp. 199–214, 2004.

[8] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *CRTS*, 2008.

[9] R. Gerber, S. Hong, and M. Saksena. Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Processes. *IEEE TSE*, 21(7):579–592, 1995.

[10] Z. Gu. Timing Analysis of Distributed End-to-End Task Graphs with Model-Checking. In *EUC*, pp. 214–223, 2005.

[11] T. A. Henzinger. The Theory of Hybrid Automata. In *LICS*, pp. 278–292, 1996.

[12] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh. Schedulability and End-to-end Latency in Distributed ECU Networks: Formal Modeling and Precise Estimation. In *EMSOFT*, pp. 129–138, 2010.

[13] M. Saksena, and S. Hong. Resource Conscious Design of Distributed Real-Time Systems: An End-to-End Approach. In *ICECCS*, pp. 306–313, 1996.

[14] Anders Wall, K. Sandström, J. Mäki-Turja, C. Norström, and W. Yi. Verifying Temporal Constraints on Data in Multi-rate Transactions using Timed Automata. In *RTCSA*, pp. 263–270, 2000.

[15] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli. Stochastic Analysis of CAN-Based Real-Time Automotive Systems. *IEEE TII*, 5(4):388–401, 2009.