Chapter 6
Timing Contracts for Multi-Core Embedded Control Systems

M. Al Khatib, A. Girard and T. Dang

Abstract In physical dynamical systems equipped with embedded controllers, timing contracts specify the time instants at which certain operations are performed such as sampling, computation, and actuation. In the first part of this chapter, we present a class of timing contracts specifying bounds on the sampling-to-actuation delay and on the sampling period. We then review existing techniques that can handle the problem of stability verification: given models of the physical plant and of the controller and a timing contract, we verify that the resulting dynamical system is stable. In the second part of the chapter, we consider the scheduling problem of embedded controllers on a multiple core computational platform: given a set of controllers, each of which is subject to a timing contract, we synthesize a dynamic scheduling policy, which guarantees that each timing contract is satisfied and that each of the shared computational resources is allocated to at most one embedded controller at any time. The approach is based on a timed game formulation whose solution provides a suitable schedule.

6.1 Introduction

Physical systems equipped with embedded controllers have a long history (aircrafts, cars, robots, etc.) and are becoming ever more complex and pervasive (smart buildings, autonomous vehicles, etc.). Efficient usage of the computational resources in embedded control systems while providing formal guarantees of stability requires a

M. Al Khatib · A. Girard (✉)
Laboratoire des Signaux et Systèmes (L2S), CNRS, CentraleSupélec, Université Paris-Sud, Université Paris-Saclay, 91192 Gif-sur-Yvette, France
e-mail: antoine.girard@l2s.centralesupelec.fr

M. Al Khatib
e-mail: mohammad.alkhatib@l2s.centralesupelec.fr

T. Dang
Verimag, Université Grenoble-Alpes-CNRS, 38000 Grenoble, France
e-mail: thao.dang@imag.fr

S. Tarbouriech et al. (eds.), *Control Subject to Computational and Communication Constraints*, Lecture Notes in Control and Information Sciences 475, https://doi.org/10.1007/978-3-319-78449-6_6

profound understanding of the interaction between their computational and physical components. Models faithfully describing such cyber-physical integration combine continuous as well as discrete dynamics whereby the former originates from the behavior of the physical systems whereas the latter results from the behavior of components like sensors, actuators, and other computation and communication resources. One direction in modeling timing of events (sampling, computation, and actuation) in this orchestration is given by timing contracts [11]. Under such contracts, the control engineers are responsible for designing a control law that is robust to all possible timing variations specified in the contract while the software engineers can focus on implementing the proposed control law so as to satisfy the timing contract. Consequently, we propose techniques that are useful within this framework.

In the first part of this chapter, we present a class of parameterized timing contracts specifying bounds on the sampling-to-actuation delay and on the sampling period. We then review existing techniques [6, 9, 10] that can handle the problem of stability verification: given models of the physical plant and of the controller and a timing contract, verify that the resulting dynamical system is stable. In this context, we briefly present our approach [2, 3], based on the notion of reachable set, and which is built on efficient over-approximation algorithms developed over the past decade (see e.g., [19]).

In the second part, we consider the scheduling problem of embedded controllers on a multiple core computational platform. Given a set of controllers, each of which is subject to a timing contract, we synthesize a dynamic scheduling policy, which guarantees that each timing contract is satisfied and that each of the shared computational resources is allocated to at most one embedded controller at any time. The approach is based on a timed game formulation [5] whose solution provides a suitable schedule. Results on this second problem partially appear, in the case of single core computational platforms in [4].

Notation

Let $\mathbb{R}, \mathbb{R}_0^+, \mathbb{R}^+, \mathbb{N}, \mathbb{N}^+$ denote the sets of reals, nonnegative reals, positive reals, nonnegative integers, and positive integers, respectively. For $I \subseteq \mathbb{R}_0^+$, let $\mathbb{N}_I = \mathbb{N} \cap I$. Finally, for a set S, we denote the set of all subsets of S by 2^S.

6.2 Problem Formulation

The model considered in the chapter is represented by the block diagram given by Fig. 6.1. Typically, the plant's state z flows under continuous dynamics. Then, at each sampling instant t_k^s, $k \in \mathbb{N}$, the plant's state is sampled by the sampler and is passed through a network to the controller. The latter computes the control input u based on $z(t_k^s)$ and updates the plant's input at instant t_k^a, $k \in \mathbb{N}$. The plant's input is then held constant by a zero-order hold until the next update arrives via the network.

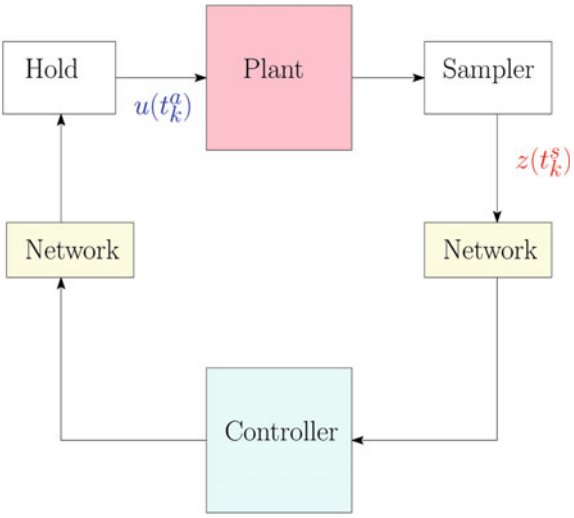

Fig. 6.1 Block diagram of a sampled-data system

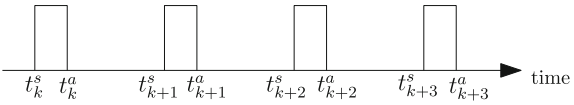

Fig. 6.2 Periodic sampled-data systems

The following model allows us to capture the continuous dynamics of the plant as well as the discrete dynamics, introduced by the sampler and the zero-order hold.

$$\dot{z}(t) = Az(t) + Bu(t), \quad \forall t \in \mathbb{R}_0^+ \tag{6.1a}$$

$$u(t) = Kz(t_k^s), \qquad t_k^a < t \leq t_{k+1}^a \tag{6.1b}$$

where $z(t) \in \mathbb{R}^p$ is the state of the system, $u(t) \in \mathbb{R}^m$ is the control input, the matrices $A \in \mathbb{R}^{p \times p}$, $B \in \mathbb{R}^{p \times m}$, $K \in \mathbb{R}^{m \times p}$, and $k \in \mathbb{N}$. In addition, it is assumed that K is designed such that the matrix $A + BK$ is Hurwitz and that for all $t \in [0, t_0^a]$, $u(t) = 0$.

Traditionally, controllers assume that sampling is performed periodically and that actuation is performed with as little latency as possible. This scenario is shown in Fig. 6.2 where the sampling instants are given by $t_k^s = kh$ for all $k \in \mathbb{N}$ and h being the sampling period. However outside this ideal case, variations in the timing sampling and actuation instants can be captured by *timing contracts*, which make it possible to take into account the temporal nondeterminism of the sequences of sampling and actuation instants $(t_k^s)_{k \in \mathbb{N}}$ and $(t_k^a)_{k \in \mathbb{N}}$.

We assume that the sequences of sampling and actuation instants $(t_k^s)_{k \in \mathbb{N}}$ and $(t_k^a)_{k \in \mathbb{N}}$ satisfy a *timing contract* $\theta(\underline{\tau}, \overline{\tau}, \underline{h}, \overline{h})$ given by

$$\begin{aligned} 0 &\leq t_0^s, \\ t_k^s \leq t_k^a &\leq t_{k+1}^s, && \forall k \in \mathbb{N} \\ \tau_k = t_k^a - t_k^s &\in [\underline{\tau}, \overline{\tau}], && \forall k \in \mathbb{N} \\ h_k = t_{k+1}^s - t_k^s &\in [\underline{h}, \overline{h}], && \forall k \in \mathbb{N} \end{aligned} \tag{6.2}$$

where $\underline{\tau} \in \mathbb{R}_0^+$, $\overline{\tau} \in \mathbb{R}_0^+$, $\underline{h} \in \mathbb{R}^+$, and $\overline{h} \in \mathbb{R}^+$ provide bounds on the sampling-to-actuation delays (which include time for computation of the control law) and sampling periods. Note that we impose $\underline{h} \neq 0$ to prevent Zeno behavior. Moreover, these parameters must belong to the following set $\mathscr{C}$ so that the time intervals given in (6.2) are always non-empty and it is always possible to choose $t_{k+1}^s \geq t_k^a$:

$$\mathscr{C} = \left\{ (\underline{\tau}, \overline{\tau}, \underline{h}, \overline{h}) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \times \mathbb{R}^+ \times \mathbb{R}^+ : \underline{\tau} \leq \overline{\tau} \leq \overline{h}, \underline{h} \leq \overline{h} \right\}.$$

Contract (6.2) is a general timing contract which includes or over-approximates the different contracts introduced in [11]. Their relation to the timing contract (6.2) is described as follows:

1. **ZET Contract**: The Zero Execution Time contract is given by (6.2) with $\underline{\tau} = \overline{\tau} = 0$ and $\underline{h} = \overline{h} = h \in \mathbb{R}^+$. In other words, the contract states that the sampling and actuation instants are periodic and simultaneous such that $t_k^s = t_k^a = kh$ for $k \in \mathbb{N}$. As mentioned in [11], this contract is hardly achievable in practice since computation always takes time in between the sampling and actuation instants.
2. **LET Contract**: The Logical Execution Time contract is given by (6.2) with $\underline{\tau} = \overline{\tau} = \underline{h} = \overline{h} = h \in \mathbb{R}^+$. The contract states that the sampling and actuation instants are periodic such that $t_0^s = 0$ and $t_k^s = t_{k-1}^a = kh$ for $k \in \mathbb{N}^+$.
3. **DET Contract**: The Deadline Execution Time contract is given by (6.2) with $\underline{\tau} = 0$ and $\underline{h} = \overline{h} = h \in \mathbb{R}^+$. The contract states that the sampling instants are periodic, or $t_k^s = kh$ for $k \in \mathbb{N}$, and actuation instants are at some point t_k^a in the interval $[t_k^s, t_k^s + \overline{\tau}]$, with $\overline{\tau} \leq h$.
4. **TOL Contract**: The Timing Tolerance contract is defined by a nominal sampling period $h \in \mathbb{R}^+$, nominal sampling-to-actuation delay $\tau \in \mathbb{R}_0^+$, and two jitters $J^h, J^\delta \in \mathbb{R}_0^+$ with $J^\tau \leq \tau$ and $J^h + J^\tau + \tau \leq h$, such that $t_k^s \in [kh, kh + J^h]$ and $t_k^a \in [t_k^s + \tau - J^\tau, t_k^s + \tau + J^\tau]$, for $k \in \mathbb{N}$ (refer to Fig. 6.3). We cannot exactly model this contract using (6.2). However, we can over-approximate it using (6.2) with $\underline{\tau} = \tau - J^\tau$, $\overline{\tau} = \tau + J^\tau$, $\underline{h} = h - J^h$, and $\overline{h} = h + J^h$.

In the following, we formulate the two problems discussed in this chapter.

6.2.1 *Stability Verification Problem*

In our problem formulation, we consider the following notion of stability for system (6.1)–(6.2), that guarantees the exponential convergence of the state to the origin, i.e., $z = 0$, with a predefined rate $\beta \in \mathbb{R}^+$:

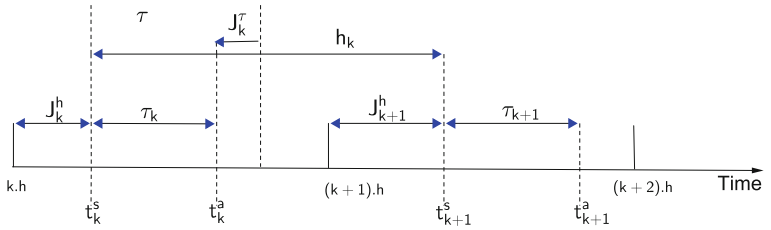

Fig. 6.3 Time variables included in a *TOL* contract. $J_k^h \in [0, J^h]$ and $J_k^\tau \in [-J^\tau, J^\tau]$

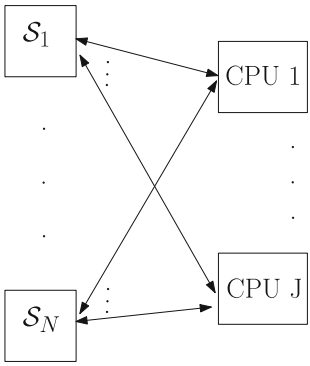

Fig. 6.4 Block diagram of N sampled-data systems sharing J CPUs

Definition 6.1 (β-*Stability*) Let $\beta \in \mathbb{R}^+$, system (6.1)–(6.2) is β-stable if there exist $C \in \mathbb{R}^+$ and $\varepsilon' \in \mathbb{R}^+$ such that

$$|z(t)| \leq Ce^{-(\beta+\varepsilon')(t-t_0^s)}|z(t_0^s)|, \ \forall t \in \mathbb{R}^+. \tag{6.3}$$

Consequently, in this work, we consider the following problem:

Problem 6.1 (*Stability verification*) Given $\beta \in \mathbb{R}^+$, $A \in \mathbb{R}^{p\times p}$, $B \in \mathbb{R}^{p\times m}$, $K \in \mathbb{R}^{m\times p}$, $(\underline{\tau}, \overline{\tau}, \underline{h}, \overline{h}) \in \mathscr{C}$, verify that (6.1)–(6.2) is β-stable.

The reader is referred to Sect. 6.3 where we provide an overview of existing techniques that can solve Problem 6.1, and we present our own approach.

6.2.2 *Scheduling Problem on Multiple CPUs*

Consider a collection of $N \in \mathbb{N}^+$ sampled-data systems $\{\mathscr{S}_1, \ldots, \mathscr{S}_N\}$ of the form (6.1) where each system $\mathscr{S}_i = (A_i, B_i, K_i)$ is subject to a timing contract $\theta(\underline{\tau}^i, \overline{\tau}^i, \underline{h}^i, \overline{h}^i)$ of the form (6.2), with parameters $(\underline{\tau}^i, \overline{\tau}^i, \underline{h}^i, \overline{h}^i) \in \mathscr{C}$, $i \in \mathbb{N}_{[1,N]}$.

In addition, we assume that these systems share J CPUs, as shown in Fig. 6.4, to compute the value of their control inputs given by (6.1b). Note that no communication

exists in between the CPUs or between the systems, but there exists communication only between the systems and all J CPUs. Furthermore, the time required by CPU j to compute inputs of system S_i is assumed to belong to some known interval $[\underline{c}^i_j, \overline{c}^i_j]$ with $0 \leq \underline{c}^i_j \leq \overline{c}^i_j$, $i \in \mathbb{N}_{[1,N]}$, and $j \in \mathbb{N}_{[1,J]}$, where $\underline{c}^i_j$ and $\overline{c}^i_j$ denote the best and worst case execution time, respectively.

The timing of events in the kth control cycle of system $\mathscr{S}_i$ starts at instant $t_k^{s_i}$ when sampling occurs. Then, system $\mathscr{S}_i$ gains access to the CPU j at instant $t_k^{b_i}$, at which computation of the control input value begins. The CPU is released at instant $t_k^{e_i}$, at which computation of the control input value ends. After that, actuation occurs at instant $t_k^{a_i}$. We denote by $\mathbb{N}(i, j)$ the set gathering indexes of the control cycles, at which system $\mathscr{S}_i$ accesses the CPU j, where $\bigcup_{j \in \mathbb{N}_{[1,J]}} \mathbb{N}(i, j) = \mathbb{N}$ for all $i \in \mathbb{N}_{[1,N]}$.

Then, the sequences $(t_k^{s_i})_{k\in\mathbb{N}}$, $(t_k^{b_i})_{k\in\mathbb{N}}$, $(t_k^{e_i})_{k\in\mathbb{N}}$, and $(t_k^{a_i})_{k\in\mathbb{N}}$ satisfy the following constraints for all $i \in \mathbb{N}_{[1,N]}$:

$$\begin{array}{ll} 0 \leq t_0^{s_i} & \\ t_k^{s_i} \leq t_k^{b_i} \leq t_k^{e_i} \leq t_k^{a_i} \leq t_{k+1}^{s_i}, & \forall k \in \mathbb{N} \\ c_k^i = t_k^{e_i} - t_k^{b_i} \in [\underline{c}^i_j, \overline{c}^i_j], & \forall k \in \mathbb{N}(i, j), \forall j \in \mathbb{N}_{[1,J]} \\ \tau_k^i = t_k^{a_i} - t_k^{s_i} \in [\underline{\tau}^i, \overline{\tau}^i], & \forall k \in \mathbb{N} \\ h_k^i = t_{k+1}^{s_i} - t_k^{s_i} \in [\underline{h}^i, \overline{h}^i], & \forall k \in \mathbb{N}. \end{array} \tag{6.4}$$

In addition, a conflict arises if several systems request access to one of the J CPUs at the same time. Let us define the following time sets, for $i \in \mathbb{N}_{[1,N]}$ and $j \in \mathbb{N}_{[1,J]}$:

$$\mathsf{Com}(\mathscr{S}_i, j) = \bigcup_{k\in\mathbb{N}(i,j)} [t_k^{b_i}, t_k^{e_i}).$$

$\mathsf{Com}(\mathscr{S}_i, j)$ is the union of time intervals when CPU j is used by system $\mathscr{S}_i$. Then, in order to prevent conflicting accesses to the CPU the following property must hold:

$$\begin{array}{c} \forall (m, n, j) \in \mathbb{N}^2_{[1,N]} \times \mathbb{N}_{[1,J]} \text{ with } m \neq n, \\ \mathsf{Com}(\mathscr{S}_m, j) \cap \mathsf{Com}(\mathscr{S}_n, j) = \emptyset. \end{array} \tag{6.5}$$

Remark 6.1 It is straightforward to verify that for any sequences $(t_k^{s_i})_{k\in\mathbb{N}}$, $(t_k^{b_i})_{k\in\mathbb{N}}$, $(t_k^{e_i})_{k\in\mathbb{N}}$, and $(t_k^{a_i})_{k\in\mathbb{N}}$ satisfying (6.4)–(6.5), the sequences $(t_k^{s_i})_{k\in\mathbb{N}}$ and $(t_k^{a_i})_{k\in\mathbb{N}}$ satisfy the timing contract $\theta(\underline{\tau}^i, \overline{\tau}^i, \underline{h}^i, \overline{h}^i)$.

We aim at synthesizing a dynamic scheduling policy, generating sequences of timing events satisfying (6.4)–(6.5). The scheduler has control over the sampling and actuation instants $(t_k^{s_i})_{k\in\mathbb{N}}$, $(t_k^{a_i})_{k\in\mathbb{N}}$ and over the instants $(t_k^{b_i})_{k\in\mathbb{N}}$ when computation begins. Also, the scheduler assigns a CPU to compute the control input for each system $\mathscr{S}_i$ at each control cycle $k \in \mathbb{N}$. However, the execution times $(c_k^i)_{k\in\mathbb{N}}$, and thus the instants when computation ends $(t_k^{e_i})_{k\in\mathbb{N}}$, are determined by the environment and are therefore uncontrollable from the point of view of the scheduler. Next, given

Table 6.1 Methods that can solve instances of Problem 6.1 with description of the modeling and computational approaches, list of restrictions, and possible extensions

References	Models	Algorithm	Restrictions	Extensions
[9]	Difference inclusions	LMI	–	$\tau_k > h_k$; controller synthesis
[10]		LMI	–	Scheduling
[15]		LMI	$\underline{\tau} = \overline{\tau} = 0$	Controller synthesis
[16]		LMI	$\underline{\tau} = \overline{\tau} = 0$	–
[22]		SOS	$\underline{\tau} = \overline{\tau} = 0$	–
[12]		Invariance	$\underline{\tau} = \overline{\tau} = 0$	–
[18]	Time-delay systems	LMI	$\underline{h} = 0$	$\tau_k > h_k$; scheduling
[14]		LMI	$\underline{h} = \overline{h}, \underline{\tau} = 0$	Controller synthesis; quantization
[20]		LMI	$\underline{\tau} = \overline{\tau} = 0$	–
[13]	Interconnected systems	LMI	$\underline{h} = \underline{\tau} = \overline{\tau} = 0$	–
[6]	Hybrid systems	SOS	–	Nonlinear dynamics; scheduling
[17]		LMI	$\underline{\tau} = 0, \underline{h} = 0$	Scheduling

that a task T_i, a task-set $\mathscr{T}$, and timing contracts Θ are characterized as

$$T_i = \left((\underline{c}_1^i, \overline{c}_1^i), \ldots, (\underline{c}_J^i, \overline{c}_J^i)\right), i \in \mathbb{N}_{[1,N]} \tag{6.6a}$$

$$\mathscr{T} = \{T_1, \ldots, T_N\}, \tag{6.6b}$$

$$\Theta = \{\theta(\underline{\tau}^1, \overline{\tau}^1, \underline{h}^1, \overline{h}^1), \ldots, \theta(\underline{\tau}^N, \overline{\tau}^N, \underline{h}^N, \overline{h}^N)\}, \tag{6.6c}$$

we define the scheduling problem informally, at this point of the chapter, as

Problem 6.2 (*Schedulability verification*) Given a set of control tasks $\mathscr{T}$ and timing contracts $\Theta = \{\theta(\underline{\tau}^1, \overline{\tau}^1, \underline{h}^1, \overline{h}^1), \ldots, \theta(\underline{\tau}^N, \overline{\tau}^N, \underline{h}^N, \overline{h}^N)\}$ as in (6.6), verify whether or not there exists a scheduling policy with sequences of timing events satisfying (6.4)–(6.5).

A precise formulation of the schedulability of the task-set $\mathscr{T}$ is provided in Sect. 6.4 along with a solution to the schedulability verification problem based on safety games over timed game automata.

6.3 Stability Verification

Several approaches are presented in the literature to solve instances of Problem 6.1. A non-exhaustive list is given in Table 6.1. From the modeling perspective, the problem can be tackled using difference inclusions, time-delay systems, or hybrid systems. On the computational side, the approaches are based on semi-definite programming (Linear Matrix Inequalities (LMI) or Sum Of Squares (SOS) formulations), invariant sets, or reachability analysis. Let us remark that approaches [6, 9, 10] appear to be able to address all instances of Problem 6.1.

Regarding our approach to Problem 6.1, we solve the same problem for a more general class of dynamic systems, given by a difference inclusion, and conclude on the stability of system (6.1)–(6.2). Meanwhile, one essential ingredient that is used in our study is the approximation scheme developed for over-approximating the reachable set of (6.1)–(6.2) from a given initial set. Such an over-approximation is provided in a previous work [2]. Let us first start by rewriting the system in terms of impulsive systems, in order to interpret the reachable set we use in the sequel.

6.3.1 Reformulation Using Impulsive Systems

In our analysis, it is more practical to transform (6.1) into an impulsive system with two types of resets each referring to a sampling or actuation instant. Such a reformulation is convenient to develop stability conditions based on reachability analysis. The system is thus given by

$$\begin{aligned} \dot{x}(t) &= A_c x(t), \quad t \neq t_k^s, t \neq t_k^a \\ x(t_k^{s+}) &= A_s x(t_k^s), \quad k \in \mathbb{N} \\ x(t_k^{a+}) &= A_a x(t_k^a), \quad k \in \mathbb{N} \end{aligned} \tag{6.7}$$

where $x(t) \in \mathbb{R}^n$ is the state of the system with $n = p + 2m$, $(t_k^s)_{k\in\mathbb{N}}$ and $(t_k^a)_{k\in\mathbb{N}}$ are given by (6.2), $x(t^+) = \lim_{\tau\to 0,\tau>0} x(t+\tau)$, and

$$\begin{aligned} A_c &= \begin{pmatrix} A & 0 & B \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; A_s = \begin{pmatrix} I_p & 0 & 0 \\ K & 0 & 0 \\ 0 & 0 & I_m \end{pmatrix}, \\ A_a &= \begin{pmatrix} I_p & 0 & 0 \\ 0 & I_m & 0 \\ 0 & I_m & 0 \end{pmatrix}, \; x(t) = \begin{pmatrix} z(t) \\ Kz(\theta^s(t)) \\ u(t) \end{pmatrix}, \end{aligned} \tag{6.8}$$

with $\theta^s(t) = t_k^s$ for $t \in (t_k^s, t_{k+1}^s]$. We consider in the following, system (6.7) under timing contract (6.2).

A notion for stability of the impulsive system guaranteeing the exponential convergence of the state to the origin with a predefined rate $\beta \in \mathbb{R}^+$ is given by

Definition 6.2 (β-*Stability*) Let $\beta \in \mathbb{R}^+$, system (6.2)–(6.7) is β-stable if there exist $C \in \mathbb{R}^+$ and $\varepsilon^* \in \mathbb{R}^+$ such that

$$|x(t)| \leq Ce^{-(\beta+\varepsilon^*)(t-t_0^s)}|x(t_0^s)|, \ \forall t \in \mathbb{R}^+. \tag{6.9}$$

Note that β-stability of system (6.2)–(6.7) is equivalent to the β-stability of (6.2)–(6.1). We are now interested in verifying stability of embedded control systems in the form given by (6.7) under one of the general timing contracts defined previously in Sect. 6.2. Indeed, we can easily show that system (6.7) under the ZET and LET contracts is stable if and only if the eigenvalues of the matrix $e^{hA_c}A_aA_s$ and $A_ae^{hA_c}A_s$ are inside the unit circle, respectively. As for the DET or TOL contracts, we have that stability of system (6.2)–(6.1) is guaranteed by the stability of system (6.2)–(6.7) with an adequate choice of the timing contract parameters. It is noteworthy that in the case of the TOL contract, stability of system (6.2)–(6.7) is only sufficient when the parameters of the over-approximating timing contract are chosen as explained in Sect. 6.2. Consequently, in this work, we consider an equivalent to Problem 6.1:

Problem 6.3 (*Stability verification*) Given $\beta \in \mathbb{R}^+$, $A_c, A_s, A_a \in \mathbb{R}^{n\times n}$, $(\underline{\tau}, \overline{\tau}, \underline{h}, \overline{h}) \in \mathscr{C}$, verify that (6.2)–(6.7) is β-stable.

6.3.2 A Stability Verification Approach Based on Difference Inclusions

Our stability verification approach to solve Problem 6.3 is based on a reformulation of the linear impulsive systems (6.2)–(6.7) in the general framework of difference inclusions. Then, for a fairly large class of difference inclusions, we recall necessary and sufficient conditions for stability, established in [3]. These conditions are based on the successive images of a set under the dynamics of the difference inclusion. For linear impulsive systems (6.2)–(6.7), these conditions allow us to design a stability verification algorithm using reachability analysis techniques developed in [2].

Let us introduce first a general formulation based on difference inclusions and later show how linear impulsive systems in the form of (6.2)–(6.7) can be embedded in this framework. We consider discrete-time dynamical systems modeled by the following difference inclusion:

$$\xi_{k+1} \in \Phi(\{\xi_k\}), \ k \in \mathbb{N} \tag{6.10}$$

where $\xi_k \in \mathbb{R}^n$ is the state of the system, and $\Phi : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$ is a set-valued map. Stability for systems of the form (6.10) is considered in the following sense:

Definition 6.3 (*GES*) System (6.10) is globally exponentially stable (GES) if there exists $(C, \varepsilon) \in \mathbb{R}^+ \times (0, 1)$ such that for all trajectories $(\xi_k)_{k\in\mathbb{N}}$ of (6.10), we have

$$|\xi_k| \leq C\varepsilon^k |\xi_0|, \ \forall k \in \mathbb{N}. \tag{6.11}$$

Next we verify the stability of a difference inclusion of the form (6.10). We make first the following assumptions on the map Φ.

Assumption 6.1 For all $\mathscr{S} \subseteq \mathbb{R}^n$, $\lambda \in \mathbb{R}_0^+$, the following assertions hold:

(i) $\Phi(\mathscr{S}) = \bigcup_{z\in\mathscr{S}} \Phi(\{z\})$;
(ii) $\Phi(\lambda\mathscr{S}) \subseteq \lambda\Phi(\mathscr{S})$;
(iii) if $\mathscr{S}$ is bounded, then $\Phi(\mathscr{S})$ is bounded.

Under item (i) of Assumption 6.1, for all $\mathscr{S}, \mathscr{S}' \subseteq \mathbb{R}^n$, it follows that $\Phi(\mathscr{S} \cup \mathscr{S}') = \Phi(\mathscr{S}) \cup \Phi(\mathscr{S}')$. Also, if $\mathscr{S} \subseteq \mathscr{S}'$, then $\Phi(\mathscr{S}) \subseteq \Phi(\mathscr{S}')$. We define the iterates of Φ as $\Phi^0(\mathscr{S}) = \mathscr{S}$ for all $\mathscr{S} \subseteq \mathbb{R}^n$, and $\Phi^{k+1} = \Phi \circ \Phi^k$ for all $k \in \mathbb{N}$. Let $(\xi_k)_{k\in\mathbb{N}}$ be a trajectory of (6.10) such that $\xi_0 \in \mathscr{S}$, then under item (i) of Assumption 6.1, for all $k \in \mathbb{N}$, $\Phi^k(\mathscr{S})$ is the set of all possible values of ξ_k.

Then, the stability verification problem, for systems of the form (6.10), can be formulated as follows:

Problem 6.4 (*Stability verification*) Under Assumption 6.1, verify that system (6.10) is GES.

Let $\beta \in \mathbb{R}^+$ and suppose that $\mathscr{S} \subseteq \mathbb{R}^n$ represents all the states of the system at sampling instant t_k^s. Then, a scaled reachable set of system (6.2)–(6.7) at instant t_{k+1}^s is given by the map $\Phi : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$ such that

$$\Phi(\mathscr{S}) = \bigcup_{\tau\in[\underline{\tau},\overline{\tau}]} \bigcup_{w\in[\max(0,\underline{h}-\tau),\overline{h}-\tau]} e^{(w+\tau)\beta} e^{wA_c} A_a e^{\tau A_c} A_s \mathscr{S}. \tag{6.12}$$

The following proposition establishes the equivalence between stability of systems (6.2)–(6.7) and (6.10).

Proposition 6.1 *Given $\beta \in \mathbb{R}^+$. System (6.2)–(6.7) is β-stable if and only if system (6.10) is GES with Φ given by (6.12).*

The next proposition shows that the map Φ in (6.12) satisfies the previous assumptions.

Proposition 6.2 *Let Φ be given by (6.12), then Φ satisfies Assumption 6.1.*

It follows from Propositions 6.1 and 6.2 that Problem 6.3 can be reduced to Problem 6.4. Therefore, in the next sections, we present necessary and sufficient theoretical conditions for stability verification and an algorithm to solve Problem 6.4.

6.3.2.1 Stability Verification: Theoretical Results

This section presents necessary and sufficient conditions, taken from [3], for stability of system (6.10). The following result characterizes the stability of system (6.10) in terms of the map Φ.

Theorem 6.2 *Let $\mathscr{S} \subseteq \mathbb{R}^n$ be bounded with 0 in its interior, under Assumption 6.1, the following statements are equivalent:*

(a) System (6.10) is GES;
(b) There exists $(k, j, \rho) \in \mathbb{N}^+ \times \mathbb{N}_{[0,k-1]} \times (0, 1)$ such that $\Phi^k(\mathscr{S}) \subseteq \rho\Phi^j(\mathscr{S})$;
(c) There exists $(k, \rho) \in \mathbb{N}^+ \times (0, 1)$ such that $\Phi^k(\mathscr{S}) \subseteq \rho \bigcup_{j=0}^{k-1} \Phi^j(\mathscr{S})$.

6.3.2.2 An Algorithm for Stability Verification

In this section, we present an algorithm for verifying the stability of system (6.10). Indeed, the maps Φ involved in Theorem 6.2 can be impractical to compute exactly. This is the case of linear impulsive system (6.2)–(6.7), which requires the computation of the reachable set given by (6.12). In that case, we may use an over-approximation $\overline{\Phi} : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$, which is easier to compute and satisfies the following assumption:

Assumption 6.3 For all $\mathscr{S} \subseteq \mathbb{R}^n$, the following assertions hold:

(i) $\Phi(\mathscr{S}) \subseteq \overline{\Phi}(\mathscr{S})$;
(ii) if $\mathscr{S}$ is bounded then $\overline{\Phi}(\mathscr{S})$ is bounded.

The iterates of $\overline{\Phi}$ are defined similarly to those of Φ. We now derive sufficient conditions for stability of system (6.10) based on $\overline{\Phi}$.

Corollary 6.1 *Under Assumptions 6.1 and 6.3, if $\mathscr{S} \subseteq \mathbb{R}^n$ bounded with 0 in its interior, and $(k, i, \rho) \in \mathbb{N}^+ \times \mathbb{N}_{[0,k-1]} \times (0, 1)$ such that $\overline{\Phi}^k(\mathscr{S}) \subseteq \rho\overline{\Phi}^i(\mathscr{S})$, then system (6.10) is GES.*

Now, we propose a stability verification algorithm to solve Problem 6.4 based on the sufficient condition given in Corollary 6.1. The algorithm consists of an initialization step and a main loop. In the initialization step, we compute an initial set $\mathscr{S}$, which is then propagated in the main loop using the map $\overline{\Phi}$ to check the stability condition given by Corollary 6.1. The choice of the initial set is important in order to try to minimize the value of the integer k such that the stability condition given by Corollary 6.1 holds. Detailed approaches to compute the initial set $\mathscr{S}$ and the over-approximation $\overline{\Phi}$ can be found in [2].

6.4 Scheduling of Embedded Controllers Under Timing Contracts

Our aim in this section is to solve Problem 6.2 on schedulability verification.

6.4.1 Timed Game Automata and Safety Games

This section is intended to briefly introduce timed automata [1], timed game automata [21], and safety games.

6.4.1.1 Timed and Timed Game Automata

Let C be a finite set of real-valued variables called clocks. We denote by $\mathscr{B}(C)$ the set of conjunctions of clock constraints of the form $c \sim \alpha$ where $\alpha \in \mathbb{R}_0^+$, $c \in C$ and $\sim \in \{<, \leq, =, >, \geq\}$. We define a timed automaton (TA) and a timed game automaton (TGA) as in [8]:

Definition 6.4 A *timed automaton* is a sextuple (L, l_0, Act, C, E, I) where

- L is a finite set of locations;
- $l_0 \in L$ is the initial location;
- Act is a set of actions;
- C is a finite set of real-valued clocks;
- $E \subseteq L \times \mathscr{B}(C) \times Act \times 2^C \times L$ is the set of edges;
- $I : L \rightarrow \mathscr{B}(C)$ is a function that assigns invariants to locations.

Definition 6.5 A *timed game automaton* is a septuple $(L, l_0, Act_c, Act_u, C, E, I)$ such that $(L, l_0, Act_c \cup Act_u, C, E, I)$ is a timed automaton and $Act_c \cap Act_u = \emptyset$, where Act_c defines a set of controllable actions and Act_u defines a set of uncontrollable actions.

Formal semantics of TA and TGA are stated in [8]. Informally, semantics of a TA is described by a transition system whose state consists of the current location and value of the clocks. Then, the execution of a TA can be described by two types of transitions defined as follows:

- time progress: the current location $l \in L$ is maintained and the value of the clocks grow at unitary rate; these transitions are enabled as long as the value of the clocks satisfies $Inv(l)$.
- discrete transition: an instantaneous transition from the current location $l \in L$ to a new location $l' \in L$ labeled by an action $a \in Act$ is triggered; these transitions are enabled if there is an edge $(l, G, a, C', l') \in E$, such that the value of the clocks satisfies G; in that case, the value of the clocks belonging to C' is reset to zero.

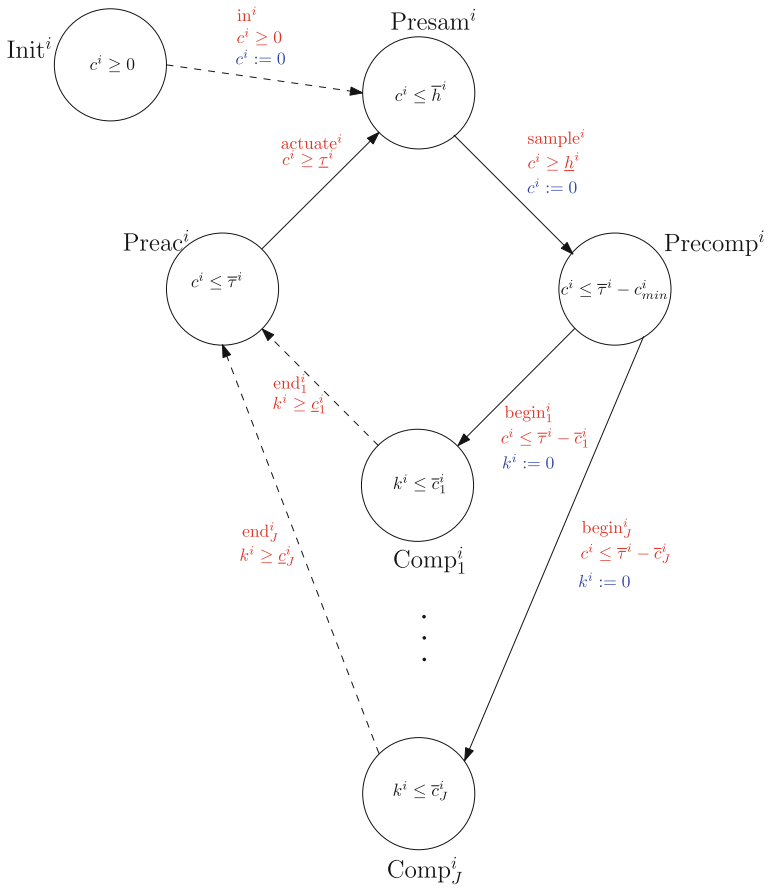

Fig. 6.5 TGA_i, where plain and dashed edges correspond to controllable and uncontrollable actions, respectively

The semantics of TGA is similar to that of TA with the specificity that discrete transitions labeled by a controllable actions (i.e., $a \in Act_c$) are triggered by a controller, while discrete transitions labeled by uncontrollable actions (i.e., $a \in Act_u$) are triggered by the environment/opponent.

6.4.1.2 Safety Games

Safety games (see, e.g., [8]) are defined by a timed game automaton and a set of unsafe locations $L_u \subseteq L$. A solution to the safety game is given by a winning strategy for the controller such that under any behavior of the environment/opponent, the set of unsafe locations is avoided by all controlled executions of the TGA.

6.4.2 Reformulation into TGA

We propose a reformulation of the schedulability verification problem using timed game automata and safety games.

We first associate to each control task and timing contract a timed game automaton, as shown in Fig. 6.5 and formally defined as follows:

Definition 6.6 Let $i \in \mathbb{N}_{[1,N]}$, the timed game automaton generated by control task $T_i = \left((\underline{c}_1^i, \overline{c}_1^i), \ldots, (\underline{c}_J^i, \overline{c}_J^i)\right)$ and timing contract $\theta(\underline{\tau}^i, \overline{\tau}^i, \underline{h}^i, \overline{h}^i)$ is

$$\mathsf{TGA}_i = (L^i, l_0^i, Act_c^i, Act_u^i, C^i, E^i, Inv^i),$$

where

- $L^i = \{Init^i, Presam^i, Precomp^i, Preac^i, Comp_1^i, \ldots, Comp_J^i\}$;
- $l_0^i = Init^i$;
- $Act_c^i = \{sample^i, begin_1^i, \ldots, begin_J^i, actuate^i\}$;
- $Act_u^i = \{end_1^i, \ldots, end_J^i, in^i\}$;
- $C^i = \{c^i, k^i\}$;
- $E^i = \{(Init^i, c^i \geq 0, in^i, \{c^i\}, Presam^i),$
  $(Presam^i, c^i \geq \underline{h}^i, sample^i, \{c^i\}, Precomp^i),$
  $(Precomp^i, c^i \leq \overline{\tau}^i - \overline{c}_1^i, begin_1^i, \{k^i\}, Comp_1^i), \ldots,$
  $(Precomp^i, c^i \leq \overline{\tau}^i - \overline{c}_J^i, begin_J^i, \{k^i\}, Comp_J^i),$
  $(Comp_1^i, k^i \geq \underline{c}_1^i, end_1^i, \emptyset, Preac^i), \ldots, (Comp_J^i, k^i \geq \underline{c}_J^i, end_J^i, \emptyset, Preac^i),$
  $(Preac^i, c^i \geq \underline{\tau}^i, actuate^i, \emptyset, Presam^i)\}$;
- $Inv^i(Init^i) = \{c^i \geq 0\},$
  $Inv^i(Presam^i) = \{c^i \leq \overline{h}^i\},$
  $Inv^i(Precomp^i) = \{c^i \leq \overline{\tau}^i - c_{min}^i\}, with \;\; c_{min}^i = \min_{j \in \mathbb{N}_{[1,J]}}(\overline{c}_j^i),$
  $Inv^i(Comp_1^i) = \{k^i \leq \overline{c}_1^i\}, \ldots, Inv^i(Comp_J^i) = \{k^i \leq \overline{c}_J^i\},$
  $Inv^i(Preac^i) = \{c^i \leq \overline{\tau}^i\}.$

Intuitively, the set of locations L^i denotes all the possible situations that a control task T_i may be in and E^i denotes all the possible transitions between locations. If we assume that the control loop has not started yet then this is modeled by the location $Init^i$. After that the control loop starts at a certain time that is determined by the environment and thus an uncontrollable transition $(Init^i, c^i \geq 0, in^i, \{c^i\}, Presam^i)$ takes place, where the task has to wait until sampling could occur. The latter is realized by the location $Presam^i$. Then whenever possible, a controller (which is the scheduler) has to decide when sampling must occur. When sampling takes place, the control task will be waiting until a CPU is assigned to compute its control input. This waiting situation is realized by the $Precomp^i$ location. The mission of assigning a CPU for task T_i is that of the scheduler, thus a possible controllable transition occurs when the assignment of CPU_j takes place declaring that the task is in a new situation realized in TGA_i by the location $Comp_J^i$. The task rests in this situation until its execution on the CPU finishes which means that this duration is decided

by the environment (which is the CPU and not the scheduler) and thus an uncontrollable transition from $Comp^i_j$ to a new location $Preac^i$ means that the execution has terminated and the control task is in the situation where actuation is to happen next. The latter decision is taken by the scheduler, and thus is controllable, where the control input is fed to the plant and the control task is back again in the pre-sampling situation realized as before by the $Presam^i$ location. In such a case, the control loop is closed and the behavior of the control task is repeated infinitely. Note that all the executions of TGA_i explained informally above must respect the semantics of the timed game automata introduced in Sect. 6.4.1.

Now let the sequences $(t_k^{s_i})$, $(t_k^{a_i})$, $(t_k^{b_i})$ and $(t_k^{e_i})$ be given by the instants of the discrete transitions labeled by actions $sample^i$, $actuate^i$, $begin^i$ and end^i, respectively. It is easy to see that these sequences satisfy the constraints given by (6.4). Conversely, one can check that all sequences satisfying (6.4) can be generated by executions of TGA_i. Moreover, let us restate that the controllable actions are $sample^i$, $actuate^i$, $begin^i$, which means that the scheduler determines the instants when sampling and actuation occur and when computation begins. However, end^i is uncontrollable, which means that the execution time, and thus the instant at which computation ends is determined by the environment.

Finally, CPU j is used by system $\mathscr{S}_i$ if the current location of TGA_i is $Comp^i_j$, with $j \in \mathbb{N}_{[1,J]}$. To take into account the constraint given by (6.5), stating that two systems cannot access any of the J CPUs at the same time, we need to define the composition of the timed game automata defined above:

Definition 6.7 The timed game automaton generated by the set of control tasks $\mathscr{T} = \{T_1, \ldots, T_N\}$, with $T_i = \big((\underline{c}^i_1, \overline{c}^i_1), \ldots, (\underline{c}^i_J, \overline{c}^i_J)\big)$ for all $i \in \mathbb{N}_{[1,N]}$, and timing contracts $\Theta = \{\theta(\underline{\tau}^1, \overline{\tau}^1, \underline{h}^1, \overline{h}^1), \ldots, \theta(\underline{\tau}^N, \overline{\tau}^N, \underline{h}^N, \overline{h}^N)\}$ is given by $\mathsf{TGA} = (\overline{L}, \overline{l}_0, \overline{Act_c}, \overline{Act_u}, \overline{C}, \overline{E}, \overline{Inv})$ where

- $\overline{L} = L^1 \times \cdots \times L^N$, thus $l = (l^1, \ldots, l^N) \in L$ denotes the location of TGA;
- $\overline{l}_0 = (Init^1, \ldots, Init^N)$;
- $\overline{Act_c} = \bigcup_{i=1}^N Act^i_c$;
- $\overline{Act_u} = \bigcup_{i=1}^N Act^i_u$;
- $\overline{C} = \bigcup_{i=1}^N C^i$;
- $\overline{E} = \{(l_m, \lambda, act, C', l_n) \in \overline{L} \times \mathscr{B}(\overline{C}) \times (\overline{Act_c} \cup \overline{Act_u}) \times \overline{L} : \exists i \in \mathbb{N}_{[1,N]}, l^j_m = l^j_n \ \forall j \neq i \text{ and } (l^i_m, \lambda, act, C', l^i_n) \in E^i\}$;
- $Inv(l) = \bigwedge_{i=1}^N Inv^i(l^i), i \in \mathbb{N}_{[1,N]}$.

TGA describes the parallel evolution of the $\mathsf{TGA}_1, \ldots, \mathsf{TGA}_N$ and thus models the concurrent execution of the control tasks $T_1, \ldots, T_N$.

6.4.3 Scheduling as a Safety Game

In our setting, we denote the safety game by $(\mathsf{TGA}, \overline{L}_u)$, where the set of locations corresponding to conflicting accesses to the CPUs $\overline{L}_u \subseteq \overline{L}$ is defined by

$$\overline{L}_u = \{l \in \overline{L} : \exists (m, n, j) \in \mathbb{N}^2_{[1,N]} \times \mathbb{N}_{[1,J]}, m \neq n, \\ (l^m = Comp^m_j) \wedge (l^n = Comp^n_j)\}. \tag{6.13}$$

From the previous discussions, we define the following property:

Definition 6.8 (*Schedulability*) $\mathscr{T}$ is *schedulable* under timing contracts Θ if and only if there is a winning strategy to $(\mathsf{TGA}, \overline{L}_u)$.

From the practical point of view, the safety game, and thus Problem 6.2, can be solved using the tool UPPAAL-TIGA [5]. The latter synthesizes also a winning strategy when it exists, which provides us with a dynamic scheduling policy for generating the sequences $(t_k^{s_i})_{k\in\mathbb{N}}$, $(t_k^{b_i})_{k\in\mathbb{N}}$, $(t_k^{e_i})_{k\in\mathbb{N}}$, and $(t_k^{a_i})_{k\in\mathbb{N}}$ satisfying (6.4)–(6.5), for all $i \in \mathbb{N}_{[1,N]}$.

6.5 Illustrative Example

In this section, we are interested in synthesizing schedules for a given number N of sampled-data systems, which are subject to timing contracts and whose control input is computed by J shared CPUs, with $J < N$. Indeed, the schedule should guarantee the stability of each system. We implemented the scheduling approach presented in Sect. 6.4 using UPPAAL-TIGA [5], and used the stability verification algorithm from [2] to verify stability.

6.5.1 One Processor

Example 6.1 We take $N = 2$ where the two systems $\mathscr{S}_1 = (A_1, B_1, K_1)$ and $\mathscr{S}_2 = (A_2, B_2, K_2)$ are taken from [7] and are given by the following matrices:

$$A_1 = \begin{pmatrix} 0 & 1 \\ 0 & -0.1 \end{pmatrix}, \; B_1 = \begin{pmatrix} 0 \\ 0.1 \end{pmatrix}, \; K_1 = \begin{pmatrix} -3.75 & -11.5 \end{pmatrix}. \tag{6.14}$$

$$A_2 = \begin{pmatrix} 0 & 1 \\ -2 & 0.1 \end{pmatrix}, \; B_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \; K_2 = \begin{pmatrix} 1 & 0 \end{pmatrix}. \tag{6.15}$$

6.5.1.1 Stability Verification

After setting $\beta = 0$, we use the stability verification algorithm in [2] to verify that systems $\mathscr{S}_1$ and $\mathscr{S}_2$ are β-stable under timing contracts $\theta(0.1, 0.35, 0.3, 0.85)$ and $\theta(0.2, 0.6, 0.8, 1.15)$, respectively. This means obviously that each of the two systems with any synthesized scheduling policy on a shared CPU, respecting the above

timing contracts, is guaranteed to be stable. The computation times required for stability verification are 1.96 and 1.5 s, respectively.

6.5.1.2 Scheduling

Now, we consider the set of control tasks $\mathscr{T} = \{T_1, T_2\}$ running on a single processor, or $J = 1$. After setting the best and worst case execution times for each task as $\underline{c}_1^1 = 0.12, \overline{c}_1^1 = 0.35, \underline{c}^2 = 0.04$, and $\overline{c}^2 = 0.12$ we define task $T_1 = ((\underline{c}_1^1, \overline{c}_1^1))$, task $T_2 = ((\underline{c}_1^2, \overline{c}_1^2))$, and the same set of timing contracts as in the previous section $\Theta = \{\theta(0.1, 0.35, 0.3, 0.85), \theta(0.2, 0.6, 0.8, 1.15)\}$.

In order to solve the scheduling problem, we associate to $\mathscr{T}$ the timed game automaton TGA as given in Definition 6.7. Following the approach in Sect. 6.4, we solve the safety game on TGA to find a strategy (if it exists) for the triggering of controllable actions that occur at $(t_k^{s_i})_{k\in\mathbb{N}}$, $(t_k^{b_i})_{k\in\mathbb{N}}$, and $(t_k^{a_i})_{k\in\mathbb{N}}$, with $i \in \mathbb{N}_{[1,2]}$, guaranteeing that the set of bad states $\overline{L}_u$ of the system, given by (6.13), is never reached regardless of when uncontrollable actions occurring at $(t_k^{e_i})_{k\in\mathbb{N}}$, $i \in \mathbb{N}_{[1,2]}$, are exactly taken.

Using UPPAAL-TIGA, we successfully proved that $\mathscr{T}$ is schedulable under timing contracts Θ, and thus a scheduling policy was found. The computation time required to solve the game was 1.37 s.

Figure 6.6 shows the timing of events resulting from this scheduling policy. The first and second plots show that the timing contracts $\theta(0.1, 0.35, 0.3, 0.85)$ and $\theta(0.2, 0.6, 0.8, 1.15)$ are respected for both systems $\mathscr{S}_1$ and $\mathscr{S}_2$, respectively. The third plot shows that only one of the two systems gains access to the shared processor at a time since it appears clearly that

$$\forall (m, n) \in \mathbb{N}^2_{[1,2]} \text{ with } m \neq n,$$
$$\mathsf{Com}(\mathscr{S}_m, 1) \cap \mathsf{Com}(\mathscr{S}_n, 1) = \emptyset.$$

One can notice that in the first three control cycles of $\mathscr{S}_2$, the beginning of the computation has to be delayed until the CPU is released by $\mathscr{S}_1$.

Using this scheduling policy, Fig. 6.7 shows results of simulating $\mathscr{S}_1$ and $\mathscr{S}_2$, when they share a single processor to compute the value of their control inputs, for the initial states $z_0^1 = \binom{2}{3}$ and $z_0^2 = \binom{2}{3}$ with $t_0^{s_1} = 0.4$ and $t_0^{s_2} = 0.9$. As shown, trajectories of both systems converge to zero and therefore the scheduling policy in this case guarantees the exponential stability of each system.

6.5.2 *Two Processors*

Example 6.2 We take $N = 3$, where we have two identical systems $\mathscr{S}_1$ and $\mathscr{S}_2$ whose matrices are given by (6.14) and another system $\mathscr{S}_3$ with matrices given by (6.15).

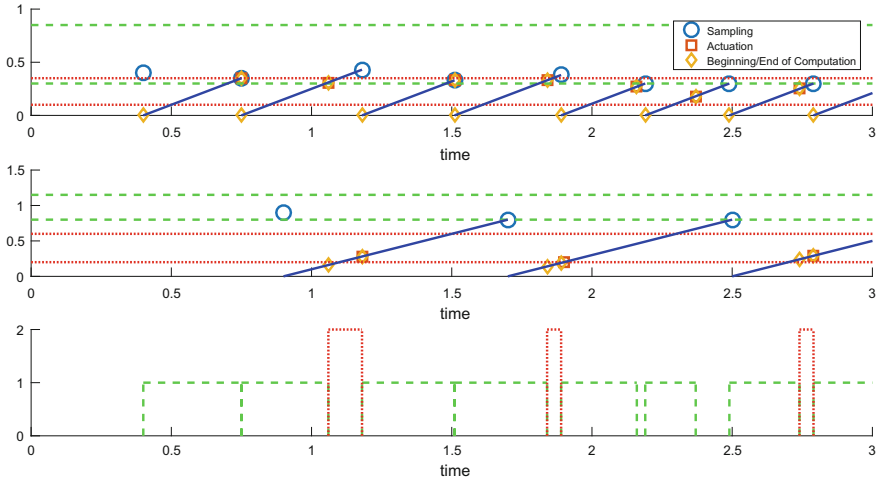

Fig. 6.6 Timing of events (sampling, beginning/end of computation, and actuation) for systems $\mathscr{S}_1$ (first plot) and $\mathscr{S}_2$ (second plot) during the first 3 s; dotted lines represent constraints on actuation instants, while dashed lines represent constraints on sampling instants. In the third plot, the dotted line represents $\mathsf{Com}(\mathscr{S}_2, t)$ (less frequent) and the dashed line represents $\mathsf{Com}(\mathscr{S}_1, t)$ (more frequent)

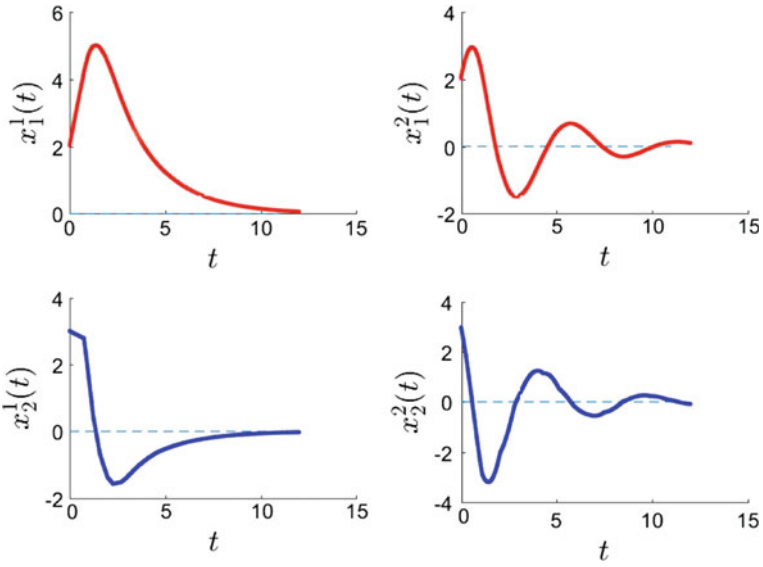

Fig. 6.7 Trajectories for systems $\mathscr{S}_1$ (left) and $\mathscr{S}_2$ (right) using the synthesized scheduling policy

First, we consider a single processor to compute the control input of the three systems (i.e., $J = 1$) where control tasks T_1, T_2, and T_3 are given by $T_1 = T_2 = (0.12, 0.25)$ and $T_3 = (0.04, 0.1)$. We consider the set of contracts $\Theta_a = \{\theta(0.1, 0.35, 0.1, 0.35), \theta(0.1, 0.35, 0.1, 0.35)\}$, $\Theta_b = \{\theta(0.1, 0.35, 0.1, 0.35), \theta(0.1, 0.2, 0.1, 0.2)\}$, and $\Theta_c = \{\theta(0.1, 0.35, 0.1, 0.35), \theta(0.1, 0.35, 0.1, 0.35), \theta(0.1, 0.2, 0.1, 0.2)\}$. Following the approach in Sect. 6.4 we can prove that each of the task-set $\{T_1, T_2\}$, the task-set $\{T_2, T_3\}$, and obviously the task-set $\{T_1, T_2, T_3\}$ is not schedulable under timing contracts Θ_a, Θ_b, and Θ_c respectively. On the other hand, this does not mean that systems $\mathscr{S}_1$, $\mathscr{S}_2$, and $\mathscr{S}_3$ cannot share two processors to compute their control input.

Now, we consider two CPUs, or $J = 2$, and define the task-set $\mathscr{T} = \{T_1, T_2, T_3\}$ with $T_1 = T_2 = ((0.12, 0.25), (0.12, 0.25))$ and $T_3 = ((0.04, 0.1), (0.04, 0.1))$. Then we associate to $\mathscr{T}$ the TGA as given in Definition 6.7 and solve the safety game on TGA to find a strategy (if it exists) for the triggering of controllable actions that occur at $(t_k^{s_i})_{k\in\mathbb{N}}$, $(t_k^{b_i})_{k\in\mathbb{N}}$, and $(t_k^{a_i})_{k\in\mathbb{N}}$, with $i \in \mathbb{N}_{[1,3]}$, guaranteeing that the set of bad states $\overline{L_u}$ of the system is never reached regardless of when uncontrollable actions occurring at $(t_k^{e_i})_{k\in\mathbb{N}}$, $i \in \mathbb{N}_{[1,3]}$, are exactly taken.

Using UPPAAL-TIGA, we successfully proved that $\mathscr{T}$ is schedulable under timing contracts Θ_c, and thus a scheduling policy was found. The computation time required to solve the game and output the scheduling policy was 10 s.

Figure 6.8 shows the timing of events resulting from this scheduling policy. The first three plots show that the contracts $\theta(0.1, 0.35, 0.1, 0.35)$, $\theta(0.1, 0.35, 0.1, 0.35)$, and $\theta(0.1, 0.2, 0.1, 0.2)$ are respected for systems $\mathscr{S}_1$, $\mathscr{S}_2$, and $\mathscr{S}_3$, respectively. The fourth and fifth plots show that only one of the three systems gains access to each of the two shared processors at a time since it appears clearly that

$$\forall (m, n, j) \in \mathbb{N}^2_{[1,3]} \times \mathbb{N}_{[1,2]} \text{with } m \neq n,$$
$$\mathsf{Com}(\mathscr{S}_m, j) \cap \mathsf{Com}(\mathscr{S}_n, j) = \emptyset.$$

At this point, we should mention that we followed the stability verification approach in [2] and proved that for $\beta = 0$, β-stability is guaranteed for systems $\mathscr{S}_1$, $\mathscr{S}_2$, and $\mathscr{S}_3$ under timing contracts $\theta(0.1, 0.35, 0.1, 0.35)$, $\theta(0.1, 0.35, 0.1, 0.35)$, and $\theta(0.1, 0.2, 0.1, 0.2)$ respectively. Using this scheduling policy, Fig. 6.9 shows results of simulating $\mathscr{S}_1$, $\mathscr{S}_2$, and $\mathscr{S}_3$ when they share two processors to compute the value of their control inputs, for the initial states $z_0^1 = z_0^2 = z_0^3 = \binom{2}{3}$ with $t_0^{s_1} = t_0^{s_2} = t_0^{s_3} = 0.01s$. As shown, trajectories of the three systems converge to zero and therefore the scheduling policy in this case guarantees the exponential stability of each system.

6.6 Conclusion

In this chapter, we proposed an approach for verifying stability and scheduling embedded control systems under timing contracts on a multi-core platform using

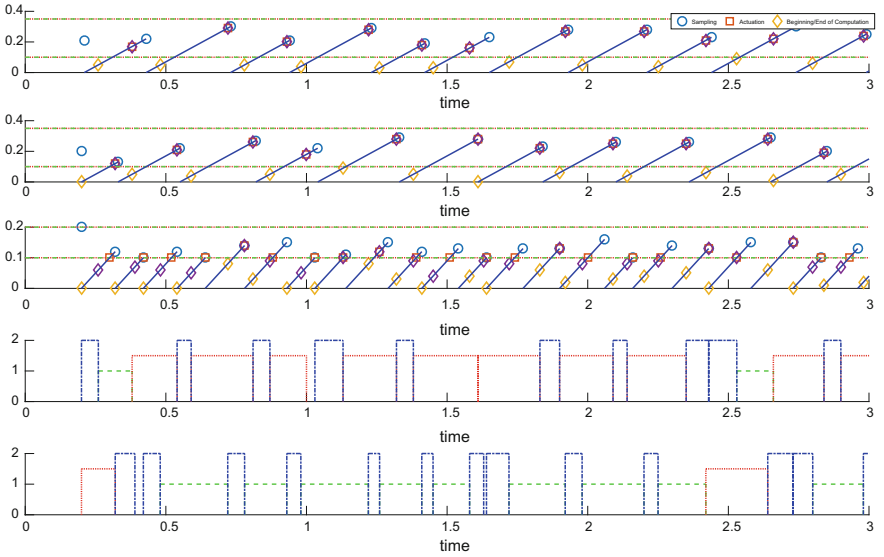

Fig. 6.8 Timing of events (sampling, beginning/end of computation, and actuation) for systems $\mathscr{S}_1$ (first plot), $\mathscr{S}_2$ (second plot), $\mathscr{S}_3$ (third plot) during the first 3 s; dotted lines represent constraints on actuation instants, while dashed lines represent constraints on sampling instants. In the fourth and fifth plot, the dashed line (magnitude 1) represents $\mathsf{Com}(\mathscr{S}_1, j)$, dotted line (magnitude 1.5) represents $\mathsf{Com}(\mathscr{S}_1, j)$, and the dotted-dashed line (magnitude 2) represents $\mathsf{Com}(\mathscr{S}_3, j)$ for $j = 1$ (fourth plot) and $j = 2$ (fifth plot)

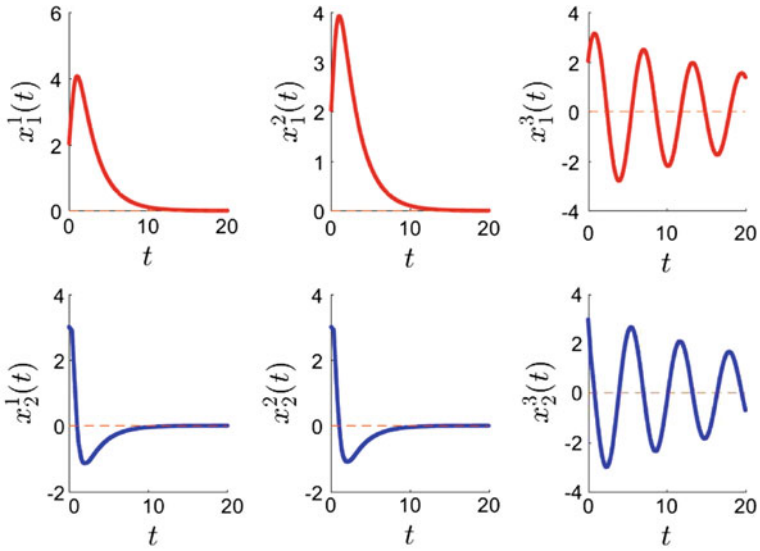

Fig. 6.9 Trajectories for systems $\mathscr{S}_1$ (left), $\mathscr{S}_2$ (middle), and $\mathscr{S}_3$ (right) using the synthesized scheduling policy

reachability analysis and safety timed games, respectively. As a future work, it would be interesting to consider preemptive scheduling since it is not trivial to extend the present work to scheduling with preemption. Another direction for improving our approach is to find optimal schedules in the sense that the control loop is to be closed as soon as possible for each task to have the best possible performance.

Acknowledgements This work was supported by the Agence Nationale de la Recherche (COMPACS project ANR-13-BS03-0004) and by the Labex DigiCosme, Université Paris-Saclay (CODECSYS project).

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
2. Al Khatib, M., Girard, A., Dang. T.: Verification and synthesis of timing contracts for embedded controllers. Hybrid Systems: Computation and Control, pp. 115–124. ACM (2016)
3. Al Khatib, M., Girard, A., Dang, T., Dang, T.: Stability verification and timing contract synthesis for linear impulsive systems using reachability analysis. Nonlinear Anal. Hybrid Syst, **25**, 211–226 (2017)
4. Al Khatib, M., Girard, A., Dang, T.: Scheduling of embedded controllers under timing contracts. Hybrid Systems: Computation and Control, pp. 131–140. ACM (2017)
5. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-TIGA: time for playing games! Computer Aided Verification, pp. 121–125. Springer (2007)
6. Bauer, N.W., Maas, P.J.H., Heemels, W.P.M.H.: Stability analysis of networked control systems: a sum of squares approach. Automatica **48**(8), 1514–1524 (2012)
7. Briat, C.: Convex conditions for robust stability analysis and stabilization of linear aperiodic impulsive and sampled-data systems under dwell-time constraints. Automatica **49**(11), 3449–3457 (2013)
8. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. Concurrency Theory, pp. 66–80. Springer, Berlin (2005)
9. Cloosterman, M.B.G., Hetel, L., Van De Wouw, N., Heemels, W.P.M.H., Daafouz, J., Nijmeijer, H.: Controller synthesis for networked control systems. Automatica **46**(10), 1584–1594 (2010)
10. Donkers, M.C.F., Heemels, W.P.M.H., Van De Wouw, N., Hetel, L.: Stability analysis of networked control systems using a switched linear systems approach. IEEE Trans. Autom. Control **56**(9), 2101–2115 (2011)
11. Derler, P., Lee, E.A., Tripakis, S., Törngren, M.: Cyber-physical system design contracts. In: International Conference on Cyber-Physical Systems, pages 109–118 (2013)
12. Fiacchini, M., Morărescu, I.-C.: Constructive necessary and sufficient condition for the stability of quasi-periodic linear impulsive systems. IEEE Trans. Autom. Control **61**(9), 2512–2517 (2016)
13. Fujioka, H.: Stability analysis of systems with aperiodic sample-and-hold devices. Automatica **45**(3), 771–775 (2009)
14. Gao, H., Meng, X., Chen, T., Lam, J.: Stabilization of networked control systems via dynamic output-feedback controllers. SIAM J. Control Optim. **48**(5), 3643–3658 (2010)
15. Hetel, L., Daafouz, J., Tarbouriech, S., Prieur, C.: Stabilization of linear impulsive systems through a nearly-periodic reset. Nonlinear Anal. Hybrid Syst. **7**(1), 4–15 (2013)
16. Hetel, L., Kruszewski, A., Perruquetti, W., Richard, J.-P.: Discrete and intersample analysis of systems with aperiodic sampling. IEEE Trans. Autom. Control **56**(7), 1696–1701 (2011)
17. Heemels, W.P.M.H., Teel, A.R., Van de Wouw, N., Nešić, D.: Networked control systems with communication constraints: tradeoffs between transmission intervals, delays and performance. IEEE Trans. Autom. Control **55**(8), 1781–1796 (2010)

18. Liu, K., Fridman, E., Hetel, L.: Networked control systems in the presence of scheduling protocols and communication delays. SIAM J. Control Optim. **53**(4), 1768–1788 (2015)
19. Le Guernic, C., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Anal. Hybrid Syst. **4**(2), 250–262 (2010)
20. Liu, K., Suplin, V., Fridman, E.: Stability of linear systems with general sawtooth delay. IMA J. Math. Control Inf. **27**(4), 419–436 (2010)
21. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Annual Symposium on Theoretical Aspects of Computer Science, pp. 229–242. Springer (1995)
22. Seuret, A., Peet, M.: Stability analysis of sampled-data systems using sum of squares. IEEE Trans. Autom. Control **58**(6), 1620–1625 (2013)