Firmness analysis of real-time tasks

Firmness Analysis of Real-Time Tasks

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op vrijdag 21 december 2018 om 11:00 uur

door

Amirreza Baghbanbehrouzian

geboren te Tabriz, Iran

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr.ir. A.B. Smolders
1^{e} promotor:	prof.dr.ir. T. Basten
copromotor:	dr. D. Goswami
leden:	dr. Z. Al-Ars (Technische Universiteit Delft)
	prof.dr. P. Eles (Linköpings Universitet)
	prof.dr. K.G.W. Goossens
	prof.dr.ir. J.P.M. Voeten

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Firmness Analysis of Real-Time Tasks

Amirreza Baghbanbehrouzian

Cover design by Yvonne Meeuwsen

Printed by ProefschriftMaken, The Netherlands

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-4659-6

To Zahra and Reza

Acknowledgements

This thesis is the result of team work and the following people directly or indirectly contributed in that

Prof. Twan Basten, my promoter, directed me in this journey. His vision on the bigger picture was the key. Thank you Twan for patiently guiding me through this project and getting me to the desired outcome.

I would like to thank Dr. Dip Goswami, my daily supervisor, for his advice on the technical aspects. The early steps towards this thesis and later shaping it would not have been possible without your contribution.

As always, I felt blessed to have my life-long friend, Hadi Alizadeh, next to me in the past four years. Thank you Hadi for your invaluable support.

A big thanks to Dr. Marc Geilen. You always offered your help despite your busy schedule and the fact that you were not my official supervisor.

I would like to thank Dr. Madjid Nabi who was a great support especially in the beginning of my PhD study.

My heartfelt gratitude goes to my fellow PhD students, Rasool Tavakoli, João Bastos, Bram van der Sanden, Joost van Pinxten, Róbinson Medina Sanchez, Umar Waqas, Juan Valencia and Reinier van Kampenhout. You were like a family to me in the past four years and I am grateful for all your help. Special thanks to Joost for all his technical and social advices. Thanks to Róbinson and João for connecting us and cheering up the group. Thank you Rasool for all the nice time we had together.

Being a member of the ES group has been a great experience. I would like to thank Mladen Skelin for all the cheerful times we spent together. I would also like to thank Marja de Mol, Margot Gordon and Rian van Gaalen for their warm welcome when I just arrived to The Netherlands and joined the ES group. Thank you Marja for taking good care of the whole group.

I would also like to thank my friends at TU/e, Samaneh Tadayon Mousavi and Amir Ghahremani for all the joyful times we had together.

My regards also go to Nelli Troushina. She inspired me with confidence in many tough moments.

I would like to thank my life-long teachers. I am grateful to Dr. Nasser Masoumi, my supervisor at University of Tehran. He trusted in me and involved me in the projects that taught me invaluable lessons in life. A big thank to Mr. Gholamreza Shadi, my 6th grade math teacher and my friend. He taught me to think and live a creative life.

Last but not least, I would like to thank my family. Thank you for forgiving my absence in the family in the past years. Thanks to my mother, Zahra Elmi. If it was not for her and her lessons, I could never have achieved any of these accomplishments. She is always the symbol of hope and wisdom in my life. I hope one day I can be the kind of person that she taught me to be. I am grateful to my father, Reza, who has always supported me. His visits to me have always been a big encouragement for me. Thank you also for your lovely messages. Thanks to my sister Elnaz and my nephew Arsalan Mehdizadeh. Your support always kept me going.

Abstract

Real-time computing is becoming pervasive in industrial and consumer applications. Users require not only valid data but also fast (real-time) response from real-time systems such as autonomous cars. This requires more and faster computing resources, and increases costs. Resource sharing is used to reduce the cost by a tighter resource dimensioning. However, resource sharing leads to interference among real-time tasks. This interference preempts and postpones task executions. A delay due to preemption which is larger than the deadline causes the task to violate timing constraints. Real-time tasks are categorized into *hard*, *soft* and *firm* in terms of their tolerance to Deadline Misses (DMs). DMs may have catastrophic consequences for hard real-time tasks. However, soft and firm real-time tasks allow occasional DMs. As opposed to soft real-time tasks, the number of DMs does not influence the quality of service in firm tasks as long as it is within an acceptable bound. As a category of firm tasks, (m, k)-firm tasks must meet the deadline of at least m jobs, i.e. each instance of a task, in any k consecutive jobs. Firmness analysis verifies the satisfaction of firmness conditions of given (m, k)-firm tasks on processors running under a given scheduling policy. This is the focus of this thesis.

We first address the process of the design of an (m, k)-firm task and its mapping on a shared resource. We illustrate the process of obtaining m and k based on the stability and performance of control feedback loops. Moreover, we propose a Multi-Constraint Resource Allocation (MuCoRA) method for co-mapping of multiple hard and (m, k)-firm tasks with throughput and latency constraints on a multiprocessor platform. The process of resource allocation for real-time tasks consists of two main steps: (i) proposing a candidate mapping for a set of tasks and (ii) verifying the satisfaction of the timing requirements of the tasks in the proposed mapping. The latter includes the schedulability analysis of hard tasks and firmness analysis of (m, k)-firm tasks. The first challenge is extensively studied in previous work. We focus on firmness analysis of (m, k)-firm tasks running under different scheduling policies.

Second, we introduce the balloons and rake problem which abstracts a common core challenge in firmness analysis of a set of periodic tasks running on a shared processor. We propose the Finite Point (FP) method to solve the balloons and rake problem. We use the balloons and rake problem and the FP method in firmness analysis problems. Scheduling policies highly influence the interference among the tasks. We propose approaches to reduce the firmness analysis problem to the balloons and rake problem for different scheduling policies.

Third, we propose the Firmness Analysis (FAn) method. This reduces the firmness analysis problem of sets of asynchronous and synchronous periodic (m, k)-firm tasks running under a Time Division Multiple Access (TDMA) scheduling policy to the balloons and rake problem. The relative release time is unknown in a set of synchronous periodic tasks, as opposed to asynchronous tasks. We consider the scenario where the resource allocation, in terms of allocated slots, causes occasional DMs of a task. We obtain the exact minimum (for a synchronous task set) and maximum (for an asynchronous task set) number of Deadline Hits (DHs) in any k consecutive jobs of an (m, k)-firm task.

Fourth, we consider a set of asynchronous periodic tasks running under a Static-Priority Preemptive (SPP) scheduling policy. We address the problem of adding a new task to an existing set of tasks. This is relevant in mixed-criticality application scenarios where applications are incrementally mapped according to their criticality-level. Therefore, we obtain the first release time of the (m, k)-firm task that corresponds to the maximum number of DHs in any k consecutive jobs.

Finally, we extend the FAn method for an (m, k)-firm task in a set of synchronous tasks running under an SPP policy. We investigate the worst-case relative release times of all tasks that cause a minimum number of DHs in a window of k consecutive jobs of the (m, k)-firm task. The FAn method for synchronous tasks yields a conservative bound on the minimum number of DHs. This result is used as a sufficient condition to verify (m, k)-firmness constraints.

We evaluate the scalability of the FAn method for different scheduling policies with that of existing work and state-of-the-art. Existing work include a reachability analysis on the timed-automata model of the corresponding system using UPPAAL tool and a brute-force search approach on all the possible tasks execution scenarios. State-of-the-art is an MILP-based method. The algorithm used in the FAn method has linear time complexity in number of tasks, hyperperiod of the tasks and number of consecutive jobs, i.e. k. As opposed to the FAn method, the MILP-based method and other state-of-the-art

methods have at least quadratic time complexity in the parameter space under the considered range. Thus, the FAn method has significantly lower run-time. Moreover, in the range of parameters considered in our experiments, designing with deadline misses increases the resource efficiency by 40% in the best case and 19% on average. The resource efficiency comes at a cost of a number of deadline misses that is tolerable under the firmness condition.

As opposed to traditional schedulability analysis with hard real-time deadlines (which is a special case of firmness analysis), the proposed methods in this thesis are a foundation for scheduling with deadline misses constrained by firmness conditions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-time systems refer to software and hardware systems that guarantee response within given timing constraints. These timing constraints are often expressed as deadlines. Real-time applications are becoming pervasive in industrial and domestic applications. Cyber-Physical Systems (CPSs) are one of the main users of real-time systems [39]. A CPS utilizes software solutions to monitor and control one or more physical systems. Examples include interventional X-Ray imaging systems [54] in the healthcare domain, anti-lock brakes [42] in the automotive domain and fly-by-wire systems [40] in the aviation domain among many others.

With technology growing, the users of CPSs require more functionalities of these systems. This results in an increased complexity in software which requires a faster computation and communication hardware at a higher expense. This increases the cost of final products which is against the market requirement. In this context, sharing resources among multiple applications reduces the cost and is of great interest among embedded software engineers. Network sharing and processor sharing are examples. This increases the utilization of a resource. Utilization of a resource is measured by the percentage of time that it is executing a set of tasks assigned to it. The major challenges in a resource efficient design are [6] (i) maximizing the utilization of the resource (and avoid idling of resources) and (ii) assigning as many applications as possible to the resource whilst guaranteeing the satisfaction of their timing requirements. To tackle these challenges, a great amount of investigation is still ongoing. The results presented in this thesis are a foundation to address some of these challenges.

1.1 Multiprocessor systems

Multiprocessor platforms support two or more Central Processing Units (CPU) in a single computer system [26]. Multiprocessing refers to the execution of multiple concurrent processes in a system, with each process running on a separate CPU, as opposed to a single running process at any one instant [61]. The revolution of switching from uniprocessor architectures to the complex parallel structures of multiprocessors occurred because of the limits reached by satisfying Moore's law's greedy demand for computing power density using classic single processor architectures [15].

With the introduction of the multiprocessor hardware systems, software system had to be adapted accordingly. One adaptation was to transform the software to be able to run in parallel, as opposed to fully sequential processing. Software applications were then designed as a set of tasks that allow for parallelism. The next adaptation was to assign tasks to processors. We refer to the latter as mapping of applications to processors. The aim is to speed up processing by providing a mapping that allows parallelism.

1.2 Real-time applications

Real-time applications are distinguished from other software applications by their requirements on correct as well as on-time delivery of outputs [6]. Two notable categories of applications in real-time CPSs are *control* and *streaming* applications [20]. The main distinguishing characteristic between these two types is their temporal constraints: latency and throughput. Streaming applications require to produce a stream of output with a minimum throughput. Video streams with a lower limit on their frame rate and audio streams with a lower limit on their bit rate are examples. Streaming applications are commonly modeled by a set of dependent tasks [67]. In the process of resource allocation to streaming applications on multiprocessor systems, different tasks can be allocated to different processors. This provides the possibility of running tasks in parallel in different processors. Fig. 1.1 illustrates the parallelism in a streaming application which consists of three tasks. Fig. 1.1(a) shows the dependency graph of the application. Circles and arrows show the tasks and dependencies between them, respectively. Fig. 1.1(b) shows the execution of different instances of the tasks on different processors. As shown in the figure, this application can have a maximum parallelism of three if every task is allocated to a separate processor.

Control applications regulate the behavior of physical systems using feed-

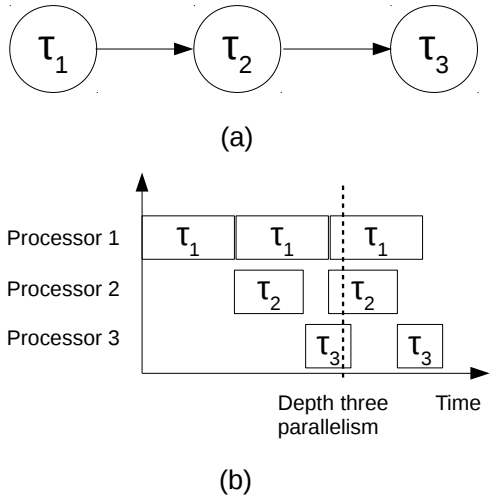

Figure 1.1: Diagram of a streaming application. (a) tasks and dependencies are shown with circles and arrows, respectively. (b) Execution of each instance of a task is shown with a rectangle. The maximum parallelism of depth three can be reached if each task is mapped to a seperate processor.

back loops. They range from a thermostat for regulating a room temperature to an auto-pilot system of an airplane to adjust the altitude and speed. A feedback control loop consists of sensing-computing-actuating actions. A feedback control loop first compares the value or status of process variable being controlled, e.g. room temperature in the thermostat example and altitude in the auto-pilot system, with the desired value or the setpoint. Then, it computes an actuating data – also known as actuating signal – based on the difference between the setpoint and sensing data. Finally, the control loop applies the actuating signal as a control input to the actuator to bring the process variable to the same value as the setpoint. In many complex CPSs, control systems are distributed [34]. In distributed control applications the sensing and actuating actions are conducted in dedicated platforms. Since the computing actions are commonly not computationally demanding, multiple control loops share a processor to conduct their computing actions [34]. The computing action runs control algorithms with the sensor data and produces corresponding input for actuators. Feedback control tasks are activated under time-triggered [75] or

event-triggered [1] [18] policies. Time-triggered control applications are naturally modeled by periodic tasks. To this end, each of the sensing, computing and actuating actions are periodic tasks with the same period. The required access time of each computing task to a processor for being completely executed is referred to as *execution time* of the task. Every sensing data that is ready to be processed has to be processed by the computing task on the processor before the actuating input is sent to the actuator. This time duration is called *execution deadline* of the computing task. Fig. 1.2 illustrates such a setup on an X-Ray imaging system in the healthcare domain [54]. The feedback control loop in the figure adjusts the position of C-arm of the X-Ray imaging system according to the required distance between the X-Ray tube and the patient. The sensor measures this distance and sends the corresponding data to the processor. In the processor, the difference between the sensing data and the setpoint is calculated. Then the actuating data is produced accordingly and sent to the actuator. We are interested in the computing task. The processor may be shared by multiple such controllers. Sharing a processor among multiple control and streaming applications, e.g. the image processing system in Fig. 1.2, introduces inter-application interference issues that are discussed later in this chapter.

1.3 Real-time scheduling

Sharing resources among real-time applications must deal with inter-application interference in order to provide real-time timing guarantees. Scheduling application execution on a shared resource is done by Real Time Operating Systems (RTOS) [33]. Examples include OSEK/VDX [53] and Autosar [48] for automotive systems, RT-POSIX [50], OSE [51] and VxWORKs [59] for general-purpose applications. Strategies to decide the execution order of different tasks must be predictable in order to guarantee the satisfaction of the timing requirements of real-time applications. Scheduling policies are used for this purpose. When multiple tasks request for the same resource, the scheduling policies decide which task will get access to the resource. Some of the commonly used policies are described in the following.

1.3.1 Time division multiple access (TDMA) scheduling policy

A TDMA policy is a budget scheduler that defines a periodic working cycle, called *time wheel*, for resources. Each time wheel is divided into a number of

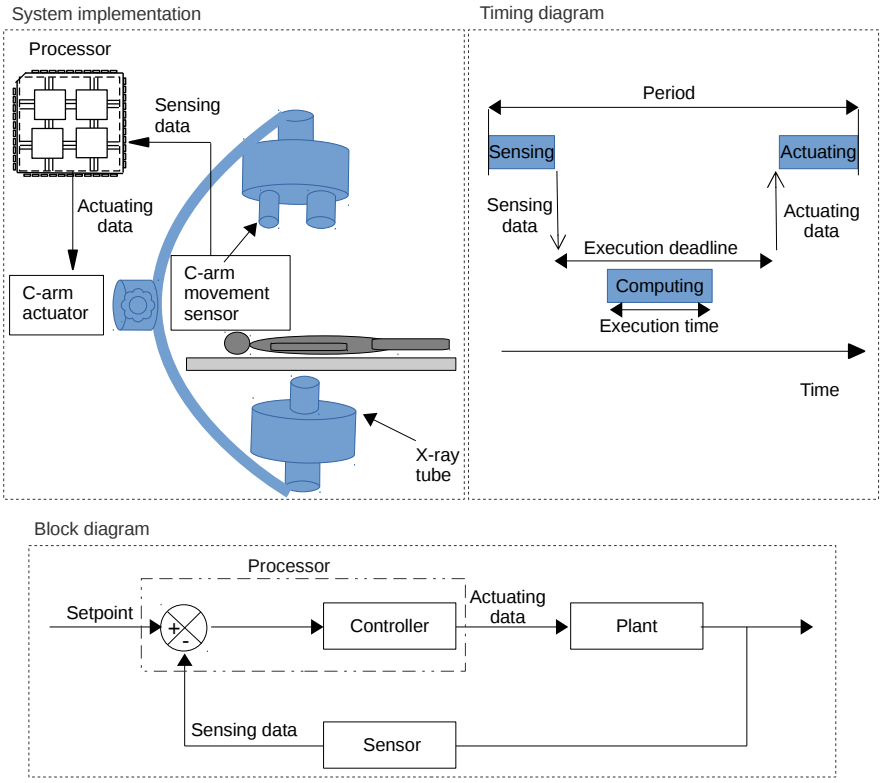

Figure 1.2: Feedback control loop of the controller of the C-arm position on an X-Ray imaging system. Sensing and actuating operations are conducted on dedicated hardware. However, the computing task is executed on a shared processor.

time slots. Usually, each time slot is assigned to only one task. Fig. 1.3 illustrates a time wheel with 10 time slots. The tasks τ_1, τ_2 and τ_3 are assigned to 2, 1 and 2 time slots, respectively. For example, the OSE [51] RTOS supports TDMA scheduling.

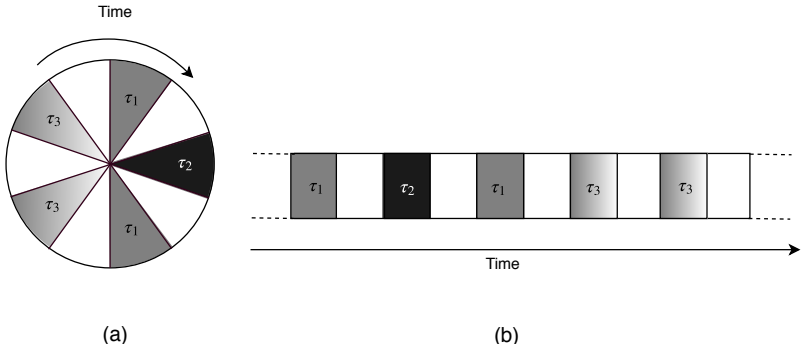

Figure 1.3: A TDMA time wheel with 10 time slots. TDMA time wheels are typically shown (a) with a circle that rotates with time and (b) horizontally corresponding to a time axis. In this example, three applications τ_1, τ_2 and τ_3 are assigned to different time slots.

1.3.2 Priority-based scheduling policy

In a priority-based policy, also known as event-driven, priorities are assigned to tasks. Based on this priority an RTOS decides which task will execute at any given time. The priority assignment can be done either statically or dynamically. In static priority schedules, fixed priorities are assigned to the tasks at *design time*, i.e.,before the implementation of a software on a platform. The priorities are often assigned based on the criticality, period and workload of the tasks. The Rate Monotonic (RM) [43] algorithm is one of the strategies used to assign priorities to tasks. This algorithm assigns a higher priority to a task with a shorter period. In dynamic-priority scheduling, the priorities of tasks are decided at *run-time*, i.e.,while applications are running on a hardware platform. The Earliest Deadline First (EDF) [24] algorithm is the most widely used dynamic-priority policy for real-time tasks. Under EDF, a task with an earlier deadline gets a higher priority. Autosar [48] supports static priority scheduling and OSE [51] supports both static and dynamic priorities.

1.3.3 Preemptive and non-preemptive scheduling

Scheduling policies can be preemptive and non-preemptive. In preemptive policies a task execution is preempted once a higher priority task is activated. In non-preemptive policies, a task completes its execution once started. Even

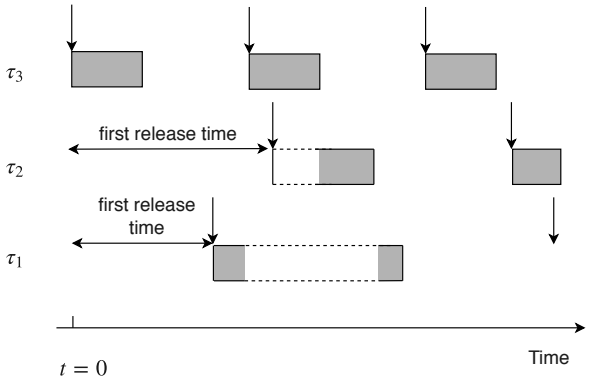

Figure 1.4: Three periodic applications running under an SPP scheduling policy. τ_3 and τ_1 have the highest and lowest priorities, respectively.

though task preemption imposes overheads, it has advantages that justify its usage in many RTOSs, e.g. RT-POSIX [50], OSEK/VDX [53] and Autosar [48]. Every preemption of a task requires to save the program execution state in memory [20]. The time spent on saving this data on the memory is part of a more general notion of *context switching overhead* introduced by preemptions [20]. Besides, a high number of preemptions increases the number of cache misses, i.e.,a failed attempt to read data from or write data to the processor cache, resulting in a main memory access which has a much higher delay [63]. However, preemptive scheduling increases the CPU utilization [20]. It also reduces the waiting time and response time of a task execution which implies a better schedulability. All in all, the degree of preemption is traded with the system schedulability. Different scheduling policies can be preemptive, e.g. priority-based and TDMA policies, and non-preemptive, e.g. first-come-first-served and priority-based policies [6]. Fig. 1.4 illustrates an instance of the execution of three periodic tasks τ_1, τ_2 and τ_3 under a Static-Priority Preemptive (SPP) schedule. Here, τ_3 and τ_1 have the highest and the lowest priorities, respectively. As shown in the figure, the execution of τ_1 for example is preempted once τ_3 is activated. RT-POSIX [50], OSEK/VDX [53] and Autosar [48] support SPP policies. OSEK/VDX [53] supports non-preemptive static priority-based scheduling.

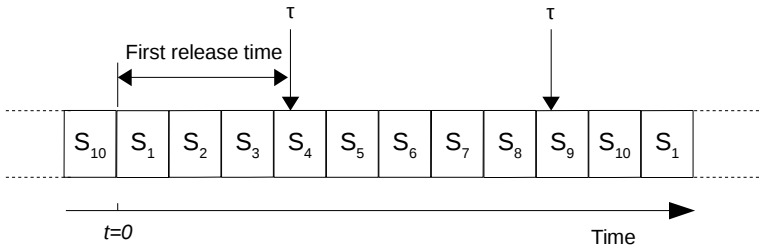

Figure 1.5: First release time of a task running under an asynchronous TDMA policy.

1.3.4 Synchronous and asynchronous scheduling

Based on the predictability of the relative activation time of tasks on a schedule, scheduling policies are divided to two categories: synchronous and asynchronous. The relative release time of tasks running under asynchronous schedules are fixed, as opposed to synchronous schedules[1]. The relative release time of tasks in an asynchronous TDMA policy is specified by the *first release time* of the tasks relative to the beginning of a TDMA time wheel. Fig. 1.5 illustrates the first release time of the task τ under a TDMA policy. In this example, the time reference $t = 0$ is the beginning of a TDMA time wheel. The relative release time of a set of tasks running under an asynchronous SPP policy is specified by the first release time of the tasks relative to a time reference, e.g. the first release time of the task with the highest priority which is assumed to be at $t = 0$. Fig. 1.6 illustrates the first release times in a set of three tasks running under an asynchronous SPP policy. In this task set, τ_3 has the higher priority and τ_1 has the lowest priority. The first release time of τ_3 is considered as the time reference $t = 0$.

1.3.5 TDMA vs SPP scheduling policies

Even though in many systems RTOSs and therefore the underlying scheduling policies are predefined to match the platform architecture, TDMA and SPP policies are preferred based on the application level requirements. The exe-

[1]We follow the definition in [23] which is opposite to the ones reported in [14][13]. The naming considers the fact that a set of synchronous tasks can be released simultaneously as opposed to asynchronous tasks that are released with given offsets.

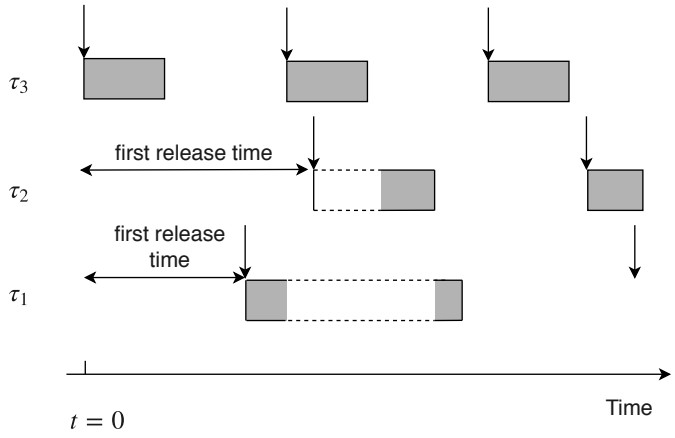

Figure 1.6: First release times of a set of tasks running under an asynchronous SPP policy. The first release time of the tasks with the highest priority, i.e.,τ_3, is considered to be the time reference $t = 0$.

cution requirements of applications with throughput constraints are usually translated to a time budget in a work cycle of the processor [66]. This budget can be translated to time slots in a TDMA time wheel. Besides, applications with latency constraints and periodic releases are preferred to be scheduled under SPP policies. This corresponds to the fact that if periodic applications are running under a TDMA policy, allocated and utilized resources are not necessarily the same. That is, an allocated slot might occur before a new release of a task and after finishing the previous release of the task. This commonly occurs due to the fact that the allocated slots are chosen based on the worst-case alignment between the task releases and the TDMA time wheel. This issue is illustrated in Fig. 1.7. Let the task τ with a period shown in the figure be completely executed with a processing time equal to one full TDMA slot. The time wheel has 10 time slots. Assuming that the deadline of τ equals its period, it requires at least two time slots allocated in each time wheel to guarantee meeting the deadlines of all possible releases. Allocating S_3 and S_7 to the task provides such a guarantee. In the instance shown in the figure, the release of τ_1 at the beginning of the time slot S_3 is fully executed in S_3. Therefore, the slot S_7 does not execute this release of τ. Note that allocation of S_7 to τ is necessary, because otherwise a release of τ at the beginning of S_4,

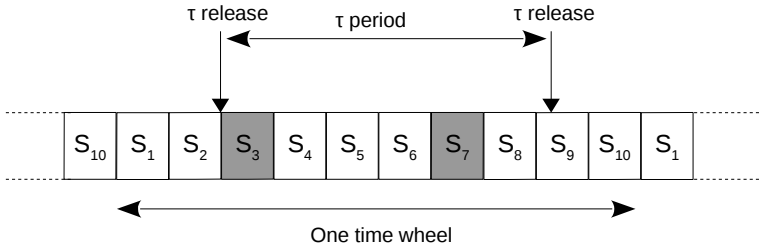

Figure 1.7: An instance of a mismatch between allocated and utilized resources in a TDMA time wheel. The release of τ at the beginning of the slot S_3 is executed in S_3. Therefore S_7 does not execute τ.

for example, will result in a deadline miss. An SPP schedule is more efficient for periodic tasks as there is no notion of allocated resource. A TDMA policy is also common for co-mapping of streaming and control applications [30]. This corresponds to the fact that streaming applications usually require more processing time.

1.4 Deadline misses in real-time tasks

Each instant of a periodic task is called a *job*. The *execution deadline* is a time bound for execution of a job starting from its release, i.e.,the task activation. Fig. 1.2 shows the execution deadline for the computing task of the C-arm controller of a X-Ray imaging system. This deadline is decided based on the release of the actuating task. Other reasons might play a role in deciding the deadline of periodic tasks. One typical reason for missing deadlines in real-time tasks is that there is not enough access of a task to a resource before its deadline. If the preemption of a task (in both TDMA and SPP) is *too long*, it results in Deadline Missed jobs (DMs). Real-time applications are divided into three main categories in terms of their tolerance to deadline misses: hard, soft and firm real-time tasks. Deadline misses have catastrophic effects on hard tasks as opposed to soft and firm tasks. Soft and firm tasks satisfy the requirements of a system even with *some* occasional deadline misses. Deadline misses reduce the quality of service of soft tasks. However, the quality of service of firm tasks is not affected by the number of deadline misses as long as this number is within an acceptable bound. Hamdoui et al. [35] specify this bound

for the number of deadline misses by introducing the notion of (m, k)-firmness constraints. That is, an (m, k)-firm task satisfies the requirements of a system if it meets the deadline of at least m jobs in any k consecutive jobs of the task. Note that some tasks are inherently soft but may be treated as firm tasks by defining a lower bound for their required quality of service that corresponds to a certain number of deadline misses. For example, in feedback control loops, deadline misses reduce the Quality of Control (QoC). We treat them as firm tasks and obtain the (m, k)-firmness bound for a given QoC requirement.

If a task can tolerate some deadline misses, the scenario above, i.e.,the lack of resources is the cause of DMs, can potentially be considered as an opportunity to leverage scarce resources. That is, a task tolerant to some DMs can potentially be mapped to a resource that has otherwise insufficient capacity. This not only allows us to map more tasks to a resource but also increase the average resource utilization. The latter corresponds to the fact that with deadline misses we can design for the better-than-worst-case scenario which usually happens (at run-time) more often than the worst-case scenario.

1.5 Feasibility and firmness analysis

A real-time schedule must guarantee the satisfaction of the timing requirements of all tasks. Feasibility analysis (also known as feasibility test and schedulability analysis) verifies the satisfaction of the timing requirements of hard real-time tasks under a scheduling policy. That is, a schedule is feasible for a task if and only if it meets all its deadlines. The feasibility analysis fundamentally differs for synchronous and asynchronous schedules. The main challenge in feasibility analysis of a periodic task under a synchronous schedule is the prediction of the worst-case timing for a job release. In a synchronous SPP policy, for example, this worst-case has correlation with the relative release time of the task and all the tasks with higher priorities than the task under analysis. This worst case scenario happens when the task under feasibility test and all the tasks with higher priorities are released simultaneously. This instance of the scheduling problem is known as the *critical instance* [23]. If a task meets its deadline in the critical instance, it certainly meets its deadline in any other scenario.

Firmness analysis verifies the satisfaction of (m, k)-firmness conditions. That is, it obtains the maximum possible number of deadline misses in any window of k consecutive jobs of an (m, k)-firm task. If this number is less than $k-m$, the task satisfies the (m, k)-firmness condition. The worst-case scenario that is considered in the feasibility analysis does not necessarily have any corre-

lation with the maximum possible number of deadline misses in any window of k consecutive tasks of an (m, k)-firm task. In fact, feasibility analysis obtains the worst-case scenario for one job release of a task. The firmness analysis considers the worst-case release for k consecutive jobs of a task. Therefore, feasibility analysis and firmness analysis do not have any correlation. A part of this thesis addresses the firmness analysis of an (m, k)-firm task under SPP and TDMA policies with both synchronous and asynchronous task sets. We refer to a schedule that is positively verified by feasibility analysis and firmness analysis as a *positive schedule.* The state-of-the-art for firmness analysis can be summarized as follows:

- Firmness analysis exists for the asynchronous SPP policy [13] [37] [71] [76]. However, this body of work assumes that the first release time for all the tasks in a schedule, including the task under firmness analysis, is given.

- Firmness analysis is proposed recently for the synchronous SPP policy [69]. This solution is based on the Mixed Linear Integer Programming technique. As shown in Chapter 5, this method is not scalable with the parameter k.

- Firmness analysis for TDMA policies was not addressed in any work prior to the work presented in this thesis.

- Multiprocessor mapping techniques taking into account (m, k)-firmness conditions have not been developed prior to the work presented in this thesis.

1.6 Problem statement and research challenges

The problem being dealt with in this thesis can be summarized as follows:

Achieving scalable and resource-efficient mapping and feasibility/firmness analysis methods for a set of hard and (m, k)-firm tasks running under SPP and TDMA scheduling policies.

We aim for a resource efficient design. Such a design is characterized by high utilization of a resource and mapping more tasks on it compared to traditional designs. The current mapping methods for hard real-time tasks result in a

considerable utilization gap between the worst-case scenario and the typical-case scenario of task execution. We aim to design real-time systems for better-than-worst-case by allowing some occasional deadline misses. We consider a setup that contains both hard and (m, k)-firm tasks. Two frameworks are required: (i) mapping and (ii) feasibility and firmness analysis. The former must consider the hard and (m, k)-firm nature of the task along side the type of constraint (latency constraint or throughput constraint) of the tasks and scheduling policy enforced by the operating system on the platform. Moreover, the mapping solution must be resource efficient and less conservative than the solution obtained by the state-of-the-art methods. The latter must guarantee that the number of deadline misses of every hard task is zero and of every (m, k)-firm task is at most $k - m$. Considering the fact that the firmness analysis of each task is conducted multiple times in the process of obtaining a positive schedule for a set of tasks, the proposed method must have an acceptable run-time. The run-time of the method must also scale well to the expected range of system parameters. The following challenges must be addressed.

Challenge A. *Translating application requirements into (m, k)-firmness conditions.*

Deadline misses decrease the quality of control in control tasks. An excessive number of deadline misses can result in an unstable system. In order to obtain a bound for the acceptable number of deadline misses, the stability and performance of a system must be considered. On one hand, the stability of control applications are sensitive to the consecutiveness of deadline misses. On the other hand, the performance of such applications are influenced by the overall number of deadline misses in long runs. The proposed method, therefore, must consider different scenarios for the allowed number of deadline misses.

Challenge B. *Development of a resource-efficient mapping solution for a set of tasks with hard and (m, k)-firm deadlines and different constraints, i.e.,throughput and latency constraints.*

The aim is to increase the utilization of a resource. We consider co-mapping of both tasks with throughput constraints, i.e.,streaming applications, and latency constraints, i.e.,feedback control loops. Given that these tasks are traditionally analyzed using different models, separate design flows are commonly followed to map them to a processor. The proposed method therefore must combine these flows. Moreover, we consider (m, k)-firmness constraints for

the tasks with latency constraints to increase the utilization of the resource. Therefore, the proposed mapping method must prioritize the resource allocation step for these types of the applications. Moreover, the proposed method must still be applicable if any type of task is missing in a task set that is intended to be mapped to a resource.

The proposed method must obtain a near-optimal positive schedule in a time acceptable for design time. We define the optimality of the resource allocation method in terms of the number of tasks that are mapped to the resource. The mapping process usually involves several iterations between obtaining a candidate mapping and feasibility/firmness analysis before converging on a positive schedule.

Challenge C. *Firmness analysis of (m, k)-firm tasks running under SPP and TDMA policies.*

Once an (m, k)-firm task is mapped to a resource, the satisfaction of its firmness conditions must be verified. A firmness analysis must consider the number of deadline misses in a window of k consecutive jobs of a task. This analysis fundamentally differs for different scheduling policies and for tasks that are synchronous or asynchronous. In a TDMA policy, each task can be analyzed separately. In TDMA, a task preemption occurs due to the unavailability of the allocated slots at a certain time. In such a case, the relative time between a task release and the TDMA cycle must be considered. However, in an SPP policy, the workload of tasks with higher priorities than the task under consideration causes preemption in the task execution. Therefore, firmness analysis must consider the relative release time among tasks.

If a task set is asynchronous, we aim to obtain the relative release times for (m, k)-firm tasks which result in the maximum number of deadline hits in any k consecutive jobs. However, if tasks are synchronous, we obtain the minimum possible number of deadline misses in any k consecutive jobs of an (m, k)-firm task. This corresponds to the fact that any relative release time among tasks – in SPP – and between each task and processor work cycle – in TDMA – may occur.

1.7 Contributions

Our contributions to address the above challenges are described in the following.

Contribution 1: *Robust co-synthesis of embedded control systems with (m, k)-firmness conditions.*

This contribution addresses *Challenge A*. Feedback control applications are robust to occasional deadline misses. This opens up the possibility of saving scarce (computation and communication) resources on embedded platforms. Stability and performance requirements of a control loop impose restrictions on acceptable patterns of deadline misses (e.g., not too many misses in a row). Such requirements are captured by (m, k)-firmness conditions. (m, k)-firm design requires (i) representation of stability and performance requirements in terms of (m, k)-firm deadlines (ii) controller synthesis taking into account the (m, k)-firmness parameters (iii) schedule analysis to verify guarantees on meeting the firmness conditions. We present a co-synthesis framework for these three design components and illustrate its applicability with examples. This contribution is explained in Chapter 2 and based on the following publication.

[8] A. Behrouzian, D. Goswami and T. Basten. Robust co-synthesis of embedded control systems with occasional deadline misses. *In 24th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS).* IEEE, 2018.

Contribution 2: *Multi-constraint multiprocessor resource allocation*

This contribution addresses *Challenge B* by proposing a Multi-Constraint Resource Allocation (MuCoRA) method for applications from multiple domains onto multiprocessors. In particular, we address a mapping problem for multiple throughput-constrained streaming applications and multiple latency-constrained feedback control applications onto a multiprocessor platform running under a TDMA policy. The control applications are both hard and (m, k)-firm real-time tasks. The main objective of the proposed method is to reduce resource usage while meeting constraints from both these two domains (i.e., throughput and latency constraints). We show by examples that the overall resource usage for this mapping problem can be reduced by distributing the allocated resource (i.e., TDMA slots) to the control applications over the TDMA wheel instead of allocating consecutive slots. MuCoRA is explained in Chapter 2 and is based on the following publication.

[9] A. Behrouzian, D. Goswami, T. Basten, M. Geilen and H. Alizadeh Ara. Multi-constraint multiprocessor resource allocation. *In International Conference on Embedded Computer Systems: Architectures, Modeling, and Simula-*

tion (SAMOS). IEEE, 2015.

Contribution 3: *Introduction of and solution to the balloons and rake problem*

In order to provide a framework to address *Challenge C*, we introduce the balloons and rake problem. This problem abstracts the core challenge in the firmness analysis problems. It essentially poses the question how many balloons in certain given periodic pattern can be hit simultaneously by a rake. We propose the Finite Point (FP) method to solve the balloons and rake problem. We illustrate other potential applications for the balloons and rake problem in other engineering domains. Chapter 3 explains the balloons and rake problem. This contribution is based on part of the following publication.

[11] A. Behrouzian, H. Alizadeh Ara, M. Geilen, D. Goswami, and T. Basten. Firmness analysis of real-time tasks (under review).

Contribution 4: *Firmness analysis of (m, k)-firm tasks running under a TDMA policy*

This contribution addresses *Challenge C* for the TDMA policy. We propose a Firmness Analysis (FAn) method for verification of firmness conditions of periodic (m, k)-firm tasks running on a shared TDMA-scheduled processor. We particularly consider DMs arising from processor sharing. Based on the available processor budget for any job that is ready for execution, we use the results of the balloons and rake problem and the FP method to quantify the maximum possible number of DMs. These results are used to guarantee the worst-case scenario for a synchronous task set. The FP method is also used to obtain the relative release time of asynchronous tasks which results in the maximum number of deadline hits in any k consecutive jobs of (m, k)-firm tasks. The FP method is further generalized using a timed-automata model to consider variation in the period of tasks. The UPPAAL tool is used to validate and analyze the timed-automata model. Considering all possible relative release times of tasks, a brute-force approach verifying the maximum number of DMs is proposed to evaluate the FAn method. The FAn method and the brute-force method give an exact bound on the number of DMs, whereas the timed-automata analysis provides a conservative bound. The methods are evaluated considering a realistic case study. Scalability analysis of the FAn method shows an acceptable verification time for different sets of parameters. This contribution is explained in Chapter 4 and is based on the following

publication.

[12] A. Behrouzian, D. Goswami, M. Geilen, M. Hendriks, H. Alizadeh Ara, E. P. Van Horssen, W. P. M. H. Heemels and T. Basten. Sample-drop firmness analysis of TDMA-scheduled control applications. *International Symposium on Industrial Embedded Systems (SIES).* IEEE, 2016.

Contribution 5: *Firmness analysis of* (m, k)*-firm tasks running under SPP policies*

This contribution addresses *Challenge C* for SPP policies. We provide firmness analysis of an (m, k)-firm task that is intended to be added to a set of asynchronous tasks scheduled under an SPP policy. We extend the FAn method to obtain a synchrony that results in the maximum number of deadline hit jobs in any k consecutive jobs of the task. Thus, our work lifts the limiting assumption in existing work that the first release time is known [13] [37]. The scalability of FAn is compared with that of existing work – a brute-force search approach – and a timed-automata model of the problem that is analysed using the reachability check of the UPPAAL model checker. The FAn method substantially reduces the complexity of the analysis. Addressing *Challenge C* for a synchronous SPP policy, we extend the FAn method to translate the firmness analysis problem of a set of synchronous tasks running under a synchronous SPP policy to the balloons and rake problem. Using the FP method we then obtain a lower bound for the minimum number of DMs in any k consecutive jobs of a task. The scalability of the FAn method is compared with that of a timed-automata approach, a brute-force approach and a Mixed Integer Linear Programing (MILP)-based method. The FAn method scales substantially better to firmness analysis problem instances with a large k and a high number of tasks. Firmness analysis for the SPP policies is presented in Chapter 5 and the following publications.

[10] A. Behrouzian, D. Goswami, T. Basten, M. Geilen, H. Alizadeh Ara and M. Hendriks. Firmness analysis of real-time applications under static-priority preemptive scheduling. *In Proceedings of Real-Time and Embedded Technology and Applcations Symposium (RTAS).* IEEE, 2018.

[11] A. Behrouzian, H. Alizadeh Ara, M. Geilen, D. Goswami, and T. Basten. Firmness analysis of real-time tasks (under review).

1.8 Thesis outline

The remainder of this thesis is organized into five chapters. Chapter 2 discusses the design of an (m, k)-firm task considering the performance and stability of a feedback control loop. This chapter also proposes a framework for co-mapping of a set of real-time tasks with multiple constraints running under a TDMA schedule. Chapter 3 introduces the balloons and rake problem and the finite point method to solve the problem. Chapter 4 translates the firmness analysis problem of (m, k)-firm tasks running under a TDMA policy to the balloons and rake problem and obtains the minimum and maximum possible number of deadline hits in any k consecutive jobs of a task. Chapter 5 provides a framework to translate the firmness analysis problem of set of synchronous and asynchronous tasks running under an SPP scheduling policy to the balloons and rake problem. Chapter 6 provides the final conclusions and directions for future work.

Chapter 2

Resource allocation with deadline misses

This chapter addresses the problem of resource allocation to multiple real-time applications on a multiprocessor system. We aim to assign as many real-time applications as possible to the processors. In particular, we enforce design with deadline misses and gain from the fact that more applications can be assigned to a processor if they can tolerate occasional deadline misses. This chapter discusses design and resource allocation of multiple hard and (m, k)-firm tasks in a set of multi-constraint (a combination of throughput and latency constraints) real-time tasks. It serves as an illustration of the use and need of designing with deadline misses, motivating the (m, k)-firmness analyses developed in the remainder of this thesis. We first introduce the model of real-time tasks considered in this thesis. Next, we illustrate the process of designing m and k parameters for feedback control applications considering their performance and stability[1]. Subsequently, inspired by the hardness, i.e. the ability to tolerate DMs, and temporal model of tasks, we propose a hierarchical mapping strategy[2]. The multi-constraint resource allocation method maps multiple real-time tasks consisting of tasks with hard and (m, k)-firm deadlines on a multiprocessor system running under a Time Division Multiple Access (TDMA) scheduling policy. This method provides a resource efficient mapping while reducing the number of iterations between obtaining a candidate mapping solution and its schedulability. It also guarantees the satisfaction

[1] These results are based on [8].
[2] These results are based on [9].

of (m, k)-firmness conditions using the FAn method which is elaborated in Chapter 4.

2.1 Motivation

2.1.1 Multi-constraint scheduling

Many application domains including automotive and healthcare systems require to run multiple real-time tasks simultaneously with stringent requirements on their timing and performance to ensure correct functionality. Often, these applications impose different domain-specific constraints for the correct functional behavior at the system-level. Examples include interventional X-Ray (iXR) systems where precision motion control applications for patient support may execute together with compute-intensive image processing applications (see Fig. 2.1). On the one hand, the precision of the motion controller is critical for patient safety and on the other hand, the image quality after the processing is crucial for clinical purposes. The design of such systems requires to meet different types of constraints. Generally, the quality requirements of streaming applications can be translated into minimum throughput constraints. In a video stream, the required frame rate is the throughput of the system. Similarly, the QoC requirement of a feedback control application can be translated into a constraint on the maximum sampling period [31]. Further, depending on the model of the feedback control loops, the constraint on the sampling period can be expressed as a maximum latency allowed for the corresponding tasks, i.e. a deadline for the task. Thus, the scheduling problem deals with a combination of applications with throughput and latency constraints.

System-level functionality requires to meet both constraints at the application level. In this process, not only performance of individual applications needs guarantees, but performance of one application should not directly influence that of others. Achieving such system-level requirements on predictability or timing is particularly challenging on shared platforms due to interference between applications through resource accesses. A budget scheduler provides temporal predictability on a shared resource by guaranteeing a fixed access time for every scheduled application [77]. Time Division Multiple Access (TDMA) is a common scheduling policy for realizing temporal predictability for such applications [31] [46]. It allocates identical constant time slots to applications in a work cycle.

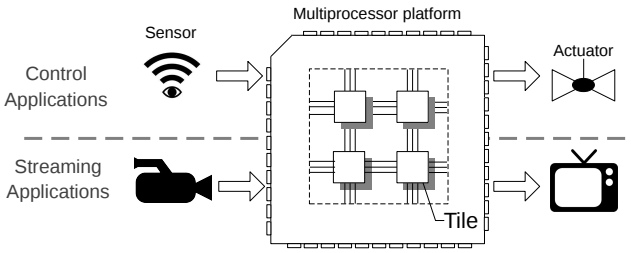

Figure 2.1: Setup under consideration: latency-constrained feedback control applications sharing processors with throughput-constrained streaming applications.

2.1.2 Designing with deadline misses

Due to the safety-critical nature of control applications, timing plays a key role in guaranteeing their Quality of Control (QoC) [2]. Running control applications on a shared processor for computation reasons can cause control samples to miss their computational deadline. This affects the QoC. A sample should be processed before the next sample arrives and therefore, each sample has a computational deadline less than or equal to the sampling period of the application. The samples with missed deadlines are referred to as Deadline Misses (DMs). A potential reason for a sample to miss a deadline can be that insufficient resources are available for the application when the sample is ready for processing. Under an (m, k)-firmness condition [35], however, a control application can still satisfy QoC requirements in presence of DMs. That is, at least m samples out of any k consecutive samples must meet the computational deadline to satisfy application level requirements. This can be used to design a system with tighter resource dimensioning by allowing some deadline misses.

2.1.3 Key observations and our contributions

The goal is to efficiently allocate resources to a given set of multi-constraint tasks on a multiprocessor system. Each processor is running under a TDMA policy. Therefore, the resource allocation process involves binding tasks to processors and assigning TDMA time slots to each task. Tasks may be of the following types: control feedback loops with either hard deadlines or (m, k)-firmness conditions and streaming applications with a hard lower bound on their throughput. The strategy follows the following observations:

- O1: The slots allocated to the feedback control loops with either hard or (m, k)-firm deadlines, modeled with periodic tasks, should be distributed as much as possible. This helps to reduce the number of time slots required for a task to satisfy its timing requirement (see Section 2.4.2).
- O2: Streaming applications, modeled and analyzed using data flow graphs, require a budget (instead of specific time slots) in terms of the percentage of each work cycle. This budget is translated to a number of slot. These slots can be chosen arbitrarily (see Section 2.4.2).
- O3: An (m, k)-firm task is allowed to have between 0 and $k-m$ deadline misses in any k consecutive executions while still satisfying the requirements of the system (see Section 2.2).
- O4: Deadline misses affect the performance and stability of a feedback control loop. Burst deadline misses jeopardize the stability of the system. As long as the system is stable, performance of the system is sensitive to the number of deadline misses in longer runs. This can be translated to (m, k)-firmness conditions (see Section 2.2).

We propose a resource allocation method that efficiently obtains a feasible schedule. We utilize the observations above to obtain a resource allocation flow for all tasks. Given Observations O1 and O2, we first bind and allocate feedback control loops with hard deadlines, because this provides maximal allocation freedom for these applications. Next, streaming applications are bound to the processors, because streaming applications may need substantial processing resources. To this end, we obtain the budget (in terms of number of slots) required for the streaming applications. Inspired by O2, we reserve those numbers of slots for streaming applications; however, we do not allocate any specific time slots to them yet. Instead, we proceed with slot allocation for (m, k)-firm tasks (considering the budget that must be reserved for streaming applications). Inspired by O4, we obtain two (m, k)-firmness conditions, one

addressing stability and the other for performance, for each feedback control loop. These two (m, k)-firmness conditions allow us to have a considerable number of deadline misses in a longer window (in view of the performance requirement) while not being overly restrictive for short runs. Then, inspired by O1 and O3, we efficiently allocate time slots to (m, k)-firm tasks distributing them as much as possible. Finally, we allocate the remaining time slots to the streaming applications as per the percentage.

The rest of this chapter is organized as follows. We first describe a method to translate the stability and performance requirements of a feedback control loop to (m, k)-firmness conditions. We explain Observation O4 mentioned above in detail. We formulate the setup under consideration in this chapter. Next, in Section 2.4.2, we explain our mapping flow – outlined above – in detail. In Section 2.5, we evaluate our proposed method with a realistic case study. Section 2.6 discusses related work. Finally, we summarize the key messages of this chapter.

2.2 (m, k)-firm task design

2.2.1 (m, k)-firmness in controller design

Allowing deadline misses in feedback control implementations is increasingly getting attention due to its potential to save (computation and communication) resources in embedded settings [35] [73]. This is an interesting use case since control systems are inherently robust to occasional deadline misses. The design of a feedback controller is mainly concerned about stability (e.g., system states are bound) and performance (e.g., how fast the system reaches the reference signal). Both stability and performance are influenced by the number of deadline misses over a window of consecutive executions of control computation jobs. Typically, such timing requirements are well captured by (m, k)-firm deadline specifications implying that at least m jobs must meet deadlines in any window of k consecutive jobs. Generally, a lower k, a short time window, is required for stable controller synthesis. Performance is evaluated over a longer window, i.e., a higher k. We represent both stability and performance (m, k)-firmness conditions in controller design and – later in this chapter – we present a scheduling analysis framework for guaranteeing both these conditions in an implementation.

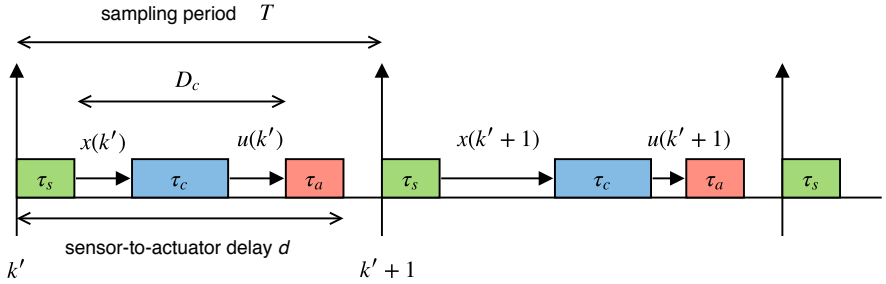

Figure 2.2: Control tasks and deadlines. The execution of τ_s (sensing), τ_c (computing) and τ_a (actuating) tasks.

2.2.2 Embedded control and deadlines

Linear state-feedback control

Feedback control applications regulate the behavior of dynamic systems (plant). A controller is realized by performing three sequential operations – sensing (reading plant states using sensors), computing (computing control actions) and actuating (applying a control signal to the plant). In an embedded implementation, these operations are performed as tasks – τ_s (sensing), τ_c (control computing) and τ_a (actuating). A control algorithm is realized by periodically repeating these three operations as shown in Fig. 2.2. Sampling period T is defined as the time interval between the start of execution of two consecutive τ_s jobs. Sensor-to-actuator delay d is defined as the time from the start of τ_s to the completion of τ_a within one sampling period. Usually, both T and d should be constant over the sampling periods. Overall system dynamics with sampling period T and delay d (for linear systems) is given by [16],

$$S_h : x(k'+1) = A_{T,d}x(k') + B_{T,d}u(k'), \tag{2.1}$$

where $A_{T,d}$ and $B_{T,d}$ are system and input matrices capturing delay and $x(k')$ are system states in the k'^{th} sampling period. The controller is $u(k')$ and the general form of a state-feedback controller is given by,

$$u(k') = Kx(k'), \tag{2.2}$$

where K is the feedback gain to be designed.

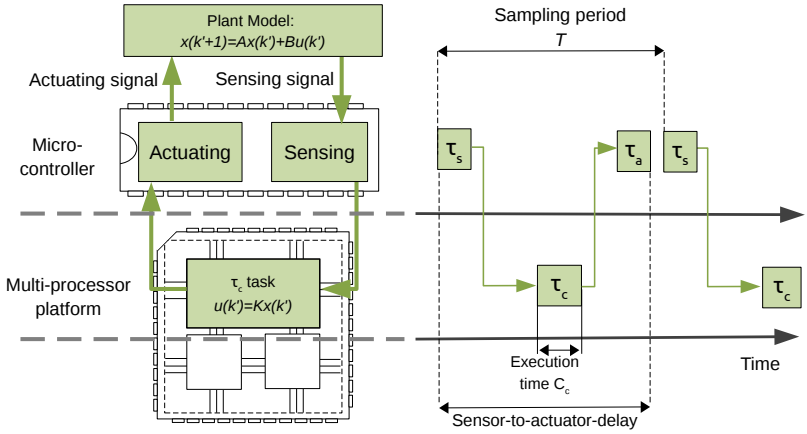

Figure 2.3: Implementation and timing diagram of feedback control loop. Sensing and actuating tasks are performed in dedicated hardware which are commonly micro-controllers.

Implementation architecture

Constants T and d are typically achieved by time-triggered executions of the τ_s and τ_a tasks. That is, τ_s and τ_a are triggered strictly periodically with period T and the offset between them is equal to deadline D_c of the task τ_c – illustrated in Fig. 2.2. τ_s and τ_a are typically executed on a separate micro-controller in a time-driven fashion. τ_c runs on a shared processor with other tasks. Fig. 2.3 illustrates this setup. The deadline D_c can be computed as,

$$D_c = d - e_s - e_a - \epsilon \tag{2.3}$$

where d is the sensor-to-actuator delay, e_s and e_a are execution times of τ_s and τ_a, and ϵ is any other delay caused by the communication between the tasks. Such setting is common in distributed implementations [32].

Deadline misses

To cope with a deadline miss (by a τ_c job), a control application may adopt various policies and the controller $u(k')$ should be designed accordingly [**?**]. There are mainly two choices for a deadline-missed control computation job: (i) setting $u(k') = 0$ (ii) using the old input, i.e., $u(k') = u(k' - 1)$. We

denote a control computation job that misses its deadline by DM and one that meets (hits) its deadline by DH. If a DM can be detected immediately after it is activated, the control computation job can be aborted, the old control signal can be held using a Zero-Order-Hold mechanism and thus, the processor utilization can be reduced. Detecting a DM is for example possible if the control computation task τ_c is executed on a shared processor with a TDMA policy. If execution time C_c of τ_c exceeds the (remaining) allocated computation budget on the shared processor until the deadline D_c, then τ_c will miss its deadline. Firmness analysis for detectable DM and DH jobs of a task running under a TDMA schedule is presented in Chapter 4. We consider such a task model where a DM is detected and aborted as soon as τ_c is released and the old control input is held. When a DH job is detected immediately after its activation, the execution of τ_c is continued in the upcoming sampling period. Subsequently, the control signal $u(k')$ is updated and passed on to τ_a. On the other hand, in the case of a DM, the τ_c job is aborted in the following sampling period, the processor is freed up and the old $u(k')$ is held. This is illustrated in Fig. 2.4.

Modeling deadline misses

Fig. 2.4(a) shows an example of three executions of control computation jobs, $DH \rightarrow DM \rightarrow DH$. Since the DM is not executed and the old control signal is held, the effective sampling period is the time between the start of two consecutive DH jobs. Hence, the sampling period for this pattern of deadline misses is $2T$ and the delay is given by $d_2 = T + d$. Overall system dynamics is denoted by,

$$S_{m,1} : x(k' + 1) = A_{2T,d_2} x(k') = B_{2T,d_2} u(k'), \tag{2.4}$$

where A_{2T,d_2} and B_{2T,d_2} are system and input matrices for the $DH \rightarrow DM \rightarrow DH$ pattern with sampling period $2T$ and delay d_2. Fig. 2.4(b) shows an example of four executions of control computation jobs with an $DH \rightarrow DM \rightarrow DM \rightarrow DH$ pattern. Similar to the previous example, the effective sampling period is $3T$ and the delay is $d_3 = 2T + d$ in this case. The resulting model is given by,

$$S_{m,2} : x(k' + 1) = A_{3T,d_3} x(k') = B_{3T,d_3} u(k'), \tag{2.5}$$

The general model for β consecutive DMs can be derived similarly considering a sampling period of βT and a delay of $(\beta - 1)T + d$.

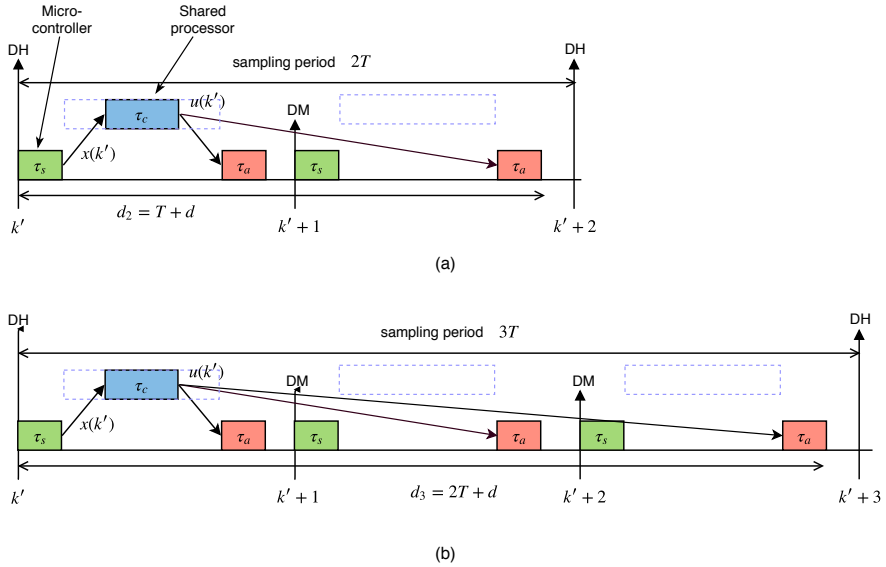

Figure 2.4: Control task with detectable deadline misses and hits. T and d are the same as shown in Fig. 2.2. DH/DM are indicated at the start of each period T. Execution patterns are (a) $DH \rightarrow DM \rightarrow DH$ (b) $DH \rightarrow DM \rightarrow DM \rightarrow DH$.

2.2.3 Stable controller synthesis and performance

Switched control

We consider state-feedback control of the form shown in Eq. 2.2. Since a DM is not executed, the control action $u(k')$ is updated only at the DH jobs. The following timing knowledge is assumed for stable controller synthesis:

- The maximum number of consecutive occurrences of DM jobs β is known at design time.
- At any DH job, the hit/miss pattern of the following $\beta + 1$ jobs is known at run-time.

These assumptions are valid for setups with a fixed set of periodic real-time (control) tasks under TDMA or static-priority preemptive scheduling.

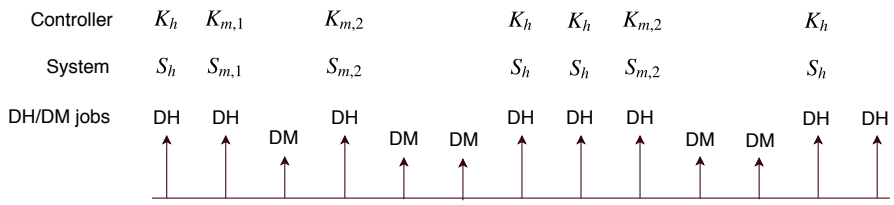

Figure 2.5: Controller execution for $\beta = 2$ (maximum two DM jobs in a row).

We illustrate the idea with an example trace with $\beta = 2$ in Fig. 2.5. There are three possible scenarios of interest:

- $DH \rightarrow DH$: When a DH job is followed by another DH job, the system S_h is as shown in Eq. 2.1. In this scenario, we use the controller in Eq. 2.2 with feedback gain $K = K_h$.
- $DH \rightarrow DM \rightarrow DH$: In this scenario, the system $S_{m,1}$ is as shown in Eq. 2.4 with sampling period $2T$ and delay $T + d$. We use the controller in Eq. 2.2 with feedback gain $K = K_{m,1}$.
- $DH \rightarrow DM \rightarrow DM \rightarrow DH$: In this scenario, the system $S_{m,2}$ is as shown in Eq. 2.5 with sampling period $3T$ and delay $2T + d$. We use the controller in Eq. 2.2 with feedback gain $K = K_{m,2}$.

Generally, there are $\beta + 1$ scenarios for a maximum of β consecutive DM jobs and each scenario uses different feedback gain K. Depending on the order of DM and DH jobs, the system switches between S_h, $S_{m,1}$ and $S_{m,2}$ in the example shown in Fig. 2.5.

Stability and performance

Switching as illustrated in Fig. 2.5 can potentially lead to instability [16]. Design of feedback gains K_h, $K_{m,1}$ and $K_{m,2}$ should ensure overall system stability using analytical tools such as common quadratic Lyapunov functions (CQLF) [44]. Generally, the DH jobs occur more frequently than the DM jobs and the system mostly runs under S_h. Therefore, in our controller synthesis, we design (i) aggressive gain K_h with a higher performance using standard techniques like pole placement [32] (ii) gains $K_{m,1}$ and $K_{m,2}$ such that a CQLF exists between S_h, $S_{m,1}$ and $S_{m,2}$ assuring overall switching stability.

The closed-loop dynamics for S_h with K_h designed as described above is given by,

$$x(k+1) = (A_{T,d} + B_{T,d}K_h)x(k) = A_{cl,T}x(k), \tag{2.6}$$

where K_h is designed using standard pole placement for high performance. $K_{m,1}$ and $K_{m,2}$ is designed as follows. If there exist Z_1, Z_2 and Y with $Y = Y^{T'} > 0$ such that the following Linear Matrix Inequalities hold,

$$\begin{aligned} Y - A_{cl,T}YA_{cl,T}^{T'} &> 0, \\ \begin{bmatrix} Y & Y^{T'}A_{2T,d_2}^{T'} + Z_1^{T'}B_{2T,d_2}^{T'} \\ A_{2T,d_2}Y + B_{2T,d_2}Z_1 & Y \end{bmatrix} &> 0, \\ \begin{bmatrix} Y & Y^{T'}A_{3T,d_3}^{T'} + Z_2^{T'}B_{3T,d_3}^{T'} \\ A_{3T,d_3}Y + B_{3T,d_3}Z_2 & Y \end{bmatrix} &> 0, \end{aligned}$$

then $P = Y^{-1}$, $K_{m,1} = Z_1P$, $K_{m,2} = Z_2P$ and the resulting switched system is stable [8]. Here, T' denotes the transpose of a matrix. This conclusion can be generalized to any β. The above design imposes the following restriction on the order of DM and DH jobs – any β DM jobs must be followed by at least one DH job. Therefore, in any consecutive $\beta+1$ executions, at least one job must meet its deadline. The above stable controller requires the $(1, \beta+1)$-firmness condition to be met. For example, $\beta = 2$ implies a $(1, 3)$-firmness condition for stability.

A performance requirement can be represented as an (m, k)-firmness condition using existing approaches [73] [32]. The performance is evaluated over a longer time window. This implies a value of k significantly larger than $\beta+1$ (i.e., the k value for stability). Thus, we have two (m, k)-firmness conditions for stability and performance.

Illustrative example

We consider an unstable second-order mass-spring-damper system with the following state-space model for illustration [73],

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -3 & 1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u,$$

with a sampling period $T = 5ms$ and delay $d = 4.95ms$. The stability condition is captured by $(1, 3)$-firm deadlines with $\beta = 2$ (i.e., maximum two DMs in a row). We design K_h, $K_{m,1}$ and $K_{m,2}$ as described above and obtain,

$$K_h = \begin{bmatrix} -3.4657 & -1.7442 & 0.8252 \end{bmatrix},$$

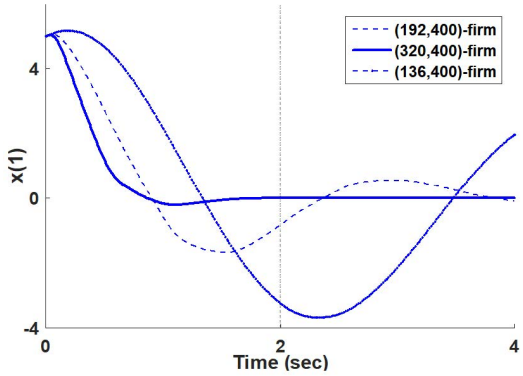

Figure 2.6: System response with 264, 208 and 80 DMs in a window of 400 samples (i.e., with (136,400)-firm, (192,400)-firm and (320,400)-firm deadlines). Only the response with the (320,400)-firmness condition meets the performance requirement of settling down within $2s$.

$$K_{m,1} = \begin{bmatrix} 0.9655 & -1.2021 & 0.0317 \end{bmatrix},$$
$$K_{m,2} = \begin{bmatrix} 0.9655 & -1.2021 & 0.0317 \end{bmatrix}.$$

We further consider a performance requirement stating that from a given initial condition of $x = \begin{bmatrix} 5 & 1 \end{bmatrix}$ the system should reach $x = \begin{bmatrix} 0 & 0 \end{bmatrix}$ the latest in $2s$. This implies that a window of $k = \frac{2s}{5ms} = 400$ samples is of interest for performance. Fig. 2.6 shows the system response of three examples with different numbers of DMs in a window of $k = 400$ samples. Clearly, the performance requirement (settling within $2s$) is not met if the number of DMs goes beyond 80 in a window of 400 samples. Therefore, the performance requirement can be captured by a (320,400)-firmness condition. Generally, such performance requirements can be derived analytically as shown in [73] [32].

2.2.4 Firmness constraints for a task

In this section, we considered stability and performance as performance metrics for control applications. This resulted in obtaining two sets of (m, k)-firmness constraints. There might be other performance metrics in different applications, e.g. settling time, overshoot, raising time etc. These metrics might

impose even more firmness constraints. On the other hand, one firmness constraint might imply another constraint. For example, $(2, 3)$-firmness implies $(6, 10)$-firmness. In the rest of this chapter for simplicity we assume exactly one (m, k)-firmness constraint for a firm task. As shown later in this chapter, multiple (m, k)-firmness constraints can be verified in our mapping approach with only a linear increase in the run-time.

2.3 Setup under consideration

2.3.1 Task models

Control applications

We consider the computing task in feedback control loops as control applications that are intended to be mapped to a multiprocessor system. We model these control applications with strictly periodic tasks. We define a set $\Pi = \{\tau_i\}$ of periodic tasks running on a shared processor. Each task $\tau_i = (C_i, T_i, D_i)$ is specified by its constant execution time C_i, activation period T_i and execution deadline D_i. We show in the successive chapters that deviation from the execution time of a task has a monotonic effect on our results and we may choose worst-case execution time in our methods. We assume that the deadlines of all tasks are at most equal to their activation period, i.e. $D_i \leq T_i$. This scenario is particularly relevant for feedback control applications where execution with outdated data might cause stability issues. We consider both control applications with hard deadlines, i.e. no deadline misses are allowed, and with (m, k)-firmness conditions, i.e. QoC of a control task is acceptable if there are at least m DHs in any k consecutive jobs of the task.

Streaming applications

We use data flow models (SDF graphs, in particular, though other variants like SADF for which multiprocessor mapping techniques exist would also work) to model and map the streaming applications and for throughput analysis [66] [65]. We illustrate our proposed method considering streaming applications such as the H.263 encoder [62] (see Fig. 2.7). In an application SDF graph (application graph), nodes (actors) communicate by sending data (tokens) over edges (channels). Upon firing of an actor, input tokens are consumed from its input channels and after a certain amount of time (execution time) new tokens are produced on its output channels; the numbers of consumed and produced

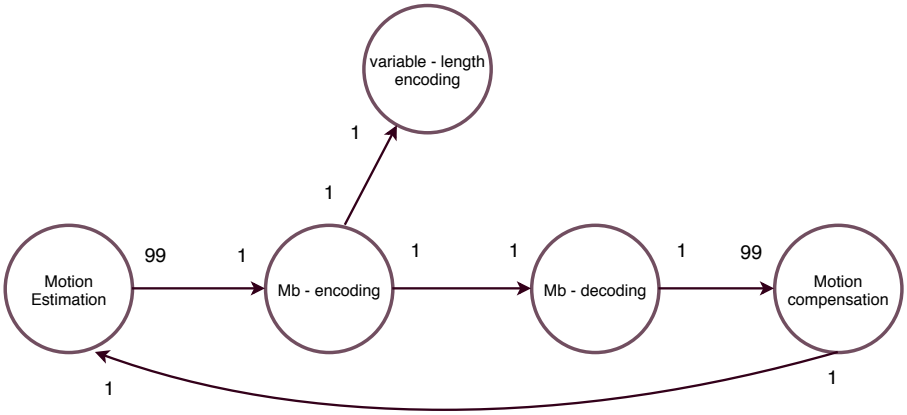

Figure 2.7: SDF graph of H.263 encoder

tokens for each firing are given by the annotations (rates) attached to the channel ends in the graph.

The execution time of some of the actors and their input/output rate may vary (e.g. for variable-length encoding actor in the example shown on Fig. 2.7) from one execution to another. By conservative overestimation it is possible to have a fixed number for them. Alternatively, more fine-grained models can be made using Scenario-Aware Dataflow Graphs [68]. For simplicity, we limit ourselves to SDF graphs. Let $SA = \{SA_1, SA_2, ...\}$ be the set of all streaming applications. Each application $SA_i \in SA (i \in \mathbb{N})$ has a throughput constraint of λ_i actor firings per time unit, for some dedicated actor.

2.3.2 Platform model

We consider a tile-based template [66] as the multiprocessor platform. Fig. 2.8 shows an example architecture platform with four connected tiles that process multiple control and streaming applications. Each tile contains a processor (P), a local memory (M), and a network interface (NI) that connects the memory to the local processor, and interconnections between tiles. Let $\Theta = \{\theta_1, \theta_2, ...\}$ denote the set of tiles under consideration.

All tiles run a TDMA scheduling policy. Under the TDMA policy, the resource access schedule repeats cyclically according to a cycle called a *time wheel*. The time wheel consists of multiple equally sized parts, called *time slots*. It should be noted that the context switching overhead between applications

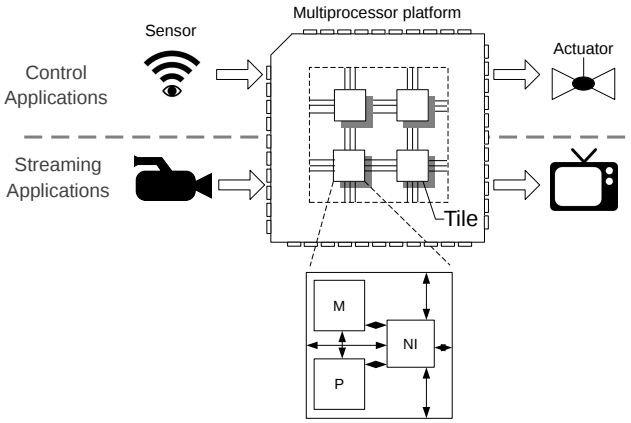

Figure 2.8: A tile-based multiprocessor system used for processing multi-constraint applications. P: processor, M: memory and NI: network interface.

is negligible compared to the size of each time slot. Many architectures [46] allow such choice of the TDMA configuration. Each time slot can be allocated to a different application during which the application can run without any interference from other applications. Each tile $\theta_i \in \Theta$ has a time wheel of length w_i consisting of q_i equally sized time slots. Each time slot has a length of v_i time units. For each tile θ_i, we define S_i as the set of all time slots as $S_i = \{S_{i,1}, S_{i,2}, ..., S_{i,q}\}$.

We consider the communication between tiles to be asynchronous. The presented analysis on timing behavior of data communicated between tiles is based on the worst-case timing scenario. In other words, any data traveling from one tile to another, is assumed to arrive at the destination tile at the moment such that it will have to wait the most until it reaches the next assigned time slot. Fig. 2.9 illustrates the worst-case timing scenario for data that is communicated between two tiles t_1 and t_2 with $w_1 = w_2 = 6$. As shown in Fig. 2.9, $S_{1,2}$ in t_1 and $S_{2,6}$ in t_2 are assigned to application SA_1. Therefore, the worst-case scenario for data sent from t_1 to t_2 is to arrive just after $S_{2,6}$ ends. In this scenario, the data has to wait for $w_2 - v_2$ time units before its processing starts.

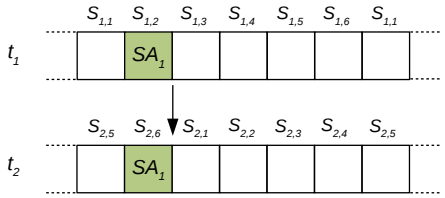

Figure 2.9: Worst-case scenario for data communication between tiles.

2.4 Mapping solutions

In this section, we propose a mapping strategy for multiple hard-deadline and firm periodic tasks running under a TDMA scheduling policy. A naive way of obtaining the optimal mapping solution for a set of tasks considered in this chapter is to try all the possible combinations of time slots for all the tasks. This is in fact a loop that repeats the feasibility analysis for hard tasks, firmness analysis for firm tasks, and throuput analysis for every streaming application for every candidate schedule. However, this is intractable. An efficient way of resource allocation would reduce the number of iterations between these two phases.

Our proposed Multi-Constraint Resource Allocation (MuCoRA) method provides tight resource dimensioning while reducing the chance of deadline misses for firm tasks. That is, MuCoRA increases the chance for a feasible schedule while obtaining a candidate mapping solution. The method is based on a mapping heuristic inspired by Observations O1, O2, O3 and O4 briefly mentioned in Section 2.1. These observations are detailed later in this section. We first formulate the problem precisely.

2.4.1 Problem formulation

The problem of mapping multiple streaming, hard and firm tasks can now be formalized. For a set of inputs as follows:

- I1: a set of streaming applications SA where $SA_i \in SA$ with a throughput constraint λ_i modeled by an SDF graph,
- I2: a set of hard-deadline control applications $\Pi_{hard} = \{\tau_{hard-i}\}$ where $\tau_{hard-i} = (C_{h-i}, T_{h-i}, D_{h-i})$,

- I3: a set of firm control applications $\Pi = \{\tau_i\}$ where $\tau_i = (C_i, T_i, D_i)$ with given (m, k)-firmness constraints and,
- I4: a set of tiles Θ, each tile $\theta_i \in \Theta$ running a TDMA policy with a time wheel size of w_i time units and q_i time slots each with a length of v_i time units.

we address the following resource allocation problem:

- P1: binding actors (for SA) and tasks (for Π_{hard} and Π) to the tiles and,
- P2: assigning time slots (i.e., $S_{i,j}$) in a tile to applications (i.e., SA, Π_{hard} and Π) bound to that particular tile,

such that the following constraints are satisfied:

- C1: throughput constraints λ_i of the streaming applications $SA_i \in SA$,
- C2: the hard latency constraint D_i of the control applications $\tau_{hard-i} \in \Pi_{hard}$ and,
- C3: the (m, k)-firmness conditions of the control applications $\tau_i \in \Pi$.

Given I1-I4, a solution to P1 and P2 which satisfies C1-C3 for all the applications is a valid solution to the above resource allocation problem.

2.4.2 MuCoRA: resource allocation flow

A resource allocation is required to address three subproblems:

- A. mapping the control applications with hard deadline,
- B. mapping the streaming applications and,
- C. mapping the control applications with firmness conditions.

Each subproblem consists of

- i. binding the applications to tiles (i.e., P1) and,
- ii. assigning the time slots to the applications (i.e., P2).

MuCoRA is following the flow shown in Fig. 2.10 and explained in the rest of this section.

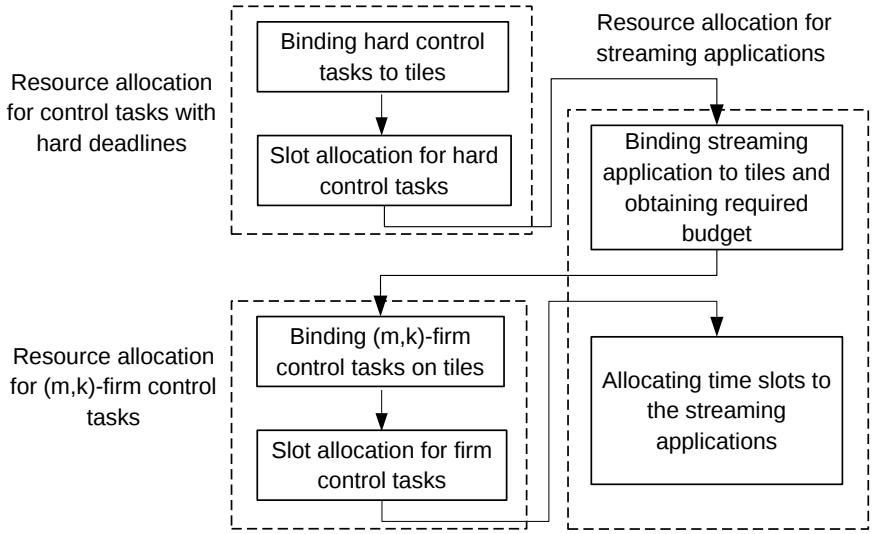

Figure 2.10: MuCoRA overview on resource allocation for multi-constraint real-time tasks including control applications with hard deadlines, streaming applications and firm control applications.

A. Mapping control applications

Considering Observation O1 in Section 2.1 elaborated below, we first map feedback control loops with hard deadlines to the processor.

Key observation: Traditionally, resource allocation for applications with different types of constraints on shared resources is performed by time-isolated resource-to-application mapping [65]. Fig. 2.11(a) and Fig. 2.11(b) show two examples of such allocation in view of the setup illustrated in Fig. 2.1. In Fig. 2.11(a), the t_a interval within a work cycle is allocated to the streaming applications while t_b is allocated to the control applications. While such scheduling policy provides application-level independence, it can be noticed that no sensing update for the control applications is possible during t_a and the sampling period of the control loops must be longer than this duration. That is, the minimum sampling period of the control (i.e., latency-constrained) applications would be the sum of t_a and the execution time of the control task C_i (i.e., $t_a + C_i$ in Fig. 2.11(a)). Towards meeting a given latency constraint, one needs to allocate enough resources in order to take into account the above

worst-case scenario, i.e., t_a should be small enough such that $t_a + C_i$ meets the latency constraint. As opposed to Fig. 2.11(a) in which there is only one partition for each of the streaming and control applications, Fig. 2.11(b) shows an allocation with distributed resource budget to control applications. The resources allocated to the latency and throughput constrained applications are still temporally isolated like the example in Fig. 2.11(a). With the distributed allocation shown in Fig. 2.11(b), the sampling period for the control application would be shorter than that of the example in Fig. 2.11(a). That is, $t_a + C_i > t'_a + C_i$. Since a shorter sampling period can potentially be translated to a higher QoC, this observation is particularly relevant for feedback control loops. While this observation suggests that the resources should be as distributed as possible for the control applications leading to a high number of time slots, each time slot needs to accommodate certain overhead for context switching from one application to the other. Such overhead restricts the granularity of the time slots allowed by the system since a higher number of slots in a work cycle imposes a higher overhead (leading to a poor resource efficiency). In this work, we assume that the time slots are long enough to ignore such overhead. Moreover, the choice of parameters of a work cycle is a design decision often taken at an early design phase and this work deals with a given set of work cycle parameters.

To guarantee performance for streaming applications, it is necessary to consider the worst-case scenario for these throughput-constrained applications. This is needed, because it is generally not possible to assume time synchronization between time wheels on different processors. As a result, a processing resource is allocated to meet the throughput constraint for any distribution of the budget over concrete time slots. Thus, our observation for allocating resources to control applications can be integrated with any method that allocates sufficient budget (i.e., does not perform application-to-slot mapping) to streaming applications for throughput guarantees. For this purpose, we rely on the method reported in [66] and the existing tooling support in SDF^3 [67]. Therefore, for a given TDMA time wheel and a set of multi-constraint applications, we first allocate slots to control applications with hard deadlines.

Given I2 and I4, we address P1 and P2 for control applications. Resource allocations for the control applications with hard deadlines consist of two stages. In the first stage, each control application is bound to only one tile (i.e. P1). First, this avoids latency overhead due to asynchronous interaction among the tiles. Second, this avoids delay caused by the interconnections. Further, the control tasks are computationally light meaning they can be run on a single tile. The binding solution of control applications $\tau_{hard-i} \in \Pi_{hard}$ must guarantee that the latency constraint C2 is met.

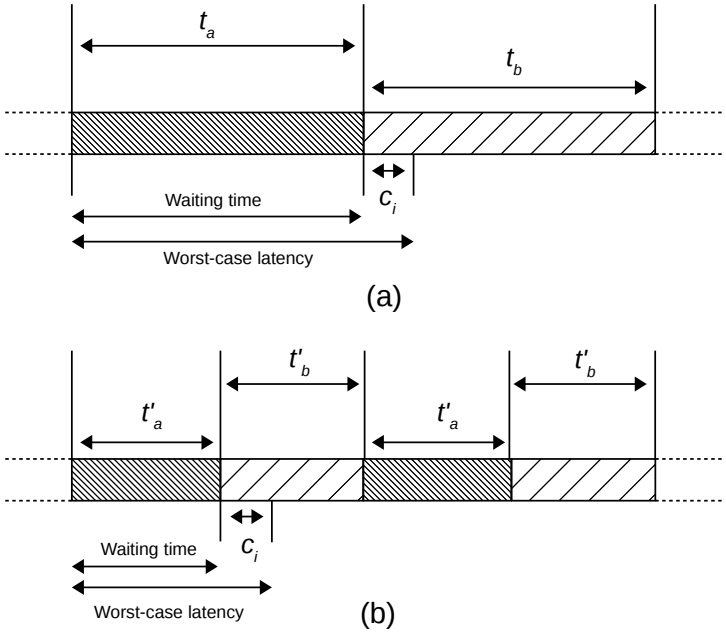

Figure 2.11: Example: each work cycle of a processor is divided into (a) two, and (b) four slots each assigned to either streaming applications or control applications. Streaming applications are processed during times t_a and t'_a while control applications are processed during times t_b and t'_b .

We use a load balancing policy for task-to-tile binding similar to [66]. Resource usage for the control applications corresponds to the number of time slots that is required to meet C2. We define the load function of control application τ_{hard-i} as

$$l_{\tau_{hard-i}} = \frac{C_{h-i}}{T_{h-i}} \tag{2.7}$$

It determines the minimum fraction of a time wheel that is required for a control application to meet C2. It is noteworthy that a total load of less than 1 is a necessary condition for schedulability on a single tile, i.e [35]:

$$\sum l_{\tau_{hard-i}} < 1. \tag{2.8}$$

To balance the load of control applications among tiles, we sort the applications in decreasing order of their loads. Then, we bind applications in that order to the tiles. For example, assume binding of four control applications with load function values of

$$[l_{\tau_{hard-1}}, l_{\tau_{hard-2}}, l_{\tau_{hard-3}}, l_{\tau_{hard-4}}] = [0.6, 0.4, 0.3, 0.1].$$

The best case for binding these applications to two tiles according to the resource load balancing policy is to bind τ_{hard-1} and τ_{hard-4} to one tile and τ_{hard-2} and τ_{hard-3} to the other tile. This addresses P1 for the control applications Π_{hard}. In the following, we describe how time slots are assigned to the control applications towards addressing P2.

Definition 1. *For any time slot $S_{i,j}$ (i.e. slot number j in tile number i), we define an occupation variable $X_{i,j,m}$ that takes 1 when slot $S_{i,j}$ is assigned to application τ_{hard-m} and 0 otherwise.*

For each application mapped on a tile we initially allocate all available time slots, i.e., $X_{i,j,m} = 1$ for all j which are not allocated to other applications and next, we attempt to unassign the time slots that are *unnecessary*. That is, we verify if an assigned slot can be unassigned without violating constraint C2.

The result of the following proposition is used to verify if an assigned slot can be unassigned.

Proposition 1. *If the release of a job in the beginning and at the end of a TDMA time slot results in DHs, the release of the job at any time within the time slot is also a DH.*

Proof. Suppose this proposition is false. Suppose the release of a job of $\tau_m = (C_m, T_m, D_m)$ at the beginning, i.e. t, and the end, i.e. $t + v_i$ where v_i is the length of every time slot in the TDMA time wheel of tile θ_i, of the time slot $S_{i,j}$ results in DHs, while its release at $t < t' < t + v_i$, results in a DM (see Fig 2.12(a)). Let us consider a window of the size D_m (see Fig 2.12(b)). In order to verify if a job of τ_m released at t is a DH, the overall duration of the time slots allocated to τ_m within the window starting at t must be at least equal to C_m. Therefore, in order to verify DMs between t and $t+v_i$, we require to slide the window between t and $t + v_i$ and verify the allocated time to τ_m within the window. The only possibility for having a DM at t' corresponds to the case when the overall allocated time within the window is reducing while sliding the window from t to t'. This can only be valid if $S_{i,j}$ is allocated to τ_m while $t + D_m$ (i.e. the end of the window while the beginning of it is at

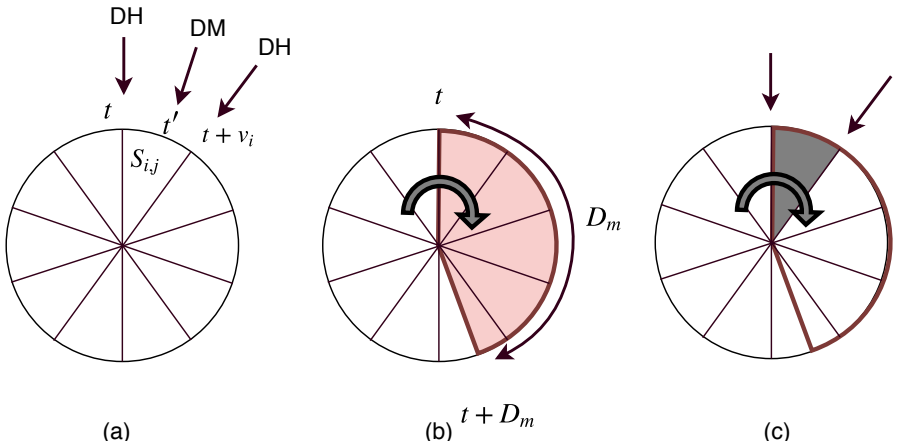

Figure 2.12: Proof by contradiction for Proposition 1. (a) The supposition. (b) sliding window to verify DHs released at t. (c) The only scenario for having a DM release at t' is valid if $S_{i,j}$ is allocated to τ_m

t) is in a time slot that is not allocated to τ_m (see Fig. 2.12(c)). Therefore, when the beginning of the window is at t' the allocated time to τ_m within the window is less than that of the case when the beginning of the window is at t. If we continue sliding the window further than t' two scenarios can happen. (i) the overall allocated time to τ_m within the window when the beginning of the window is at $t + D_m$ is less than that of the window with its beginning at t'. This scenario happens when $t' + D_m$ (the end of the window while the beginning of it is at t') is in the time slot that is not allocated to τ_m. (ii) the allocated time to τ_m within the window stays the same as the case when the beginning of the window is at t'. This scenario happens when $t' + D_m$ is in the beginning of a time slot. In none of these scenarios the job released at $t + v_i$ can hit the deadline. Hence, the initial supposition is false and the proposition is correct. □

Proposition 1 implies that to verify the schedulability of a periodic task with hard deadline allocated to a TDMA schedule it is sufficient to consider its releases in the beginning of each time slot. Therefore, assigned time slot $S_{i,j}$ (i.e. slot number j in tile number i) to a control application $\tau_{hard-m} = (C_m, T_m, D_m)$ can be unassigned without violating C2 if the fol-

lowing inequality is satisfied for all $j \leq n \leq j + D_m$

$$\sum_{k=n-\lceil \frac{D_m}{v_i} \rceil+1}^{n} X_{i,k,m} \geq \lceil \frac{C_m}{v_i} \rceil + 1 \tag{2.9}$$

where v_i is the size of each time slot. Eq. 2.9 basically verifies whether τ_{hard-m} hits its deadline by having a release at the beginning of the slot $S_{i,k}$ where $n - \lceil \frac{D_m}{v_i} \rceil + 1 \leq k \leq n$. Considering the jobs of τ_{hard-m} released at the beginning of all the time slots, assigning or unassigning $S_{i,j}$ to τ_{hard-m} may only affect the jobs of the task when they are released at the beginning of this set of slots. In other words, at least $\frac{C_m}{v_i}$ number of time slots must be allocated to τ_{hard-m} in any $\lceil \frac{D_m}{v_i} \rceil$ time slots to guarantee the satisfaction of the timing requirements of τ_{hard-m}.

Fig. 2.13 illustrates the process of unassigning the slot $S_{i,3}$ of the TDMA time wheel shown in the figure for task $\tau_m = (2, 5, 4)$. Eq. 2.9 is used to verify the satisfaction of the timing requirements of the task while unassigning the slot $S_{i,3}$. In the figure, all the time slots are allocated (shown with dark grey color) to τ_m except $S_{i,1}$ and $S_{i,5}$ (shown with light grey). Therefore, $X_{i,1,m} = 0$ and $X_{i,5,m} = 0$. Eq. 2.9 considers all the combination of four consecutive slots including $S_{i,3}$ and obtains the summation $\sum X_{i,j,m}$ for these slots. If this summation for all the combinations is equal to or larger than $C_m + 1 = 2 + 1 = 3$, $S_{i,3}$ can be unassigned for τ_m. From the results shown in Fig. 2.13, it is concluded that $S_{i,3}$ can be unassigned for τ_m without causing any DM.

As illustrated in Fig. 2.11, a distributed allocation of resources for control applications improves the resource efficiency requiring less time slots. Therefore, we basically aim to *minimize the maximum distance between any two consecutive slots that are assigned to a control application.* We assign the slots in such a way that the maximum number of slots between every two assigned time slots is

$$\left\lceil \frac{D_m}{C_m} \right\rceil. \tag{2.10}$$

In the above process, some slots might already be occupied by other applications and it is not possible to assign those slots to the control application. In those cases, more slots might be required to guarantee the performance, i.e., meet the latency constraint C2. To maximize the distribution in resource allocation, the position of slots that are considered for unassignment is chosen periodically with a period as Eq. 2.10.

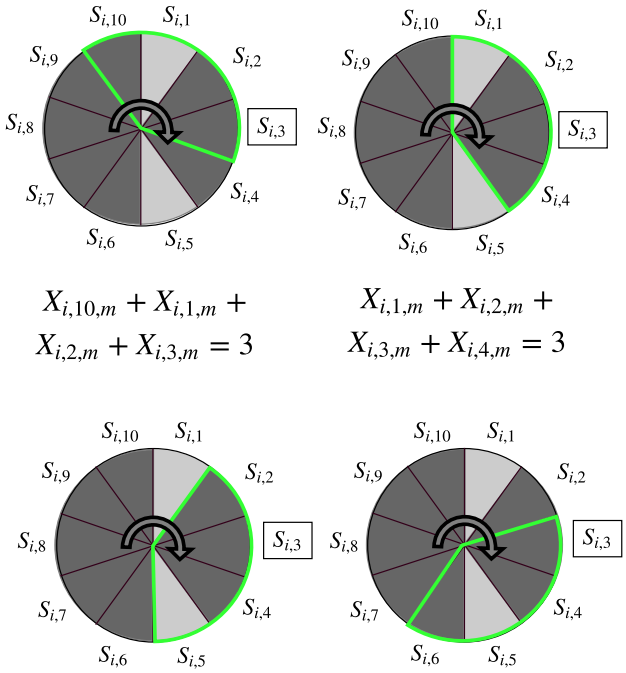

Figure 2.13: All combinations of four consecutive time slots including $S_{i,3}$ are verified by equation Eq. 2.9 to learn wether $S_{i,3}$ can by unassigned for τ_m.

Fig. 2.14 shows two orders of selecting time slots in the example in Fig. 2.13 for unassignment. In Fig. 2.13(a), consecutive available slots are considered for unassignment. Fig. 2.13(b) shows the resulting allocation. In Fig. 2.13(c), the time slots are chosen with a period of 2 obtained by Eq. 2.10. The order of selecting the slots would be

$$\{S_{i,1}, S_{i,3}, S_{i,5}, S_{i,7}, S_{i,9}, S_{i,2}, S_{i,4}, S_{i,6}, S_{i,8}, S_{i,10}\}.$$

$S_{i,1}, S_{i,5}$ are not allowed to be assigned to τ_{hard-m}. By analyzing (using

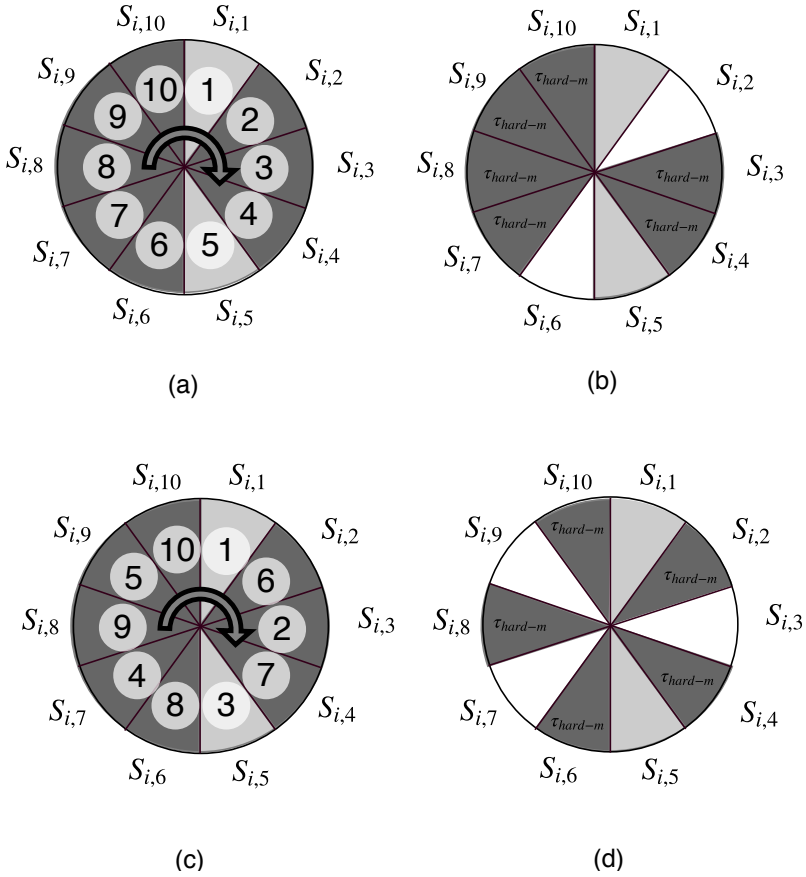

Figure 2.14: Unassigning the assigned time slots for τ_{hard-m} using (a) sequential order which leads to the slot allocation (b), and (c) the MuCoRA method which results in the slot allocation (d). The slots colored in dark gray are allocated to τ_{hard-m}. The slots colored in light gray are previously allocated to other applications. The slots colored with white are emptied after unassigning the slots for τ_{hard-m}.

Eq. 2.9 with the period 2.10) all other slots, we find that $S_{i,3}, S_{i,7}, S_{i,9}$ can be unassigned. Fig. 2.14(d) shows the resulting allocation. Although both mapping solutions (Fig. 2.14(b) and Fig. 2.14(d)) satisfy the latency constraints

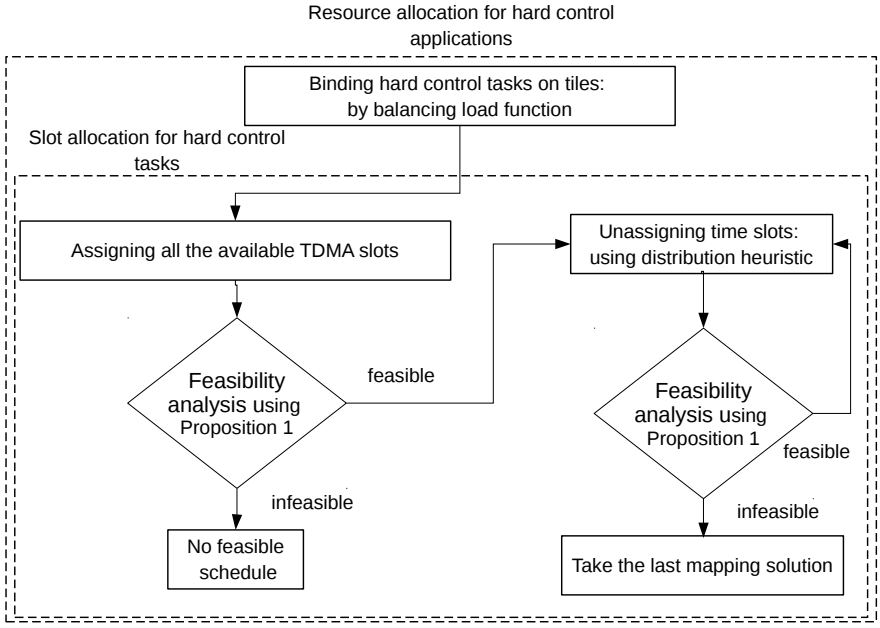

Figure 2.15: Flowchart of resource allocation for hard control applications.

of application τ_{hard-m}, the second schedule, i.e. Fig. 2.14(d), requires less time slots than the first one, i.e. Fig. 2.14(b). Maximum distance between two consecutive assigned time slots in the second solution is 2 as per Eq. 2.10. Thus, the results are in line with the observation described in Section 2.1. Fig. 2.15 shows the flowchart of MuCoRA for the resource allocation of hard control applications. The slot allocation obtained by MuCoRA is, however, not guaranteed to be the optimal solution in terms of allocating the minimum possible number of slots for all tasks.

B. Mapping streaming applications

We use the existing method of [66] as implemented in the SDF^3 tool [67] to allocate resources to streaming applications. The reader is referred to [66] for details. Here, we only summarize the essential aspects.

Binding actors to tiles. This step binds every actor of a streaming application SDF graph to a tile. The actors that have large impact on the throughput

are bound first. The impact of an actor is too expensive to compute exactly. Therefore, it is estimated for each cycle in the SDF graph in which it participates, by the total computation time of the actor divided by the pipelining parallelism in that cycle. The pipelining is, in turn, estimated by the total number of initial tokens in the cycle, normalized by the actor firing rates on the corresponding edge on which they reside. Since an actor may be part of multiple cycles in the SDF graph, the most critical cycle is considered to be an estimate for the impact of the actor.

After sorting the actors according to their impact on throughput they are bound to the tiles. Binding tries to achieve load balancing, where the load of a tile is estimated as the total execution time of firings of actors bound to the tile divided by the total execution time of all actor firings. The main strategy is a greedy algorithm that binds an actor to the least loaded tile. The SDF^3 tool balances the load of tiles not only in terms of computation, but also in terms of memory usage and bandwidth requirements, both of which we do not consider in this chapter. SDF^3 allows us however to set weighting factors for these aspects, which then enables us to perform load balancing in terms of computation only.

Constructing firing order. After binding the actors of an application to the tiles, the static order of their firing is obtained using a list-scheduler.

Budget allocation. A binary search algorithm is used in combination with a throughput analysis algorithm to find the minimum number of time slots required on each tile to satisfy the required throughput rate. The throughput analysis algorithm that is used is pessimistic, both in view of which of the time slots in the time wheel will be allocated, and in terms of the misalignment of slots on different tiles due to asynchronous execution of their TDMA wheels. Thus, problem P1 is solved for the streaming applications and problem P2 is trivially satisfied by any slot allocation we may choose. Therefore, after obtaining the budget (i.e. number of time slots) that is required for streaming applications by SDF^3, we proceed with the resource allocation process with resource allocation for firm control applications. This corresponds to the fact that, in contrast to the streaming applications, the position of allocated slots matters for firm control applications (as for hard control applications). However, while allocating time slots to firm control applications, we consider the number of slots that are required for the streaming applications in each tile. After resource allocation for firm tasks we (randomly) allocate slots to streaming applications from the remaining time slots.

C. Mapping (m, k)-firm control applications

Mapping firm tasks to tiles also consists of two steps: (i) binding tasks to processor tiles and (ii) slot allocation for each task on a tile. In order to bind firm tasks to tiles, we take the same approach as for hard control applications. That is, we balance the load among tiles. If there are two firm tasks, for example, we bind the task with a higher load function (Eq. 2.7) to a tile that has more unallocated time slots. The main difference between mapping hard and firm control applications is in the slot allocation process. We start this process similar to that of the applications with hard deadlines. That is, we first assign all the available time slots to a firm task $\tau_i = (C_i, T_i, D_i)$. Then we attempt to unassign the slots with the period obtained by Eq. 2.10. However, in each iteration of unassigning an initially assigned slot, we verify the satisfaction of the firmness conditions for the task. Chapter 4 proposes the Firmness Analysis (FAn) method that verifies the satisfaction of a firmness condition of a firm task on a given TDMA policy. Note that, after unassigning the time slots, the remaining slots must be at least equal to the number of slots that is required for streaming applications (which is already obtained in the previous step). Fig. 2.16 shows the flowchart of the resource allocation process of firm tasks that is explained above.

As an example, assume control applications with hard deadlines are previously allocated to the slots that are shown with light gray in Fig. 2.17(a). We intend to efficiently allocate time slots (from the slots $S_{i,3}, S_{i,7}, S_{i,9}$) to the task $\tau_1 = (1.5, 9, 8.5)$. For simplicity of the example, assume a single $(8, 10)$-firmness condition. As explained before, firm control tasks can generally have multiple (m, k)-firmness conditions. The presented flow applies to any number of (m, k)-firmness conditions. So we consider a single condition in this example. We first assign all the three time slots to τ_1. We learn from the FAn method that allocating these slots to τ_1 satisfies the (m, k)-firmness condition. Then we unassign $S_{i,7}$ and verify the firmness condition using the FAn method. We learn that $S_{i,3}$ and $S_{i,9}$ allocated to τ_1 still satisfies the firmness condition (see Fig. 2.17(b)). Unassigning any of $S_{i,3}$ or $S_{i,9}$ leads to violating the firmness condition based on the FAn method though. Therefore, if assigning only one slot of this tile, i.e. $S_{i,7}$, to streaming applications satisfies their throughput requirement, the mapping solution shown in Fig. 2.17(b) is a feasible schedule for the set of tasks in this example.

Consider a case when a task has multiple (m, k)-firmness constraints. In such a case, we follow the same flow as explained above with multiple firmness analysis every time that we unassigned a time slot. That is, every time that a time slot is unassigned for a task, we verify the satisfaction of all the firmness

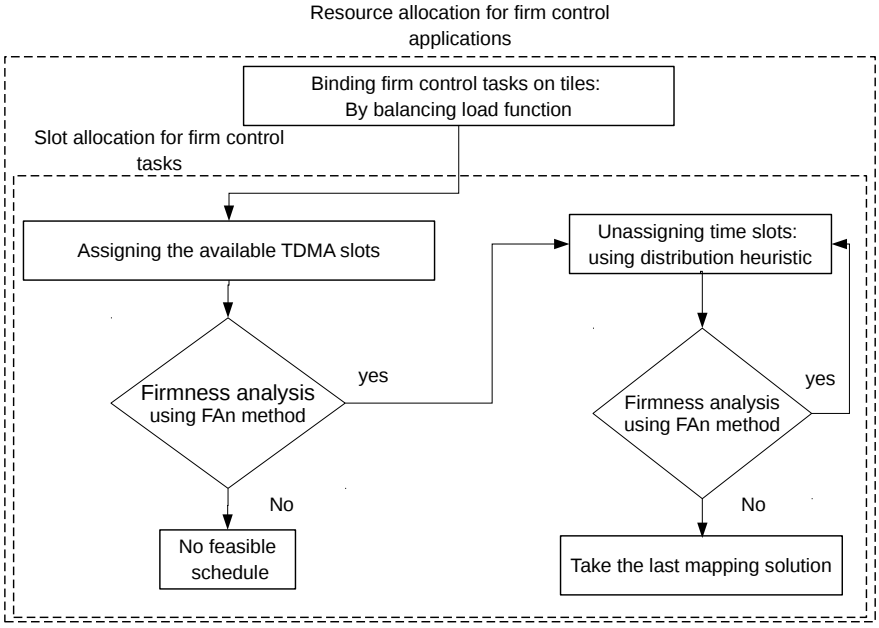

Figure 2.16: Flowchart of the resource allocation process for firm control applications.

constraints of the task.

2.5 Experimental evaluation

2.5.1 System description

In this section we evaluate the resource allocation method MuCoRA discussed in this chapter. We consider multiple combinations of streaming applications, control applications with hard deadlines and firm control applications which are intended to be mapped to a multiprocessor system. The multiprocessor platform has two tiles both running under TDMA policy. We intend to map the applications to the processors such that their timing requirements, i.e. throughput and latency (deadlines, firmness), are satisfied. Each TDMA time wheel consists of 10 time slots. Each time slot has length $1ms$. The streaming

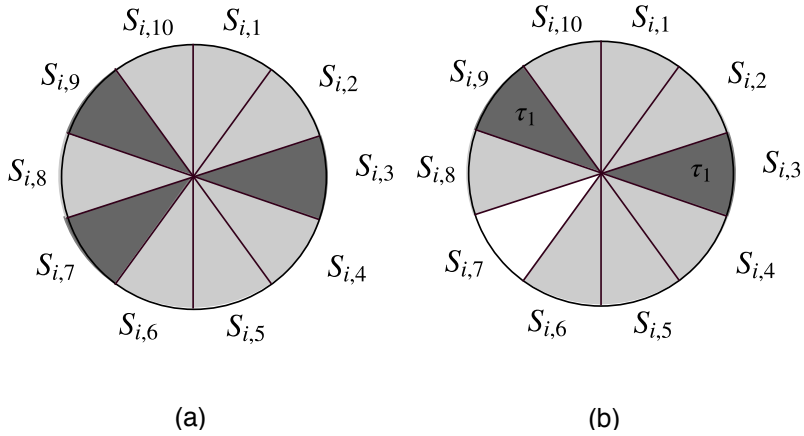

Figure 2.17: Allocation of the remaining time slots in Fig. 2.14(d) to the $(8, 10)$-firm task τ_1. (a) initially all the available slots $S_{i,3}, S_{i,7}, S_{i,9}$ are allocated to τ_1, which satisfies the firmness condition. (b) final slot allocation with the minimum possible slots allocated to τ_1.

applications are an H.263 video encoder (SA1) [62] and an H.263 decoder (SA2) [64] with the SDF graphs as shown in Fig. 2.7 and Fig. 2.18, respectively. The sampling periods of control applications are randomly chosen in a range where it is smaller than, comparable to and larger than the time wheel size, i.e. for a control application τ_m, $T_m \in [5, 100]ms$. The execution time of the control applications are randomly chosen in a way that the overall utilization of control applications in a tile θ_i, i.e. $U_{i-\tau_m} = \frac{C_m}{T_m}$, is in the range $U_{i-\tau_m} \in [0.2, 0.6]$. Firmness conditions are also chosen randomly in meaningful ranges. k is chosen randomly in the range $3 \leq k \leq 150$. The MuCoRA method is implemented in MATLAB and compiled on a computer with a quad-core processor and a clock frequency of 2.6GHz.

2.5.2 Success rate

Brute-force approach: we evaluate the success rate of our method in obtaining a feasible schedule with that of a brute-force method. The brute-force method follows the same approach in the task-to-tile binding process as MuCoRA. However, in the slot allocation process, the brute-force approach verifies the allocation of all the possible combinations of available slots to a control

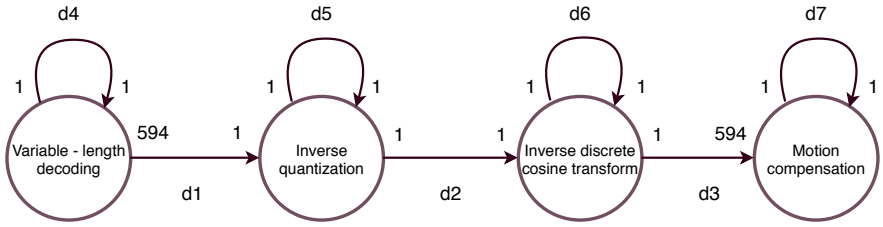

Figure 2.18: SDF graph of H.263 decoder

task to obtain the most efficient mapping solution, i.e. a mapping solution with the minimum possible number of slots allocated to the task. The brute-force approach uses SDF^3 to decide the required budget for streaming applications as in MuCoRA.

We take 100 combinations of multi-constraint tasks as follows. The periods of control tasks with both hard and (m, k)-firm deadlines are randomly chosen in a range where it is smaller than, comparable to and larger than the time wheel size, i,e, for a control application τ, $T \in [5, 100]ms$. The deadline of each task τ is randomly picked in the range $0.8T \leq D \leq T$. The execution time of hard control applications are chosen such that their overall utilization in two tiles is in the range $U \in [0.4, 1.2]$, i.e. $U \in [0.2, 0.6]$ on average for each task. The utilization of the task τ on the tile θ_i is obtained by $U_{i-\tau_h ard} = \frac{C_h}{T_h}$. The same assumption is made for the execution time of firm tasks.

Using both the MuCoRA and brute-force methods, we first map all hard control applications. From the SDF^3 analysis we obtain that allocation of 3 time slots in one time wheel and 4 time slots in the other guarantees the throughput constraint of the H.263 decoder and encoder. For firm tasks however, we intend to map as many tasks to each tile as possible. Therefore, for each experiment with a predefined set of tasks after the resource allocation of all hard control tasks and streaming applications we map the maximum possible number of firm tasks of that task set. In 94 cases (out of 100) both methods obtain mapping solution for the same set of tasks. In 86 cases both methods provide a mapping solution with the same number of slot allocations assigned to the applications. That is, in 8 cases, although the brute-force method could not map more firm tasks on any of the two tiles, the number of remaining (unassigned) slots is higher in the mapping solutions obtained by the brute-force method. The results of this experiment show that the distribution heuristic used by MuCoRA results in an efficient resource allocation in

a majority of cases while iterating less between obtaining candidate mapping solution and feasibility and firmness analysis. Next, we discuss more on the run-time of MuCoRA.

2.5.3 Scalability

We conducted another set of experiments in which we address resource allocation of hard and firm tasks separately to evaluate the scalability of MuCoRA. In these experiments, various values for the parameters that influence the run-time of MuCoRA are chosen. We first evaluate the scalability of MuCoRA for control applications with hard deadlines which are intended to be mapped to a multiprocessor systems with two tiles. We assume that tiles are initially empty, i.e. no other tasks are running on the tiles. The run-time of MuCoRA depends on the number of repetitions of the calculation of Eq. 2.9 for all tasks. Recall that Eq. 2.9 is used to unassign the time slots that are initially assigned to a task from all empty slots. Therefore, the number of tasks in a task set is one factor in the run-time of MuCoRA. For each task, Eq. 2.9 is calculated to verify whether an initially assigned time slot can be unassigned or not. Therefore, the number of time slots in each time wheel influences the run-time of MuCoRA. The execution time, period and deadline of control applications are chosen as explained in the beginning of Section 2.5. The experiments differ with each other in the number of tasks in each task set and the number of time slots in each TDMA time wheel. Fig. 2.19 shows the run-time of MuCoRA for our experiments. Fig. 2.19(a) depicts the run-time of MuCoRA for 100 task sets with different numbers of tasks M. The run-time of the method for the combinations of tasks for which the resource allocation is terminated without a feasible mapping solution are denoted with No, otherwise Yes. Our experiments imply that MuCoRA has complexity $O(M^{c_1})$ where $1 \leq c_1 \leq 2$. Fig. 2.19(b) shows the run-time of MuCoRA for case-studies with a variable number of time slots in each time wheel. MuCoRA has the complexity $O(q_i^{c_2})$ where $2 \leq c_2$. Here q_i denotes the number of time slots in each time wheel of the tile θ_i. Comparing c_1 and c_2 suggests that the run-time of MuCoRA for the tasks with hard deadlines is more affected by the number of slots in a TDMA time wheel than the number of tasks. This corresponds to the fact that for a given time wheel size, the run-time of Eq. 2.9 depends on the number of time slots in the time wheel. That is, by increasing the number of time slots in the time wheel, not only Eq. 2.9 is used more often, but also the run-time of every calculation of Eq. 2.9 increases. Considering both M and q_i, MuCoRA for a set of tasks with hard deadlines has complexity $O(M^{c_1} q_i^{c_2})$. As shown in Fig. 2.19 the run-time of MuCoRA is acceptable for both design time and

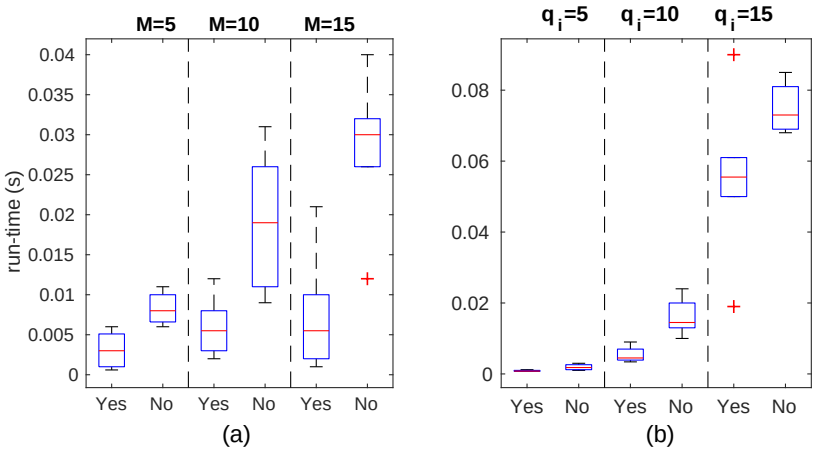

Figure 2.19: The run-time of MuCoRA for slot allocation of hard control applications with (a) various numbers of tasks in task sets and (b) various numbers of time slots in a TDMA time wheel.

run-time usage.

Next, we evaluate the run-time of MuCoRA for the resource allocation of firm tasks. The number of firm tasks M and the number of time slots in a time wheel q_i affect the run-time of MuCoRA for a set of firm tasks as well. However, to verify whether an assigned time slot can be unassigned for a task, the FAn method should be applied to verify satisfaction of firmness conditions. The run-time of the firmness analysis of a firm task (using the FAn method) is more than that of the feasibility test of a task with hard deadlines. Therefore, we expect the run-time of MuCoRA for firm tasks to be more than that of MuCoRA for control applications with hard deadlines. The complexity of the FAn method for TDMA scheduling is independent of the number of tasks M and the number of time slots in a time wheel q_i (see Chapter 4). Therefore, we expect MuCoRA for firm tasks to be less affected by the number of slots in a time wheel than MuCoRA for tasks with hard deadlines. Moreover, we expect MuCoRA to scale linearly in the number of (m, k)-firmness conditions per task. To evaluate the scalability of MuCoRA for a set of firm tasks, we therefore considered 100 different combinations of randomly chosen firm tasks with a single (m, k)-firmness condition. In

each experiment, we intend to map firm tasks to two TDMA time wheels (of the two tiles) which are previously partially allocated to other applications. Fig. 2.20(a) and Fig. 2.20(b) show the run-time of MuCoRA for variable M and q_i. Our experiments show that MuCoRA for a set of (m, k) tasks running under a TDMA policy has complexity $O(w_i k M^{c_3} q_i^{c_4})$ where w_i is the length of each time wheel of the time θ_i, k is the number of consecutive samples in the firmness condition and $1 \leq c_3 \leq 2$ and $1 \leq c_4 \leq 2$. Comparison between c_4 and c_2 implies that the run-time of MuCoRA for firm tasks is indeed less affected by q_i than MuCoRA for hard tasks. This corresponds to the fact that the run-time of the FAn method is not affected by the number of slots in a time wheel as explained above. w_i and k affect the run-time of the FAn method that will be discussed in Chapter 4. Comparison of the absolute values of the run-time of MuCoRA in Fig. 2.19 and Fig. 2.20 shows that the dominant factor in the run-time of MuCoRA for firm tasks is the run-time of the firmness analysis, i.e., the run-time of the FAn method. Therefore, given that the FAn method has considerably lower run-time compared to the state-of-the-art methods for firmness analysis (as shown in Section 4.6 and Section 5.5), MuCoRA is also expected to have a lower run-time compared to the cases where firmness analysis is performed by other techniques.

We take the same 100 combinations of firm task sets to evaluate the scalability of MuCoRA with the number of firmness constraints for each task. We considered fixed values for M and q_i, i.e.,$M = 5$ and $q_i = 10$. We compare the run-time of positive schedules and non-positive schedules for varying numbers of firmness constraints for each tasks in the range $[1, 4]$. m and k for firmness constraints are chosen randomly with the following conditions on their range: $5 \leq k \leq 150$ and $m < k$. Fig. 2.21(a) and Fig. 2.21(b) show the run-time of positive, shown with Yes, and non-positive, shown with No, schedules against the number of firmness constraints for each task. Fig. 2.21 implies that the run-time of MuCoRA scales linearly with the number of firmness constraints for each task. This corresponds to the fact that the run-time of MuCoRA for firm tasks is comprised mostly of the run-time of the FAn method (see Chapter 4). During a resource allocation with MuCoRA, the number of calls of the FAn method scales linearly with the number of firmness constraints because for each candidate schedule the (m, k)-firmness conditions must be verified.

2.5.4 Resource efficiency

The main motivation for considering firm tasks in real-time system design is to provide an efficient resource allocation. An efficient resource allocation can be evaluated with different design metrics. Mapping a higher number of tasks

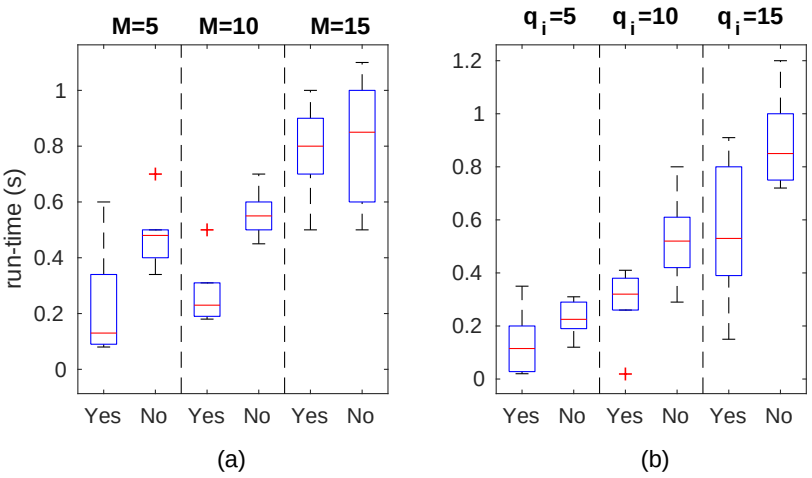

Figure 2.20: Run-time of MuCoRA for resource allocation of firm tasks with (a) varying number of tasks and (b) varying number of time slots in a TDMA schedule.

on a resource is one of the design metrics with which we can evaluate the efficiency of a design with deadline misses. We take the same combinations of firm tasks for which we evaluated the scalability of MuCoRA for mapping firm tasks explained in Section 2.5.3. We further use MuCoRA to allocate resources to the same task sets considering hard deadlines for all the tasks. That is, each task has the same configuration except the firmness conditions, which is (k, k), i.e.,all the deadlines must be met. We consider the number of positive schedules for hard and firm task sets obtained by MuCoRA out of the 100 case studies. 43 positive schedules for hard tasks and 71 positive schedules for firm tasks are obtained using MuCoRA. This shows that in many cases accepting some occasional deadline misses allows mapping more tasks to resources. Considering firm tasks, the resource utilization increases by 19% in average and 40% in the best case in our experiments. Note that this resource efficiency is obtained with the cost of reduction in QoC because of deadline misses. Designing with deadline misses is only applicable in situations where this reduction in QoC is acceptable for the system.

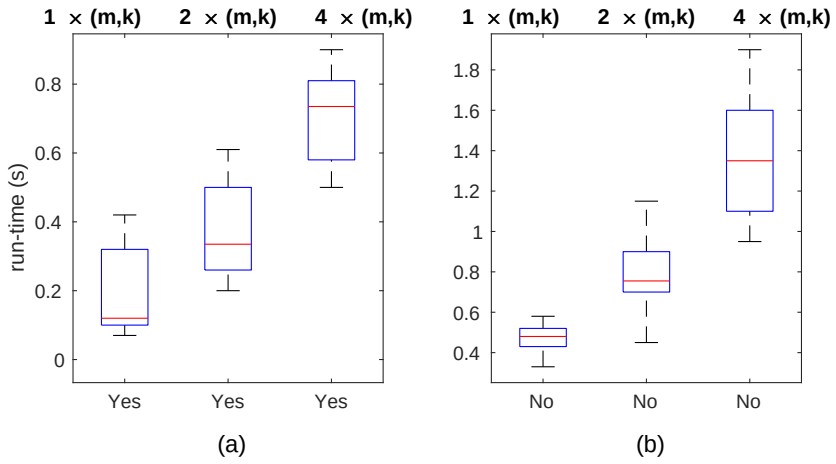

Figure 2.21: Run-time of MuCoRA for multiple (m, k)-firmness constraints of each task for (a) the combinations that result in positive schedules and (b) the combinations that do not result in positive schedules.

2.6 Related work

In view of the setup we consider in this chapter (Fig. 2.1), the existing literature can be classified into two categories: (i) resource allocation and mapping methods for feedback control applications and general latency-constrained applications with hard and (m, k)-firm deadlines (ii) resource allocation/mapping methods to meet or optimize throughput for streaming applications. Our work addresses a combination of (i) and (ii). No existing work addresses this combined problem, but the literature on both (i) and (ii) is relevant.

Over the last decade, there has been a considerable amount of work on control/architecture co-design. The idea is to allocate or optimize shared communication or computation resources for implementing one or multiple control loops with hard or (m, k)-firm deadlines. This is in line with direction (i). The problem is addressed considering different abstractions both for system dynamics (i.e., control point of view) and the architecture model. The method in [22] formulates a dynamic program to find the optimal scheduling for multiple control tasks with hard deadlines running under a non-preemptive priority-based policy and similarly, [17] presents a method to find the optimal sampling pe-

riods (i.e., latency constraints) in a similar setup as [22]. In [4], the resource requirements of control loops are translated into a compute-bandwidth constraint (rather than latency constraint). In [45], [31], scheduling control applications for both communication and computation resource is considered. In these classes of work, several scheduling policies in communication (e.g,, TDMA, Fixed-Priority Non-Preemptive) and computation (e.g., rate monotonic, EDF) are considered to derive the stability and performance region for a given combination of scheduling policies. The event-based controller proposed in [74] performs sampling (i.e., sensing, computing and actuating) only at the occurrence of events (e.g., crossing an error threshold) and saves communication bandwidth by avoiding unnecessary sampling. The mapping and scheduling problem for a combination of real-time and control applications is considered for automotive architectures in [31]. It should be noted that traditional real-time scheduling methods [43] are not directly applicable to schedule feedback control applications since it needs to explicitly consider the impact of jitter unlike deadline-driven real-time applications. In the context of obtaining m and k parameters for (m, k)-firm control applications, [47] analyzes network control systems considering packet drops governed by firmness constraints. [73] analyzes the performance of controllers under (m, k)-firmness conditions in the context of linear systems.

Along direction (ii), a large body of work has been reported on resource allocation for mapping of throughput-constrained applications onto multiprocessors [66], [70], [36]. MuCoRA works with all budget allocation techniques for streaming applications. [66] proposes resource allocation techniques for task binding and scheduling directly on an SDF graph of the throughput-constrained applications. In [36], the throughput-constrained streaming applications are modeled and mapped to the multiprocessor by an acyclic model in which an application task graph allows to run each tasks once. [70] encodes the problem of mapping and scheduling of dataflow applications on a multiprocessor platform in the form of logical constraints and presents it to a Satisfiability Modulo Theory (SMT) solver.

The mapping and scheduling to answer the combination of both (i) and (ii) directly have not yet been explored. However, some work addresses isolation of applications in shared resources. Based on the Worst-Case Execution Time (WCET) of intra-application tasks running on multiprocessors, [5] proposes a bus scheduling method to share a Network on Chip (NoC) among several processors. The approach in [29] uses a virtualization method for scheduling several applications on different shared resources (e.g. processors, interconnects and memory). In this method, using a TDMA policy, each resource is divided into smaller virtual resources. This allows independent design, verifi-

cation and execution of multiple applications.

2.7 Summary

In this chapter, we addressed the mapping problem of multiple control and streaming applications onto a TDMA-scheduled multiprocessor, to illustrate that designing with deadline misses is meaningful and leads to efficient resource usage. We considered both control applications with hard deadlines and firm tasks. We discussed the process of obtaining firmness parameters for firm control applications considering its performance and stability. While state-of-the-art resource allocation methods either consider one type of constraint or translate all constraints to one category, our method treats the constraints from different domains separately, but in a holistic fashion. We have shown that distributing the allocated resources for control applications reduces the resource requirement by control applications and that this improves overall resource efficiency. For streaming applications, it suffices to initially allocate only a budget and postpone slot allocation to the end. Although we have presented a specific instance of the mapping flow considering three types of applications, it also works for any combination of one or two application types. Moreover, the observations underlying its design may also be meaningful for other setups. The results of this chapter form a foundation for designing real-time systems with deadline misses. The remaining chapters of this thesis focus on firmness analysis.

Chapter 3

Balloons and rake problem

This chapter introduces the balloons and rake problem. This problem addresses the synchrony between two periodic events. One of the events occurs finitely many times and the other one occurs infinitely many times. We intend to obtain the maximum possible number of times that two events have a given relative occurrence time. In the following chapter we show that this problem represents a common challenge in schedulability analysis of (m, k)-firm real-time tasks. We propose a solution to this problem based on the Finite Point (FP) method [12]. We also illustrate other instances of this problem in other application domains at the end of this chapter.

3.1 Problem formulation

Definition 2. *(Balloons and Rake Problem)*
Let a line of infinitely many balloons be formed of N possibly different balloons which periodically repeat infinitely many times with period p. Each balloon is specified by its width and its position in a period of the balloon line. Determine the maximum possible number of balloons that can be struck with one hit by a rake with r identical blades on the line of balloons. The blades of the rake have width 0 and distance d from each other. The number of struck balloons is the number of the blades located on balloons.

For every balloon, we refer to its edge with the lower value on the x axis, as the *left end* and with the higher value of x as the *right end* (see Fig. 3.1(a)). Starting the counting from a random balloon, we denote the i^{th} balloon in a period of the line of balloons with $b_i = (w_i, x_i^L)$ where w_i and x_i^L are its width

and left-end position, respectively. Here $i \in [1, N]$. We denote the right end of the balloon by x_i^R.

Example 3.1.1. *Consider a line of balloons formed of two balloons, i.e.,*$N = 2$*; in each period (see Fig.3.1(a)),* $b_1 = (2.5, 0)$ *and* $b_2 = (2, 3)$*. This pattern of balloons repeats with period* $p = 6$*. We intend to obtain the maximum number of struck balloons with one hit by a rake with* $r = 5$ *blades which have a distance* d *of* 2.5 *from each other. Without loss of generality, we assume that a period of the balloon line starts at* $x = 0$ *and the left end of the balloon* b_1 *is located at* $x_1^L = 0$*. Therefore, there are balloons identical to* b_1 *with left end at* $..., -12, -6, 0, 6, 12, ...$

Definition 3 (Balloon Function)**.** *Balloon function* $f : \mathbb{R} \to \{1, 0\}$ *takes* 1 *if there exists a balloon at* x *and* 0 *otherwise. Formally*

$$f(x) = \begin{cases} 1 & \text{if } \exists i \in [1, N] : \exists z \in \mathbb{Z} : \quad x_i^L + zp \leq x < x_i^R + zp \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Note that the right end of a balloon is not part of the balloon, as opposed to the left end of it. That is, the intervals where $f(x) = 1$ have a closed left and open right. Therefore, the intervals where $f(x) = 0$ have closed left and open right too. This allows us to determine the minimum number of struck balloons, i.e.,the minimum number of blades located at a position x with $f(x) = 1$, by obtaining the maximum number of blades located at a position x with $f(x) = 0$ and subtracting the result from r. This is discussed later in this chapter.

Definition 4 (Strike Function)**.** *Strike function* $m : \mathbb{R} \to \{1, 0\}$ *denotes the number of struck balloons when the first blade of the rake is located at* x*.* $m(x)$ *is obtained as follows:*

$$m(x) = \sum_{n=0}^{r-1} f(nd + x). \tag{3.2}$$

We refer to the position of the first blade as the *rake offset*. Fig. 3.1(b) and Fig. 3.1(c) show $f(x)$ and $m(x)$ in Example 3.1.1.

3.2 Finite Point (FP) method

We observe that the FP method introduced in [12] can be adapted to solve the balloons and rake problem. Since the balloons are positioned periodically

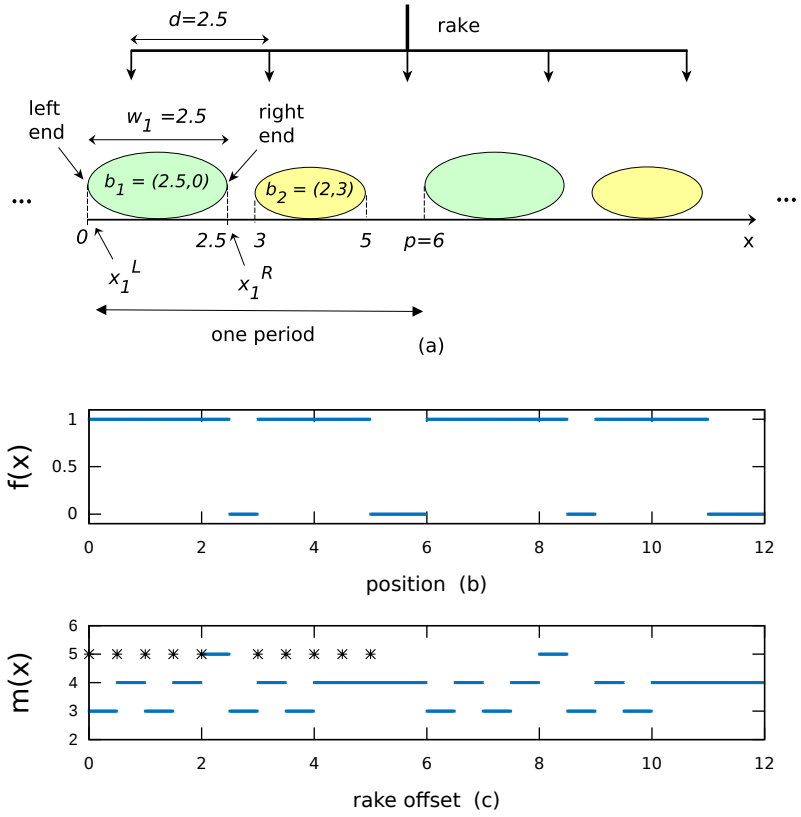

Figure 3.1: Example. 3.1.1. (a) the balloon line and the rake. (b) the balloon function $f(x)$. (c) the strike function $m(x)$. $O_{cnd-max}$ is shown by stars.

with a period of p, it is concluded that $f(x)$ is also periodic with the same period. From this and Eq. 3.2, it is concluded that $m(x)$ is also periodic with the same period. Therefore, considering all possible rake offsets in one period of balloons, i.e.,$0 \leq x < p$, guarantees obtaining all the possible combinations of struck balloons. However, the number of possible rake offsets in one period of the line of balloons is still infinite. We use the result of the following FP theorem to obtain the maximum number of struck balloons by evaluating $m(x)$ for a finite number of different rake offsets.

Theorem 1. *(Finite Point Theorem) Any increase in the value of $m(x)$ occurs on a rake offset where at least one of the blades is on the left end of a balloon. Formally,*

$$m(x-\epsilon) < m(x) \;\Rightarrow\; \forall \epsilon > 0 : \exists z \in \mathbb{N} : \; \exists n \in [1, r] : \\ nd + x - \epsilon < zp + x_i^L \leq nd + x$$

Proof. From $m(x-\epsilon) < m(x)$ and Eq. 3.1, we conclude that

$$\sum_{n=0}^{r-1} f(nd + x - \epsilon) < \sum_{n=0}^{r-1} f(nd + x). \tag{3.3}$$

Since the summations in the two sides of Eq. 3.3 have the same number of terms, there is at least one n for which $f(nd+x-\epsilon) < f(nd+x)$. Moreover, from Def. 3, $f(x)$ takes only 0 and 1. Therefore, $f(nd+x-\epsilon) = 0$ and $f(nd+x) = 1$. From Eq. 3.1 we conclude that $nd + x - \epsilon < zp + x_i^L \leq nd + x$ for some z and n. □

Theorem 1 implies that, to obtain the maximum number of struck balloons we need to obtain $m(x)$ only for those rake offsets in one period of the balloon line where a blade is located at the left end of a balloon. Comparing the values of $m(x)$ for these finite number of offsets, we obtain the maximum value of $m(x)$.

Let $O_{cnd-max}$ be a set of rake offsets in which a blade is located on the left end of a balloon, i.e.,x_i^L. $O_{cnd-max}$ is obtained as follows

$$O_{cnd-max} = \{mod(mod(x_i^L - nd, p) + p, p) \mid i \in [1, N], n \in [0, r-1]\} \tag{3.4}$$

Here, *mod* is the modulus operator defined as follows

$$mod(x, y) = x - y \left\lfloor \frac{x}{y} \right\rfloor. \tag{3.5}$$

In Eq. 3.4, i and n can take N and r different values, respectively. Therefore, $O_{cnd-max}$ can have at most $N \times r$ elements. Thus, to obtain the guaranteed maximum number of struck balloons we need to compute $m(x)$ for only $N \times r$ different rake offsets. m_{max} denotes the maximum possible number of struck balloons.

$O_{cnd-max}$ for Example. 3.1.1 is shown with stars on Fig. 3.1(c). The fact that there is at least one star at any offset where there is an increase in the

value of $m(x)$ validates the FP method. $O_{cnd-max}$ has $2 \times 5 = 10$ elements. Comparing the value of $m(x)$ for all these rake offsets, we conclude that the maximum of $m_{max} = 5$ balloons are struck with the rake offset of 2.

Overall, to solve a balloons and rake problem the two main steps are obtaining $O_{cnd-max}$ using Eq. 3.4 and comparing the value of $m(x)$, obtained from Eq. 3.2, for all these offsets.

3.3 Extension of the balloons and rake problem

Def. 2 requires the maximum possible number of struck balloons with one hit of the rake as the output of the balloons and rake problem. This section explains how we can obtain the minimum possible number of struck balloons.

The minimum possible number of struck balloons corresponds to the maximum number of rake blades that hit the areas between any two consecutive balloons. We refer to these areas as anti-balloon zones.

Definition 5 (Anti-balloon Function)**.** *Anti-balloon function* $f' : \mathbb{R} \rightarrow \{1, 0\}$ *takes* 0 *if there exists a balloon at* x *and* 1 *otherwise. Formally*

$$f'(x) = 1 - f(x) \tag{3.6}$$

where $f(x)$ *is the balloon function.*

From Eq. 3.6 and Eq. 3.1 we conclude that $f'(x)$ has the same properties in terms of the left and right end of the anti-balloon zones. That is, the right end of an anti-balloon is not a part of the anti-balloon, as opposed to the left end of it. Therefore, the intervals where $f'(x) = 1$ have closed left and open right similar to balloon areas where $f(x) = 1$.

Definition 6 (Anti-strike Function)**.** *Anti-strike function* $m' : \mathbb{R} \rightarrow \{1, 0\}$ *denotes the number of the rake blades that are positioned in an anti-balloon when the first blade of the rake is located at* x*.* $m'(x)$ *is obtained as follows*

$$m'(x) = r - m(x). \tag{3.7}$$

where $m(x)$ *is the strike function (see Definition 4)*

Let m'_{max} be the possible number of rake blades that hit an anti-balloon. Using the FP method we can obtain m'_{max}. Then using Eq. 3.7 we obtain

$$m_{min} = r - m'_{max}$$

where r is the total number of the rake blades and m_{min} is the minimum possible number of rake blades that hit balloons.

3.4 Applications of the balloons and rake problem

In this section we discuss the possible applications of the balloons and rake problem in different domains of engineering. We first illustrate a traditional challenge in the aviation industry that can be translated to the balloons and rake problem and be solved using the FP method. Finally, we explain the essential features of a problem to be solvable using the results of the balloons and rake problem.

3.4.1 Rifle and propeller problem

Before introduction of jet propulsion [52], single-engine tractor-type aircraft (see Fig. 3.2(a)) were commonly used as a fighter aircraft. Such aircraft has a spinning propeller in the front that converts rotational motion of the engine into thrust that pulls the aircraft forward. In early editions of such fighters the armament was attached to the front part of the aircraft. This armament is required to fire through the arc of the spinning propeller. Therefore, the main challenge is to synchronize automatic guns and rotation of the propeller so that the bullets do not hit the blades of the propeller. That requires deciding the relative timing between start of the gun firing and certain position of the propeller so that none of the successive n bullets hits a blade of the propeller. Fig. 3.2(b) shows an instance in which a bullet hits a blade of a fighter propeller. The challenge above is a special case of the more general *rifle and propeller* problem defined as follows.

Definition 7. *(Rifle and Propeller problem)*
Let a propeller with N possibly different blades rotate with a speed of v. A rifle with r bullets is firing towards the propeller with a period of p. Obtain the start time of the rifle firing, i.e.,shooting offset, so that no bullet hits a propeller blade.

Fig. 3.3 shows the busy window of an example of firing four bullets towards a propeller with three blades.

The time duration that a blade b_1 blocks the rifle shooting direction is calculated as follows.

$$t_1 = \frac{w}{\pi d^2 v}$$

where d is the distance between the rifle barrel and the center of a propeller and w is the width of the blade b_1.

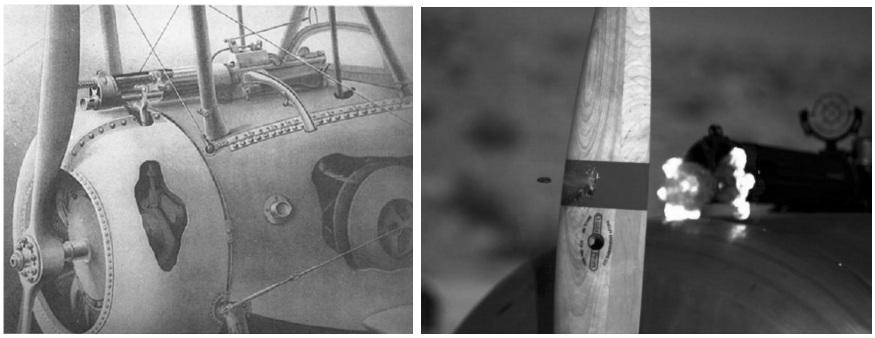

Figure 3.2: (a) a single-engine tractor-type aircraft with an armament located behind the propeller [55]. (b) an example of a failure scenario when the bullet hits a blade of the propeller [56].

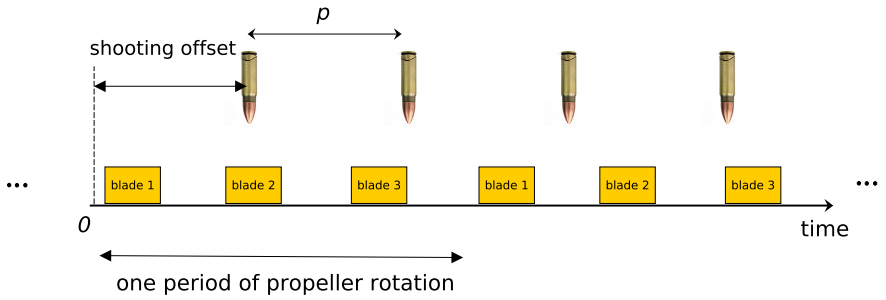

Figure 3.3: An instance of a rifle firing and a propeller rotating in front of the rifle.

This problem can be reduced to the balloons and rake problem. In this regard, the parameter *time* in the rifle and propeller problem must be substituted by the parameter *position* in the balloons and rake problem. Moreover, we consider the time interval that each blade is passing in front of the rifle as the size of a balloon. Besides, the period of the rifle shooting is considered as the distance between two successive blades of a rake. Then, the number of rifle bullets is interpreted as the number of rake blades. Table 3.1 shows the parameters of the rifle and propeller problem corresponding to those of the

balloons and rake problem.

Table 3.1: Balloons and rake problem parameters corresponding to the rifle and propeller problem

type	Balloons and rake problem	Rifle and propeller problem
input	# of balloons in each period N	# of propeller blades N
	# of rake blades, r	# of the rifle bullets b
	size of each balloon	propeller blade block time $t_1 = \frac{w}{\pi d^2 v}$
	distance between rake blades, d	rifle shooting period p
output	rake offset for zero balloon hits	shooting offset for zero propeller hits

In the rifle and propeller problem we intend to obtain the shooting offset that results in shooting all the bullets at instances when there is no blade blocking the rifle. This is corresponding to obtaining an offset for the rake that results in no struck balloons. That is, we need to obtain the offsets that result in the minimum number of struck balloons. This minimum number must be equal to zero; otherwise we conclude that shooting all the bullets without hitting a propeller blade is impossible. Therefore, one of the offsets that results in the zero struck balloons can be considered as the shooting offset which results in no blades hit.

It is noteworthy that the rifle and propeller problem was solved by a hardware solution, i.e.,*synchronization gear* [55], in later editions of the fighter (see Fig. 3.4). A synchronization gear pushes the trigger of the rifle at certain times relative to the position of the propeller so that the bullet passes through the arc of the propeller without hitting a blade.

3.4.2 Essence of the balloons and rake problem

This section answers the question of which problems can be solved using the FP method. We discuss the distinguishing essence of the balloons and rake problem and the rifle and propeller problem that allows us to use the FP method.

The balloons and rake problem and the rifle and propeller problem are basically the interaction of two periodic events that are either synchronous or

Figure 3.4: The synchronization gear of a Messerschmitt Bf 109E [55]. A wooden disc is attached to the propeller to idicate where each bullet is passing through the propeller in each round.

are intended to be asynchronous. This periodicity can be either in time or position. As examples, the periodicity of the rifle and propeller problem is in time and the periodicity of the balloons and rake problem is in position. In the case that two events are synchronous, we are usually looking for either the best case or worst case. The balloons and rake problem falls in this category as the main purpose of the problem is to obtain the maximum and minimum possible number of struck balloons. However, in the case that two periodic events can be synchronized, we usually intend to obtain a particular synchrony between the events. The rifle and propeller problem is of this type as we intend to obtain a fixed shooting offset that results in shooting all the bullets without hitting a propeller blade. The periods of the two events are fixed. This feature in the balloons and rake problem can mean that the the rake cannot be stretched to have different distance between blades. This also means that in the rifle and propeller problem the firing frequency is fixed. Finally, it is important to notice that in the balloons and rake problem one of the two periodic events has zero width while the other one has non-zero width. That is, the blades of the rake have zero width and balloons have non-zero width. In the rifle and propeller problem, we assume that the shooting process has zero duration

while the propeller blade blocks the shooting direction for a time period.

3.5 Summary

In this section, we introduced the balloons and rake problem that obtains particular relative alignment between two periodic events with finite and infinite repetitions. We used the Finite Point method to solve the problem. We use the results of this discussion in the following chapters to solve firmness analysis problems. One possible extension of the balloons and rake problem is the case where there are more than two periodic events. That is, there is more than one line of balloons, for example. These lines of balloons are located on top of each other with arbitrary alignment. The challenge in this problem is to obtain the maximum/minimum number of overall struck balloons in all the layers with one hit of a rake.

Chapter 4

Firmness analysis for a TDMA policy

A Time Division Multiple Access (TDMA) scheduling policy is used as a time isolated budget scheduler for a set of real-time tasks sharing a resource. This scheduling policy is of a particular interest for scheduling a set of multi-constraint applications, i.e. throughput and latency constraints. Lack of budget allocated to a task – in terms of allocated time slots in a work cycle of the resource – may cause occasional deadline misses of the task. However, missing deadlines is tolerable to (m, k)-firm tasks as long as the number of Deadline Hit (DHs) jobs, i.e. each instance of a task, is at least equal to m in any k consecutive jobs of the task. Firmness analysis guarantees the satisfaction of such conditions in a given schedule.

We propose the Firmness Analysis (FAn) method that obtains the minimum number of DHs for a set of tasks running under a given TDMA schedule[1]. The FAn method translates the firmness analysis problem to that of the balloons and rake problem and uses the Finite Point (FP) method to solve it. We extend the FAn method to that of a timed-automata model to consider jitter in the period of the task under the firmness analysis. Moreover, we use a brute-force search approach to evaluate the run-time of the FAn method and the timed-automata method. Comparing the run-time of the three methods for a realistic case study shows an acceptable run-time for both FAn and time-automata methods. However, the FAn method scales better with the size of the TDMA time wheel and the length of the firmness window, i.e.,k.

[1]These results are based on [12].

4.1 Motivation

Real-time applications in different domains, e.g. healthcare and automotive, require to run multiple real-time applications simultaneously. Sharing resources among applications is a widely used trend towards a cost-efficient product development. This imposes new challenges in hardware and software design, as motivated in Chapter 1. Inter-application interference, for example, on a shared resource is a potential issue. A *Budget scheduler* provides temporal predictability on a shared resource by guaranteeing a fixed access time for every scheduled application [20]. Time Division Multiple Access (TDMA) is a common scheduling policy for realizing temporal predictability for such applications [2] [9]. It allocates identical slots with predefined length to applications in a work cycle.

Control applications are used to regulate the behavior of a physical dynamic system. Due to the safety-critical nature of control applications, timing plays a key role in guaranteeing their Quality of Control (QoC) [27]. Running control applications on a shared processor for computation reasons can cause control samples to miss the computational deadlines. We refer to control samples as control jobs that need to be processed on a processor. Missing deadlines for a control application affects the QoC. A job should be processed before the next job is released and therefore, each job has a computational deadline less than or equal to the activation period of the application. The jobs with missed deadlines are referred to as Deadline Misses (DMs). A potential reason for a job to miss its deadline is that insufficient resources are available for the application when the job is ready for processing. However, if a control application can tolerate occasional DMs, the above phenomenon can be considered as an opportunity. That is, if a task can tolerate some DMs, it can be mapped on a resource which is otherwise insufficient, as detailed in Chapter 2.

Under the (m, k)-firmness condition [35] a control application can still satisfy QoC requirements in presence of DMs. That is, at least m jobs out of k consecutive jobs must meet the computational deadline to satisfy application level requirements. In other words, $k - m$ jobs out of k consecutive jobs can miss the computational deadline without violating the requirements.

The possibility of missing computation deadlines of a given task running on a TDMA-scheduled processor is traditionally verified by calculating the best-case and worst-case response time of the task [60]. That is, if the activation period is larger than the worst-case response time of the control task no job misses the deadline. On the other hand, if the activation period is less than the best-case response time of the control application, all the deadlines are missed.

We consider control applications running on a shared processor under a TDMA policy. We are particularly interested in a range of activation periods that lies between the best and the worst-case response time of the control task. For such a range of activation periods, a certain (m, k)-firmness condition is given for each control application. We aim to formally verify the satisfaction of such conditions and in effect, guarantee QoC. We propose an analytic method to quantify the number of DMs. We propose a method to obtain the maximum number of DMs by verifying the number of DMs for a finite window of release of jobs. We further extend the method by considering parameter variation, e.g. jitter in the activation period. A timed-automata based method is presented to address this aspect. Because of an over-approximation in building the state space, the results of the timed automata method are conservative though.

4.2 Setup under consideration

We consider a control application with a given (m, k)-firmness requirement derived as explained in Section 2.2. The execution instances of the corresponding control task on a processor is referred to as *jobs*. In other words a job is an instance of a task. We consider a system in which dedicated hardware is responsible for operating the sensors and the actuators in a typical application scenario, as illustrated in Fig. 2.3. The sensors send the jobs, i.e.,sensor data, to a main processor unit, e.g. a controller Electronic Control Unit (ECU) [32], through a communication network. The time instance a job is ready to be executed on the processing unit is referred to as *release time* of the job. Similar to Chapter 2, we consider the processing unit to run under a TDMA policy. In a TDMA policy, a work cycle named *time wheel* consists of multiple *time slots* – allocated to the tasks. This provides a fixed periodic access time for each task. Once a job is released on the processor, it is processed only if the current time slot is allocated to the task that the job belongs to. Otherwise the job has to wait until the time in which the processor is allocated to the task. Each job needs to be executed on the processor for a certain amount of time to generate the corresponding actuating data. This time duration is referred to as *execution time* of a job. If the execution of a job cannot be completed before the next one is released for a given slot assignment, we encounter a DM.

Based on the fact that the activation clock (on the dedicated hardware) is usually asynchronous with respect to the processor clock, the release time of a job is non-deterministic. Depending on the release time of a job, the pattern and the number of DMs can therefore vary. Then, for a given set of tasks and platform settings, the challenge is to obtain the absolute maximum number

of DMs to verify (m, k)-firmness conditions. While the relative release time of a job is non-deterministic, the relative release time of the successive jobs can be expressed in terms of the release time of the first job (in a window of k consecutive jobs). In Section 4.4, we use this fact to propose a method for maximum DMs quantification.

4.3 Problem definition

We model a control task in the same way as in Chapter 2. We repeat the definition for completeness.

Definition 8 (Control task)**.** *A control application is a tuple $\tau = (C, T, D)$ consisting of execution time C, activation period T and execution deadline D.*

We discuss in the successive sections that deviation in the execution time of a task has monotonic effect on our results and we may choose the worst-case execution time in our method. We assume that the deadline of all tasks are less than or equal to their activation period, i.e.,$D \leq T$.

We consider a TDMA schedule with finite number of time slots allocated to tasks such that each task may be executed on the time slots allocated to it. As a timing-compositional scheduling policy, TDMA allows us to analyze each task separately. From an application point of view, a TDMA time wheel can therefore be classified into two types of intervals: *allocated intervals* and *non-allocated intervals*. An allocated interval refers to an interval in which the processor is assigned to the application. A non-allocated interval includes the intervals that are allocated to other applications or not used for processing purposes (e.g. context switching overhead [72]).

Definition 9 (TDMA Schedule)**.** *A TDMA schedule is specified by a tuple $TDMA = (w, A)$ where w is the length of the TDMA time wheel and A is a non-empty subset of the finite set of intervals $\{(t^i_{start}, t^i_{end}) \in \mathbb{R} \times \mathbb{R} \mid 0 \leq t^i_{start} < t^i_{end} < w\}$ such that no two intervals overlap.*

In Def. 9, we may consider switching overhead [2] between two consecutive time slots. In such a case, $t^i_{end} \neq t^{i+1}_{start}$.

We consider that the relative release time between every task and the TDMA time wheel is not given. Therefore, a firmness analysis should obtain the minimum possible number of DHs in any k consecutive jobs of an (m, k)-firm task. Having this number equal to or more than m guarantees the satisfaction of the (m, k)-firmness condition.

4.4 DMs quantification: FAn method

4.4.1 Hit zone calculation

Let τ be a periodic task running under a TDMA schedule $TDMA = (w, A)$ where the set of time slots $\{(t^i_{start}, t^i_{end}), 1 \leq i \leq q_i(\#\text{ slots in } \theta_i)\} \subset A$ in the tile θ_i are assigned to τ. The *Accessibility Function* specifies the time instances when τ has access to the processor.

Definition 10 (Accessibility Function)**.** *The Accessibility Function* $l : \mathbb{R} \rightarrow \{0, 1\}$ *takes* 1 *if the processor is accessible to* τ *at* t *and* 0 *otherwise.*

From Def. 10 the Accessibility Function $l(t)$ is calculated as follows

$$l(t) = \begin{cases} 1 & \text{if } \exists n, i \in \mathbb{Z} : \ t^i_{start} + nw \leq t < t^i_{end} + nw, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Execution completion of a job of τ released at t before its deadline depends on the overall access time of the task to the processor from t to $t + D$ which is captured by *Total Accessibility Function.*

Definition 11 (Total Accessibility Function)**.** *The Total Accessibility Function* $g : \mathbb{R} \rightarrow [0, D]$ *indicates the overall access of* τ *to the processor between* t *and* $t + D$*.* $g(t)$ *is obtained as follows*

$$g(t) = \int_t^{t+D} l(x)dx. \tag{4.2}$$

Definition 12 (Deadline Hit job (DH))**.** *A job of* τ *released at* t *is a DH if* $g(t) \geq C$*.*

A job of τ that is not a DH is referred to as a DM. We assume that a job which has missed its deadline is no longer executed. In other words, once a task has a release at t there is no demanded (unfinished) workload from previous jobs of the task. This execution strategy is particularly applicable to feedback control applications where execution of outdated samples can cause instability (see Section 2.2). Moreover, this helps us to leverage processor by executing less. Def. 12 implies that we can decide whether a job of τ released at t is a DH immediately after its release. This can be used by a run-time scheduler to detect a DM immediately after its release and do not execute that job at all. This allows to have even a tighter resource dimensioning.

Table 4.1: Balloons and rake problem parameters corresponding to the FAn method

type	balloons and rake	FAn: TDMA schedule
input	N and p in the form of f	h
	# of rake blades, r	k
	distance between blades, d	T
output	min # of struck balloons	min # of DHs

Definition 13 (Hit Zone Function)**.** *Hit zone function* $h : \mathbb{R} \rightarrow \{0, 1\}$ *takes* 1 *if a job of* τ *released at* t *is a DH and* 0 *otherwise. Formally,*

$$h(t) = \begin{cases} 1 & if \quad C \leq g(t) \\ 0 & if \quad C > g(t) \end{cases} \tag{4.3}$$

Any time interval $t \in [t1, t2)$ with $h(t) = 1$ is referred to as a *hit zone.*

Example 4.4.1. *Let* $\tau_1 = (2.2, 7, 7)$ *be a* $(8, 10)$*-firm task running under a TDMA schedule* $TDMA = (5.5, A)$ *where* $A = \{(0, 1), (1.1, 2.1), (2.2, 3.2), (3.3, 4.3), (4.4, 5.4)\}$ *(times in ms). Note that, there is a switching overhead of* $0.10ms$ *between every two consecutive slots. The second and the forth time slots are allocated to* τ_1*. Fig. 4.1(a) depicts Accessibility Function* $l_1(t)$ *obtained from Eq. 4.1. Substituting* $l_1(t)$ *in Eq. 5.1 results in Total Accessibility Function* $g_1(t)$ *shown in Fig. 4.1(b). Here,* $C_1 = 2.2$ *denotes the execution time of* τ_1*. Fig. 4.1(c) shows Hit Zone Function* $h_1(t)$*.*

4.4.2 Computing minimum DHs

In order to obtain the minimum number of DHs in any k consecutive jobs of τ, we must obtain the minimum number of jobs that are released in hit zones out of any k consecutive jobs. We refer to the balloons and rake problem illustrated in Chapter 3. In order to connect the problem above to the balloons and rake problem we should define which parameters represent rake blades and balloon function f. In that context, each release of τ is considered as a blade of a rake with k blades for k consecutive jobs. The distance between every two blades is T (i.e.,period of τ). Next, we obtain the balloon function. In this regards, we assume that the intervals in which $h(t) = 1$, i.e.,hit zones, are the balloon areas.

Fig. 4.1(d) shows the number of DHs against the offset of a window of 10 consecutive jobs of τ_1. Moreover, the stars on Fig. 4.1(d) shows the candidate offsets of the minimum number of DHs obtained from the FP method. The fact that there is a star at any point where the number of DHs decreases validates the FAn method.

We obtain the minimum possible number of DHs with an assumption that τ has a constant execution time. Variation in the execution time of τ has a monotonic effect on the number of DHs [12]. Therefore, we may choose the worst-case execution time in the FAn method which still results in the minimum number of DHs. That is, in a case of a change in either of the relative release time and execution time of τ, the number of DHs may increase. In such a case, the FAn method therefore yields conservative results which are used to validate the satisfaction of the (m, k)-firmness condition. In Example 4.4.1, the FAn method results in the minimum of 7 DHs for τ in any 10 consecutive jobs which implies that $(8, 10)$-firmness condition of τ is not satisfied.

4.5 DMs quantification: Timed-Automata method

DMs quantification for a case with a variation in the release times of the jobs of an (m, k)-firm task requires an exhaustive search over all the possible release times of all the jobs of the task in any window of k consecutive jobs of the task. In such a case, the FP method is no longer applicable as is assumes fix activation periods for the task. In this section we use a timed automata-based model as a general method to address variation in the value of such parameters.

Timed automata are finite state automata extended with real-valued clocks [3]. This formalism provides a dense time domain and allows non-deterministic choices with respect to both discrete transitions and the progress of time. There are several tools that support exhaustive analysis of timed automata with respect to temporal logic properties. In this paper, we use the Uppaal tool which is a widely used model checking tool used by both industry and academia [7]. Uppaal supports several extensions that increase the usability, such as a C-like language to manipulate discrete variables. Furthermore, it also supports generalizations of timed automata, i.e., stopwatch automata and linear hybrid automata. These formalisms are more expressive than timed-automata, but the types of analysis that can be done is more restricted: stopwatch automata [21] only allows an analysis based on an over-approximation of

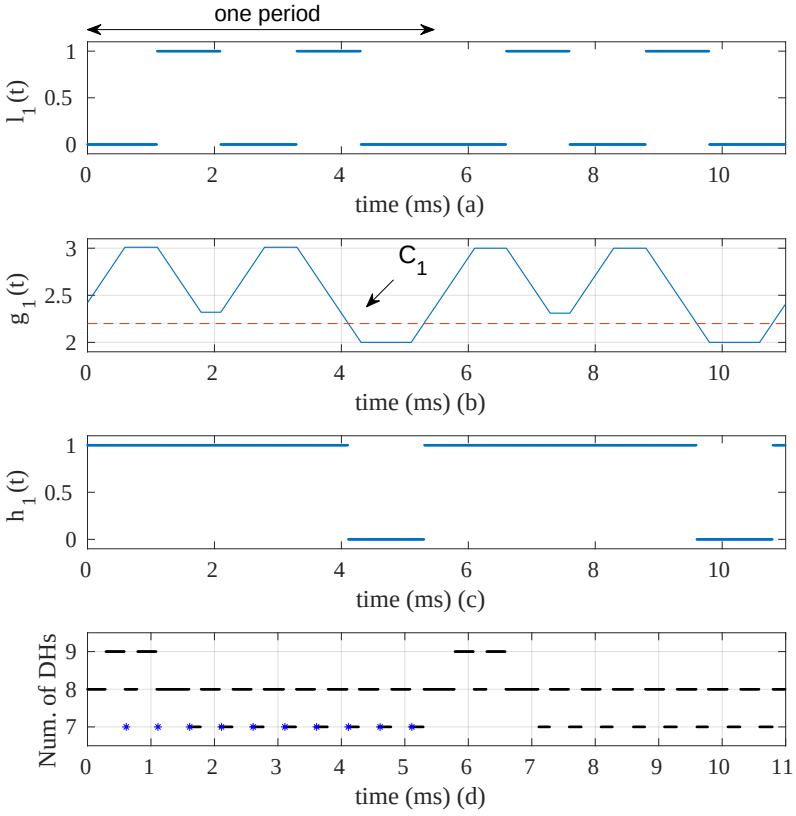

Figure 4.1: Example 4.4.1. (a) Accessibility Function $l_1(t)$. (b) Total Accessibility Function $g_1(t)$. (c) Hit Zone Function $h_1(t)$. (d) number of DHs against the offset of 10 consecutive jobs. The stars show candidate offsets for minimum number of DHs.

the state space, and linear hybrid automata allows analysis based on statistics and simulation techniques.[21] analyzes and reasons that the prize of over-approximation in not too high though.

The fact that Accessibility Function is periodic (see Section 4.4) allows us to model it as a timed automata. Let us take the example shown on Fig. 4.1. The TDMA time wheel is divided up into five time slots – two slots allocated to the control application, as shown on Fig. 4.1(a). By taking the periodicity

into account and by choosing a different offset (which does not change the problem), we see that the time domain is composed of consecutive intervals $[0, 10], (10, 22), [22, 32], (32, 55) \ldots$ of allocated intervals (closed intervals) and non-allocated intervals (open intervals). Note that, all the timings are multiplied in 10. This corresponds to the fact that, the expressions involving clock variables are limited to the integer domain.

Fig. 4.2 shows the UPPAAL encoding of the sequence of allocated and non-allocated intervals. We refer to this automaton as *TDMA automaton*. It has a local clock variable x to keep track of the time. Starting from an initial location, *AllocatedInterval1*, this automaton changes the location whenever x equals to the boundaries above and updates a boolean variable *execution* which indicates if the processor is allocated to the application in the next location. The automaton goes to the initial state once $x = w$ where w is the size of time wheel. The invariant $x \leq 10$ in combination with the guard $x == 10$ on the outgoing transition to *Non-AllocatedInterval1* ensures that the automaton can stay in *AllocatedInterval1* for exactly 10 time units before it is forced to go to *Non-AllocatedInterval1*. The other locations have the same behavior. Note that this automaton does not exactly model the Accessibility Function, because at the edges of the intervals, the automaton can be in both current and next locations, whereas the function defines that the automaton should either be in location *Non_AllocatedInterval1* or *AllocatedInterval2*. The behavior of the function is, however, included in the behavior of the automaton, which therefore provides a *conservative abstraction*. Besides using stopwatch, this is the second reason of over-approximation of the timed automata method.

Fig. 4.3 shows an automaton which keeps track of job generation time considering jitter bound for activation period T. This automaton is referred to as *sample provider*. It has a clock variable x, which specifies the generation time of the next job according to a job rate. Furthermore, it has three constant integer parameters: T, k and j. The parameter T is equal to the activation period, i.e.,$T = 70$ ($\times 0.1ms$) in the example shown on Fig. 4.1, and used to specify the time between generated jobs. The parameter k is used to specify the number of consecutive jobs that is generated. j determines the jitter bound if there is any, otherwise it takes zero. This automaton starts from the initial location *INIT*. The transition from *INIT* to *JITTER* is non-deterministic i.e.,it can be taken at any value of x less than w. This captures every possible offset of activation less than w of the activation with respect to the TDMA time wheel. The automaton can leave the location *JITTER* at any time less than j. This allows us to model jitter [57]. This non-determinism again causes the tool to consider all possible amounts for jitter during the maximum DMs quantification. Once the automaton takes a transition towards *SAMPLING*,

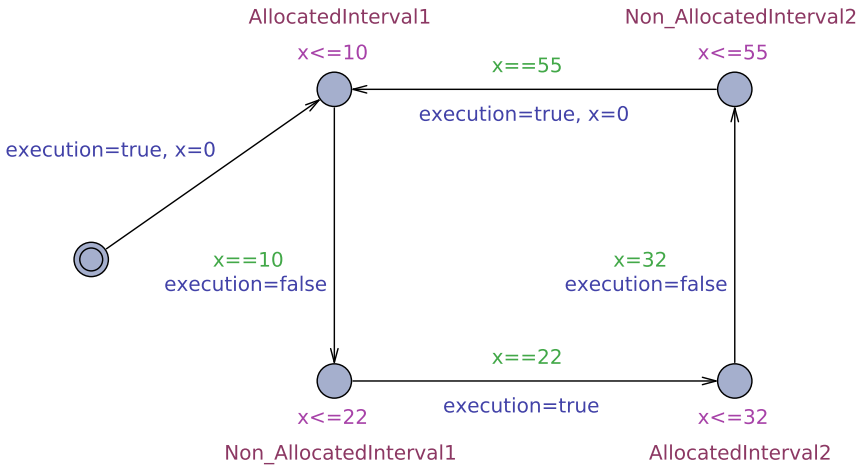

Figure 4.2: TDMA automaton for modeling a TDMA time wheel.

the value of the parameter s is incremented by one. Then it takes the transition to *SAMPLING* if we have not finished with the k jobs, increments the job counter n, and resets the clock. The automaton stays in the location *SAMPLING* until the clock equals the activation period and then resets the clock and takes the transition back to *JITTER*. Once all the k jobs have been generated, the transition to the location *END* is taken. This is an *urgent* location which must be left immediately, otherwise it causes a deadlock. This feature is used to stop the verification process in this time-automata model.

Based on the value of the parameters s and *execution* coming from automata *SampleProvider* and *TDMA*, a third automaton, named *Control application*, decides if a generated job has been executed properly or not. Fig. 4.4 shows the *ControlApplication* automaton. Here the variable s is a binary counter and the variable x is the clock of the automaton. As shown on Fig. 4.4, this automaton is initially located on *NOT_READY*. Once a job is generated, i.e.,the value of s equals 1, the automaton takes the transition to the location *READY* and decreases the value of s by one. This shows that the generated job is ready to be executed. Note that this transition like any other transition that has a synchronization *hurry* should be taken immediately once the guard of the transition ($s > 0$ in this case) is satisfied. The automaton stays in *READY* location until one of the following transitions are enabled: 1) The self loop is enabled if a new jobs is generated. That is, the previous job is

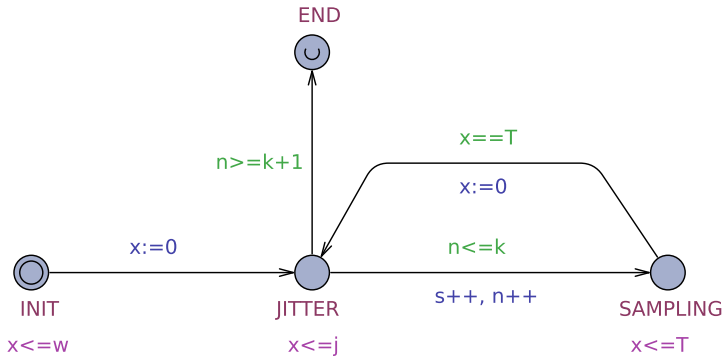

Figure 4.3: SAMPLE PROVIDER automaton to keep track of the generated jobs.

overwritten while it had not been started to be executed. This transition therefore increases the number of the missed jobs by one. 2) The transition to the location *RUNNING* is enabled if *execution* takes true, which shows that the automaton has gone to one of the locations *AllocatedInteval1* or *Allocated-Inteval2*. This transition to the location *RUNNING* also resets the clock x. In this automaton, the clock keeps track of the total time that *execution* has the value *true*, which shows that the processor is allocated to the application to execute the last job. If *execution* takes false, i.e.,the control application is pre-empted in real world, the automaton *ControlApplication* takes the transition to the location *fast decision*, which should be left immediately. If $x >= C$, which shows that the execution of the previous job has been completed, the automaton takes the transition to the initial location, otherwise it goes to the location *PREEMPTED* and stops the clock. Using stop watch causes an over-approximation of the state space which leads to conservative results. The automaton stays in this location until either the variable *execution* is changed to true or a new job is generated. The maximum number of DMs can be found using the UPPAAL query `sup : m`, which exhaustively analyzes the state space for the supremum value of variable m. The variable `m` indicates the number of DMs that is increased by 1 whenever the function `miss()` is run. For $k = 10$, for instance, UPPAAL reports maximum 3 DMs. Due to the over-approximation of this method we conclude that real number of DMs for this case is equal or less than 3. The verification takes $46ms$. We experiment more with the same example, for $k = 50$ and $k = 100$ UPPAAL reports 16 and 28 DMs with verification times of $760ms$ and $2s$, respectively.

In the example shown on Fig. 4.3, assume that there is a jitter of 2 in

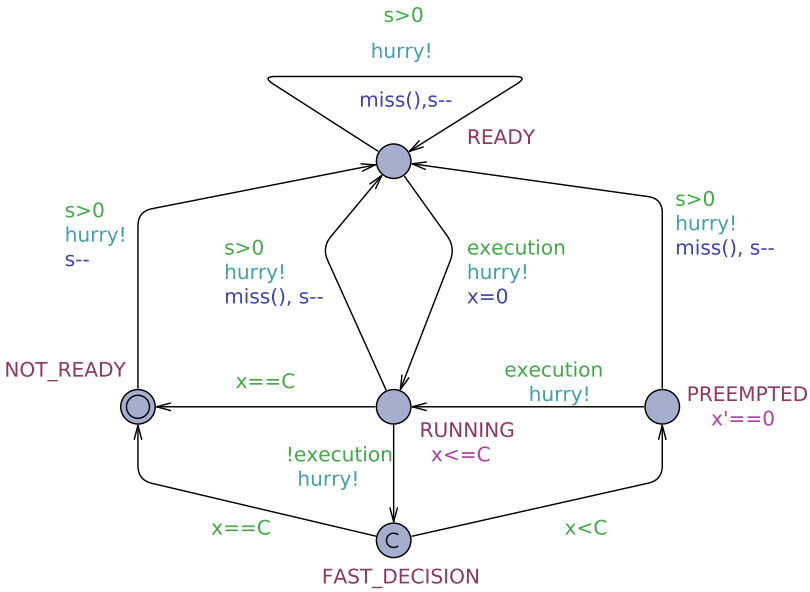

Figure 4.4: Control Application automaton for counting DMs.

the activation period for $h = 70$ (i.e.,$68 \leq h \leq 72$). All potential values for activation period should be considered for verification of the maximum number of DMs. This situation can be specified in the automaton *SampleProvider* by defining the jitter parameter $j = 4$. That is, the location *JITTER* can be left once $x \geq 0$ and must be left before $x = 4$. Applying this change in our case, it yields 5 DMs.

A potential limitation with respect to the use of timed automata is that the expressions involving clock variables are limited to the integer domain, i.e.,it takes only integer values. Thus, for example, modeling the situation in which the activation period equals to 2π time units is not directly possible. In that case, using same method as explained above, we can specify that the activation period lies between 6 and 7 time units by having an invariant $x \leq 7$ and a guard on the self loops with $x \geq 6$ in Fig. 4.3. In this case, UPPAAL reports an upper bound of 10 out of 10 samples which is not very useful. Of course, we can "scale" the time axis to increase the accuracy of this over-approximation, e.g., by using invariant $x \leq 628320$ (2×314160) and the guard $x \geq 628319$ and by multiplication of the other constants in clock guards with 10^5. Now UPPAAL reports an upper bound of 3 out of 10 misses. The analysis still is very efficient: the increased size of the clock constants and the timing uncertainty does not visibly add to the computation time (the UPPAAL GUI still reports 0 seconds).

4.6 Experimental evaluation

In this section, we first introduce another method, i.e.,brute-force search approach, to evaluate the results obtained by the FAn method and the timed-automata method. We then explain our experimental results of applying the proposed methods on a realistic case. Next, we make hypotheses about the scalability of the methods. We then show our results and compare the runtime of the three methods. Finally, we illustrate the effect of designing with deadline misses in the resource efficiency for one of our experiments.

4.6.1 Brute-force approach

We use a brute-force approach in obtaining the number of DMs for all the possible first release time of a window of k consecutive jobs. First release time of a job can take only the values that are integer multiply of the clock cycle of the processor. Therefore, given that the hit zone function is periodic with the period of the TDMA time wheel size, considering all the possible first

release time of the window in one period of the TDMA time wheel guarantees to obtain all the possible combination of DMs and DHs in the window.

4.6.2 Scalability analysis

The run-time of the FAn method mostly depends on the process of obtaining hit zones. This process depends on the period of the hit zone function. The period of the hit zone function equals to the length of a time wheel w in a TDMA policy. This justifies that the FAn method for the TDMA policy has complexity $O(w)$.

The process of calculating hit zones exists in the BF method too. Moreover, in the TDMA policy, the run-time of the BF method depends on the number of clock cycles in one period of the TDMA time wheel. Therefore, the complexity of the BF method for the TDMA policy is $O(wf^2)$ where f is the clock frequency of the underlying operating system.

The run-time of the timed-automata model depends on the size of the state space built during the reachibility analysis. We believe that in any timed-automata model of the TDMA policy, switching between different time slots and releasing a job of a task are considered as a new state. Our experiments show that the TA method for TDMA has complexity $O(k^2q^{c_1})$ where q is the number of time slots in one TDMA time wheel and $c_1 \geq 3$.

4.6.3 Case study

For illustration of the applicability of our proposed methods and their scalability evaluation, we consider a control application with sampling period of $2ms$. We consider a window of $k = 125$ consecutive samples in our experiments. Based on the specifications of the platform on which the application is running, execution time of the control task is determined. We took different sets of platform-related settings to verify the (m, k)-firmness properties of each. The FP method and the Brute-Force (BF) method was implemented in MATLAB and compiled on a computer with a quad-core processor and a clock frequency of $2.6GHz$. The same system was used to verify the Timed-automata (TA) model using Uppaal.

Table 4.2 shows four examples of the settings and results of our experiments. The values for the parameters in this example are chosen inspired by a realistic platform [29] for implementation of feedback control loops. In the table, w indicates the size of the TDMA time wheel, C denotes the execution time of the control application, and *run-time* shows the verification time. The last column of the table shows the maximum number of DMs for each case.

Table 4.2: Different sets of platform settings and verification results using Finite-point and timed-automata methods ($k = 125$)

w	allocated interval (μs)	C	run-time			max # of DMs
			FAn	TA	BF (1GHz)	
$1.3ms$	[0,80], [440,520], [870,950]	400μ	$520\mu s$	$31s$	$18s$	58
$1.3ms$	[0,175], [870,1000]	400μ	$430\mu s$	$22s$	$17s$	10
$700\mu s$	[0,250]	$600\mu s$	$225\mu s$	$12s$	$6s$	125
$700\mu s$	[0,250]	$600\mu s$	$265\mu s$	$18s$	$6s$	54

No jitter in sampling period is considered in the experiments reported in the table. It is noteworthy that despite the over-approximation in the timed automata method, the results obtained by the three methods are the same for this example. However, as it can be seen in Table 4.2, the verification times are quite different. The verification time of the UPPAAL model and BF method are at least two order of magnitude larger than that of the FP method. This justifies the use of the FP method for cases without jitter. Our experiments show that this difference in the verification time between the two methods increases as k increases. However, this verification time is still acceptable as a part of the design process. The timed-automata model can be simplified for a control application without jitter in the sampling period. Then the verification time would be shortened, though still longer than the verification time of the FP method. However in this work we introduce the timed automata model to address the cases with jitter.

Fig. 4.5 depicts the maximum number of the DMs against the sampling period for the set of settings in the first row of Table 4.2. From classical response time analysis it can be obtained that the worse-case and best-case response time for this example are $2.135ms$ and $1.77ms$. That is, a sampling period shorter than $1.77ms$ will result in all DMs while a sampling period longer than $2.135ms$ is enough to meet all the deadlines. The maximum number of DMs for sampling periods between the values above can be obtained using the method proposed in this work. The range of sampling periods between the above values gives different number of DMs as shown on Fig. 4.5. For a given platform

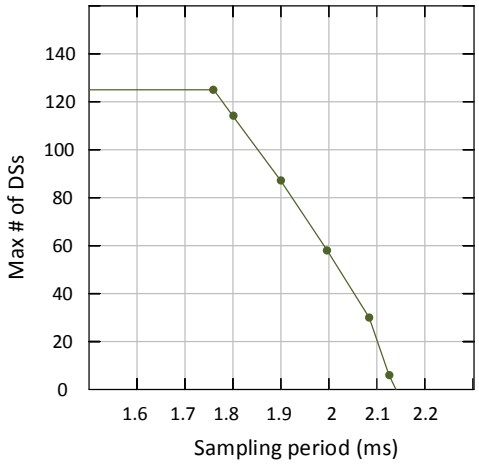

Figure 4.5: Maximum number of DMs against sampling period for the case in the first row of Table 4.2

settings, this analysis can be used to choose a suitable sampling period to meet an (m, k)-firmness bound (hence, to meet the QoC requirement). That is, considering (m, k)-firmness properties, we can reduce the sampling period to a value less than $2.135ms$ without allocating more processor resource to the application. Besides, we can reduce the allocated resource instead of changing the sampling period to have a resource efficient allocation. In the first set of settings in Table 4.2, for example, considering a sampling period of $2.135ms$ which results in no DMs, we can reduce 11% of the length of each allocated slice, i.e.,33% less resources allocated, to have 25 DMs.

4.7 Related work

A control application that has DMs has been analysed from two points of view in previous work: (i) Study the effects of DMs on the performance of control applications and improvement of controller design to tolerate the DMs. [27] investigates the effects of DMs in terms of stability and performance on the control loops closed over a network. [47] analyses the stability and the opti-

mality of a system governed by the (m, k)-firmness condition. [38] and [28] assign a scheduling architecture for a set of control applications to meet their requirements in presence of DMs. (ii) Quantification of a bound for DMs for a given set of applications and platform parameters. [32] addresses the DMs arising from the delay of the communication network between sensors and processor. The work investigates bounds on the number of deadline misses such that the control application retains its stability and satisfies the QoC requirements. The focus of [32] is mainly on an automotive specific communication network as a platform and hard to generalize for typical multi-processor settings. [76] provides a general formulation in the context of systems where some tasks occasionally experience sporadic overload for obtaining a tight bound of (m, k)-firmness properties. The need for overload models makes results in [76] not directly applicable to periodic control tasks. Our work belongs to category (ii). We focus on (m, k)-firmness analysis of a given TDMA-scheduled platform. We provide guarantees on the satisfaction of firmness constraints. Our results on (m, k)-firmness bounds can be taken into account in the controller design phase (i.e., category (i)) and vice versa.

4.8 Summary

An analytical method was proposed to quantify the maximum number of deadline misses for a given control task governed by an (m, k)-firmness condition running on a TDMA–scheduled processor. We showed that the proposed FAn method verifies (m, k)-firmness properties with acceptable verification time for use at design time. We extended the problem for the cases with a variation in the sampling period. To this end, a timed-automata based method was used to model a TDMA time wheel and the sampling process of a control application. The timed-automata based method has a longer verification time compared to the FAn method for cases where both methods are applicable. However run times are still acceptable for practical purposes. The FAn method provides an exact bound, whereas the timed automata method yields conservative results. The results of the FAn method have been used in Chapter 2 for resource efficient mapping of a set of multi-constraint applications, i.e.,applications with throughput and latency constraints, in the MuCoRA method.

Chapter 5

Firmness analysis for SPP policies

A Static Priority Preemptive (SPP) scheduling policy provides a resource-efficie-nt schedule for a set of periodic tasks with latency requirements. Besides, considering (m, k)-firm tasks, it allows us to design with tighter resource dimensioning. However, firmness analysis is required to guarantee the satisfaction of the (m, k)-firmness condition in an SPP policy. In this chapter, we propose an analytical method for firmness analysis of a set of periodic tasks running under an SPP policy. We extend the FAn method (proposed for firmness analysis of a TDMA policy in the previous chapter) to translate the firmness analysis problem of an SPP policy to that of the balloons and rake problem. We consider synchronous and asynchronous sets of tasks running under an SPP policy. Our method provides an exact solution to obtain the maximum possible number of Deadline Hits (DHs) for an (m, k)-firm task that is intended to be added to a set of tasks with given first release times under an asynchronous SPP policy[1]. We also obtain the first release time for such a task which results in the maximum number of DHs. We propose another method that obtains the minimum possible number of DHs of an (m, k)-firm task in a set of synchronous tasks running under an SPP policy[2]. Our analysis provides conservative results (due to over-approximation) which can be used as a sufficient condition to guarantee the satisfaction of (m, k)-firmness conditions.

[1] These results are based on [10].
[2] These results are based on [11].

5.1 Motivation

Schedulability analysis for shared resources is necessary to provide temporal predictability for real-time systems. Static-Priority Preemptive (SPP) scheduling is commonly used in many real-time systems, such as Autosar-based operating systems [49]. In an SPP system, fixed priorities are assigned to tasks and they can be preempted by activations of higher priority tasks. If preemptions cause *too much* delay in a task execution, the task misses its deadline. A firmness analysis aims to guarantee that the distribution of DHs is consistent with requirements of (m, k)-firmness conditions.

One crucial factor in the number of DHs is synchrony among the tasks, which significantly influences the interference. In this regard, strictly periodic tasks are divided into two classes: synchronous and asynchronous [23]. They differ in assigning a value to the release time of the first job of the task. For synchronous tasks, the *first release time* (see Chapter 1) is not fixed, relative to the other tasks, whereas for asynchronous tasks it is fixed[3]. With knowledge of the release time of the first job of asynchronous tasks, the interference from higher priority tasks becomes predictable.

In the process of mapping multiple hard and (m, k)-firm real-time tasks on a shared processor, the hard real-time tasks are usually mapped first in order to provide guarantees on their schedulability. Next, the (m, k)-firm tasks are mapped and they require further analysis to ensure satisfaction of their constraints. We consider a case in which a new (m, k)-firm asynchronous task is intended to be added to a set of asynchronous tasks running under an SPP policy. This is common in many mixed-criticality application scenarios where applications are incrementally mapped according to their criticality-level. We aim to identify the best-case release time of the newly added asynchronous task to obtain the *maximum minimum* (max-min) number of DHs in any window of k consecutive jobs. It is the lower bound for the maximum possible number of DHs and we refer to it as max-min DHs. While *maximizing* DHs is preferable for a better performance of the system, a *minimum* number of DHs is required for a guaranteed lower bound on the number of DHs. This guarantee is obtained by considering the worst-case execution time for all tasks. We show that the number of DHs can only grow if execution time of one or multiple jobs (of the task under firmness analysis or the task with the higher

[3]We follow the definition in [23]. If the first release times of a set of tasks are not known, they may potentially be released simultaneously (synchronous tasks). For the asynchronous tasks, the first release times of a set of tasks are known and task releases (typically) do not happen simultaneously. Note that the task definitions are exactly opposite in the work reported in [14][13].

priorities) becomes less than its worst-case execution time.

Moreover, we extend the FAn method to solve the firmness analysis problem for synchronous SPP policies. We obtain a conservative results for the minimum possible number of deadline misses in any k consecutive jobs of a task. These results are used as a sufficient condition to guarantee the satisfaction of (m, k)-firmness conditions.

The scalability of FAn for SPP policies is compared with that of three existing approaches: 1) a brute force approach which verifies all the possible alignments between the requested and available computing resources in a window of k consecutive jobs inspired by [14]; 2) a MILP-based method inspired by [69]; 3) a timed-automata model of the problem inspired by [10]. The FAn method scales substantially better to the problem instances with a large k and a high number of tasks than the timed-automata, brute force and the MILP approaches.

5.2 Setup under consideration

Let $\Pi = \{\tau_i\}_{1 \leq i \leq M}$ be a set of M periodic tasks sharing a processor. As in earlier chapters, each task $\tau_i = (C_i, T_i, D_i)$ is specified by its constant execution time C_i, activation period T_i and execution deadline D_i. We discuss in the successive sections that deviation in the execution time of a task has monotonic effect on our results and we may choose the worst-case execution time in our method. We assume that the deadline of all tasks are less than or equal to their activation period, i.e.,$D_i \leq T_i$. Each instance of a task is referred to as a *job*. The activation time of each job is referred to as the *release time* of the job. The earliest time t when a job of τ_i has been executed for C_i time unit is referred to as the *response time* of the job. The accessibility of τ_i to a processor at time t is modeled with *Accessibility Function*.

Definition 14 (Accessibility Function)**.** *The Accessibility Function* $l_i : \mathbb{R} \rightarrow \{0, 1\}$ *takes* 1 *if the processor is accessible to* τ_i *at* t *and* 0 *otherwise.*

Execution completion of a job of τ_i released at t before its deadline depends on the overall access time of the task to the processor from t to $t + D_i$ which is captured by *Total Accessibility Function*.

Definition 15 (Total Accessibility Function)**.** *The Total Accessibility Function* $g_i : \mathbb{R} \rightarrow [0, D_i]$ *indicates the overall access of* τ_i *to the processor between* t *and* $t + D_i$*.* $g_i(t)$ *is obtained as follows*

$$g_i(t) = \int_t^{t+D_i} l_i(x)dx. \tag{5.1}$$

Definition 16 (Deadline Hit job (DH)). *A job of τ_i released at t is a DH if $g_i(t) \geq C_i$.*

A job of τ_i that is not a DH is referred to as a DM. We assume that a job which has missed its deadline is no longer executed. As explained in Chapter 4, once a task has a release at t there is no demanded (unfinished) workload from previous jobs of the task. This execution strategy is particularly applicable to feedback control applications where execution of outdated samples can cause instability. Moreover, this helps us to leverage processor by executing less. Def. 16 implies that we can detect whether a job of τ_i released at t is a DH immediately after its release. This can be used by a run-time scheduler to detect a DM immediately after its release and do not execute that job at all. This allows to have even a tighter resource dimensioning.

Definition 17 (Hit-Zone Function). *Hit-Zone Function $h_i : \mathbb{R} \to \{0, 1\}$ takes 1 if a job of τ_i released at t is a DH and 0 otherwise. Formally,*

$$h_i(t) = \begin{cases} 1 & \text{if} \quad C_i \leq g_i(t) \\ 0 & \text{if} \quad C_i > g_i(t) \end{cases} \tag{5.2}$$

Any time interval $t \in [t1, t2)$ with $h_i(t) = 1$ is referred to as a *hit-zone*.

In this chapter we consider a set of hard and (m, k)-firm tasks that are running under an *SPP scheduling policy*.

Definition 18 (SPP Schedule). *An SPP schedule for M periodic tasks is defined as the set $SPP = \{(\tau_i, P_i)\}_{1 \leq i \leq M}$ of M tuples. Each tuple (τ_i, P_i) assigns the priority P_i to the task τ_i. A job of τ_i released at t' is executed at any time t (where $t' \leq t < t' + D_i$) if: A) $h_i(t) = 1$; B) τ_i has been executed no more than C_i time unit before t; C) there is no demanded workload from a task with a higher priority than that of τ_i.*

We assume that the task under firmness analysis has the lowest priority since the tasks with lower priorities do not cause interference to its execution.

5.3 FAn method for asynchronous tasks

5.3.1 Problem definition

This section proposes the FAn method for the firmness analysis of an (m, k)-firmness task $\tau_i = (C_i, T_i, D_i, O_i)$ in a set of asynchronous tasks $\Pi = \{\tau_i\}_{1 \leq i \leq M}$

running under an SPP policy. Note that each asynchronous task is specified with one more parameter O_i compared to the task model introduced in Section 5.2. O_i indicates the first release time of the task τ_i relative to the first release time of the highest priority task. Adding a given (m,k)-firm task τ_1 to a set of asynchronous tasks running under an SPP scheduling policy involves deciding relative first release time of τ_1. The first release time of τ_1 influences the number of DHs as it affects the interference of the tasks with the higher priorities than τ_1. In this section, we obtain the first release time O_1 which results in the maximum number of DHs in any k consecutive jobs of τ_1.

5.3.2 Hit-zone calculation

The relative release times of all the tasks with the higher priorities than τ_1 are specified by the parameters O_i and are given. Therefore, we can obtain their interference on a job of τ_1 released at t. This interference is periodic with the period of the hyper-period of all the task with the higher priorities than τ_1 which is specified with H_{1^+}. Formally, $H_{1^+} = lcm\{T_j | \tau_j \in hp(\tau_1)\}$ where $hp(\tau_1)$ is the set of all the tasks with the higher priorities than τ_1 and lcm is the least common multiple operator. We justify the existence of the lcm above based on the fact that activation period of every task is an integer multiple of the system clock cycle. Therefore Accessibility Function (see Section 5.2) can be obtained as follows

$$l_1(t) = \begin{cases} 0 & \text{if } \exists \tau_j \in hp(\tau_1); \exists n \in [1, M_{j-hyp}],\ \ B_j^n \leq t < R_j^n \\ 1 & \text{otherwise} \end{cases} \tag{5.3}$$

where B_j^n and R_j^n are the release time and the response time of the n^{th} job of the task $\tau_j \in hp(\tau_1)$. Here $M_{j-hyp} = H_{1^+} \,/\, T_j$ is the number of releases of τ_j within $[0, H_{1^+})$. The release time of each job is obtained from $B_j^n = nT_j + O_j$. [14] proposes a method to obtain R_j^n for an individual job. While efficient for individual jobs, this method is not efficient when we obtain R_j^n for all the jobs of $hp(\tau_1)$ within $[0, H_{1^+})$ since a part of calculations is repeated for every job in a set of consecutive activations. We propose an iterative method instead where calculation of R_j^n and $l_1(t)$ are interdependent as explained in the following.

We initially assume that $l_1(t) = 1$ for $t \in [0, H_{1^+})$. Starting from the first job of the highest priority task released at $t = 0$, in each iteration we first obtain the response time of a job. Second, we update $l_1(t)$ by considering $l_1(t)|_{B_j^n \leq t < R_j^n} = 0$. Third, we use the updated $l_1(t)$ to obtain the response

time of the next job of the same task in one hyper-period H_{1^+}. This process is elaborated as follows.

Let τ_{hst} be the highest priority task. Without loss of generality, we assume that the first release time of t_{hst} is 0, i.e.,$B^1_{hst} = 0$. $R^1_{hst} = C_{hst}$ as τ_{hst} is not preempted by any other task. Therefore, $l_1(t)|_{0 \leq t < C_{hst}} = 0$. R^n_j for the second job of τ_{hst} and all jobs of $hp(\tau_1)$ in the first hyper-period H_{1^+} equals to the smallest $t > 0$ satisfying the following equation [14]

$$t = nC_j + Idle_j(B^n_j) + \sum_{\tau_{jj} \in hp(\tau_j)} \left\lceil \frac{t - O_{jj}}{T_{jj}} \right\rceil C_{jj}. \tag{5.4}$$

In Eq. 5.4, nC_j is the execution time of the n jobs of τ_j. $Idle_j(B^n_j)$ is the overall time when the resource is not used to execute any job of τ_j or $hp(\tau_j)$ in $[0, B^n_j]$ which is obtained as follows.

$$Idle_j(B^n_j) = \int_0^{B^n_j} 1 - l_1(t)dt \tag{5.5}$$

The summation in Eq. 5.4 captures the overall interference of the jobs of the tasks with a higher priority than τ_j within $[0, t]$. Eq. 5.4 can be solved using the fixed point iteration method [19] as follows.

$$\begin{cases} t^{(0)} = nC_j + Idle_j(B^n_j) \\ t^{(\gamma+1)} = nC_j + Idle_j(B^n_j) + \sum\limits_{\tau_{jj} \in hp(\tau_j)} \left\lceil \dfrac{t^{(\gamma)} - O_{jj}}{T_{jj}} \right\rceil C_{jj} \end{cases} \tag{5.6}$$

where $\gamma \in \mathbb{Z}^+$. We calculate a new value of t^γ in each iteration by incrementally increasing γ until $t^{\gamma+1} = t^\gamma$ is satisfied. Alg. 1 shows a general fixed point iteration method to solve recursive equations that are in the form of $t = Con. + f(t)$ where $Con.$ is a constant value. To solve Eq. 5.7 with Alg. 1 $Con.$ and $f(t)$ are substituted as follows.

$$Con. = nC_j + Idle_j(B^n_j), f(t) = \sum_{\tau_{jj} \in hp(\tau_j)} \left\lceil \frac{t - O_{jj}}{T_{jj}} \right\rceil C_{jj}.$$

Once we obtained R^n_j, we update $l_1(t)$ and continue with obtaining the response time of R^{n+1}_j. After obtaining R^n_j for all jobs of τ_j in one hyper-period H_{1^+} and updating $l_1(t)$ correspondingly, we obtain the response time of the first job of the next highest priority task in $hp(\tau_1)$. We continue this process until we obtain the response time of all the jobs of the tasks with the

Algorithm 1 Iterative fixed point solution for $t = Con + f(t)$

Input $f(t)$, Con ▷ Con is the constant term of a recursive formula
Output t ▷

1: $t^{(0)} = Con$
2: $t^{(-1)} = Con + 1$ ▷ needed to initiate the while loop
3: $\gamma = 0$
4: **while** $t^{(\gamma)} \neq t^{(\gamma-1)}$ **do**
5: $\gamma++$
6: $t^{(\gamma)} = Con + f(t^{(\gamma-1)})$
7: **end while**
8: $t = t^{(\gamma)}$

higher priorities than τ_1 in one hyper period and update $l_1(t)$ for all of these jobs. The final result of $l_1(t)$ is used as Accessibility Function of τ_1.

Let us consider the case where there is an (m,k)-firm task τ_j with the higher priorities than τ_1 in the set Π. Recall that DMs are never executed. We need to take this assumption into account in the process of obtaining the response time of $hp(\tau_1)$ and updating $l_1(t)$ that mentioned previously in this section. In this regard, for DMs of τ_j we assume that $B_j^n = R_j^n$.

Example 5.3.1. *We intend to add* $\tau_1 = (1.5, 13, 11, O_1)$ *to a set of asynchronous tasks* $\Pi = \{\tau_2, \tau_3\}$ *where* $\tau_2 = (3, 8, 7, 4)(\times 100\mu s)$ *and* $\tau_3 = (2, 6, 3, 0)$ $(\times 100\mu s)$ *are running under an SPP policy with priorities* $P_3 > P_2 > P_1$. *We intend to obtain an offset* O_1 *where* τ_1 *has the maximum possible number of DHs in any* k *consecutive jobs. Fig. 5.1(a) depicts the execution of* τ_2 *and* τ_3 *in one hyper period of* $H_{1^+} = 24$. $Idle_2(B_2^n)$ *and* $Idle_3(B_3^n)$ *are denoted by* Id_2 *and* Id_3 *on the figure, respectively. Fig. 5.1(b) shows Accessibility Function* $l_1(t)$.

Once Accessibility Function is obtained, the Total Accessibility Function and Hit-Zone Function can be obtained using Eq. 5.1 and Eq. 5.2. Fig. 5.2(a) and Fig. 5.2(b) show $g_1(t)$ and $h_1(t)$ for Example 5.3.1, respectively.

5.3.3 Computing the maximum DHs

For an (m,k)-firm task that is intended to be added to a set of asynchronous tasks, we need to obtain the maximum number of DHs and the corresponding first release time. Having the maximum number of DHs bigger than or equal to m satisfies the (m,k)-firmness condition. In order to obtain the maximum

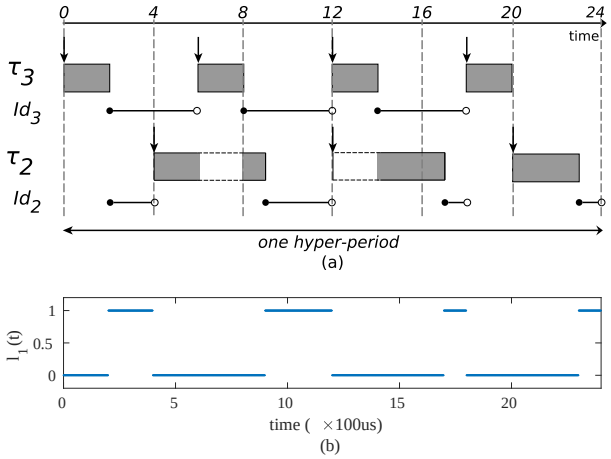

Figure 5.1: Calculation of $l_1(t)$ in Example 5.3.1. (a) Id_2 and Id_3 show the intervals where $Idle_2 = 1$ and $Idle_3 = 1$, respectively. (b) $l_1(t)$ in one hyper-period.

number of DHs in any k consecutive jobs of τ_1 we must obtain the maximum number of jobs which are released in hit-zones in any k consecutive jobs. This problem can be solved using the FP method (see Chapter 3) and substituting the parameters of this problem corresponding to those of Balloon and Rake problem as listed in Table 5.1. In this regard, we consider a window of k consecutive jobs of τ_1 as a rake with k blades. Therefore, the distance between every two consecutive blades equals to T_1. Moreover, we consider each hit zone as a balloon. Therefore, the Hit-Zone Function h_1 represents the balloon function f. Fig. 5.2(c) shows the number of DHs in 10 consecutive jobs of τ_1 against the first release times of τ_1 in two hyper-period $2H_{1^+}$. The dots on Fig. 5.2(c) show the candidate offsets for the maximum number of DHs in 10 consecutive jobs of τ_1 and the corresponding number of DHs. The fact that there is at least one dot at any time where there is an increase in the number of DHs, validates the results of FAn method.

Recall that we consider constant execution times of all the tasks in a task set. Based on Proposition 2, considering the worst-case execution time for all tasks under an SPP policy results in the lower bound for the number of DHs for a given first release time of τ_1.

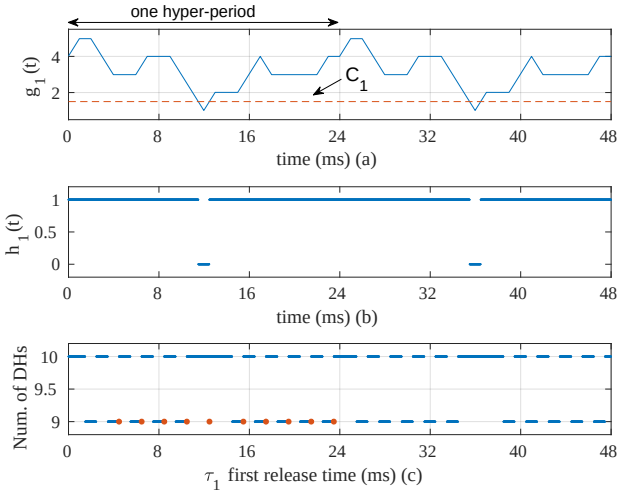

Figure 5.2: Results of FAn method applied on Example 5.3.1. $g_1(t)$ (a), $h_1(t)$ (b), and the number of DHs (c) in two hyper-period of $H_{1^+} = 24$.

Proposition 2. *Consider two applications* $\tau_1 = (C_1, T_1, D_1)$ *and* $\tau_2 = (C_2, T_2, D_2)$ *running under SPP policy with priorities* $P_2 > P_1$. *When the execution times* C_1' *and* C_2' *of the two tasks* τ_1 *and* τ_2 *are smaller than their worst-case execution times, i.e.,*$C_1' \leq C_1$ *and* $C_2' \leq C_2$, *the resulting Hit-Zone Function* $h_1'(t)$ *satisfies* $h_1'(t) \geq h_1(t)$ *for any* $t \geq 0$.

Proof. Let us first assume that only the execution time of the task with higher priority is smaller than its worst case execution time, i.e., $C_1' = C_1$ and ${C_2}' < C_2$. The corresponding value of Accessibility Function, Total Accessibility Function and Hit-Zone Function of τ_1 considering C_2' is denoted by $l_1'(t)$, $g_1'(t)$ and $h_1'(t)$ and considering C_2 is denoted by $l_1(t)$, $g_1(t)$ and $h_1(t)$. From Eq. 5.3 and Eq. 5.1

$$\forall t \geq 0, l_1'(t) \geq l_1(t), g_1'(t) \geq g_1(t).$$

Therefore,

$$\forall C_2' < C_2, h_1'(t) \geq h_1(t).$$

Next, we assume that $C_1' < C_1$ and ${C_2}' \leq C_2$. From Eq. 5.2, we obtain $h_1''(t) \geq h_1(t)$ where $h_1''(t)$ is the corresponding Hit-Zone Function for C_1'. □

Table 5.1: Balloon and Rake problem parameters corresponding to FAn method for multiple asynchronous tasks

type	Balloon and Rake	FAn: asynchronous tasks
input	N and p in the form of f	h_1
	# of rake blades, r	k
	distance between blades, d	T_1
output	max # of struck balloons	max # of DHs
	offset of the rake	first release time of τ_1

Proposition 2 implies that variation in the execution time of any task in a given set of asynchronous tasks has monotonic effect in the number of DHs. Considering the worst-case execution time for all the tasks therefore results in the minimum-maximum number of DHs. That is, the number of DHs may increase with a lower execution time of any task with the same or the higher priority than τ_1. Thus, the obtained results from the FAn method is the lower bound for the maximum number of DHs.

5.4 FAn method for synchronous tasks

This section uses the solution of the balloons and rake problem to address the firmness analysis of multiple synchronous tasks running under an SPP policy.

5.4.1 Problem definition

This section proposes the FAn method for a set $\Pi = \{\tau_i\}_{1 \leq i \leq M}$ of multiple *synchronous* tasks $\tau_i = (C_i, T_i, D_i)$ running under an SPP schedule (see Def. 18). Note that, in contrast to asynchronous tasks, relative first release times are not given for synchronous tasks. Let $\tau_1 \in \Pi$ be the lowest priority task which is under firmness analysis. The highest priority task is denoted by τ_{hst}. The set of all the other tasks with priorities higher than τ_1 and lower than τ_{hst} is denoted by Π_{mdl}. The tasks with the higher priorities than τ_1 may or may not have hard deadlines. Formally,

$$\Pi = \{\tau_{hst}, \tau_1\} \cup \Pi_{mdl}, \text{ such that } \{\tau_{hst}, \tau_1\} \cap \Pi_{mdl} = \phi.$$

Different relative release time of the tasks with the higher priorities than τ_1 leads to different interferences on the execution of τ_1. Therefore, to guarantee

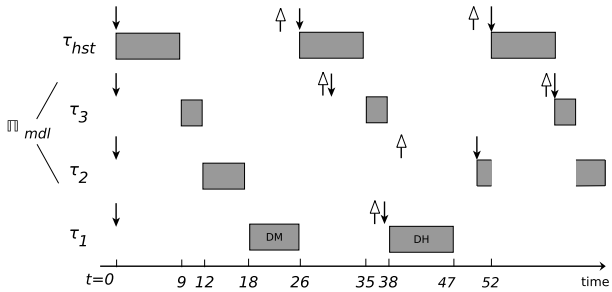

Figure 5.3: The critical instant for the tasks in Exampe 5.4.1. Release times of the tasks and their deadlines are shown with downward and upward arrows, respectively.

the satisfaction of the firmness conditions, we need to identify the worst-case release time that leads to the minimum number of DHs in a window of k consecutive jobs.

The worst-case first release time – also known as *critical instant* [23] – for a single job of the task τ_1 in a set of synchronous tasks occurs when τ_1 and all the tasks with the higher priorities than τ_1 have simultaneous releases. This job then experiences the Worst Case Response Time (WCRT) explained in the next subsection. The results of the WCRT analysis would be an initial step to detect whether it is possible for τ_1 to have a DM. Critical instant, however, has no correlation with the minimum number of DHs in k consecutive jobs of τ_1.

Example 5.4.1. *We take* 4 *synchronous tasks* $\tau_1 = (9, 38, 37), \tau_2 = (6, 50, 40)$, $\tau_3 = (3, 31, 30)$ *and* $\tau_{hst} = (9, 26, 23)$ *where* $P_{hst} > P_3 > P_2 > P_1$ *(all times are in ms).* τ_1 *is an* $(7, 10)$*-firm task. In this example,* $\Pi_{mdl} = \{\tau_2, \tau_3\}$*. We intend to obtain the minimum number of DHs in any* 10 *consecutive releases of* τ_1*. Without loss of generality, we assume that* τ_{hst} *has a release at* $t = 0$*. Fig. 5.3 depicts the critical instances for all the tasks.* τ_1 *might experience DMs, as shown on the figure.*

In the rest of this section, we first briefly explain the schedulability analysis of a task in a set of synchronous tasks. Next, we obtain the sufficient conditions for having a DH job without having to consider the relative release time of all the tasks with the higher priorities than τ_1. We then use this method to

identify a set of periodic intervals in which a release of τ_1 is definitely a DH. Finally, we formulate the problem in the form of a balloons and rake problem and obtain a lower bound for the minimum number of DHs in any k consecutive jobs of τ_1.

5.4.2 Schedulability analysis of synchronous tasks

Using WCRT analysis for synchronous tasks, regardless of the number of tasks, the worst possible release time of the tasks – also known as *critical instant* [23] – is considered for schedulability analysis of synchronous tasks. The critical instant for a synchronous task τ_i occurs when τ_i and all the tasks with the higher priorities release a job at the same time. The basic idea behind WCRT analysis is that if the job of a task that is released at the critical instant meets its deadline, all other jobs of the task meet their deadlines.

Let us assume a critical instant for the task τ_i with a release of τ_i and all the tasks with the higher priorities at $t = 0$. The WCRT of τ_i, which is denoted by $WCRT_i$, is the minimum $t > 0$ that satisfies the equation below [23].

$$t \geq C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j, \tag{5.7}$$

where $hp(\tau_i)$ is the set of all the tasks with the higher priorities than τ_i. Note that this solution considers no context switch overhead of the resource. A closed-form solution of Eq. 5.7 does not exist because of the nonlinear nature of the ceiling operator inside the summation. As an alternative, the fixed point iteration method [19] is used to solve Eq. 5.7 as follows.

$$\begin{cases} t^{(0)} = C_i \\ t^{(\gamma+1)} = C_i + \sum\limits_{\tau_j \in hp(\tau_i)} \left\lceil \dfrac{t^{(\gamma)}}{T_j} \right\rceil C_j \end{cases} \tag{5.8}$$

where $\gamma \in \{0, 1, 2, ...\}$. These iterations have to be continued by incrementally increasing the value of γ until the value of $t^{(\gamma)}$ does not change anymore, i.e.,$WCRT_i = t^{(\gamma+1)} = t^{(\gamma)}$. That is, the last result is the actual $WCRT_i$. Note that Eq. 5.7 converges only if the overall resource utilization is at most 1 [13]. To solve Eq. 5.7 with Alg. 1 $Con.$ and $f(t)$ are substituted as follows.

$$Con. = C_i, f(t) = \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$

Example 5.4.2. *Fig. 5.4 illustrates WCRT computation of synchronous tasks. Fig. 5.4(a) shows the critical instance for three synchronous tasks τ_1, τ_2 and τ_3 with the priorities $P_3 > P_2 > P_1$ in an SPP schedule. We assume that all the tasks release their first job at $t = 0$. Fig. 5.4(b) shows WCRT computation of τ_1 using the fixed point iteration method. The figure depicts t of Eq. 5.7 on the x-axis and $Con + f(t)$ on the y-axis. Note that the horizontal axes in both Fig. 5.4(a) and Fig. 5.4(b) are time with the same scale for both figures.*

$WCRT_1$ is obtained after four iterations of Eq. 5.8, as shown in Fig. 5.4(b). The first iteration $t^{(0)} = C_1$ results in the response time of τ_1 without any interference from the higher priority tasks. In the second iteration, $t^{(1)} = C_1 + C_2 + C_3$ determines the response time of τ_1 considering the maximum interference during an interval of the previous response time estimate ($C1$). If there is no additional workload demanded before $t = t^{(1)}$, $t^{(1)}$ is the response time of τ_1. However, since before $t = t^{(1)}$, τ_3 releases another job, the response time of τ_1 is postponed to $t = t^{(2)} = C_1+C_2+2C_3$ that is captured with the third iteration of Eq. 5.8. This process continues until $t = t^{(3)} = C_1 + 2C_2 + 2C_3$ predicts the actual response time of τ_1. We compute this as $t^{(3)} = t^{(4)} = t^{(\gamma)} = C_1 + 2C_2 + 2C_3$ for all $\gamma > 4$.

If $WCRT_i > D_i$, τ_i is therefore not schedulable under the SPP policy. For the task τ_i that is not schedulable under SPP, a release at the critical instant is a sufficient but not a necessary condition for a DM. That is, the critical instance is not necessarily the only instance that a job of τ_i misses its deadline. In the next subsection, we discuss the necessary conditions for the release time of τ_i relative to those of the higher priority tasks such that τ_i meets or misses its deadline.

Besides, the classical schedulability analysis obtains the possibility of missing the deadline for *one* job of a task. This however does not necessarily have any correlation with the maximum number of DMs/DHs in a window of k consecutive jobs of the task. In the rest of this section we obtain the minimum possible number of DHs in any k consecutive jobs of a task.

5.4.3 Interference model

In order to obtain whether a released job of τ_1 is a DH, we need to obtain the response time of the job and compare it with its deadline D_1. Therefore, we need to obtain the interference of τ_{hst} and all the tasks in Π_{mdl} on that particular job of τ_1. In this section, we propose a model for the interference of all the tasks on τ_1. This model considers the exact interference of τ_{hst} and an upper bound interference of the tasks in Π_{mdl} on τ_1. This allows us to

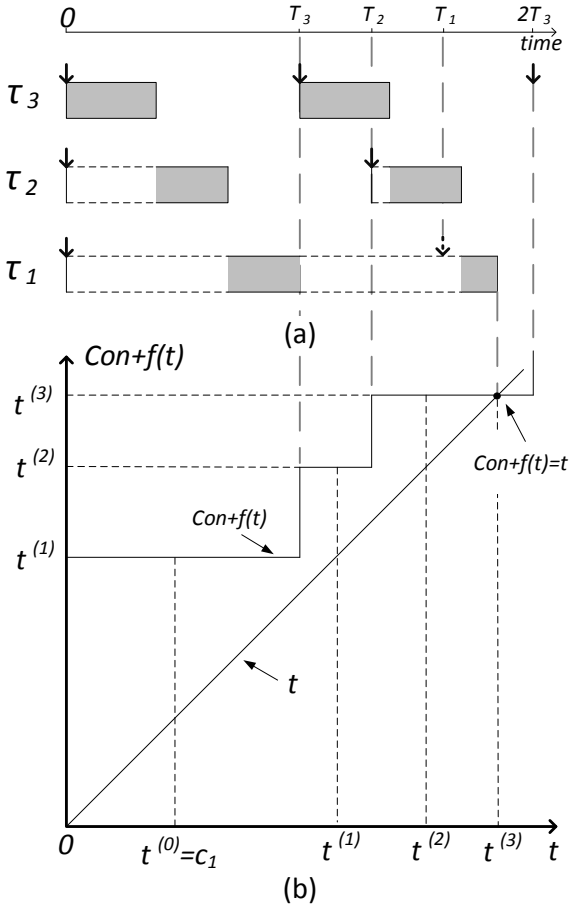

Figure 5.4: WCRT computation of τ_1 in Ex. 5.4.2. (a) The critical instant for three synchronous tasks τ_1, τ_2 and τ_3 with priorities $P_3 > P_2 > P_1$. (b) WCRT computation of τ_1 using the fixed point iteration method. The vertical and horizontal axes are the right and left hand sides of Eq. 5.7, respectively.

consider only the first release time of τ_1 relative to that of τ_{hst}.

For a job of τ_1 released at t, we consider a time window of the length D_1 –

starting from t – when the job can be executed before its deadline. Therefore, the job is a DH if

$$D_1 \geq C_1 + intf_{hst}(D_1, t) + \sum_{\tau_i \in \Pi_{mdl}} intf_{ub-\tau_i}(D_1), \tag{5.9}$$

where $intf_{hst}(D_1, t)$ is the overall interference of τ_{hst} within $[t, D_1)$. $intf_{ub-\tau_i}(D_1)$ is an upper bound of the interference of $\tau_i \in \Pi_{mdl}$ in any window of the length D_1 which is independent of t. Although it is a pessimistic assumption, considering a constant value for the interference of all $\tau_i \in \Pi_{mdl}$ reduces the complexity of the problem as we ignore the relative release time between each of these tasks and τ_1. Next, we obtain $intf_{hst}(D_1, t)$ and $intf_{ub-\tau_i}(D_1)$, respectively.

$intf_{hst}(D_1, t)$ is obtained as follows

$$intf_{hst}(D_1, t) = \int_{t}^{t+D_1} \beta(x)dx \tag{5.10}$$

where the function $\beta : \mathbb{R}^+ \to \{1, 0\}$ models the execution time of τ_{hst} by taking 1 at any time t when τ_{hst} is executed and 0 otherwise. Since τ_{hst} has the highest priority in Π, it is not preempted by any other task. Therefore, $\beta(t)$ is periodic with the same period as τ_{hst}, i.e.,T_{hst}. $\beta(t)$ for one period of τ_{hst} starting from 0, i.e.,$0 \leq t < T_{hst}$, is calculated as follows

$$\beta(t) = \begin{cases} 1 & \text{if} \quad 0 \leq t < C_{hst} \\ 0 & \text{if} \quad C_{hst} \leq t < T_{hst} \end{cases} \tag{5.11}$$

From Eq. 5.10 and the fact that $\beta(t)$ is periodic, it is concluded that $intf_{hst}(\delta, t)$ is also periodic with the same period, i.e.,T_{hst}. Fig. 5.7(a) and Fig. 5.7(b) depicts $\beta(t)$ and $intf_{hst}(D_1, t)$ for two periods of τ_{hst} in Example 5.4.1.

Next, we obtain $intf_{ub-\tau_i}(D_1)$. We first introduce the notion of *upper bound interference* of a task τ_i on a task with a lower priority than that of τ_i.

Definition 19 (Upper bound Interference (UI))**.** *Let τ_i be a periodic task in a set of synchronous tasks running under an SPP policy. UI function $intf_{ub-\tau_i} : \mathbb{R}^+ \to \mathbb{R}^+$ specifies an upper bound of the overall time that τ_i is executed in a time interval of the length δ.*

Considering $\tau_i \in \Pi_{mdl}$, $intf_{ub-\tau_i}(D_1)$ is then equivalent to an upper bound of the interference of τ_i on any job of τ_1. We need the results of the following proposition to calculate $intf_{ub-\tau_i}(\delta)$.

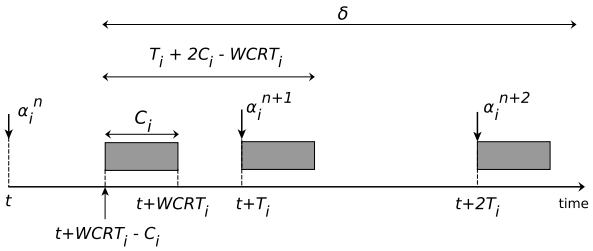

Figure 5.5: An execution scenarion for τ_i which results in an upper bound for the execution of the task in an interval with the length δ.

Proposition 3. *Let τ_i be a periodic task in a synchronous task set under an SPP policy. An upper bound for overall execution of τ_i in an interval of the length δ is obtained by considering the beginning of the interval at $t+WCRT_i-C_i$ when a job of τ_i – released at t – is executed between $t+WCRT_i-C_i$ and $t+WCRT_i$ and all the successive jobs are executed immediately after their releases (see Fig. 5.5).*

Proof. Let α_i^n denotes the n^{th} job of τ_i released at t (see Fig. 5.5). $t+WCRT_i$ is then the latest possible response time of α_i^n where $WCRT_i$ denotes the worst-case response time of τ_i. Therefore, the minimum possible time difference between the response of one job and the execution commencement of the next job is $T_i - WCRT_i$. This is the scenario where the next job, i.e.,α_i^{n+1}, commences its execution immediately after its release, i.e.,at $t+T_i$. Therefore, the minimum interval in which $2C_i$ execution of τ_i occurs equals to $T_i + 2C_i - WCRT_i$ which is the interval starting from $t+WCRT_i-C_i$ as shown in Fig. 5.5. Then, $intf_{ub-\tau_i}(T_i + 2C_i - WCRT_i) = 2C_i$.

An upper bound for the closest executions to these two executions are the successive releases after α_i^{n+1} if they are completely executed immediately after their releases (e.g. the execution of α_i^{n+2} in Fig. 5.5). Therefore, an upper bound for the execution of τ_i in an interval of length δ is obtained by considering the scenario when a job of τ_i is executed between $t+WCRT_i-C_i$ and $t+WCRT_i$ while the successive jobs are executed without preemption immediately after their releases. In such a scenario, the interval of the length δ starting from $t+WCRT_i-C_i$ contains an upper bound execution of τ_i. □

$intf_{ub-\tau_i}(\delta)$ is obtained as follows

$$intf_{ub-\tau_i}(\delta) = min\{\delta, C_i\} + max\left\{0\ ,\ \left\lfloor \frac{\gamma}{T_i} \right\rfloor C_i + min\{mod(\gamma, T_i), C_i\}\right\} \tag{5.12}$$

where

$$\gamma = \delta + WCRT_i - T_i - C_i.$$

Eq. 5.12 is explained as follows. From Proposition 3, we consider the case where a job of τ_i released at t is executed between $t + WCRT_i - C_i$ and $t + WCRT_i$ (see Fig. 5.5). The successive jobs of τ_i are executed immediately after their releases. In Eq. 5.12, the first term, i.e.,$min\{\delta, C_i\}$, captures the execution of τ_i within $[t+WCRT_i-C_i, t+T_i)$. $\delta < T_i-(WCRT_i-C_i)$ – which is equivalent to $\gamma < 0$ – is corresponding to the case where the interval with the length δ starting from $t + WCRT_i - C_i$ ends before the release of the next job which occurs at $t + T_i$. In such a case, $intf_{ub-\tau_i}(\delta) = C_i$; unless $\delta < C_i$ which results in $intf_{ub-\tau_i}(\delta) = \delta$. The second term in Eq. 5.12, with the max operator, results in a non-zero value if $\delta \geq T_i - (WCRT_i - C_i)$ (i.e.,$\gamma \geq 0$) which is the case when there are more than one execution of τ_i in the interval of length δ. The executions of all the successive jobs are captured with this term. $\lfloor \gamma/T_i \rfloor C_i$ captures the execution of all the jobs released in the interval except the first and the last jobs. Finally, the execution of the last job that is released within the time interval δ is captured with $min\{mod(\gamma, T_i), C_i\}$ in Eq. 5.12. Note that for any $\tau_i \in \Pi_{mdl}$ the minimum value of $intf_{ub-\tau_i}(D_1)$ equals $min\{D_1, C_i\}$. That is, a job of τ_1 is preempted by at least one complete execution of $\tau_i \in \Pi_{mdl}$ in the worst-case interference, unless $D_1 < C_i$ (see Eq. 5.12).

In Example 5.4.1, $WCRT_3 = 12ms$ and $WCRT_2 = 18ms$ which are obtained from Eq. 5.7. Let both τ_2 and τ_3 have releases at $t = 0$. Considering an interval with the length $D_1 = 37$ starting from $t + WCRT_3 - C_3 = 9$ and $t + WCRT_2 - C_2 = 12$, two executions of τ_3 and one execution of τ_2 occur at most in the interval as shown in Fig. 5.6. Therefore,

$$\sum_{\tau_i \in \Pi_{mdl}} intf_{ub-\tau_i}(D_1) = intf_{ub-\tau_2}(D_1) + intf_{ub-\tau_3}(D_1) = C_2 + 2C_3 = 12.$$

Fig. 5.7(c) depicts the two sides of Eq. 5.9 for Example 5.4.1. The straight line in the figure shows D_1.

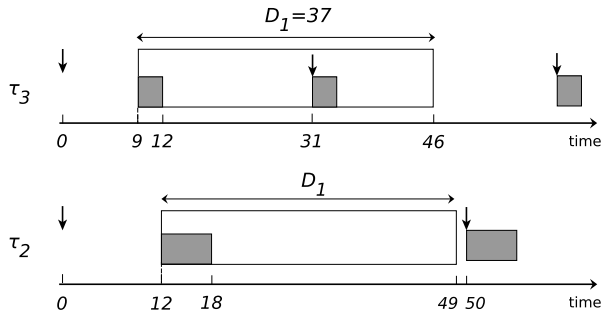

Figure 5.6: Calculation of the upper bound interference of τ_2 and τ_3 in any interval of the length D_1.

5.4.4 Common Hit-Zone calculation

From Section 5.2, if a job of τ_1 released at t meets its deadline, t is in a hit-zone. Besides, Eq. 5.9 provides sufficient condition to decide whether this job is a DH. That is, if the job is not a DH based on Eq 5.9, it is not necessarily a DM. This case then depends on the releative release time of the tasks in Π_{mdl}. Therefore, using Eq. 5.9 we obtain a subset of hit zone intervals which we refer to as *common hit-zones*, since they are common among all the hit-zones obtained by considering different release time of the tasks in Π_{mdl}.

Definition 20. *(Common Hit-Zone (CHZ) function) Let τ_1 be in the set Π of multiple synchronous tasks. Common Hit-Zone Function $h'_1(t) : \mathbb{R} \rightarrow \{0, 1\}$ specifies common hit zones by taking $h'_1(t) = 1$ for any time t when Eq. 5.9 is satisfied.*

Fig. 5.7(d) shows $h'_1(t)$ for Example 5.4.1 that is obtained by comparing the two sides of Eq. 5.9 that is shown in Fig. 5.7(c).

5.4.5 Computing the minimum DHs

Obtaining the minimum number of jobs in any k consecutive jobs of τ_1 that can be released in CHZ, i.e.,when $h'_1 = 1$, provides a conservative bound for the minimum number of DHs. Having the minimum number of DHs more than or equal to m (in (m, k)-firmness conditions), guarantees the satisfaction of (m, k)-firmness conditions. From the fact that $Intf_{hst}(D_1, t)$ – the only

Table 5.2: The parameters of the balloons and rake problem corresponding to those of FAn for synchronous tasks

type	balloons and rake	FAn: synchronous tasks
input	N and p in the form of f	h'_1
	# of rake blades, r	k
	distance between blades, d	T_1
output	min # of struck balloons	min # of DHs

time-dependent term in Eq. 5.9 – is periodic with the period T_{hst}, we conclude that h'_1 is also periodic with the same period. Then, we refer to the balloons and rake problem described in Chapter 3. That is, we consider a window of k consecutive jobs of τ_1 as a rake with k blades. There is a distance T_1 between every two consecutive blades of the rake. We take h'_1 as the balloon function f. Then, the result of the balloons and rake problem for the minimum number of struck balloons with the parameters above is the minimum number of τ_1 released in CHZs. Table 5.2 lists the mapping of parameters between the analysis presented in this section and the balloons and rake problem.

Fig. 5.7.e shows the number of DHs in a window of 10 consecutive jobs of τ_1 within two periods of τ_{hst} execution, i.e.,$[0, 52)$ in Example 5.4.1. The stars on the figure show the candidate offsets of the window to obtain the minimum number of jobs released at CHZ. The fact that there is at least one star in any point where there is a decrease in the number of DHs validates the FAn method.

5.5 Experimental evaluation

In this section, we evaluate the scalability and accuracy of the FAn method. We first explain three existing approaches: (i) a Brute-Force (BF) search approach on all possible relative alignments between tasks inspired by [14], (ii) a method that formulates the firmness analysis problem as a MILP instance, inspired by [69] and, (iii) a reachability analysis based on the Timed-Automata (TA) model of the problem realized in the UPPAAL tool inspired by [12]. Second, we provide hypotheses on the complexity of the FAn method and the existing methods. Third, we introduce our experimental setups used for evaluation purposes. Finally, we discuss the accuracy and the scalability of all the methods based on our experiments.

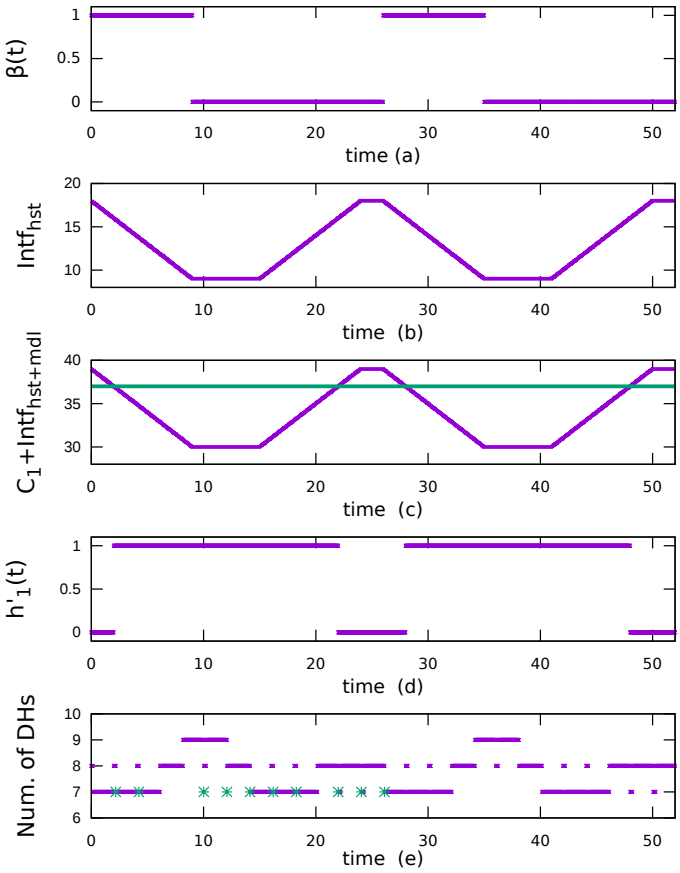

Figure 5.7: The solution of Example 5.4.1 using the FAn method. (a) $\beta(t)$ obtained from Eq. 5.11. (b) Interference of τ_{hst} on τ_1. (c) Both sides of Eq. 5.9 for Example 5.4.1. The straight line indicates D_1. (d) CHZ function. (e) Number of DHs versus the offset of the windows of 10 consecutive jobs of τ_1. Stars show the candidate offsets for the minimum number of DHs.

5.5.1 Brute-Force search approach

Bernat et al. [14] proposes a firmness analysis method for a given alignment between all the tasks in a task set. For a set of tasks running on a shared

processor, all the timing parameters, e.g. job releases, execution commencement, execution time, task period, etc., are integer multiples of a processor clock cycle. This allows us to obtain all the possible alignments among all tasks, in an SPP policy. Then, for each alignment, we use the method in [14] to obtain the minimum (in synchronous SPP policy) and the maximum (in the asynchronous SPP policy) number of DHs in any k consecutive jobs of τ_1. Comparing the results of this method for all the possible alignments, we verify the satisfaction of a given (m, k)-firmness condition.

5.5.2 MILP-Based method

Sun et al. [69] recently proposed a MILP-based method for the feasibility test of a given (m, k)-firm periodic task in a set of synchronous tasks running under an SPP policy. The main difference between the problem addressed in [69] and that of the FAn method is the task model under consideration. [69] assumes that a job of a task that has missed the deadline is executed completely. However, in our task model, a task is executed only if it does not miss its deadline. We adapted the MILP-based method [69] to the task model as considered in this work. To this end, we modified constraints on the response time of a task. That is the response time of a task can be at most equal to the deadline of the task. Otherwise the corresponding job of the task is considered as a DM. The MILP-based method is based on an over-approximation which results in a conservative output. However, comparison between the result of the MILP method and the BF method shows exact results in the majority of our experiments.

5.5.3 Timed-Automata model

The nature of the firmness analysis problems allows us to model the system with timed automata [3]. Our model is inspired by timed-automata model proposed in Chapter 4 which models a set of periodic tasks running under a TDMA policy. We use the Uppaal tool to verify properties of our model. Run-time of such verification highly depends on the complexity of the timed-automata model. Therefore, we compare the complexity of our timed-automata with that of the FAn method and the BF method only on the parameters that we believe exist in any timed-automata model of the system.

We modeled each periodic task with two automata. One automaton decides the release time of each job of the task and the other automaton keeps track of the overall time that a job has been executed. Another automaton decides which task has access to the processor based on their priorities. We

Table 5.3: CCCA system parameters (time in ms)

Task	$\tau_i = (C_i, T_i, D_i, O_i)$	P_i	m	k
Breaking control (hard)	$(5, 30, 28, 0)$	4	-	-
Collision avoidance (hard)	$(15, 50, 48, 19)$	3	-	-
Engine control ((m, k)-firm)	$(15, 30, 28, 12)$	2	140	170
Display control ((m, k)-firm)	$(25, 50, 48, 7)$	1	140	170

use stopwatch automata [21] which allows us to stop the clock that represents the overall execution of a task in case of a task preemption. Analysis of stopwatch automata is based on an over-approximation on the state space. Comparison between the final results of the BF and FAn methods shows no over-approximation in the number of DHs in our case-studies though.

5.5.4 Experimental setup

We took 50 different setups for each of scheduling policies, i.e.,synchronous and asynchronous SPP policies. Each setup consists of multiple periodic tasks and a scheduling policy under which the tasks are running. The parameters for our experiments are inspired by the Cruise Control with Collision Avoidance (CCCA) system of [37]. This system consists of two hard real-time tasks, i.e.,Breaking Control (BC) and Collision Avoidance (CA), and two (m, k)-firm tasks, i.e.,Engine Control (EC) and Display Control (DC). Table 5.3 shows the parameters of these tasks. We took multiple instances of one task when we required to have a combination of more than four tasks. In our experimental setups $k \in \{10, 50, 100, 170\}$ and $M \in \{4, 6, 10, 20, 50\}$. Moreover, we choose the activation period of tasks in a task set to obtain a wide range of hyperperiod of the tasks, i.e.,the hyperperiod of all tasks H and the hyperperiod of the tasks with the higher priorities than that of the task under the firmness analysis H_{1^+}. We implemented the BF method and the FAn method in the C++ programming languages. All the experiments were run on a system with a $2.6GHz$ quad core CPU and 4GB RAM.

5.5.5 Scalability analysis

Table 5.4 compares the complexity of the four methods that are considered in this work obtained from our experiments. In order to obtain the complexity of different methods, we conducted experiments by varying one of the parameters

Table 5.4: Complexity of firmness analysis methods

Met.	Asyn. SPP	Syn. SPP
FAn	$O(MH_{1^+})$	$O(MT_{hst})$
BF	$O(kH_{1^+}^2Mf)$	$O(kH^3M^2f)$
TA	$O(k^2H_{1^+}^3M^{c_2})$	$O(k^2H^{c_3}M^{c_4})$
MILP	-	$O(k^{c_5}HM^2)$

affecting the run-time of the methods while assuming constant value for the rest of the parameters in the range that is mentioned in Section. 5.5.4.

The run time of the FAn method mostly depends on the process of obtaining hit zones. This process depends on the period of the Hit-Zone Function. The period of the Hit-Zone Function equals the hyperperiod H_{1^+} in the asynchronous SPP policy and the period of the task with the highest priority T_{hst} in the synchronous SPP policy. Moreover, the run time of obtaining the Hit-Zone Function in asynchronous and synchronous tasks scales with the number of tasks M as it influences the number of times that Eq. 5.3 and Eq. 5.9 are calculated, respectively. Therefore, the FAn methods for asynchronous and synchronous SPP policies have complexity $O(MH_{1^+})$ and $O(MT_{hst})$, respectively.

The process of calculating hit zones exists in the BF method too. Moreover, in the asynchronous SPP policy, the number of all possible offsets of a window of k consecutive jobs of τ_1 scales with M and H_{1^+}. Besides, the run-time of the calculation of the Hit-Zone Function depends on H_{1^+} as the number of jobs in one hyperperiod H_{1^+} depends on the length of the hyperperiod. Thus, this approach has complexity $O(kH_{1^+}^2Mf)$ where f is the clock frequency of the processor. In the synchronous SPP policy, all possible alignments between all tasks have to be considered. Therefore, the hyperperiod of all tasks H and the number of tasks M have a bigger impact on the number of possible release times of all tasks than that of an asynchronous SPP policy. The BF method for synchronous SPP policy has complexity $O(kH^3M^2f)$.

The run-time of the timed-automata model depends on the size of the state space built during the reachibility analysis. In a timed-automata model of asynchronous and synchronous SPP policies, the size of the state space depends on the number of task switches in hyperperiod H_{1^+} and H, respectively. Our experiments show that the TA method for synchronous and asynchronous SPP policies has complexity $O(k^2H_{1^+}^3M^{c_2})$ and $O(k^2H^{c_3}M^{c_4})$, respectively, with $c_2 \geq 2$, $c_3 \geq 3$ and $c_4 \geq 2$.

In the range of parameters that we considered in our experiments, the run-time of the MILP algorithm has complexity $O(k^{c_5}HM^2)$ where $c_5 \geq 3$. Our experiments confirm the fact that the number k is a dominant factor in the complexity of the MILP-based method.

Table 5.5 shows the results of the TA, FAn and BF methods for three of our experiments out of 50 experiments running under asynchronous SPP policy. In all the experiments in Table 5.5, τ_1 is under firmness analysis. We show case-studies with different numbers of tasks M and various hyperperiod H_{1^+} to illustrate their effect on the run-time of the three methods above. Π_1 and Π_2 have the same number of tasks $M = 4$ and different hyperperiod of the tasks with the higher priorities than τ_1, i.e.,H_{1^+}. Π_1 and Π_3 have hyperperiod H_{1^+} in the same range whilst different number of tasks, i.e.,$M = 4$ for Π_1 and $M = 6$ for Π_3. Note that, comparing the run-time of Π_2 and Π_3 is not useful for any method since more than one parameter influencing the scalability of the methods, i.e.,M and H_{1^+}, differ between these two task sets. In the case of a SPP policy, the priority of the tasks in each task set is in the same order as the index of the tasks names, e.g. τ_4 and τ_1 have the highest and the lowest priority in Π_1, respectively. The table confirms that the run-time of the TA method is more affected by change in the number of tasks than that of the FAn and the BF method. Moreover, the run time of the BF and the TA method is more affected by H_{1^+} than that of the FAn method (see Table. 5.4).

Table 5.6 shows the results of firmness analysis of the same task sets as in Table 5.5 using TA, FAn, BF and MILP-based methods running under a synchronous SPP policy. Using the MILP-based method, none of the three experiments terminated within an hour limit. We later show that the MILP-based method does not scale for the cases with high value of k, e.g. $k = 170$. While the run time of the BF method in some setups is not problematic for design time analysis, the FAn method may still be preferred when the design of system parameters, e.g. priorities in SPP policies, is done in several iterations. In such a process, each design iteration requires a firmness analysis.

Fig. 5.8 compares the run-time of the four firmness analysis methods on 50 synchronous task sets with $M = 5$ and three values of $k \in \{10, 50, 100\}$ running under the SPP policy. The FAn method terminates within 35s for all the experiments. Moreover, Fig. 5.9 depicts the run-time of these methods for 50 other combinations of tasks with $k = 10$ and $M \in \{5, 15, 30\}$. Fig. 5.8 and Fig. 5.9 show that the application of the FAn method becomes more crucial in the cases with a high number of tasks M and a large value of k. The MILP-based method may be applied for the cases with a lower k and not very high number of tasks, i.e.,$M \leq 15$ in our experiments. The BF method might be used in the cases where the period of the Hit-Zone Function is short.

Table 5.5: Three examples of 50 experimental setups considered in this work running under asynchronous SPP policy

Task set (τ_1 with $(150, 170)$-firm condition is under firmness analysis)	H_{1^+}	Asyn. SPP				
		O_{max}	DHs	run time		
				FAn	TA	BF
$\Pi_1 = \{\tau_4 = (5, 50, 48, 0)$ $, \tau_3 = (7, 50, 47, 12)$ $, \tau_2 = (12, 30, 30, 19)$ $, \tau_1 = (17, 57, 55, O_1)\}$	150	11.88	164	$<$ 0.1s	50s	30s
$\Pi_2 = \{\tau_4 = (5, 50, 48, 0)$ $, \tau_3 = (7, 47, 46, 12)$ $, \tau_2 = (11, 29, 28, 19)$ $, \tau_1 = (16, 57, 55, O_1)\}$	68150	41530	161	6s	>1h	40m
$\Pi_3 = \{\tau_6 = (5, 50, 58, 0)$ $, \tau_5 = (5, 50, 56, 12)$ $, \tau_4 = (7, 30, 28, 19)$ $, \tau_3 = (7, 50, 50, 37)$ $, \tau_2 = (5, 20, 17, 21)$ $, \tau_1 = (13, 57, 55, O_1)\}$	300	231.08	157	5s	>1h	>1h

The number of tasks M is the dominant factor with respect to the complexity of the TA method. Although the majority of our experiments with $M < 6$ terminated using UPPAAL verification, none of the experiments with $M > 8$ terminated within one hour. However, UPPAAL verification is not as much affected by the value of k as it is by M.

5.5.6 Accuracy of FAn for synchronous SPP

We define the error of our results (for synchronous SPP policy) as the absolute difference between the minimum DHs obtained by the FAn method and the minimum DHs obtained by the BF method divided by the minimum DHs obtained by the BF method. Therefore, we obtain the accuracy of the FAn method as follows

$$Acc_{FAn} = 1 - \frac{|min_{BF}(DHs) - min_{FAn}(DHs)|}{min_{BF}(DHs)} \tag{5.13}$$

where $min_{BF}(DHs)$ and $min_{FAn}(DHs)$ are the minimum DHs obtained by the BF and the FAn methods, respectively. The average accuracy of the FAn

Table 5.6: Three examples of 50 experimental setups considered in this work running under synchronous SPP policy

Task set (same as Table 5.5)	H_{1+}	DHs	Syn. SPP run time			
			FAn (accuracy)	TA	BF	MILP
Π_1	150	149	< 0.1s (92%)	40m	610s	>1h
Π_2	68150	162	21s (85%)	55m	>1h	>1h
Π_3	300	138	<1s (86%)	>1h	>1h	>1h

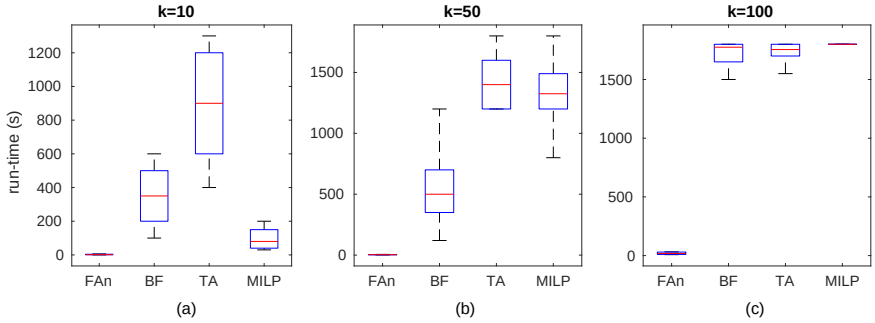

Figure 5.8: Run-time comparision among different firmness analysis methods with $M = 5$ and (a) $k = 10$, (b) $k = 50$, (c) $k = 100$.

method for all 50 synchronous task sets is 87%. The accuracy of the MILP-based method can be calculated using Eq. 5.13 by substituting $min_{MILP}(DHs)$, i.e.,the minimum number of DHs obtained by the MILP-based method, with $min_{FAn}(DHs)$. The MILP-based method is 97% accurate on an average in those of our experiments which are terminated within the time limit.

5.6 Related work

Feasibility analysis of hard real-time tasks are thoroughly studied in literature. Liu and Layland [43] proposed the feasibility analysis of a set of periodic tasks

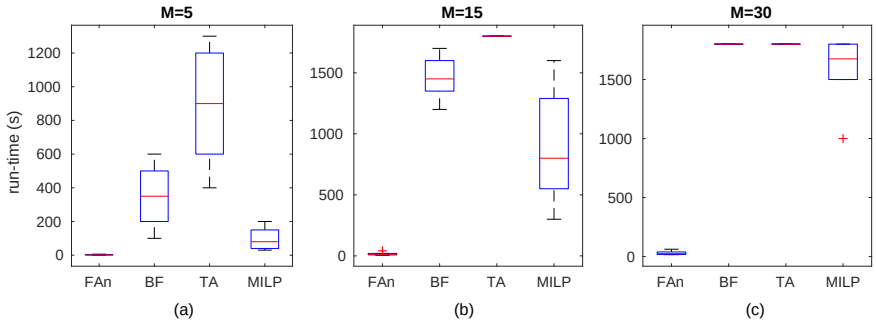

Figure 5.9: Run-time comparision among different firmness analysis methods with $k = 10$ and (a) $M = 5$, (b) $M = 15$, (c) $M = 30$.

running under a synchronous SPP policy. The notion of *critical instance* is used to obtain the worst-case response time of a task. If this value is less than the deadline of a job, it is guaranteed that no job will miss its deadline. Leung and Whitehead [41] proposed a method for feasibility analysis of a periodic task running under an asynchronous SPP policy. It is proved that, starting from any job of a task, it is necessary and sufficient to simulate the execution of all jobs in one hyper-period of the tasks with priorities equal to and higher than that of the task under the feasibility analysis. This corresponds to the fact that the schedule repeats itself in this finite time interval.

Bernat [13] proposed a relevant method as in [41] for weakly-hard real-time tasks (including (m, k)-firm tasks) running under an asynchronous SPP schedule. Obtaining all the executions of a task in one hyper-period, [41] considered all the combinations of k consecutive jobs to obtain the maximum possible number of DHs in any k consecutive jobs. Another line of work provides a lower bound on the number of DHs based on the typical worst-case analysis on the execution trace of a set of tasks running under an SPP [71] [76] and Static Priority Non-Preemptive (SPNP) [76] [58] policies. [25] proposes probabilistic analysis on the number of deadline misses in k consecutive jobs of a task running under an asynchronous SPP policy. All the work above assume that the relative release times of all tasks including the task under firmness analysis are given. This is not the case in the process of mapping a new task to an existing set of asynchronous tasks. The first release time has to be obtained. Recently [69] proposes a firmness analysis method for synchronous SPP policy. This method is based on a Mixed Integer Linear Programming

(MILP) problem formulation. As it was shown in Section 5.5, the FAn method scales substantially better for firmness analysis problem instances with a large k and a high number of tasks.

5.7 Summary

The method proposed in this chapter is an extension of the FAn method to firmness analysis of a set of tasks running under an SPP policy. The FAn method translates the firmness analysis problem to that of the balloons and rake problem. We consider the verification of the firmness conditions for a task at design time. Therefore, the FAn method provides the exact value of the maximum possible number of DHs and the corresponding first release time for a task that is intended to be added to a set of asynchronous tasks running under an SPP policy. Moreover, the FAn method obtains a conservative bound for the minimum possible number of DHs for a task running under a synchronous SPP policy. The proposed method is a foundation for scheduling with deadline misses within firmness conditions. This allows us to consider better-than-the-worst-case design for real-time tasks as opposed to traditional design with hard deadlines (which is a special case of firmness analysis). One possible future work is to analyse the impact of designing with deadline misses on resource dimensioning for SPP policies, like we did in Chapter 2 for TDMA.

Chapter 6

Conclusions and future work

Real-time applications are omnipresent in user applications and industrial cyber-physical systems. The trend is to introduce more and more functionalities on a single system. Resource sharing is an obvious approach to address the cost issues in the development of such systems. Dealing with inter-application interference and guaranteeing execution predictability is a general challenge in the design of such systems where multiple functionalities share resources. Various scheduling mechanisms and policies have been developed over the last decades to achieve real-time guarantees.

Due to safety-critical nature of the real-time systems, it is not acceptable to miss any computation deadlines. Hence, the design of a real-time system is traditionally performed considering the worst-case system and timing behavior even though the worst case may occur rarely. Such worst-case design leads to a large resource over-dimensioning with respect to a typical case. Our approach is to relax such design pessimism by allowing occasional deadline misses while still guaranteeing functional correctness. That is the better-than-worst-case design philosophy. To this end, this thesis aimed to answer the following questions:

- What type of tasks tolerate deadline misses without compromising functional correctness? How do we obtain an acceptable bound on the number of deadline misses?
- How do we allocate resources for the tasks with predefined bounds on

the number of deadline misses? We consider a set of tasks with both hard and firmness constraints.

- For a given task in a set of real-time tasks scheduled on a resource, how do we verify the satisfaction of deadline miss bounds?

6.1 Conclusions

We addressed the design and resource allocation problem of (m, k)-firm tasks. Such tasks satisfy the functional requirements if they meet the deadline of at least m jobs out of any k consecutive jobs. Our contributions are as follows.

As the first contribution, we proposed a method to obtain the m and k parameters for feedback control loops based on their stability and performance requirements. We obtained two tuples (m, k) for a task. One considers a small window of consecutive jobs and obtains an m that guarantees the stability of the control loop. The other tuple considers a longer window of consecutive jobs considering the performance requirement.

Second, we proposed the Multi-Constraint Resource Allocation (MuCoRA) framework for co-mapping multi-constraint (throughput and latency constraints) real-time applications on a multiprocessor platform running under a TDMA policy. We considered the tasks with both hard and (m, k)-firmness constraints. The proposed method aims to allocate the minimum possible number of slots to a task while satisfying its timing constraints. We showed that designing with deadline misses is meaningful (since it provides resource efficiency) and tractable (because the analysis is scalable).

Third, we proposed the balloons and rake problem which represents the core challenges in the firmness analysis problems. We solved the balloons and rake problem using the Finite Point method. We believe that the solution of the balloons and rake problem has applications in other engineering domains. We provided an example of this application by translating the Rifle and Propeller problem, i.e.,a traditional challenge in the aviation domain, to the balloons and rake problem.

As the fourth contribution, we addressed the firmness analysis of a set of tasks running under a TDMA policy. We proposed the Firmness Analysis (FAn) method to translate this problem to that of the balloons and rake problem. For a given (m, k)-firm task in a set of synchronous and asynchronous tasks, the FAn method obtains the maximum and minimum number of the deadline misses in any k consecutive jobs of the tasks, respectively. We generalized the FP method using a timed-automata model of the problem to

evaluate the FAn method. The UPPAAL tool was used to validate and verify the timed-automata model. Considering all the relative release times of tasks, a brute-force approach verifying the maximum and minimum number of deadline misses was also proposed to evaluate the FAn method. The FAn method shows an acceptable run-time for different sets of parameters. The FAn method scales considerably better than the other methods.

The fifth contribution of this thesis is the translation of the firmness analysis problem in a set of asynchronous tasks running under an SPP policy to the balloons and rake problem. Since in such a task set the relative release times are fixed, we obtained the first release time for an (m, k)-firm task that is intended to be added to the task set. This results in the maximum number of deadline hits in any k consecutive jobs of the task. The scalability of the FAn method is compared with that of existing work – a brute-force search approach and a timed-automata model of the problem that is analyzed using the reachability analysis of the UPPAAL model checker. The FAn method scales significantly better than the others for a high number of tasks and a higher k.

Finally, we proposed a scalable method for obtaining the maximum number of deadline misses in any k consecutive jobs of a task running under a synchronous SPP policy. We extended the FAn method for the synchronous SPP policy. We used three methods to evaluate the FAn method: i) a state-of-the-art Mixed-Integer Linear programming (MILP) -based method; ii) a timed automata model of the problem that is analyzed using the UPPAAl tool; and iii) a brute-force search of all the possible release times among tasks. While obtaining conservative results, the FAn method scales substantially better for the cases with a large window of consecutive jobs, i.e.,higher k.

6.2 Future work

The result of this work is a foundation for the design of real-time systems with deadline misses. We have shown the challenges involved and many of them are addressed in this thesis. However, there can be a number of (relevant) variants of the problem. Some of the problems are described in the following.

6.2.1 Accurate firmness analysis for synchronous SPP

The Balloons and Rake problem that is introduced in Chapter 3 considers a line of balloons with infinitely many balloons. An alternative to this problem is a case in which there exists more than one line of balloons. The lines of

balloons are located on top of each other and in parallel. The parameters related to the balloon line, i.e., balloon line period, balloons size etc., may vary in different lines. The lines of balloons have random relative positions. The objective is to obtain the maximum number of struck balloons with one time pushing the rake to the balloon lines.

The solution of this problem can be used to obtain an exact solution for the firmness analysis problem of a set of synchronous tasks running under an SPP policy. Considering a set of periodic tasks running under an SPP policy, we can obtain hit-zone functions for tasks with higher priorities than the task under firmness analysis – which we assume to be the lowest priority task. Then, considering each hit-zone function as a balloon function, we can translate the firmness analysis problem of a set of synchronous tasks running under an SPP policy to that of the balloons and rake problem with multiple lines of balloons.

6.2.2 Firmness analysis for non-preemptive schedules

The scheduling policies considered in this thesis are preemptive. Non-preemptive schedules are also used in various operating systems, e.g. OSEK/VDX [53], because of several reasons like lower context switching overhead in non-preemptive schedules. First-come-first-served and priority-based policies are examples of non-preemptive policies [6]. Similar to preemptive schedules, firmness analysis however requires to verify the satisfaction of the firmness conditions. Therefore, one possible extension of this work is to obtain the maximum possible number of deadline misses of a firm task running under a static or dynamic non-preemptive policy.

6.2.3 General firmness analysis framework

A possible extension of this thesis is to propose a framework for design of real-time systems which covers a wider range of policies including more complex industrial systems as well as combinations of multiple scheduling policies. One example is to consider distributed control systems with communication and computation tasks each running under different policies. In such a system, not only insufficient processing time causes deadline misses but also too much delay in a shared communication system, e.g. between the sensor and processor or between processor and actuator, might cause deadline misses.

6.2.4 Firmness analysis of soft real-time tasks

Quality of control changes with the number of deadline misses for soft real-time tasks. This introduces statistical interpretations for the performance of the soft real-time tasks relative to the number of deadline misses. The average number of deadline misses in a given window of consecutive jobs of a task is an example. Obtaining such information requires statistical analysis of the response time of tasks.

Bibliography

[1] Åarzén, K. (1999). A simple event-based pid controller. *IFAC Proceedings Volumes*, 32(2):8687–8692.

[2] Akesson, B., Minaeva, A., Sucha, P., Nelson, A., and Hanzalek, Z. (2015). An efficient configuration methodology for time-division multiplexed single resources. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), Proceedings of*. IEEE.

[3] Alur, R. and Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235.

[4] Aminifar, A., Bini, E., Eles, P., and Peng, Z. (2014). Bandwidth-efficient controller-server co-design with stability gualizadehntees. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. European Design and Automation Association.

[5] Andrei, A., Eles, P., Peng, Z., and Rosen, J. (2008). Predictable implementation of real-time applications on multiprocessor systems-on-chip. In *VLSI Design, 21st International Conference on*. IEEE.

[6] Baruah, S., Bertogna, M., and Buttazzo, G. (2015). *Multiprocessor scheduling for real-time systems (PhD thesis)*. Springer.

[7] Behrmann, G., David, A., Larsen, K., Hakansson, J., Petterson, P., Yi, W., and Hendriks, M. (2006). Uppaal 4.0. In *Quantitative Evaluation of Systems, Third International Conference on*. IEEE.

[8] Behrouzian, A., Goswami, D., and Basten, T. (2018a). Robust co-synthesis of embedded control systems with occasional deadline misses. In *On-Line Testing and Robust System Design (IOLTS), International Symposium on*. IEEE.

[9] Behrouzian, A., Goswami, D., Basten, T., Geilen, M., and Alizadeh, H. (2015). Multi-constraint multi-processor resource allocation. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), International Conference on*. IEEE.

[10] Behrouzian, A., Goswami, D., Basten, T., Geilen, M., Alizadeh, H., and Hendriks, M. (2018b). Firmness analysis of real-time applications under static-priority preemptive scheduling. In *Real-Time and Embedded Technology and Applcations Symposium (RTAS), Proceedings of*. IEEE.

[11] Behrouzian, A., Goswami, D., Basten, T., Geilen, M., Alizadeh, H., and Hendriks, M. (2018c). Firmness analysis of real-time tasks. *Submitted*.

[12] Behrouzian, A., Goswami, D., Geilen, M., Hendriks, M., Alizadeh, H., van Horssen, E., Heemels, W., and Basten, T. (2016). Sample-drop firmness analysis of tdma-scheduled control applications. In *Industrial Embedded Systems (SIES), International Symposium on*. IEEE.

[13] Bernat, G. (1998). *Specification and analysis of weakly hard real-time systems (PhD thesis)*. Universitat de les Illes Balears, Spain.

[14] Bernat, G., Burns, A., and Llamosi, A. (2001). Weakly hard real-time systems. *Computers, IEEE Transactions on*, 50(4):308–321.

[15] Bertogna, M. (2008). *Real-time scheduling analysis for multiprocessor platforms*. Scuola Superiore Sant'Anna, Pisa.

[16] Bhave, A. and Krogh, B. (2008). Performance bounds on state-feedback controllers with network delay. In *CDC*. IEEE.

[17] Bini, E. and Cervin, A. (2008). Delay-aware period assignment in control systems. In *Real-Time Systems, Symposium on*. IEEE.

[18] Borgers, D., Postoyan, R., Anta, A., Tabuada, P., Nešić, D., and Heemels, W. (2018). Periodic event-triggered control of nonlinear systems using overapproximation techniques. *Automatica*, 94:81–87.

[19] Borwein, D. and Borwein, J. (1991). Fixed point iterations for real functions. *Journal of Mathematical Analysis and Applications*, 157(1):112–126.

[20] Buttazzo, G. (2011). *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media.

[21] Cassez, F. and Larsen, K. (2000). The impressive power of stopwatches. In *International Conference on Concurrency Theory*. Springer.

[22] Cervin, A. and Alriksson, P. (2006). Optimal on-line scheduling of multiple control tasks: A case study. In *Real-Time Systems, Euromicro Conference on*. IEEE.

[23] Coutinho, M., Rufino, J., and Almeida, C. (2008). Response time analysis of asynchronous periodic and sporadic tasks sheduled by priority preemptive algorithm. In *Real-Time Systems (ECRTS), Euromicro Conference on*. IEEE.

[24] Dertouzos, M. (1974). The procedural control of physical processes. In *the IFIP Congress, In Proceedings of*.

[25] Díaz, J., García, D., Kim, K., Lee, C., Bello, L., López, J., Min, S., and Mirabella, O. (2002). Stochastic analysis of periodic real-time systems. In *Real-Time Systems, Symposium on*, pages 289–300. IEEE.

[26] Ebbers, M. (2012). *Introduction to the New Mainframe: z/OS Basics*. IBM Redbooks.

[27] Felicioni, F., Jia, N., Song, Y., and Simonot-Lion, F. (2006). Impact of a (m, k)-firm data dropouts policy on the quality of control. In *6th IEEE International Workshop on Factory Communication Systems*. IEEE.

[28] Flavia, F., Ning, J., Simonot-Lion, F., and YeQiong, S. (2008). Optimal on-line (m, k)-firm constraint assignment for real-time control tasks based on plant state information. In *Emerging Technologies and Factory Automation, IEEE International Conference on*. IEEE.

[29] Goossens, K., Azevedo, A., Chandrasekar, K., Gomony, M., Goossens, S., Koedam, M., Li, Y., Mirzoyan, D., Molnos, A., Nejad, A., et al. (2013). Virtual execution platforms for mixed-time-criticality systems: the compsoc architecture and design flow. *ACM SIGBED Review*, 10(3):23–34.

[30] Goossens, K., Koedam, M., Nelson, A., Sinha, S., Goossens, S., Li, Y., Breaban, G., van Kampenhout, R., Tavakoli, R., Valencia, J., et al. (2017). Noc-based multiprocessor architecture for mixed-time-criticality applications. *Handbook of hardware/software codesign*, pages 491–530.

[31] Goswami, D., Lukasiewycz, M., Schneider, R., and Chakraborty, S. (2012). Time-triggered implementations of mixed-criticality automotive

software. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium.

[32] Goswami, D., Schneider, R., and Chakraborty, S. (2014). Relaxing signal delay constraints in distributed embedded controllers. *IEEE Transactions on Control Systems Technology*, 22(6):2337–2345.

[33] Guan, N. (2016). *Techniques for building timing-predictable embedded systems*. Springer.

[34] Gupta, R. and Chow, M. (2010). Networked control system: Overview and research trends. *IEEE transactions on industrial electronics*, 57(7):2527–2535.

[35] Hamdaoui, M. and Ramanathan, P. (1995). A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *Computers, IEEE Transactions on*, 44(12):1443–1451.

[36] Hu, J. and Marculescu, R. (2005). Energy-and performance-aware mapping for regular noc architectures. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 24(4):551–562.

[37] Ismail, H., Jawawi, D., and Isa, M. (2015). A weakly hard real-time tasks on global scheduling of multiprocessor systems. In *Software Engineering Conference (MySEC), Proceedings of*. IEEE.

[38] Jia, N., Song, Y.-Q., and Simonot-Lion, F. (2007). Task handler based on (m, k)-firm constraint model for managing a set of real-time controllers. In *15th International Conference on Real-Time and Network Systems-RTNS*.

[39] Khaitan, S. and McCalley, J. (2015). Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, 9(2):350–365.

[40] Kim, S., Bothwell, C., and Fortenbaugh, R. (2018). Rotorcraft fly-by-wire control laws. US Patent App. 15/688,164.

[41] Leung, J. Y.-T. and Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250.

[42] Lin, Z., Yuegang, L., Jing, K., and Zhanqun, S. (2015). Anti-lock braking system's performance of vehicle using hardware in-the-loop techniques. In *2015 Sixth International Conference on Intelligent Systems Design and Engineering Applications (ISDEA)*, pages 151–154. IEEE.

[43] Liu, C. and Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61.

[44] Mason, O. and Shorten, R. (2004). On common quadratic lyapunov functions for stable discrete-time lti systems. *IMA Journal of Applied Mathematics*, 69(3):271–283.

[45] Naghshtabrizi, P. and Hespanha, J. (2009). Analysis of distributed control systems with shared communication and computation resources. In *American Control Conference, 2009. ACC'09.* IEEE.

[46] Nejad, A., Molnos, A., and Goossens, K. (2013). A software-based technique enabling composable hierarchical preemptive scheduling for time-triggered applications. In *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA).* IEEE.

[47] Ning, J., Song, Y., and Rui-Zhong, L. (2005). Analysis of networked control system with packet drops governed by (m, k)-firm constraint. In *Fieldbus Systems and Their Applications, International Conference on.* IFAC.

[48] Online. Automotive open system architecture. URL http://www. autosar. org.

[49] Online. Autosar automotive open system architecture (2016). http:// www.autosar.org/.

[50] Online. A backgrounder on ieee std 1003.1. http://www.opengroup.org/ austin/ papers/backgrounder.html.

[51] Online. Enea ose: High-performance, posix compatible, multicore real-time operating system. https://www.enea.com/globalassets/downloads/ operating-systems/enea-ose/datasheet-enea-ose.pdf.

[52] Online. Jet propulsion. https://en.wikipedia.org/wiki/Jet_propulsion.

[53] Online. Osek/vdx operating system specification. Website: http://www. osek-vdx. org.

[54] Online. Philips interventional x-ray system. https://www.usa.philips.com/ healthcare/ solutions/interventional-xray.

[55] Online. Synchronization gear. https://en.wikipedia.org/wiki/Synchronization-gear.

[56] Online. Why synchronizer gears were so important in world war i? https://www.thefirearmblog.com/blog/2016/11/25/bullets-versus-propellers-synchronizer-gears-important-world-war-slowmo-guys/.

[57] Perathoner, S., Wandeler, E., and Thiele, L. (2005). Timed automata templates for distributed embedded system architectures. *Tech. Rep.*, page 233.

[58] Quinton, S., Bone, T., Hennig, J., Neukirchner, M., Negrean, M., and Ernst, R. (2014). Typical worst case response-time analysis and its use in automotive network design. In *Design Automation, Proceedings of the 51st Annual Conference on*. ACM.

[59] River, W. (2007). Vxworks. *URL: http://www. windriver. com*.

[60] Schranzhofer, A., Pellizzoni, R., Chen, J., Thiele, L., and Caccamo, M. (2010). Worst-case response time analysis of resource access models in multi-core systems. In *Proceedings of the 47th Design Automation Conference*. ACM.

[61] Shibu, K. (2009). *Introduction to embedded systems*. Tata McGraw-Hill Education.

[62] Siyoum, F., Akesson, B., Stuijk, S., Goossens, K., and Corporaal, H. (2011). Resource-efficient real-time scheduling using credit-controlled static-priority arbitration. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), International Conference on*. IEEE.

[63] Sousa, P. (2013). *Real-Time Scheduling on Multi-core: Theory and Practice*. Faculdade de Engenharia da Universidade do Porto.

[64] Srinivasan, K., Chatha, K. S., and Konjevod, G. (2006). Linear-programming-based techniques for synthesis of network-on-chip architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(4):407–420.

[65] Sriram, S. and Bhattacharyya, S. (2009). *Embedded multiprocessors: Scheduling and synchronization*. CRC press.

[66] Stuijk, S., Basten, T., Geilen, M., and Corporaal, H. (2007). Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Proceedings of the 44th annual Design Automation Conference*. ACM.

[67] Stuijk, S., Geilen, M., and Basten, T. (2006). Sdf3: Sdf for free. In *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*. IEEE.

[68] Stuijk, S., Geilen, M., Theelen, B., and Basten, T. (2011). Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pages 404–411. IEEE.

[69] Sun, Y. and Natale, M. (2017). Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):171.

[70] Tendulkar, P. (2014). *Mapping and scheduling on multi-core processors using SMT solvers*. Universite de Grenoble I-Joseph Fourier.

[71] Tobuschat, S., Ernst, R., Hamann, A., and Ziegenbein, D. (2016). System-level timing feasibility test for cyber-physical automotive systems. In *Industrial Embedded Systems (SIES), International symposium on*. IEEE.

[72] Valencia, J., Goswami, D., and Goossens, K. (2015). Composable platform-aware embedded control systems on a multi-core architecture. In *Digital System Design (DSD), Euromicro Conference on*. IEEE.

[73] van Horssen, E., Behrouzian, A., Goswami, D., Antunes, D., Basten, T., and Heemels, M. (2016). Performance analysis and controller improvement for linear systems with (m, k)-firm data losses. In *European Control Conference (ECC)*. IEEE.

[74] Wang, X. and Lemmon, M. (2011). Event-triggering in distributed networked control systems. *IEEE Transactions on Automatic Control*, 56(3):586.

[75] Xie, Y., Yan, H., and Pang, Z. (2016). Mixed time-triggered and event-triggered controller for industrial iot applications. In *Industrial Technology (ICIT), International Conference on*, pages 2064–2067. IEEE.

[76] Xu, W., Hammadeh, Z., Kröller, A., Ernst, R., and Quinton, S. (2015). Improved deadline miss models for real-time systems using typical worst-case analysis. In *Real-Time Systems (ECRTS), 27th Euromicro Conference on*. IEEE.

[77] Zhang, K., Yao, Y., Labanni, O., Lu, Z., and Wu, X. (2012). A new universal-environment adaptive multi-processor scheduler for autonomous cyber-physical system. In *Computer and Information Science (ICIS), International Conference on*. IEEE.

Curriculum Vitæ

Amirreza Baghbanbehrouzian was born on 24-4-1987 in Tabriz, Iran. After finishing college in 2005 at National Organization for Development of Exceptional Talents in Tabriz, Iran, he studied Bachelor of Science at Sahand University of Technology. He studied Master of Science at University of Tehran in Iran. In 2012 he graduated within Advanced VLSI Lab on On-Chip Interconnects Modeling. From June 2014 he started a PhD project at Technische Universiteit Eindhoven in Eindhoven, The Netherlands, of which the results are presented in this dissertation. Since 2018 he is employed at Océ.

List of publications

Publications covered in the thesis

A. Behrouzian, D. Goswami, T. Basten, M. Geilen, H. Alizadeh Ara, and M. Hendriks, "Firmness analysis of real-time tasks," Submitted.

A. Behrouzian, D. Goswami, T. Basten, M. Geilen, H. Alizadeh Ara, "Efficient Resource Allocation for Real-Time Systems with Deadline Misses," Submitted.

A. Behrouzian, D. Goswami, T. Basten, M. Geilen, H. Alizadeh Ara, and M. Hendriks, "Firmness analysis of real-time applications under static-priority preemptive scheduling," *in Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018.

A. Behrouzian, D. Goswami, and T. Basten, "Robust co-synthesis of embedded control systems with occasional deadline misses", *In On-Line Testing and Robust System Design (IOLTS), 24th IEEE International Symposium on.* IEEE, 2018.

A. Behrouzian, D. Goswami, M. Geilen, M. Hendriks, H. Alizadeh Ara, E. van Horssen, W. Heemels, and T. Basten, "Sample-drop firmness analysis of tdma-scheduled control applications," *in Industrial Embedded Systems (SIES), International Symposium on.* IEEE, 2016.

A. Behrouzian, D. Goswami, T. Basten, M. Geilen, and H. Alizadeh Ara, "Multi-constraint multi-processor resource allocation," *in Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), International Conference on.* IEEE, 2015.

Publications not covered in the thesis

H. Alizadeh Ara, M. Geilen, A. Behrouzian and T. Basten, "Throughput-buffering trade-off analysis for scenario-aware dataflow models," *in Proc. 26th International Conference on Real-Time Networks and Systems (RTNS).* ACM, 2018.

H. Alizadeh Ara, M. Geilen, A. Behrouzian, T. Basten and D. Goswami, "Compositional dataflow modelling for cyclo-static applications," *in Proc. Euromicro Conference on Digital System Design (DSD).* IEEE, 2018.

H. Alizadeh Ara, A. Behrouzian, M. Hendriks, M. Geilen, D. Goswami and T. Basten, "Scalable analysis for multi-scale dataflow mdels," *ransactions on Embedded Computing Systems.* 17, 4, Article 80. ACM, 2018.

M. Hendriks, M. Geilen, A. Behrouzian, T. Basten, H. Alizadeh Ara, and D. Goswami, "Checking metric temporal logic with TRACE," *in Proc. 16th International Conference on Application of Concurrency to System Design (ACSD). Tool paper.* IEEE, 2016.

H. Alizadeh Ara, M. Geilen, T. Basten, A. Behrouzian, and D. Goswami, "Tight temporal bounds for dataflow applications mapped onto shared resources," *in Industrial Embedded Systems (SIES), International Symposium on.* IEEE, 2016.

E. van Horssen, A. Behrouzian, D. Goswami, D. Antunes, T. Basten, and M. Heemels, "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses," *in European Control Conference (ECC).* IEEE, 2016.

S. Adyanthaya, H. Alizadeh Ara, J. Nogueira Bastos, A. Behrouzian, R. Medina Sanchez, J. van Pinxten, L. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, M. Hendriks, S. Stuijk, M. Reniers, and J. Voeten, "xCPS: A tool to explore cyber physical systems," *ACM SIGBED Review, vol. 14, no. 1, pp. 81–95, 2016.*

H. Alizadeh Ara, A. Behrouzian, M. Geilen, M. Hendriks, D. Goswami and T. Basten, "Analysis and visualization of execution traces of dataflow applications," *In 2nd International Workshop on Integrating Dataflow, Embedded computing and Architecture, IDEA 2016, Abstract proceedings.* ESR, 2017.

S. Adyanthaya, H. Alizadeh Ara, J. Nogueira Bastos, A. Behrouzian, R. Medina Sanchez, J. van Pinxten, L. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, S. Stuijk, M. Reniers, and J. Voeten, "xCPS: A tool to explore cyber physical systems," *Workshop on Embedded and Cyber-Physical Systems Education (WESE).* ACM, 2015.

A. Behrouzian and N. Masoumi, "Analytical solutions for distributed interconnect models–part ii: Arbitrary input response and multicoupled lines," *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems, vol. 23, no. 9, pp. 1879–1888, 2014.*

A. Behrouzian and N. Masoumi, "Analytical solutions for distributed interconnect models–part i: Step input response of finite and semi-infinite lines," *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems, vol. 22, no. 12, pp. 2596–2606, 2013.*

A. Behrouzian and N. Masoumi, "Arbitrary point transient response of rlc interconnects based on composed Fourier analysis," *in 13th Mediterranean Microwave Symposium (MMS). IEEE, 2013.*