State-based switching multi-rate controller for improving resource utilization on predictable and composable platforms

Microprocessors and Microsystems 91 (2022) 104517

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

State-based switching multi-rate controller for improving resource utilization on predictable and composable platforms

Mojtaba Haghi [*], Shengru Yu, Dip Goswami, Kees Goossens, Martijn Koedam, Andrew Nelson

Eindhoven University of Technology, The Netherlands

A R T I C L E   I N F O

Keywords:
Embedded Control Systems
Embedded platforms
Predictable and composable platforms
Resource utilization
Multi-rate control

A B S T R A C T

Resource sharing is a crucial design consideration in design of embedded systems for cost and resource utilization reasons. The system-level performance is negatively influenced by resource sharing due to inter-application interference. For a control application, this further implies a trade-off between the resource utilization and the control performance. For a control application, the sampling rate is an important knob to perform trade-off between resource utilization and the control performance. In this paper, we present a state-base switching multi-rate controller (SSMC) scheme targeting predictable and composable multi-core platforms. In the proposed scheme, the controller switches between multiple *sampling* rates (or application periods) based on the state of the system i.e., the transient and steady state. We propose two multi-rate control laws targeting SSMC — one using single gain and one using multiple gains over different *actuating* points. We address the impact of model uncertainty by using a parallel observer system. We validate the effectiveness of the proposed scheme performing hardware-in-the-loop simulations targeting an industrial multi-core platform — Verintec, synthesized on a PYNQ Z2 FPGA board. Finally, we demonstrated that the proposed scheme outperforms the state-of-the-art techniques in terms of resource utilization and the control performance.

1. Introduction

In recent years, the usage of embedded systems to implement various applications has rapidly increased. The application complexity varies from simple internet-of-things (IoT) devices to complex real-time applications in vehicles. In modern cars, numerous applications/functions of different types run on processing units with limited processing resource and memory. Currently, safety critical control functions like cruise control, braking system, automatic parking and so on run on such dedicated low-capacity units called electronic control units (ECUs). Such federated architectures [1] with exclusive ECU per function ease the integration and validation process, particularly when functions are developed by different third party suppliers. However, such architectures lead to a high number of ECUs per car since the typical number of applications in a car has increased from 20 in 2000s to more than 100 applications in recent years. With the current federated approach the trend of increasing applications/functions would eventually result in a higher cost and possibly infeasible Electronic/Electrical architecture. To keep the embedded implementations cost low, the recent trend is to consider integrated architectures where multiple functions run on a single ECU [2]. An integrated architecture reduces the per vehicle production cost between 60 to 100 USD [2]. In that spirit, in this paper, we study the implementation of feedback controllers sharing an embedded platform with other applications. By reducing effective resource utilization, we try to enable a scenario where multiple (or higher number of) applications share the platform leading to a lower cost.

The multi-application scenario, that is expected in an integrated architecture, generally cause inter-application interference resulting in execution jitter and execution drift which are particularly undesirable in feedback controllers [3]. The control applications expect strictly periodic and deterministic execution of control software which is particularly challenging in multi-application scenarios. This imposes several important requirements on the target implementation platform — composability, predictable, and determinism are the three most notable ones. We consider a composable and predicable platform — CompSOC, which ensures deterministic and interference-free execution [4]. Due to these properties, the CompSOC platform is naturally suitable for embedded control implementation [5,6]. It should be noted that our proposed approach is general enough to target a wider class of platforms (see Section 3).

The aim of a control application is generally to govern the inputs of a dynamical system and to drive its output from an initial state to a

[*] Corresponding author.
E-mail addresses: s.m.haghi@tue.nl (M. Haghi), s.yu@student.tue.nl (S. Yu), d.goswami@tue.ml (D. Goswami), k.g.w.goossens@tue.ml (K. Goossens), m.l.p.j.koedam@tue.nl (M. Koedam), a.t.nelson@tue.nl (A. Nelson).

https://doi.org/10.1016/j.micpro.2022.104517
Received 10 September 2021; Received in revised form 24 February 2022; Accepted 28 March 2022
Available online 9 April 2022

Table 1
List of important symbols and acronyms.

Symbol	Meaning	Acronyms	Meaning
h_s, h_f	Sampling period	SSMC	State-based multi-rate switching scheme
$r(t)$	Desired output signal	SSR	Slow single-rate
t_s	Settling time	FSR	Fast single-rate
T_r	Input signal period	HIL	Hardware-in-the-loop
r_a	Actuating period	SIL	Software-in-the-loop
r_s	Sensing period	FPGA	Field-programmable gate array
ω	Size of CoMik slot	OS	Operating system
ψ_i	Size of *i*th partition slot	CoMik	Composable and predictable micro-kernel
K	Feedback gain	VEP	Virtual execution platform
F	Feedforward gain	WCET	Worst-case-execution-times
R	Resource utilization	TDM	Time-division multiplexing
		RM	Reconfiguration manager
		QoC	Quality of control
		LTI	Linear-time-invariant
		SSC	State-based switching scheme
		CQLF	Common quadratic lyapunov function
		POS	Parallel observer system
		SOS	Slow observer system
		FOS	Fast observer system
		SMR	Single-gain multi-rate
		MMR	Multi-gain multi-rate
		ETS	Event-triggered scheme

desired and pre-specified output (reference), while ensuring the system *stability*. At any time instant, the difference between the reference and actual output is called the *error* signal. Based on the range of the error signal, the state of the physical system is categorized as follows. The *transient* state is from the start time when the system output is at an initial condition until the output reaches and stays with 2% error bound around of the reference. As long as the system output is within this specified error bound, the system is in the *steady state*. If the error exceeds the steady state bound, for example, when the reference changes by a large margin, the system state switches back to the transient state. Usually, a controller performance is higher if the system reaches the steady state sooner while respecting the input signal constraints, e.g., the maximum input voltage. Therefore, a shorter transient state results in a higher control performance.

A control application sequentially and periodically executes three operations within a specified interval called sampling interval – *sensing*, *computing*, and *actuating*. The interval between two consecutive start of sensing operations is called the *sampling period*. The sequential order of the execution implies that a shorter sampling period results in a shorter actuating period, which is the period between two consecutive actuating operations. Generally, a shorter actuating period improves the controller performance by shortening the transient state [7,8]. However, a shorter sampling period means a shorter period to execute the control application (i.e., three operations). Let us consider two controllers with sampling periods of h_s and $h_f = h_s/4$. Fig. 1.*a* and Fig. 1*c* demonstrate the response of slow single-rate (SSR) and fast single-rate (FSR) controllers with sampling periods of h_s and h_f respectively. While scheduling FSR with h_f can potentially result in a better control performance (see the results described in Section 5), it requires to execute the control application four times instead of one in each period of h_s, leading to an increased processing resource utilization. Important symbols and acronyms are listed in Table 1.

Let us define *resource utilization* of an application as the amount of processing resource required to execute it. We assume that there is enough memory resource available on the platform for the application. In a multi-application scenario, the straightforward approach for reducing the resource utilization would be to increase the sampling period. However, the downside of such approach is the degradation of the control performance.

We consider the following two important observations to perform the trade-off between resource utilization and control performance.

- A shorter sampling period significantly improves the performance in the transient state. However, the improvement due to a shorter sampling period is minimal in steady state. Therefore, it makes sense to run a controller with a longer sampling period in the steady state to reduce the overall resource utilization with little performance degradation.
- The execution time of the combination of sensing and computing operations is significantly higher than that of the actuating operation [6]. Therefore, offering a multi-rate scheme that executes the sensing and computing operations less frequently than the actuating operation may reduce the resource utilization in transient state. One way to exploit this observation is to increase the period of the sensing and computing operations while decreasing the period of actuating operation. Note that such a multi-rate scheme should, in principle, reduce resource utilization in both computing-heavy [6] and sensing-heavy [9] control systems.

Main idea: The aim of this paper is to propose a switching scheme to achieve a performance close to the one with a short sampling period (i.e., high performance) while keeping the resource utilization close to the one with a longer sampling period (i.e., low resource utilization) as depicted in Fig. 2. The observation is that the transient state requires a shorter sampling period compared to the steady state to meet a given performance requirement. The key idea is to use a shorter *effective* sampling period in the transient state. That is, in the transient state, we use a long sensing period and a shorter actuating period along with a parallel observer resulting in a shorter effective sampling period. We use a longer sampling period when the system reaches the steady state. We refer to such a scheme as state-based switching multi-rate control (SSMC) scheme.

Illustrative example: We illustrate the SSMC scheme with an example shown in Fig. 1. In this example, a plant is controlled by the SSMC scheme to reach the desired output signal $r(t)$ which is a periodic square wave signal defined as:

$$r(t) = Y_0 + \frac{Y_d - Y_0}{2}(1 + sgn(sin(\frac{2\pi t}{T_r}))), \tag{1}$$

where sgn is the signum function and T_r is the signal period. The system output goes from Y_0 at $t = 0$ to Y_d at $t = t_s$ which is the phase when the system is in transient state. In the transient state, the controller should operate with a short actuating period of $r_a = h_f$. This can be done by using a FSR controller with a sensing period $r_s = h_f$. The timing behavior of the FSR controller is depicted in Fig. 1a. An alternative is to use a multi-rate controller (depicted in Fig. 1b) in the transient state which reduces the resource utilization. Under the multi-rate scheme depicted in Fig. 1b, the sensing and computing operations

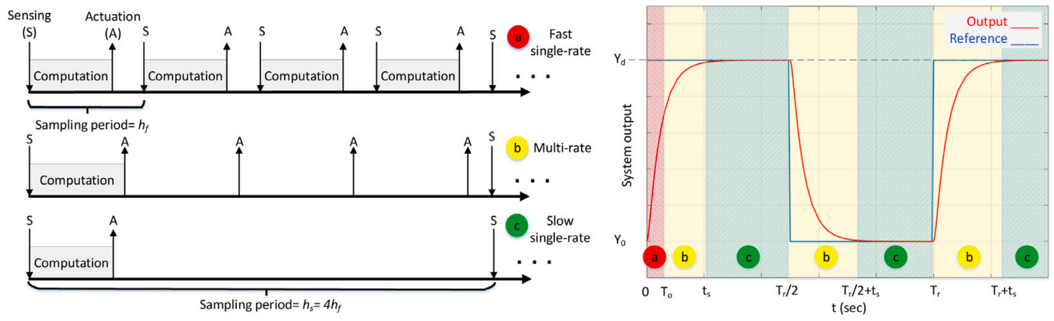


Fig. 1. Different scheduling used in the proposed approach where ∘*a* and ∘*c* are the single-rate controllers with the sensing period of h_f and $h_s = 4 \times h_f$, and ∘*b* is the multi-rate controller.

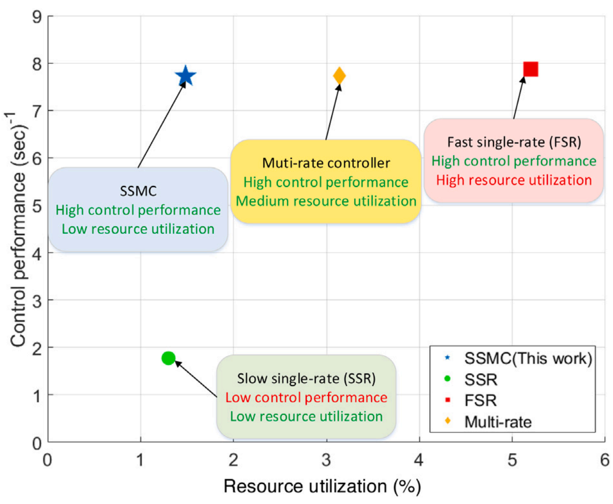


Fig. 2. Resource utilization and control performance comparison between the proposed SSMC and two single-rate controllers with short and long sampling periods.

are executed with a longer period h_s than the actuating period h_f. In essence, the effective sampling period is $h_f = h_s/m$, where $m = 4$ in this example. Under this scheme, the controller calculates m input values (among which $m-1$ are for the future actuating operations) in the computing operation using the model of the system, and actuate the system with an actuating period of $r_a = h_f$. The $m-1$ future actuating operations depend on the accuracy of the model and can be negatively influenced by the presence of model uncertainties. To deal with model-uncertainties, we consider a parallel observer system (POS) to estimate the system output [10], given a measured sensing output value. The POS requires a short initialization phase until the prediction error goes below an user-defined threshold. In this phase, which is shown as $0 < t \leq T_o$ in Fig. 1, the POS requires a shorter sensing period and hence, we use the FSR controller with the sampling period h_f. Note that the design of POS ensures that $T_o \ll t_s$. Therefore, in the transient state, the SSMC scheme starts with a FSR controller plus a POS. Next, at $t = T_o$, it switches to multi-rate plus the POS in the interval $T_o < t \leq t_s$.

When the system reaches the steady state (indicated with green hashed pattern in Fig. 1), SSMC switches to the SSR controller with a sampling period of h_s (depicted in Fig. 1c), increasing the actuating period compared to the multi-rate scheme. This further decreases the resource utilization of the control application. As soon as the system switches back to the transient state at $t = \frac{T_r}{2}$, the SSMC scheme switches back to the multi-rate scheme. The system remains in the transient state in the interval $\frac{T_r}{2} < t < \frac{T_r}{2} + t_s$ under the multi-rate scheme.

In summary, in the first period T_r of $r(t)$, during $0 < t \leq T_o$, a FSR plus a POS is active. In $T_o < t \leq t_s$ the multi-rate scheme is active and in $t > t_s$, the SSR controller is active as long as the system is in the steady state. Whenever the system state switches back to the transient state, at $t = \frac{kT_r}{2}$ and $k \in N$, the controller switches back to the multi-rate scheme.

Our contributions: In this paper,

- We propose the state-based switching multi-rate control (SSMC) scheme targeting composable and predictable multi-core platforms. The proposed scheme switches between a multi-rate controller in the transient state and a SSR controller in the steady state to reduce the resource utilization of embedded controllers while maintaining control performance.
- We demonstrate the effectiveness of the scheme in reducing the resource utilization by designing two different multi-rate schemes with single and multiple gains over the actuating operations. Since the performance of such model-based multi-rate schemes is heavily influenced by the accuracy of the models, we address the model uncertainty by using a *parallel-observer* system within the multi-rate scheme.
- We demonstrate the trade-off between the resource utilization and the quality of control. We compare the two proposed multi-rate control schemes with respect to the resource utilization, the control performance and provide insight on the usability of the controllers.
- We validate the effectiveness of SSMC and the designed controllers by performing hardware-in-the-loop (HIL) simulations targeting an industrial multi-core platform — Verintec, synthesized on a PYNQ Z2 FPGA board.
- We compare our approach with the state-of-the-art methods presented in [11,12] to show that our method outperforms the existing techniques in terms of resource utilization.

2. Related works

This work focuses on the resource-efficient implementation of control applications. In the following, we discuss related work that contributed to improving resource-efficiency for the control applications from different perspectives.

Event-triggered control: Event-triggered control uses an aperiodic sampling scheme where the sensing and actuating operations are performed based on the occurrence of a specific event. An event is defined as the scenario when the error signal exceeds a pre-defined threshold value. The sampling and actuating operations are performed by an event-triggered scheme (ETS), instead of periodic execution [13]. An ETS design mainly focuses on optimizing the network utilization in the context of networked control systems (NCSs) where the network is shared among a number of applications [14]. The ETS design techniques can be categorized based on the nature of their triggering schemes as continuous ETS (CETS) [15–17], Periodic ETS (PETS) [18, 19], or Self triggered scheme (STS) [20,21].

While the ETS approaches tackle resource utilization in general, they mostly focus on reducing communication bandwidth usage, while offering a desired control performance. First, in this paper, we focus on the efficient utilization of the processing resource which is not investigated in the context of ETS. More importantly, ETS does not reduce resource usage in the transient state, which is addressed in this study by using our proposed multi-rate scheme. Second, the assumption of an aperiodic computation and availability of sensing/actuating signals in ETS is not applicable to many real-life platforms (including CompSOC) due to restrictions imposed by scheduling policy used in the operating system (OS).

Optimal sampling scheme: This direction of research focuses on deriving a sequence of optimal sampling periods that minimizes a predefined cost function. In these studies, in contrast to periodic sampling schemes, the sampling time instances $t_0, t_1, \dots, t_N$ are not equidistant, and hence, the sampling periods $\tau_k = t_k - t_{k-1}$ might have different values over different samples. These studies mainly focus on finding an optimal sequence of τ_k aiming to minimize a cost function based on the system input and states. In [22], a solution for optimal sampling periods is proposed for linear systems considering a scenario with no disturbance. While the sampling intervals can be any arbitrary solution in [22], the study [11,23] propose an optimal sequence of sampling periods which are chosen from finite set of possible values imposed by the operating system limitations. While works reported in [11,22,23] find the optimal sequence of sampling period in design time, [24] investigates the online optimal sampling period for the control application.

In general, a similar approach can be considered to choose the sampling periods for the transient and steady state in our study. However, the state-of-the-art approaches either do not consider the model uncertainty (to facilitate the optimization problem) or does not consider the platform limitation in realizing possible sampling periods. In comparison to the approach in [23], our approach addresses a broader range of platforms and chooses the sampling periods based on the output states respecting the scheduling restrictions of the platform. This is particularly relevant for real-world implementations.

Efficient processor utilization: The researches on the processing resource efficiency can be classified into single-core and multi-core analysis. The analysis techniques targeting single-core systems focus on the inter-application interference analysis. A body of work focuses on proposing a scheduling technique which guarantees the worst-case execution time (WCET) of the control application [25]. Another line of research tries to model the execution jitter of the application caused by inter-application interference or cache memory utilization [26]. Research in the direction of the weakly hard real-time model considers the possibility of missing some occasional deadlines in the control application [27,28]. Such studies are relevant for an implementation where the platform cannot guarantee the controller's predictability. Therefore, the delay from executing other applications on the platform might violate the periodic execution of the controller [29]. Accordingly, [30] proposes a deadline-miss aware controller (DMAC). DMAC is a stabilizing controller which is robust against deadline misses and is optimal concerning the chance of missing important information. In a similar study, [31] proposes a sporadic (instead of periodic) execution of the controller. In [31], an adaptive controller is proposed that guarantees system stability in the presence of sporadic overruns by other applications. Generally, the related work targeting multi-core systems focuses on the effect of shared components (such as shared memory and network-on-chip) on the timing behavior of the control application [32].

Our work contributes to an efficient single-core multi-application implementation by exploiting multiple control modes on a platform that allows for a fast and predictable switching between the modes. We explicitly consider the composable and predicable platforms since such platforms ensure deterministic executions easing the worst-case and schedulability analysis, which poses different set of challenges.

Table 2
Execution time of control operations in clock cycles.

Contributions	[11,23]	[34]	[6]	[12]	This work
Multi-rate in transient state	–	✓	–	–	✓
Resource-efficient switching scheme	✓	–	✓	✓	✓
State-based scheduling	–	–	–	✓	✓
Model-uncertainty consideration	–	✓	–	–	✓
Evaluation framework	HIL	SIL	HIL	SIL	HIL

Multi-rate and switching control: There has been an extensive literature in control theory on multi-rate and switched controllers. The literature in this direction mostly focus on the stability analysis of the multi-rate [7,8,33–35] and/or switching systems [36,37]. In general, these studies propose theoretical tools such as switched Lyapunov functions (SLF) to guarantee the closed-loop stability of the switching system with an arbitrary switching behavior. Building on the results of such studies (specially [34] and the studies on SLFs), our study focuses on the implementation aspects of such controllers on a predictable and composable platform. We specifically study the effect of the implementation on the control performance and the switching behavior. Along this line of research, a dual-mode strategy is proposed using a state-based switching scheme in [12]. However, this work does not look into aspects of resource utilization in the transient state. They use a FSR controller which, as demonstrated in our results, utilizes more resource compared to the multi-rate controller proposed in our paper.

Resource-aware designs: Another body of work focuses on considering the platform model and its temporal behavior in control design. The effect of sensor-to-actuator delay [38], which is an artifact of the platform implementation, is considered through the controller design. In [39], a multi-core implementation is proposed for iterative learning controllers (ILCs) where the multi-core nature of the targeted platform is utilized to reduce the sensor-to-actuator delay, resulting a higher control performance. However, while studying sensor-to-actuator delay in a multi-core implementation is relevant for resource-efficiency reasons, these approaches are orthogonal to the resource-utilization aspects using multi-rate control scheme which is the focus of our work.

In [6], a non-uniform sampling technique is implemented on a composable and predictable platform. In this study, an effective utilization of the allocated platform in the form of a switching system is studied. However, compared to our work, the proposed method in [6] is limited to platform properties such as limited scheduling options. It also utilizes a single-rate scheme for both the transient and steady state which results in a higher resource utilization compared to SSMC proposed in our paper.

Our method proposes a state-based switching scheme which can be seen as a combination of multi-rate controllers in transient state, and a ETS on its switching scheme. The main purpose of our study is to propose an resource-efficient scheduling of control application on embedded platforms while maintaining the control performance at the same level with traditional periodic and single-rate controllers. As explained, our work is closely related to the works reported in [6,12,23,34]. Table 2 provides an overview of the major contributions of our work compared to these specific related works.

3. Predictable and composable platform

In this section, we first outline the requirements of implementing the proposed SSMC scheme on the platform. Then, we translate these requirements to platform properties which are essential to implement the SSMC. Finally, after comparing the possible platforms, we describe the chosen platform CompSOC and demonstrate how the platform meets all of the requirements.

3.1. SSMC implementation requirements

The SSMC scheme uses a set of controllers (i.e., FSR, multi-rate, SSR) one of which is engaged based on the system state at any given point in time. Therefore, the SSMC scheme requires platforms that allow for periodic execution and event-triggered switching between the pre-defined set of controllers.

- **Strictly periodic start time of the sensing operations in each mode:** The interval between start time of two consecutive sensing operations is called the sampling period. For implementing a precisely periodic sampling period, it is required to implement strictly periodic activation of the sensing operations.
- **Predictable execution of the computing operations:** The computing operation should be executed in a predictable fashion, allowing to perform precise and tight worst-case response time analysis.
- **Deterministic execution of the actuating operations:** The interval between the start of the sensing operation and the end of the actuating operation is called the sensor-to-actuator delay which should be constant. Therefore, the actuating operations should be finished strictly periodically. This further implies the requirement of deterministic execution of the actuating operations.
- **Online switching between the controllers:** The SSMC scheme requires to switch the controllers online. This implies that the targeted platform should be able to re-schedule the execution of the applications at run-time in a predictable and bounded amount of time. Therefore, a part of the processing resource may be dynamically allocated or unallocated to an application.

These requirements hold for any model-based development of a control application; e.g., in state-of-the-art approaches listed in Table 2. In these studies, the requirements are either assumed to hold implicitly in SIL evaluations or taken into account in HIL evaluations.

3.2. Platform properties

The aforementioned properties can be translated to platform properties as follows.

Composability: A composable platform can execute multiple applications on a single core (or possibly multiple cores) independently by using a specific scheduling mechanism to ensure temporal isolation between the applications. One way to achieve this is to use *virtual execution platforms* (VEPs) to allow an independent design, implementation, and execution of the applications [4]. For a control application, this translates to an interference-free execution that further meets the requirement on strictly periodic activation of the sensing operations.

Another direct implication of composability is the ability to tackle mixed-critically applications [4]. This means that different applications of non-real-time, soft and hard real-time requirements can be implemented and be independently executed within the same platform. This is important for SSMC implementation since a part of freed up resource resulted from the SSMC scheme is used to execute non-real-time applications.

Predictability: In general, predictability means that applications implemented on the platform have performance bounds such as a worst-case response time. The knowledge of precise and tight worst-case response time enables to meet the requirement on execution of the compute operations.

Online reconfiguration: The online switching of the SSMC scheme requires the platform to reschedule the application execution at run-time (we will explain the switching scheme in detail in Section 5.2). Such reconfiguration must have specific properties to realize the online switching proposed in the SSMC scheme.

- The switching in SSMC is event-triggered. That is, the platform should reconfigure the application schedule at the occurrence of an event (e.g., state).
- The platform should not pause or interrupt the existing running applications while performing the reconfiguration. Any pause/ interruption in execution may violate the periodic execution of the SSMC scheme.
- The reconfiguration should have a short WCET which ideally is shorter than the sampling period of the SSMC scheme. A long WCET implies that the switching happens later than the expected time, which might cause performance degradation of the SSMC scheme.

3.3. Possible embedded platforms

Among the embedded platforms available in the literature and the industry, some platforms have similar properties as required for our proposed SSMC scheme.

T-CREST: T-CREST is a predictable multi-core platform that targets the safety-critical application [40]. This platform aims to optimize the WCET of the application by proposing a predictable hardware architecture. They provide more efficient resource utilization and improved WCET by a multi-core implementation. However, since T-CREST is not composable, it does not guarantee an interference-free execution in multi-application scenarios.

Time-triggered architecture (TTA): Time-triggered architectures and embedded platforms are popular for real-time applications [41]. Unlike event-triggered architectures, TTAs execute the applications (or their tasks) by following a predetermined schedule. The scheduler invokes the execution of a single or a group of tasks sequentially and repeats the execution order periodically. The only source of preemption in TTA is assumed to be the scheduler.

While a TTA might be a necessary condition of a composable platform, it is not sufficient. For example, in TTC (time-triggered co-operative), the implemented applications are scheduled in groups. The applications within a group run sequentially, and their WCETs are interfere with each other. Therefore, the application isolation requires further consideration [42]. Moreover, the scheduler is assumed to run a periodic sequence of applications indefinitely. Therefore, the platform does not support the time-triggered switching of SSMC, where the sequence of execution changes based on the system state.

FlexPRET: FlexPRET is a precision-timed machine [43]. It is a multi-threaded processor which targets mixed-critical implementations. The applications running on the platform are either hard-real-time threads (HRTT) or soft-real-time threads (SRTT). FlexPRET uses a thread scheduler that offers temporal and hardware isolation for HRTTs, and efficient implementation for SRTTs. The scheduler interleaves the threads so that all the HRTTs meet their strictly periodic execution. At the same time, SRTTs utilize all the cycles that the processor is idle using a round-robin schedule.

FlexPRET is a suitable implementation target for SSMC. The temporal and hardware isolation offered by FlexPRET ensures the strictly periodic execution of SSMC operations as HRTTs. However, whether the reconfiguration process is predictable or composable is not specified in this platform.

PTIDES: PTIDES is a programming model for cyber–physical systems [44]. PTIDES is a special implementation of a discrete-event model of computation. In PTIDES, every hardware component (such as sensors and actuators) and applications (such as control) are actors which communicate through time-stamped events. PTIDES considers all applications as hard-real-time events and schedules them on the processor following earliest-deadline-first (EDF) scheduling method.

Table 3
Platforms comparison.

Requirements	T-CREST	TTA	FlexPRET	CompSOC
Strictly periodic execution	✓	–	✓	✓
Temporal isolation	–	✓	✓	✓
Jitter-free execution	✓	–	✓	✓
Online reconfiguration	–	–	✓	✓

PTIDES alone is not a complete platform and depends on the availability of an operating system and a hardware implementation. Currently, PTIDES uses PtidyOS as a real-time OS to perform such scheduling, which requires further hardware implementation. Therefore, comparing PTIDES with other platform is plausible only if the properties of PtidyOS and hardware be considered in the comparison.

Table 3 summarizes the comparison between the studied platforms and their ability to satisfy our requirements.

3.4. Compsoc

CompSOC is a tile-based embedded platform of processor tiles, local and shared memories, and interconnections.

Hardware: The platform's tile-based nature allows for an adaptation of hardware architecture (e.g., number of tiles and size of shared memory footprint). We focus on a commercial instance of CompSoC called Verintec [verintec.com] for the hardware architecture, where the platform has two MicroBlaze processor tiles connected through shared memory. Verintec is an FPGA-based platform, which we synthesized on a Xilinx PYNQ-Z2 FPGA board [www.tul.com.tw]. The PYNQ-Z2 is a low-cost research hardware and offers a limited memory capacity (16-64 KB) and processing resource (3 processing tiles with a maximum of 3 applications per tile). While this is sufficient for the application under consideration in this paper, the Verintec platform has also been implemented on industrial FPGA boards such as Zynq UltraScale+ ZCU106. Such instances allow more processor tiles and/or larger memories, operating at a higher frequency. SSMC can target any Verintec instance that is mapped on an FPGA board that has sufficient capacity for the use case at hand.

Operating system: In CompSOC, resources are space- and time-partitioned to create Virtual Execution Platforms (VEPs) in which applications run independently. The platform can execute multiple VEPs on each processor tile using a predictable micro-kernel (CoMik). CoMik uses a strict time-division-multiplexing (TDM) policy on processor tiles, resulting in cycle-accurate slots. These slots allow multiple applications to execute in their independent VEPs, following the TDM schedule. Since the platform is predictable and composable, it can schedule the applications perfectly periodically. Each cycle of TDM is the execution of a table of N partition slots, which can have different lengths in clock cycles, defined by ψ_i.

The scheduler separates partition slots by N CoMik slots, which have a constant length of $\omega = 4096$ clock cycles. The CoMik slots are responsible for a jitter-free context switching between the partition slots. Towards this, a CoMik slot pauses the execution of the previous VEP and stores a snapshot of all the VEP memory information and register values. It then loads the previously-stored snapshot of the next VEP and resumes its execution [4]. Each of the VEPs is allocated to (possibly) multiple partition slots on (possibly multiple) processors. The VEPs are swapped in and out periodically and transparently by CoMik. Applications run in VEPs as if they run on a bare machine.

Each pair of processor tiles can communicate through a dedicated shared memory. The communication with the shared memory guarantees a specific atomic data size. The processors either use flag-based communications or the implemented FIFOs to prevent data overwrite. Fig. 3 represents a possible instance of the platform with two soft-core MicroBlaze processor tiles. The TDM scheduling of Processor Tile 1 illustrates a possible TDM table with three partition slots (of possibly different sizes) allocated to three different VEPs.

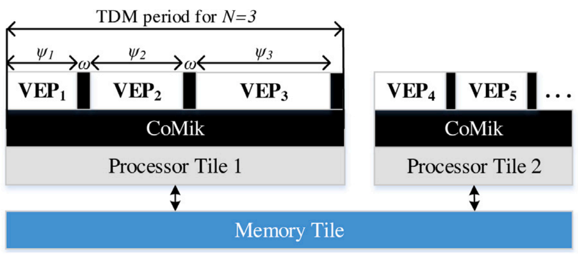


Fig. 3. Predictable embedded platform under consideration.

Online reconfiguration: CompSOC allows changing the TDM properties such as size and number of partition slots at run-time. Therefore, the platforms can perform various types of online reconfiguration. The reconfiguration of SSMC switches between different schedules represented by their TDM tables. Fig. 11 (which is discussed in detail in Sections 5 and 6) depicts these schedules. The switching in SSMC is event-triggered, and the computing operation (C_0 in Fig. 11) requests the switching based on the system state.

A specific application *reconfiguration manager* (RM) handles the reconfiguration requests. This application can be scheduled as a VEP in the TDM table and possibly in a different processor than the one that runs SSMC. Since CompSOC is composable, the execution of VEPs in TDM does not experience pause or interference during the reconfiguration. When a VEP requests a reconfiguration, RM prepares the corresponding TDM table and sends it to CoMik. During the TDM table preparation, the VEPs keep running on the platform following the current TDM table. When CoMik receives the new TDM table from RM, it waits until the current TDM table execution ends. It is also possible to delay the switching for more than one TDM table execution. CoMik then switches to the new TDM table without any pause in the execution. The next cycle of the TDM follows the new TDM table, and the VEPs continue executing in their dedicated partition slots, following the new schedule.

The reconfiguration in CompSOC is predictable and thus has a WCET. The WCET depends on whether the switching of TDM tables happens at pre-defined time instants decided at design-time or time instants decided at run-time. SSMC switches between three TDM tables defined in design-time alongside their corresponding controllers. Therefore, these TDM tables are initially prepared and saved in RM. When SSMC requests a switching, RM only sends the corresponding TDM table to CoMik.

Reconfiguration overhead: Depending on the TDM table size, the reconfiguration takes one or more TDM cycles. For a typical application with the required SSMC properties, the WCET of the reconfiguration process on CompSOC is 13 ms. Fig. 4 demonstrates two examples. RM is implemented on a separate core in these examples. We assume that the RM tile TDM has only one partition slot dedicated to RM, i.e., the RM always runs on its dedicated processor.

In Fig. 4. I, the TDM table has 5 partition slots and partition slots $1, 3, 5$ are allocated to VEP$_1$. VEP$_1$ request a reconfiguration in the TDM in ith TDM iteration. The request is to remove the fifth partition slot, allocate the third partition slot to VEP$_5$, and re-size the partition slots. The RM receives the request through the shared memory and invokes the reconfiguration before the current TDM cycle finishes. In this case, the reconfiguration takes one TDM cycle and the updated TDM is executed in $(i+1)^{th}$ TDM iteration.

In Fig. 4. II, the TDM table has 4 partition slots and partition slots 1,3 are allocated to VEP$_1$. VEP$_1$ requests a reconfiguration in the TDM, asking to allocate the third partition slot to VEP$_5$. This time RM invokes the reconfiguration when the current TDM has finished. In this case, the reconfiguration takes two TDM cycles and the updated TDM is executed in $(i+2)^{th}$ TDM iteration.

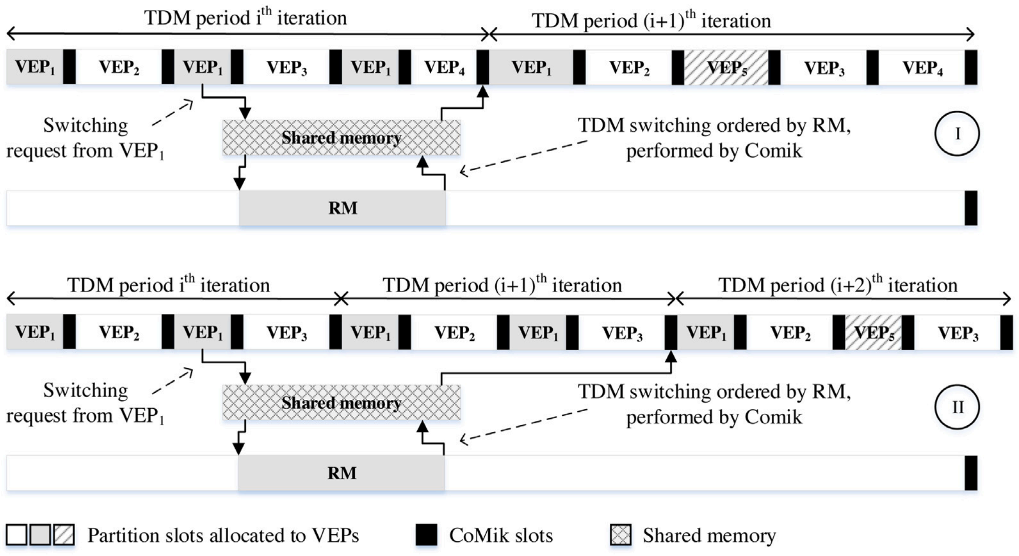

TDM period i^th iteration
TDM period (i+1)^th iteration
VEP1
VEP2
VEP3
VEP4
VEP5
Switching request from VEP1
Shared memory
TDM switching ordered by RM, performed by Comik
RM
I
TDM period (i+2)^th iteration
II
Partition slots allocated to VEPs
CoMik slots
Shared memory

Fig. 4. Two examples of online reconfiguration in CompSOC. In the first example (I on top), reconfiguration process is finished before the current TDM cycle finishes, without any TDM cycle delay. In the second example (II on bottom), the reconfiguration process is finished after the current TDM cycle finished and the reconfiguration is performed with a TDM cycle delay.

4. System architecture

4.1. Embedded control applications

We are interested in the implementation of control applications. We consider a controllable linear time-invariant (LTI) system. The state-space of the system is given by,

$$\dot{X}(t) = AX(t) + BU(t) + W(t),\\ Y(t) = CX(t), \tag{2}$$

where $X(t)$ is the system state vector, $U(t)$ is the input, $Y(t)$ is the output of the system, and $W(t)$ denotes any uncertainties in the system such as input disturbance and model uncertainty. A control application consists of three main operations executed sequentially and periodically — sensing, computing, and actuating. In the sensing operation, system sensors read the states of the system $X(t)$ at the time instances t_k where,

$$x[k] := X(t_k), \quad k \in N_{\geq 1}. \tag{3}$$

In the computing operation, the controller calculates the control values $u[k]$ at t_k. In the actuating, system actuators update the control value $u[k]$ of the system. We define the time between two consecutive sensing operations (which is equal to the time between two consecutive actuating operations in a single-rate system) as sampling period h. The assumption of strictly periodic execution means that h must be kept constant in the implementation. One way to implement a constant/uniform h is to design a TDM table of the length h and execute the control operations sequentially once per execution of the TDM table. Suppose a TDM table has N partition slots. In this case, the length of the TDM table in clock cycles is equal to the sum of the length of all the partition slots and their corresponding CoMik slots. Therefore h in *seconds* is equal to,

$$h = (N \times \omega + \sum_{i=1}^{N} \psi_i)/F_p, \tag{4}$$

where $F_p = 100$ MHz is the operating frequency of the platform. Considering the sampling period h, the model Eq. (2) is discretized as:

$$x[k+1] = A_d x[k] + B_d u[k] + W[k],\\ y[k] = Cx[k], \tag{5}$$

where,

$$A_d(h) = e^{Ah}, \quad B_d(h) = \int_0^h e^{As} B ds.$$

The platform maps each of the control operations to one of the VEPs, to implement the control application. The remaining resource can be assigned to the applications other than the control application. The platform allows for partition slots of any size [4]. Therefore, we define the size of VEPs to be either equal or slightly bigger than the WCET of their corresponding operation, defined as S, C, and A for sensing, computing and actuating, respectively.

Control performance: To quantify the performance of the control application, we use settling time as the performance metric. The settling time is the required time for the application to reach the steady state. That is, we define settling time t_s as the time that the system output requires to start from an initial state $y[0]$ to reach and stay within 2% bound of the desired output Y_d. Since shorter settling time translates to higher control performance, we define the performance metric *the quality of control* (QoC) as $QoC = {t_s}^{-1}$.

State-feedback control: Without losing generality, we opt for a state-feedback controller of the following form:

$$u[k] = K \times x[k] + F \times r[k], \tag{6}$$

where K is the state-feedback gain and F is the feedforward gain. The feedback gain K is designed using a *pole placement* technique, placing the poles of the closed-loop system at desired locations. The static feedforward gain F is designed to make the system output $y[k]$ follow the desired reference $r[k]$. If the system under control is stable, it means that in the steady state $x[k+1] = x[k]$. Substituting Eq. (6) in Eq. (2) and assuming the steady state has been reached ($x[k+1] = x[k]$) we have:

$$F = \frac{1}{C_d(I - A_d - B_d K)^{-1} B_d}. \tag{7}$$

Any of the state-of-the-art design techniques can replace the chosen controller. However, considering a more complex control objective or a design technique is orthogonal to the control application's implementation aspects. We illustrate our proposed implementation technique considering a representative control application.

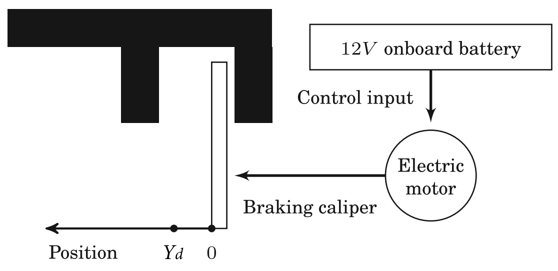


Fig. 5. Schematics of the braking system [23].

4.2. Case-study: electro-mechanical braking

We study the electro-mechanical braking (EMB) system for automobiles [23]. Fig. 5 represents the schematics of this system. When the driver presses the brake paddle in this system, the braking lever position should reach a corresponding position Y_d within the desired settling time t_s. The control objective is to reach the desired reference output $r(t)$ within the shortest t_s possible while keeping the control input below a predefined bound $|u[k]| \leq u_{max}$. In our case study, the desired reference output is Eq. (1) with $Y_d = 2$ mm. The maximum control input is $u_{max} = 12$ V, equal to the maximum possible voltage to apply to the electric motor. It is assumed that $t_s < T_r$ which means the controller reacts quicker than the reference changes. Eq. (8) presents the system dynamics.

$$\dot{X}(t) = \begin{bmatrix} -520 & -220 & 0 & 0 & 0 \\ 220 & -500 & -999994 & 0 & 2\times 10^8 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 66667 & -0.1667 & -1.3333\times 10^7 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} X(t) + \begin{bmatrix} 1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} U(t), \quad y(t) = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} X(t). \tag{8}$$

5. State-base switching controller

This section demonstrates the effect of the sampling period on resource utilization and QoC by comparing two SSR and FSR controllers. Next, from the comparison results, we propose the state-based switching scheme.

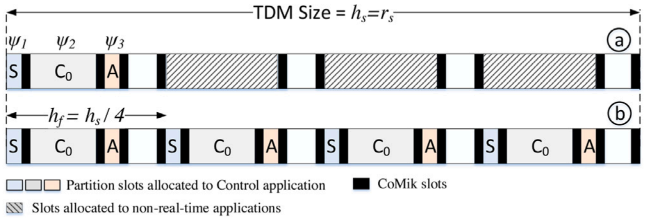
5.1. Resource utilization vs QoC in single-rate controllers

The resource utilization of an application means the amount of processing used by the application. In CompSOC, since the platform periodically executes the TDM table, the resource utilization of an application is defined based on the partition slots plus the CoMik slots allocated to it in the TDM. We recall ω and ψ_i as the size of the CoMik slot and the ith partition slot respectively. Therefore, the resource utilization is defined as:

$$R = \frac{l \times \omega + \sum_{i=1}^{l} \psi_i}{N \times \omega + \sum_{i=1}^{N} \psi_i} \tag{9}$$

where l is the number of partition slots allocated to an application, and N is the number of partition slots in the TDM table.

The resource utilization of a control application depends on the chosen sampling period (or the TDM size). Let us consider two control systems of the SSR controller with h_s and the FSR controller with $h_f = h_s/4$. Fig. 6.a and Fig. 6.b demonstrate the TDM schedule of these two systems, respectively. To make the controllers comparable, we keep the size of the TDM as h_s for FSR and run the controller 4 times in the TDM table instead. Referring to the TDM schedules, SSR and FSR utilize 3 out of 10 and 12 out of 16 sets of partition and CoMik slots in the TDM, respectively. While partition slots have different sizes in the two schedules, The size of the TDM table is the same and is equal to h_s. Using Eq. (9) the resource utilization of these systems are:



TDM Size = h_s=r_s
ψ_1
ψ_2
ψ_3
a
S
C0
A
h_f= h_s / 4
b
Partition slots allocated to Control application
CoMik slots
Slots allocated to non-real-time applications

Fig. 6. TDM schedule of (a) single-rate with $h = h_s$, and (b) single-rate with $h = h_f$. S, C_0, and A are the sensing, computing, and actuating operations. The non-real-time application executes on hashed pattern partition slots when the system is in the steady state.

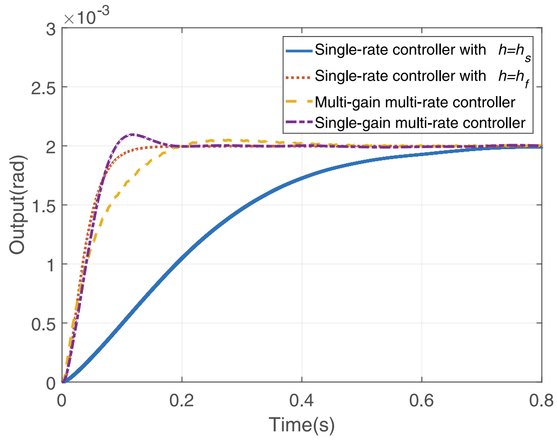


Fig. 7. The output response of the braking system considering four different controllers.

$$R_s = \frac{3\times\omega + \sum_{i=1}^{3}\psi_i}{10\times\omega + \sum_{i=1}^{10}\psi_i}, \quad R_f = \frac{12\times\omega + 4\times\sum_{j=1}^{3}\psi_j}{16\times\omega + \sum_{j=1}^{16}\psi_j} = 4\times R_s \tag{10}$$

where R_s and R_f are resource utilization of SSR and FSR respectively. ψ_i and ψ_j denote the size of partition slots of SSR and FSR scheduling respectively. Here ψ_1, ψ_2, ψ_3 are allocated to S, C_0 and A which are the sensing, computing and actuating operations respectively. As expected, there is an inverse relationship between R and h, where a smaller h results in higher resource utilization.

To compare the QoC for these two control systems, we perform model-in-the-loop (MIL) simulations on the representative braking system described in Section 4.2. Since we focus on resource utilization here, we can assume that the system model does not have uncertainties and $W[k] = 0$ for all k values. Note that we will consider the model uncertainty in multi-rate design.

Fig. 7 shows the response of the controllers with $h_s = 10$ ms and $h_f = 2$ ms. In both simulations, the feedback gain K in (6) is designed to place the closed-loop poles at $[0, 0, 0, 0.9, 0.9]$ (Similar design as [11]). Considering the defined QoC in Section 4.1, we obtain Table 4, which indicates that the FSR controller achieves a shorter settling time t_s and thus a higher QoC.

5.2. The switching scheme

Comparing the resource utilization and QoC of FSR and SSR, we observe:

Table 4
Resource utilization and QoC of single-rate controllers.

Sampling period	Resource utilization (%)	QoC ([s]$^{-1}$)
h_s	$R_s = 1.3\%$	1.77
h_f	$R_f = 4 \times R_s = 5.2\%$	8.23

- **Resource utilization**: As shown in Table 4, FSR has a higher resource utilization than SSR. The higher resource utilization results from the higher frequency of execution of the control application (4 times higher in the case-study example) requiring higher computing resources.
- **Steady state response**: The FSR controller offers a shorter transient state and thus higher QoC than SSR (as can be seen in Fig. 7). However, both controllers give similar responses in terms of keeping the output within a 2% range of $r(t)$ in the steady state. Therefore, using either of the controllers in the steady state would result in a similar response.

Hence, we propose a scheme that uses the FSR controller in the transient state and switches to the SSR controller in the steady state. Our proposed scheme would result in a similar QoC to that using only the FSR controller in both states. At the same time, the scheme would demand a lower resource utilization in the steady state. The freed-up resource in the steady state can be allocated to other platform applications. We define this as the state-based switching scheme (SSC).

SSC design: The SSC scheme consists of an FSR controller in the transient and an SSR controller in the steady state. By assuming that both FSR and SSR controllers are state-feedback controllers defined in Eq. (6), the SSC can be modeled as:

$$x[k+1] = \begin{cases} \left(A_d(h_f) + B_d(h_f)K_f\right) x[k] + F_f r[k], & \text{if FSR is active} \\ \\ \left(A_d(h_s) + B_d(h_s)K_s\right) x[k] + F_s r[k], & \text{if SSR is active} \end{cases} \tag{11}$$

where $\{A(h_f), B(h_f)\}$ and $\{A(h_f), B(h_f)\}$ are defined in Eq. (5) with $h = h_f$ and $h = h_s$ respectively. Since the SSC scheme includes switching between FSR and SSR, K_f and K_s cannot be designed independently. Even with a stable design of K_f, K_s (for example, by a proper *pole placement* technique), the stability of the overall switching system requires a separate analysis. Therefore, a stable design must guarantee the stability of the transient and the steady state and the state switching simultaneously. The stability analysis and designing of K_f, K_s are described in the following.

Switching Stability and control design: SSC can be modeled as a switching system, where the system dynamics switches between two subsystems modeled by Eq. (5) with $h = h_f$ and $h = h_s$ for FSR and SSR, respectively. We model the two switching subsystems as:

$$x[k+1] = A_k x[k] + Fr[k], \tag{12}$$

where $A_k = A(h_k) + B(h_k)K_k, \quad \forall h_k \in \{h_f, h_s\}$.

Theorem 5.1 ([45,46]). *Consider A_k are LTI discrete-time switching subsystems defined in Eq. (12). $V(x) = x^T Px$ is the common quadratic Lyapunov function (CQLF) of the subsystems A_k if there exist $P = P^T > 0$ and $Q = Q^T > 0$ that satisfy the following matrix equations,*

$$A_k{}^T PA_k, -P = -Q < 0. \tag{13}$$

If such $V(x)$ exists, then the switching system described in Eq. (12) is stable.

The designed controllers for the subsystems should guarantee the existence of the CQLF function to ensure the stability of the overall switching system in Eq. (12). Towards this, we first design the controller K_f for $h_k = h_f$. We define the closed-loop system for $h_k = h_f$ as (the feedforward input is omitted since that does not influence system stability):

$$x[k+1] = A_{cl,f}\, x[k], \tag{14}$$

where $A_{cl,f} = (A(h_f) + B(h_f)K_f)$. To design the SSR controller K_s with $h_k = h_s$, we use the following theorem.

Theorem 5.2 ([47]). *Consider the switching subsystems defined in Eq. (12). If there exist a $Y = Y^T > 0$ and a Z such that the following linear matrix inequality (LMIs) hold,*

$$\begin{bmatrix} Y & YA(h_s)^T + Z^T B(h_s)^T \\ A(h_s)Y + B(h_s)Z & Y \end{bmatrix} > 0, \\ A_{cl,f}^{-1} Y - YA_{cl,f}^T > 0, \tag{15}$$

then the switching system has a CQLF with the following feedback gain for $h = h_s$:

$$K_s = ZY^{-1}. \tag{16}$$

Therefore, by designing the FSR feedback gain using the *pole placement* technique and the SSR feedback gain using Eq. (16) the switching stability of the SSC is ensured.

Resource Utilization: To analyze the SSC resource utilization, let us first summarize the sequence of the execution for two controllers. The FSR controller is active from $t = 0$ until $t = t_s$ when the system output reaches the steady state. At t_s, the controller switches to SSR and continues executing SSR as long as the system stays at the steady state. Suppose we assume a periodic reference as defined in Eq. (1). In that case, the same sequence of execution repeats for every $T_r/2$ where the reference value switches between Y_0 and Y_d. Therefore, we define resource utilization over a cycle of the reference considering a periodic reference as a plausible assumption. In many dynamic systems, the output switches between two or more pre-defined modes. For example, the periodic reference translates to pushing and releasing the brake pedal and changing the output between Y_0 and Y_d in our case study.

The resource utilization depends on the active duration of different controllers. The FSR is active in $0 \le t < t_s$ and $\frac{T_r}{2} \le t < \frac{T_r}{2} + t_s$, and SSR is active in $t_s \le t < \frac{T_r}{2}$ and $\frac{T_r}{2} + t_s \le t < T_r$. Therefore, we derive SSC resource utilization R_{SSC} as:

$$R_{SSC} = \frac{2t_s}{T_r} R_f + \frac{T_r - 2t_s}{T_r} R_s. \tag{17}$$

If we define $R_f = \gamma R_s, \gamma > 1$, then,

$$R_{SSC} = [1 - \frac{2(\gamma - 1)t_s}{T_r}]R_f < R_f, \tag{18}$$

which indicates that $R_{SSC} < R_f$, as long as $\gamma > 1$. Therefore, the SSC utilizes less resource while providing similar QoC as FSR, by utilizing SSR in the steady state.

Platform Implementation: The SSC switching implementation on CompSOC is an online reconfiguration process described in Section 3.4. When the system state changes from the transient state to the steady state, the SSC scheme requests a switching from the FSR to the SSR controller. The request happens in the computing operation. After the switch, the non-real-time application uses freed-up partition slots allocated to SSC. Fig. 8.a depicts this switching behavior. In this example, in the ith TDM cycle, the running FSR controller requests a switch. Since the size of the TDM cycle is $h_s = 10$ ms, and it is shorter than the WCET of RM (i.e. 13 ms), the switching from FSR to SSR takes two TDM cycles. Therefore, while RM performs the reconfiguration, the FSR keeps being the active controller in the $(i+1)^{th}$ TDM cycle. It is then switched to SSR in $(i+2)^{th}$ TDM cycle.

Similarly, when the system state changes from the steady state to the transient state, the SSC scheme requests a switching from the SSR to the FSR controller. After the switch, the non-real-time application stops executing, and its resource is reallocated to SSC. Fig. 8.b depicts

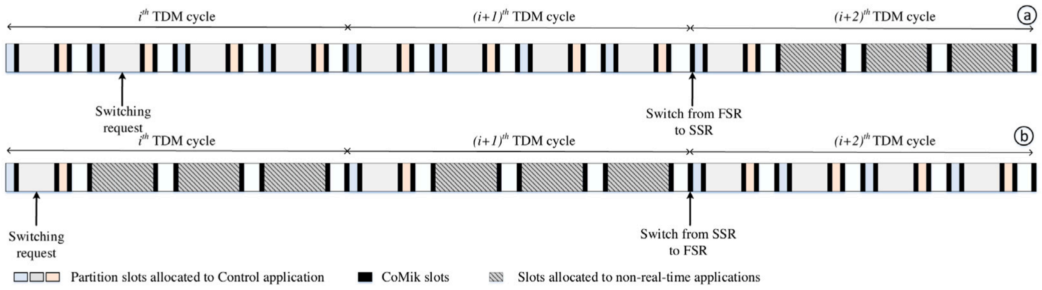


Fig. 8. SSC switching modes where (a) is a switching from the transient to steady state and (b) is a switching from the steady state to transient. The non-real-time application executes on hashed pattern partition slots when the SSC scheme is in the steady state.

this switching behavior where the switching from SSR to FSR takes two TDM cycles. Therefore, while SSR request switching in ith TDM cycle, the switching to FSR occurs in $(i+2)^{th}$ TDM cycle.

While SSC provides lower resource utilization in the steady state, it still uses the same amount of resources as FSR in the transient state. In the next section, we propose a multi-rate controller as a substitute to FSR in the transient state, which has resource utilization similar to SSR and achieves a QoC similar to FSR. The proposed state-based switching multi-rate scheme (SSMC) applies a multi-rate controller in the transient state and an SSR controller in the steady state.

6. State-based multi-rate switching scheme

In this section, we first describe the multi-rate controller, which is active in the transient state of the SSMC scheme. Next, we describe the switching scheme and the resource utilization in the SSMC scheme.

In this paper, a multi-rate controller translates to a control loop with a shorter actuating period than the sensing period. Such controllers use the information obtained from sensing to compute and actuate more than one control inputs. The motivation to use such a controller is to actuate the system with a shorter period than the sensing, which results in a lower resource utilization.

6.1. The multi-rate model

Let us define the actuating period as r_a. One sensing operation is performed every m actuating operations. We define the sensing period $r_s = m \times r_a$. In our embedded implementation, m represents the number of actuating per TDM period. We now define the following discrete-time model of Eq. (5) discretized at actuating period r_a between two consecutive measurements $x[k]$ and $x[k+1]$.

$$\begin{bmatrix} x(k,n+1) \\ w(k,n+1) \end{bmatrix} = \begin{bmatrix} A_f & B_f \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x(k,n) \\ w(k,n) \end{bmatrix} + \begin{bmatrix} B_f \\ 0 \end{bmatrix} u(k,n). \tag{19}$$

Here, A_f and B_f are discrete model parameters with the period of r_a obtained from Eq. (5), k is the index of sensing operation (output measurement), n is the index of the actuating operations (control inputs) and $n = 0, 1, \dots, m-1$ with $x[k] = x(k,0)$ and $x[k+1] = x(k,m)$. Also, $w(k,n)$ is a scalar quantity and represents the uncertainties resulted from any parametric uncertainties in the system model as well as input disturbance. Accordingly, $B_f w(k,n) = \Delta A_f x(k,n) + \Delta B_f u(k,n) + B_f \Delta u(k,n) + \Delta B_f \Delta u(k,n)$, where ΔA_f and ΔB_f are parametric uncertainties and $\Delta u(k,n)$ is added input disturbance.

6.2. The parallel observer system

The sensing information is only available once every m actuating operations. Also, the system might have model uncertainties, disturbances, or both. To address these two problems, we use a parallel observer system (POS) as a state estimation technique [10]. Fig. 9 provides an overview of the parallel system. The POS consists of two observers that run in parallel but with different sampling periods. The

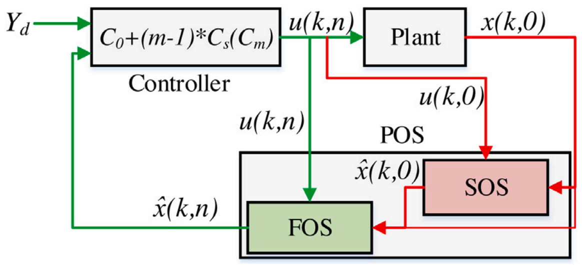


Fig. 9. The schematics of a system with a parallel observer.

slow observer system (SOS) runs at sensing period r_s and the fast observer system (FOS) runs at actuating period r_a.

The benefit of using the parallel observer instead of a single fast rate observer is dealing with the model uncertainty present in System Eq. (19). In the parallel observer system, the SOS functions as a filter to the FOS and provides stable full state order estimate to FOS. In this paper, we choose both SOS and FOS to be as Luenberger observers [10]. The dynamic model of the SOS is shown in Eq. (20):

$$\begin{aligned} \hat{x}_s[k+1] &= A_s \hat{x}_s[k] + B_s u[k] + B_s \hat{w}_s[k] + L_s C e_s[k], \\ \hat{w}_s[k+1] &= \hat{w}_s[k] + l_{sw} e_s[k], \\ \hat{y}_s[k] &= C \hat{x}_s[k], \end{aligned} \tag{20}$$

where A_s and B_s are the discretized state space matrices in Eq. (5) with the sampling period of $h = r_s$. $\hat{w}_s[k]$ is the SOS estimate of the scalar model disturbance, $e_s[k] = x[k] - \hat{x}_s[k]$, and L_s and l_{sw} are the Luenberger gains for state and uncertainty respectively.

SOS estimates the system states of Eq. (19) during the samples when the sensing operation is performed (i.e., $n = 0$). The output of SOS are the estimated states $\hat{x}_s(k) = \hat{x}(k,0)$. These estimated states are fed to the fast observer to estimate the system states $\hat{x}_f(k,n)$ at the points where the sensing information is not available ($n \neq 0$). These estimates are defined as $\hat{x}_f(k,n) = \hat{x}(k,n)$. The dynamic model of the FOS is:

$$\begin{aligned} \hat{x}_f(k,n+1) = {} & A_f \hat{x}_f(k,n) + B_f u(k,n) + B_f \hat{w}_f(k,n) + L_f e_{sf}(k,n) \\ & + L_{fw} e_{wsf}(k,n), \end{aligned} \tag{21a}$$

$$\hat{w}_f(k,n+1) = \hat{w}_f(k,n) + l_f e_{sf}(k,n) + l_w e_{wsf}(k,n), \tag{21b}$$

$$e_{sf}(k,n) = \begin{cases} \hat{x}_s[k] - \hat{x}_f[k], & \text{if } n = 0 \\ 0, & \text{if } n \neq 0 \end{cases} \tag{21c}$$

$$e_{wsf}(k,n) = \begin{cases} \hat{w}_s[k] - \hat{w}_f[k], & \text{if } n = 0 \\ 0, & \text{if } n \neq 0 \end{cases} \tag{21d}$$

where A_f and B_f are the discretized state space matrices in Eq. (5) with the sampling period of $h = r_f$. $\hat{w}_f(k,n)$ is the FOS estimate of the scalar model disturbance, and L_f, L_{fw}, l_f, and l_w are the Luenberger gains. Eq. ((21)a) and ((21)b) represent the dynamics of the state estimates $\hat{x}_f(k,n)$ and the disturbance estimates $\hat{w}_f(k,n)$ respectively. By including $e_{sf}(k,n)$ and $e_{wsf}(k,n)$ in FOS dynamics, the SOS estimated

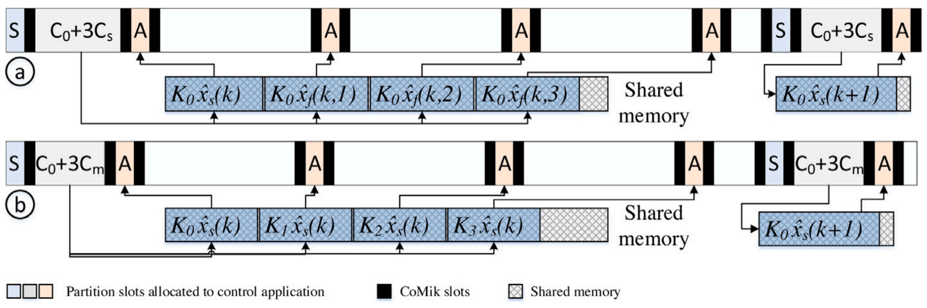


Fig. 10. Overall timing of (a) SMR controller, and (b) MMR controller for $m = 4$.

states $\hat{x}_s(k)$ are fed to FOS. Accordingly, $e_{sf}(k,n)$ and $e_{wsf}(k,n)$ are the error between SOS and FOS estimates of system states and model disturbance. These errors are only calculated at the points where SOS estimates are available ($n = 0$) and are assumed zero otherwise ($n \neq 0$).

6.3. Single-gain multi-rate scheme

The single-gain multi-rate (SMR) controller uses observer output in the computing operation. The computing operation is divided into two parts. The first part is denoted by C_0, where the controller calculates the first input value as $u(k,0) = K_0\hat{x}_s(k) + Fr(k)$ using the output of SOS $\hat{x}_s(k)$ (K_0 is designed using control law in Eq. (6) realizing the desired closed-loop poles). The second part is denoted by C_s, where the controller calculates the set of input values $u(k,n)$ for $n = 1,\dots,m-1$ for future actuating points. The control inputs are calculated as follows:

$$u(k,n) = K_0\hat{x}_f(k,n) + Fr(k,n), \tag{22}$$

using the output of FOS $\hat{x}_f(k,n)$. Fig. 10.a illustrates the overall timing of the SMR for $m = 4$.

From the implementation perspective, the computing of the SMR controller sequentially performs state estimation and control input calculation.

6.4. Multi-gain multi-rate scheme

In multi-gain multi-rate (MMR) controller, instead of using $\hat{x}_f(k,n)$, the controller calculates the input using $\hat{x}_s(k)$ and a time-lifted model as explained in the following. Similar to SMR, the MMR controller consists of two operations. C_0 calculates $u(k,0) = K_0\hat{x}_s(k) + Fr(k)$ using the output of SOS $\hat{x}_s(k)$ (K_0 is designed using control law in Eq. (6)). Using the derived $u(k,0)$ and system Eq. (5), one can define $\hat{x}(k,n)$ as:

$$\begin{aligned}\hat{x}(k,1) &= A_f\hat{x}_s(k) + B_f u(k,0) = A_f\hat{x}_s(k) + B_f K_0\hat{x}_s(k),\\ \hat{x}(k,2) &= A_f\hat{x}(k,1) + B_f u(k,1)\\ &= (A_fA_f + A_fB_fK_0)\hat{x}_s(k) + B_f u(k,1).\end{aligned} \tag{23}$$

by defining $A_1 = (A_fA_f + A_fB_fK_0)$, and by substituting this in Eq. (23) we have:

$$\hat{x}(k,2) = A_1\hat{x}_s(k) + B_f u(k,1). \tag{24}$$

Now, we can design the controller $u(k,1)$ for the new state-space model of Eq. (24) as follows:

$$u(k,1) = K_1\hat{x}_s(k) + Fr(k,1), \quad K_1 = LQR(A_1, B_f), \tag{25}$$

where $K_1 = LQR(A_1, B_f)$ is the feedback gain in control law in Eq. (6) for the state space of (A_1, B_f). This approach is repeated to calculate $u(k,n)$ for $n = 1,\dots,m-1$ as follows:

$$\begin{aligned}u(k,n) &= K_n\hat{x}_s(k) + Fr(k,n),\\ K_n &= LQR(A_n, B_f),\\ A_n &= (A_fA_{n-1} + A_fB_fK_{n-1}).\end{aligned} \tag{26}$$

The controller does not require FOS predictions $\hat{x}_f$. The calculation of $u(k,n)$ (defined as C_m for $n \neq 0$) is limited to the calculation of the $K_n\hat{x}_s(k)$ where K_n values are calculated offline and are provided to the controller before executing the control application. Fig. 10.b depicts the overall timing of MMR for $m = 4$.

From the implementation perspective, the MMR controller requires less computing effort than the SMR controller. In MMR, calculating K_n is done in design time, reducing the computation effort at run-time. Moreover, the MMR controller does not use FOS estimates and only uses SOS to estimate $\hat{x}_s(k)$.

6.5. POS initialization state

As mentioned in the introduction, the observer system requires a certain time interval to reach within a user-defined error threshold for estimation. To ensure this, the scheduling of the transient state would include an initial phase in time duration $0 \le t < T_0$ (indicated with the red color in Fig. 1) where the FSR is active alongside the POS until the observer error goes below the threshold. The time instance T_o is when the controller switches from FSR to multi-rate. For the stability analysis of the switching, as long as the LTI system under consideration and the POS system are separately stable, the separation principle ensures the overall system stability [48].

6.6. Performance analysis

We perform MIL simulations to compare the performance of the multi-rate and single-rate controllers. In MIL simulations, we modeled all the controllers (FSR,SSR, SMR, and MMR) in Simulink [49] model-based environment, and by giving a same input we compare the resulting output. Fig. 7 illustrates the results of a simulation without considering the model uncertainty, comparing both multi-rate controllers with two single-rate controllers. The sampling period for the single-rate controllers are $h_s = 10$ ms and $h_f = h_s/5 = 2$ ms. For both multi-rate controllers, the sensing period is $r_s = h_s$ and the actuating period is $r_a = h_f$. Both the SMR and the MMR perform better than the SSR with $h = h_s$. The SMR response is similar to the FSR with $h = h_f$ since both controllers follow the same control law.

6.7. SSMC scheduling and stability analysis

As described in Section 5, in the SSMC scheme, we substitute the FSR controller in the transient state with the multi-rate controller. The benefit of using SSMC over SSC is a lower resource utilization in the transient state. We would further analyze the resource utilization of SSMC in Section 7.

The stability of SSMC can be ensured by following a design and analysis similar to SSC. SSMC can be modeled as a switching system, where the system dynamics switches between two modes, which are modeled by Eq. (19) with $m = n$ and $m = 1$. The system with $m = n$ represents the multi-rate controller, and the system with $m = 1$ represents the SSR controller. Like SSC, we consider a system with the sampling period switching between the elements of the set $H = \{h_f, h_s\}$. The switching subsystems can be modeled as shown in Eq. (12). By designing the multi-rate using the proposed method in this section and by designing the SSR controller using *Theorem* 5.2, the stability of SSMC is ensured.

Like SSC, the switching behavior of SSMC is an online reconfiguration process described in Section 3.4. The switching of SSMC is invoked by the change in state (i.e., the steady state and the transient) and is implemented in the same way as described in Section 5.2.

6.8. SSMC resource utilization

To analyze the resource utilization of the SSMC, let us first summarize the sequence of the execution for controllers. Referring back to Fig. 1, the first scheduled controller is the FSR controller to help the observer reach below the predefined error threshold (as described in Section 6.5). Next, the controller switches to multi-rate in T_o until the system output reaches the steady state t_s. At t_s, the controller switches to SSR as long the system stays steady. If we assume a periodic reference as defined in Eq. (1), the described execution sequence only happens in the first period. From the second iteration onward, the execution sequence would only include the multi-rate controller in the transient state and the SSR controller in the steady state.

The resource utilization for the first period of the reference R_{SSMC_1} is the combined resource utilization of all active controllers based on their duration. Referring back to Fig. 1, FSR is active in the interval $0 \leq t < T_o$, the multi-rate is active in the interval $T_o \leq t < t_s$ and $\frac{T_r}{2} \leq t < \frac{T_r}{2} + t_s$, and SSR is active in the interval $t_s \leq t < \frac{T_r}{2}$ and $\frac{T_r}{2} + t_s \leq t < T_r$. Therefore,

$$R_{SSMC_1} = \frac{T_o}{T_r} R_f + \frac{2t_s - T_o}{T_r} R_m + \frac{T_r - 2t_s}{T_r} R_s, \tag{27}$$

where R_f and R_s are defined in Eq. (10), and R_m is resource utilization of multi-rate controller. Similarly, the resource utilization for the i_{th} period of the reference for $i = 2, 3, \dots$ is defined as:

$$R_{SSMC_i} = \frac{2t_s}{T_r} R_m + \frac{T_r - 2t_s}{T_r} R_s. \tag{28}$$

To further analyze the resource utilization of SSMC, we should calculate R_m. We continue with an embedded implementation technique of the multi-rate controllers and analyzing the resource utilization of SMR, MMR, and SSMC. We consider 5% modeling inaccuracy ($\Delta A_f = 0.05A_f, \Delta B_f = 0.05B_f$) compared to its nominal values in the state-space model shown in Eq. (5). We also consider no input noise which leads to $\Delta u[k, n] = 0$.

7. Embedded implementation and resource utilization

The proposed implementation for the multi-rate controllers is the execution of Algorithm ?? in every execution of the TDM table, following the scheduling demonstrated in Fig. 11.c. The figure is an example where the sensing period is $r_s = h_s$ and the actuating period is $r_a = h_f = h_s/4$.

The implementation must ensure a periodic execution for both sensing and actuating since the controller has different sensing and actuating periods. We ensure the periodic execution of the sensing by choosing the TDM table size equal to r_s. We ensure the periodic execution of the actuating by choosing the proper values for two parameters i_1 and i_2, which we define in the following.

We recall m as the number of actuations per sensing. The execution of the multi-rate controller depends on m. We describe this dependency per operation:

Sensing: The execution of sensing operation is independent of m since the number of sensing operations is always one per m actuating operation.

Computing: The computing operation calculates m control values in each execution. Referring to Section 6, the procedure of computing

Algorithm 1 Multi-rate control application

```
Sensing (S):
  Read the state values $x(k, 0)$
  Wait until the end of the partition slot
Computing (C):
  $n \leftarrow 0$
while $n < m$ do
  if $n = 0$ then
    Calculate $u(k, 0)$ by executing $C_0$
  else {$n \neq 0$}
    Calculate $u(k, n)$ by executing $C_s$ (or $C_m$)
  end if
  Update $u(k, n)$ in the shared memory
  $n \leftarrow n + 1$
end while
if Reach the steady state == True then
    Issue switching request
end if
  Wait until the end of the partition slot
Actuating (A):
Initialize $n \leftarrow 0$
  Update control input by $u(k, n)$
  $n \leftarrow n + 1$
if $n = m$ then
  $n \leftarrow 0$
end if
  Wait until the end of the partition slot
```

the control inputs is divided into two parts. We define T_{c_0} as the execution time of C_0 and T_{c_s} and T_{c_m} as the execution time of C_s and C_m respectively. Since the targeted platform offers jitter-free and constant execution times, T_{c_s} and T_{c_m} are constant for all the input values. Therefore the execution time of the whole computing T_c is given by:

$$T_c = T_{c_0} + (m - 1) \times T_{c_s} (or\ T_{c_m}). \tag{29}$$

Actuating: The execution time of actuating operation is constant. However, the operation is periodically executed m times over the TDM table. We realize this periodic execution by specifying the size of the partition slots in TDM. Let us define i_1 as the length of the partition slots between two consecutive actuating operation and i_2 as the length of the last partition slot in TDM (see Fig. 11.c). By assuming that for every control operation ψ_i is equal to the execution time of the corresponding operations, and by replacing T_c as in Eq. (29), we rewrite Eq. (4) as:

$$\begin{aligned} h_s &= T_s + T_{c_0} + (m - 1)T_{c_m} + mT_a \\ &+ (m - 1)i_1 + (2 + 2\ m)w + i_2. \end{aligned} \tag{30}$$

to ensure a periodic execution, the distance between two consecutive actuating must be the same. Therefore:

$$i_1 = T_s + T_{c_0} + (m - 1)T_{c_m} + 2w + i_2. \tag{31}$$

Using the realized Eqs. (30) and (31), the controller can be implemented with any desirable number of actuating per sensing as long as the resulted i_1 and i_2 from the scheduling are positive values. The number of actuating per sensing is mostly studied in respect of QoC in the presence of model uncertainty. The proposed method suggests that the available resource should also be considered alongside QoC to ensure a feasible implementation.

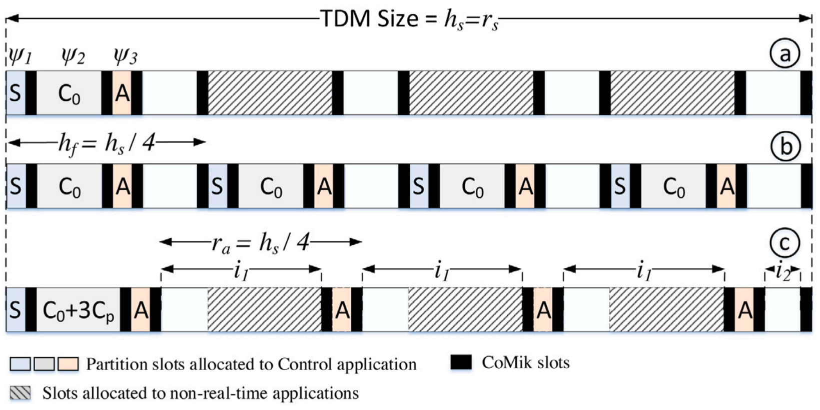


Fig. 11. TDM schedule of (a) single-rate with $h = h_s$, (b) single-rate with $h = h_f$, and (c) multi-rate with $m = 4$ and $r_s = h_s$. S, C_0, and A are the sensing, computing, and actuating operations. The non-real-time application executes on hashed pattern partition slots.

7.1. Multi-rate controller resource utilization

The resource utilization of the multi-rate controller R_m depends on m. By using Eqs. (9), and (30) we obtain:

$$R_m = \frac{T_s + T_{c_0} + (m-1)T_{c_m}(\text{or } T_{c_s}) + mT_a + (2+m)\omega}{h_s} \tag{32}$$

To compare multi-rate controllers with the single-rate controllers, we redefine R_s and R_f by referring to Eq. (10). We assume the partition slots size are equal to the WCET of their allocated operations. Therefore:

$$R_s = \frac{T_s + T_{c_0} + T_a + 3\omega}{h_s}, \quad R_f = m \times R_s \tag{33}$$

Comparing Eq. (32) and Eq. (33) we observe that, as discussed in Fig. 11, $R_s < R_m$. Therefore, both multi-rate controllers uses more resource that the SSR with $h = r_s$. Comparing to the FSR with $h = r_f$ we expect that:

$$R_m < R_f (= m \times R_s), \tag{34}$$

By substituting Eqs. (32), (33) in Eq. (34) we obtain:

$$\begin{aligned} \frac{T_s + T_{c_0} + (m-1)T_{c_m}(\text{or } T_{c_s}) + mT_a + (2+m)\omega}{h_s} &< \frac{m(T_s + T_{c_0} + T_a + 3\omega)}{h_s} \\ \Longrightarrow (m-1)T_{c_m}(\text{or } T_{c_s}) + mT_a &< (m-1)(T_s + T_{c_0}) \\ &\quad + mT_a + 2(m-1)\omega \\ \Longrightarrow T_{c_m}(\text{or } T_{c_s}) &< T_s + T_{c_0} + 2\omega \end{aligned} \tag{35}$$

For MMR, based on Eqs. (25), (26), T_{c_m} is equal to T_{c_0} for all input values of $u(k,n)$ for $n = 1, \dots, m-1$. Therefore, the inequality in Eq. (34) always holds. For the SMR controller, based on Eq. (22), T_{c_s} depends on the execution time of FOS. Whether Eq. (35) holds depends on the FOS execution time. An important observation following from Eq. (35) is that as long as the inequality holds, multi-rate controllers (including the SSMC) have a lower resource utilization than FSR. Note that this also holds when sensing takes longer than computing $T_s > T_{c_0}$, as can be the case in data-intensive control approaches [9].

We further validate this for the case-study by measuring the execution times and performing HIL simulations in Section 8.

8. Performance analysis with HIL simulations

To validate the performance of the multi-rate controllers and analyze the resource utilization, we performed HIL simulations on the Verintec instance of CompSOC platform [verintec.com]. The platform depicted in Fig. 3 is synthesized on a PYNQ Z2 FPGA board

Table 5
Execution time of control operations in clock cycles.

T_s	T_a	T_{c_0}	T_{c_s} (SMR)	T_{c_m} (MMR)
78	211	1123	1821	1123

[http://www.tul.com.tw]. The architecture consists of two MicroBlaze tiles. We use one tile to implement the control application. The other one is used to implement the system model in Eq. (5) with a sampling period of 100μs to mimic the continuous behavior of the system. The tiles can communicate through shared memory. The plant reads the input values from the shared memory and writes the state values. Similarly, the controller reads the state values and updates the input values.

Table 5 provides the execution times of the different tasks of the control application executed separately on the platform. In this table, T_{c_0} represents the computing of C_o in multi-rate controllers and the execution time of the computing in the single-rate controller. As expected, T_{c_s} is larger than T_{c_m} since it only uses the FOS and not the SOS.

Parallel observer convergence time: As described in Section 6.2, we chose both SOS and FOS in POS as Luenberger observers. The observer gains are chosen by following the Theorem 1 in [10] to ensure the stability of the observer. We have designed the observer with faster dynamics than the case-study system so that the observer quickly reaches a user-defined error threshold for estimation. We defined this estimation error threshold as $e_s[k] < 0.1 \times Y_d = 0.0002$. The convergence time of the observer T_o depends on design considerations:

- The initial condition: The mismatch between the system initial conditions $x(0)$ and the observer initial conditions $\hat{x}(0)$ affects T_o, where higher mismatch results in longer T_o.
- Model uncertainty: The amount of model uncertainty affects the observation accuracy and can increase T_o.

As indicated in Section 6, we assume a 5% model mismatch in our design. To focus on the effect of model uncertainty, we consider $x(0) = \hat{x}(0)$ which is reasonable to assume in the most real systems. With these considerations, the observer quickly converges to the system output, which results in a $T_o \simeq 0$.

Multi-rate controllers: By having the execution times measured, we perform HIL simulations considering the schedules depicted in Fig. 11 for different values of m. For multi-rate controllers the sampling period is assumed as $r_s = 10$ ms, which results into the actuating period of $r_f = r_s/m$. For the single-rate controllers, the simulations is done using different sampling periods as $h = r_s/m$. Fig. 12 illustrates the resource utilization and QoC of the single-rate and two multi-rate controllers for different values of m.

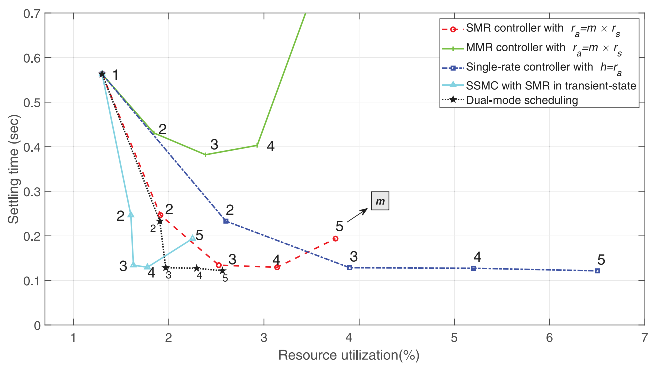


Fig. 12. QoC vs resource utilization comparison between single-rate controller, two multi-rate controllers, and SSMC.

These results validates that:

- The SMR controller results in a QoC similar to one achieved by a single-rate controller with the same actuating period while utilizing a significantly lower resource. Also, while having a better QoC, it uses more resources than the MMR controller. For example, for $m = 4$, both controllers achieve a $t_s = 0.13$ s where $R_m = 3.2\%$ and $R_f = 5.2\%$.
- The MMR controller always consumes the lowest amount of resources with the same actuating period. For the same m as the previous example, $R_m = 2.9\%$, which is lower than SMR and FSR. However, this controller does not offer a QoC similar to one achieved by a single-rate controller with the same actuating period. For example for $m = 4$, $t_s = 0.4$ s for MMR while $t_s = 0.13$ for FSR.
- In both multi-rate controllers the QoC degrades for $m > 4$. This degradation is due to the presence of the model uncertainty. With a higher value of m, the observer system has a shorter time for the convergence, and its transient state affects the resulted QoC.

Switching scheme: To validate the benefits of the proposed SSMC, we used the SMR controller for the transient state and the SSR controller with $h = r_s$ for the steady state. The reference is a periodic signal as Eq. (1) with $T_r = 0.4$ s. As demonstrated in Fig. 12, SSMC provides a similar QoC as the SMR controller while using fewer processing resources. We recall the configuration overhead of 13 ms for SSMC (as discussed in Section 3.4). This indicates a delay of $2-6$ samples in the switching scheme depending on the value of m. Fig. 12 indicates that such delay has a negligible effect on the overall QoC and the resource utilization of SSMC.

In summary, the results validate that multi-rate controllers offer a comparable control performance to single-rate controllers (with the same actuating period) while using fewer resources. The MMR controller is ideal for cases where resource utilization is the most critical constraint. In contrast, the SMR controller is the best when only QoC is of interest. Using SSMC further reduces resource utilization by offering a lower resource utilization in the steady state.

Input signal: As explained in Section 4.2 the input signal should be kept below the predefined bound, e.g. $u_{max} = 12$ V in our constraints. Fig. 13 depicts the input signals of the simulation for $m = 1, 2, \ldots, 5$. While larger values of m result in larger u_{max}, the figure validate that $u_{max} < 12$ V for all values of m.

Input noise: While we did not consider input noise in our designs, it is essential to study its effect on the performance of the SSMC scheme. Fig. 14 depicts the system's output (with $m = 4$), considering different levels of input noise ($\Delta u[k, n]$) ranging between 0–15% (a common input noise level is 5% [50]). As the figure indicates, increasing the input

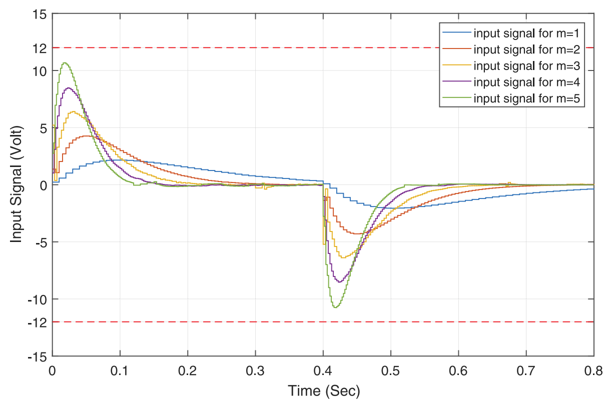


Fig. 13. The input signal $u[k]$ for different values of m. The maximum input value u_{max} is lower than 12 V for all values of m.

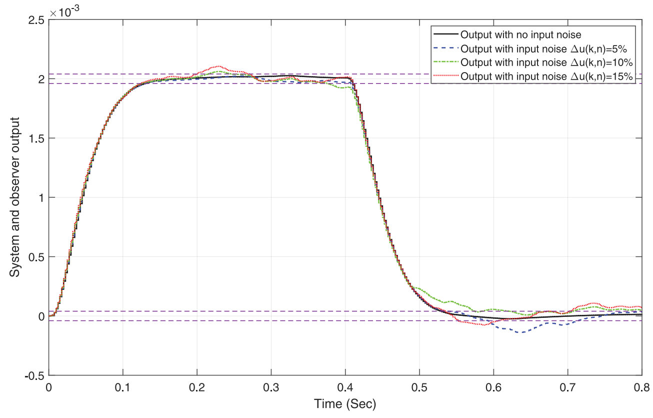


Fig. 14. System output for $m = 4$ with different values of input noise (Δu).

noise increases the steady state fluctuations. It also can cause SSMC to switch back to transient since the output exceeds the steady state error threshold. These fluctuations can be addressed by designing a proper SSR controller to cancel the effect of the input noise in the steady state.

Model uncertainty: As described earlier, the model uncertainty influences the performance of both the controller and the observer. Fig. 15 depicts the output of the system and the observer considering different model inaccuracy levels. An SSMC controller with an SMR controller with $m = 4$ in the transient state is implemented in these simulations. As the figure indicates, the designed observer performs well for $\Delta A_f, \Delta B_f = 5\%$. However, using the same observer for higher values of model uncertainty results in an increased observer error in the steady state. Compensating this error is possible by fine-tuning the observer.

Steady state error threshold: We opted for 2% of $r(t)$ as the steady state error threshold since it is accepted as a suitable threshold in control applications [51]. Increasing this threshold increases the steady state error, which is usually not ideal in control applications. However, suppose the steady-state error is less critical than the resource utilization. In that case, it is possible to increase such threshold, and accept a higher steady-state error to utilize less resource. Therefore, as long as the steady state error value is acceptable, the designer can increase the threshold, causing the SSMC to switch to the steady state quicker and use less resource. Table 6 presents the resource utilization value for different error threshold values ranging between $2-25\%$. As the table suggests, increasing the threshold level decreases resource utilization.

System Dynamic: We further study the impact of the system dynamics on the potential benefits from the SSMC scheme. In addition to the

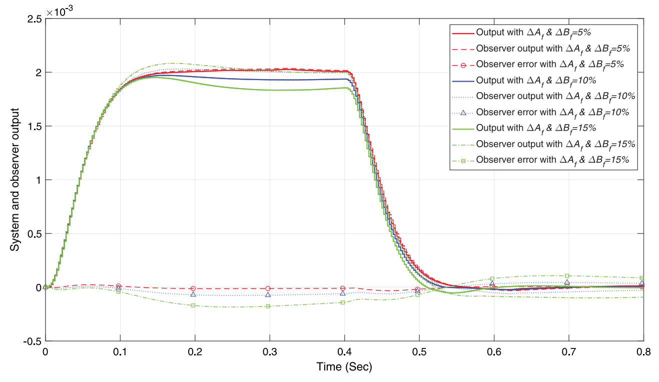


Fig. 15. The output of the system and the observer for different levels of Model Uncertainty(MO). Increasing model uncertainty increases the observer error.

Table 6
Different steady state error threshold vs resource utilization.

Error threshold	2%	5%	10%	25%
R	1.77	1.70	1.64	1.53

Table 7
Execution time of control operations for the dual rotary system in clock cycles.

T_s	T_a	T_{c_0}	T_{c_s} (SMR)	T_{c_m} (MMR)
78	211	425	1125	425

EMB system studied so far, we consider the dual-rotary (DR) dynamic system. The state-space representation of Eq. (2) for the DR system can be found in [39]. To design the SSMC controller for the DR system, we first designed a state-feedback controller (as described in Section 4.1) to place the closed-loop poles at [0.96, 0.96, 0.5, 0.5]. We then chose $r_s = 10$ ms and designed the SSMC controller with SMR in transient-state (as described in Section 6.3). We measured the execution time of control operations for the dual rotary system which are reported in Table 7. To validate the control performance, we performed MIL simulations. The resulted t_s and resource utilization for the DR system for $m = 1$ are 1.17 s and 1.3% respectively.

Fig. 16 depicts a QoC vs resource utilization for both EMB and the DR system. We normalized the QoC and resource utilization by dividing them by the values for $m = 1$. As the figure indicates, the designed SSMC controller for the DR system results in a similar behavior as the controller designed for the EMB system. The designed SSMC results in a lower resource utilization compare to single-rate controller for $m > 1$ while offering a similar control performance (model uncertainty is not considered in the DR system).

8.1. Comparison with state-of-the-art

We compare the resource efficiency and control performance of the proposed SSMC with two of the state-of-the-art techniques proposed in the literature, non-uniform sampling and dual-mode scheduling.

Non-uniform sampling: The study in [11] proposed a method to choose an optimal sequence of sampling periods that minimize the resource utilization in an OSEK/VDX OS-based platform while meeting the control requirements. Similar to our method, [11] offers a switching scheme between a finite set of sampling periods imposed by the operating system. The proposed approach in [11] uses less resource than uniform sampling, which is a scheme similar to the single-rate case in our study. The essential differences between our study and [11] are two aspects. First, in [11], the switching scheme of the system depends on the scheduling of the platform and is statically defined, while the switching is state-based in our study. Second, the proposed multi-rate

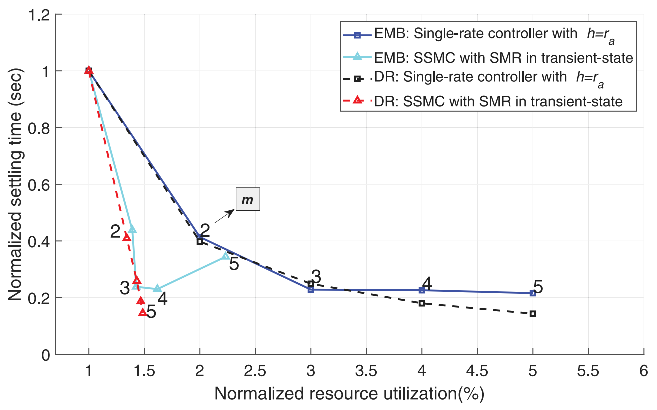


Fig. 16. Normalized QoC vs resource utilization comparison by applying the controllers on two different system dynamics. The solid lines represent the EMB system and the dashed lines represent the DR system.

controller for the transient state in our study is missing in [11] and only the switching behavior between a set of sampling periods is studied.

To compare the non-uniform approach with SSMC, two controllers are designed for the braking system defined in Section 4.2 using two different approaches. The control law for both approaches is state-feedback defined in Eq. (6) with similar K and F values. Table 8 provides the results of comparison between different approaches. In Case III of [11], an FSR controller with $r_s = r_a = 2$ ms is designed. In Case V of [11], both r_s and r_a switches between 2 ms and 5 ms following a sequence defined at design time. In the SMR approach, only an SMR controller is used for both transient and the steady state with $r_s = 5$ ms, $r_a = 2.5$ ms. Finally SSMC uses the controller proposed in our study with $r_s = 5$ ms, $r_a = 2.5$ ms for the transient state and $r_s = r_a = 5$ ms for the steady state. As the results indicate, the proposed SSMC outperforms the non-uniform scheduling in resource utilization. Comparing with the FSR controller proposed in [11] (Case III), SSMC offers a fairly similar settling time while utilizing about 60% less resource. At the same time, the maximum input value $|u[k]|_{max}$ in SSMC is 2.22 V smaller than Case III, and 2.72 V smaller than Case V of [11].

Dual-mode scheduling: The study in [12] proposes dual-mode scheduling for the control application to minimize resource utilization. Similar to the switching scheme proposed in our study, the dual-mode scheduling proposes a state-based switching scheme based on the output state of the system. Here, the controller switches between two single-rate controllers based on the error level between output and the reference. It is validated in [12] that the proposed approach allows more non-control tasks to be allocated alongside the controller. At the same time, the required control performance is guaranteed. However, in [12], the system only switches between FSR and SSR controllers and reduces resource utilization only when the error is lower than the error threshold of 2% (which can be translated to the steady state in our study). In essence, the worst-case of resource utilization occurs in the transient state. While providing a similar improvement to [12] in the steady state, our approach further reduces the resource utilization in the transient state by offering a multi-rate controller. Such improvement over [12] is reflected in Table 8 and is confirmed by the reported values of resource utilization. Generally, a longer transient state (or a shorter steady-state) results in an overall lower resource utilization in our approach, compared to dual-mode scheduling.

Similar to the comparison with non-uniform scheduling, two controllers designed for the braking system using dual-mode and SSMC approaches with a similar control law. In the dual-mode controller, the sampling period switches between 2.5 ms and 5 ms based on the output state. The designed SSMC is similar to the previous comparison.

Table 8
Results comparison between non-uniform, multi-rate, dual-mode, and SSMC scheduling. TR and SS indicates the transient and steady state respectively.

Design	r_s (ms)	r_a (ms)	t_s (ms)	R	$\|u[k]\|_{max}$ (V)
Case III in [11]	2	2	150	10	10.7
Case V in [11]	$\{2,5\}$	$\{2,5\}$	200	5.7	11.2
SMR	5	2.5	190	3.82	8.48
Dual-mode [12]	$TR:2.5 \quad SS:5$	$TR:2.5 \quad SS:5$	180	3.81	8.48
SSMC	5	$TR:2.5 \quad SS:5$	190	**3.20**	8.48

The results in Table 8 indicate that the proposed SSMC outperforms the dual-mode scheduling in terms of resource utilization. To further compare the Dual-mode scheduling with SSMC, the QoC and resource utilization comparison are also elevated for the Dual-mode. As depicted in Fig. 12, SSMC uses lesser resource than the Dual-mode for all m values. At the same time, SSMC provides a comparable QoC (model uncertainty is not considered in Dual-mode scheduling).

9. Conclusions

In this paper, we presented the implementation of the SSMC scheme on predictable and composable embedded platforms. We demonstrated that the multi-rate controllers offer a similar QoC compared to single-rate controllers while utilizing a lower amount of resources. We validated the implementation benefits by designing two multi-rate controllers and performing HIL simulations. We further improved the resource utilization by proposing a switching scheme that uses a multi-rate controller in the transient state and a single-rate controller in the steady state. The implementation of SSMC requires the targeted platform to reschedule the applications online. There are some possible extensions to this study. One direction is to realize an analytical method over the relationship between QoC and the model uncertainty. Moreover, it is interesting to optimally choose the values of m (number of actuating per sensing) and r_s (sensing period) used in SSMC, which result in optimal performance and resource utilization.

Declaration of competing interest

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

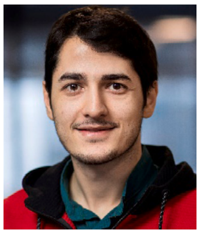
Acknowledgments

This work was partially supported by the project IMOCO4.E (GA no.101007311), and the H2020 project I-MECH (GA no.737453).

References

[1] R. Obermaisser, C. El Salloum, B. Huber, H. Kopetz, From a federated to an integrated automotive architecture, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 28 (7) (2009) 956–965.
[2] Automotive Electronic Control Unit (ECU) consolidation, Tech. Rep, Inted corporation, 2019.
[3] P. Marti, et al., Jitter compensation for real-time control systems, in: Real-Time Systems Symposium (RTSS), 2001, 2001, pp. 39–48.
[4] K. Goossens, et al., NOC-based multiprocessor architecture for mixed-time-criticality applications, in: Springer, Handbook of Hardware/Software Codesign, Springer, 2017, pp. 491–530.
[5] M. Haghi, F. Wenguang, D. Goswami, K. Goossens, Delay-based design of feedforward tracking control for predictable embedded platforms, in: ACC, 2019.
[6] J. Valencia, et al., Resource utilization and quality-of-control trade-off for a composable platform, in: DATE, 2016, pp. 654–659.
[7] M. Tomizuka, Multi-rate control for motion control applications, in: AMC, IEEE, 2004, pp. 21–29.
[8] J. Ding, F. Marcassa, S.-C. Wu, M. Tomizuka, Multirate control for computation saving, IEEE TCST (2005).
[9] S. Mohamed, D. Goswami, V. Nathan, R. Rajappa, T. Basten, A scenario- and platform-aware design flow for image-based control systems, Microprocess. Microsyst. (ISSN: 0141-9331) 75 (2020) 103037.
[10] M.-W. Thein, E.A. Misawa, A parallel observer system for multirate state estimation, in: ACC, 1999.
[11] D. Goswami, et al., Multirate controller design for resource-and schedule-constrained automotive ECUs, in: DATE, 2013.
[12] X. Dai, W. Chang, S. Zhao, A. Burns, A dual-mode strategy for performance-maximisation and resource-efficient CPS design, ACM Trans. Embed. Comput. Syst. (TECS) 18 (5s) (2019) 1–20.
[13] C. Peng, F. Li, A survey on recent advances in event-triggered communication and control, Inform. Sci. 457 (2018) 113–125.
[14] W. Wang, R. Postoyan, D. Nešić, W. Heemels, Periodic event-triggered control for nonlinear networked control systems, IEEE Trans. Automat. Control 65 (2) (2019) 620–635.
[15] P. Tabuada, Event-triggered real-time scheduling of stabilizing control tasks, IEEE Trans. Automat. Control 52 (9) (2007) 1680–1685.
[16] J. Lunze, D. Lehmann, A state-feedback approach to event-based control, Automatica 46 (1) (2010) 211–215.
[17] X. Wang, M.D. Lemmon, Event-triggering in distributed networked control systems, IEEE Trans. Automat. Control 56 (3) (2010) 586–601.
[18] R. Postoyan, P. Tabuada, D. Nešić, A. Anta, A framework for the event-triggered stabilization of nonlinear systems, IEEE Trans. Automat. Control 60 (4) (2014) 982–996.
[19] M.G. Villarreal-Cervantes, J.P. Sánchez-Santana, J.F. Guerrero-Castellanos, Periodic event-triggered control strategy for a (3, 0) mobile robot network, ISA Trans. 96 (2020) 490–500.
[20] M. Mazo Jr., A. Anta, P. Tabuada, An ISS self-triggered implementation of linear controllers, Automatica 46 (8) (2010) 1310–1314.
[21] X. Wang, M.D. Lemmon, Self-triggering under state-independent disturbances, IEEE Trans. Automat. Control 55 (6) (2010) 1494–1500.
[22] E. Bini, G.M. Buttazzo, The optimal sampling pattern for linear control systems, IEEE Trans. Automat. Control 59 (1) (2013) 78–90.
[23] W. Chang, et al., OS-aware automotive controller design using non-uniform sampling, ACM Trans. CPS (2018).
[24] A. Cervin, M. Velasco, P. Martí, A. Camacho, Optimal online sampling period assignment: Theory and experiments, IEEE Trans. Control Syst. Technol. 19 (4) (2010) 902–910.
[25] A. Aminifar, S. Samii, P. Eles, Z. Peng, A. Cervin, Designing high-quality embedded control systems with guaranteed stability, in: 2012 IEEE 33rd Real-Time Systems Symposium, IEEE, 2012, pp. 283–292.
[26] A. Aminifar, P. Eles, Z. Peng, Jfair: A scheduling algorithm to stabilize control applications, in: 21st RTAS Symposium, IEEE, 2015, pp. 63–72.
[27] Y. Sun, M.D. Natale, Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks, ACM Trans. Embed. Comput. Syst. (TECS) 16 (5s) (2017) 1–19.
[28] A. Behrouzian, H.A. Ara, M. Geilen, D. Goswami, T. Basten, Firmness analysis of real-time tasks, ACM Trans. Embed. Comput. Syst. (TECS) 19 (4) (2020) 1–24.
[29] Y. Xu, K.-E. Årzén, A. Cervin, E. Bini, B. Tanasa, Exploiting job response-time information in the co-design of real-time control systems, in: 2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications, IEEE, 2015, pp. 247–256.
[30] P. Pazzaglia, C. Mandrioli, M. Maggio, A. Cervin, DMAC: Deadline-miss-aware control, in: 31st Euromicro Conference on Real-Time Systems, ECRTS 2019, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
[31] P. Pazzaglia, A. Hamann, D. Ziegenbein, M. Maggio, Adaptive design of real-time control systems subject to sporadic overruns, in: Design, Automation & Test in Europe Conference Exhibition, Vol. 2021, DATE, 2021.
[32] Z. Li, C. Huang, X. Dong, C. Ren, Resource-efficient cyber-physical systems design: A survey, Microprocess. Microsyst. 77 (2020) 103183.
[33] S.-C. Wu, M. Tomizuka, Multi-rate digital control with interlacing and its application to hard disk drive servo, in: ACC, 2003.
[34] H. Fujimoto, Y. Hori, High-performance servo systems based on multirate sampling control, Control Eng. Pract. (2002).
[35] L. Yang, M. Tomizuka, Multi-rate short-seeking control of dual-actuator hard disk drives for computation saving, in: ACC, 2005.

[36] H. Lin, P.J. Antsaklis, Stability and stabilizability of switched linear systems: a survey of recent results, IEEE Trans. Automat. Control 54 (2) (2009) 308–322.
[37] J. Jasani, F. Presswala, N. Bhatt, Comparative study on switching techniques: A survey, Int. J. Innovat. Emerg. Res. Eng. 2 (2015).
[38] L. Maldonado, W. Chang, D. Roy, A. Annaswamy, D. Goswami, S. Chakraborty, Exploiting system dynamics for resource-efficient automotive CPS design, in: (DATE), IEEE, 2019, pp. 234–239.
[39] M. Haghi, Y. Yao, D. Goswami, K. Goossens, Parallel implementation of iterative learning controllers on multi-core platforms, in: DATE, IEEE, 2020, pp. 1704–1709.
[40] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, et al., T-CREST: Time-predictable multi-core architecture for embedded systems, J. Syst. Archit. 61 (9) (2015) 449–471.
[41] N. Thaker, S.K. Babu, Analysis of event-triggered and time-triggered architecture for a reliable embedded system, in: 8th ICCCNT, IEEE, 2017, pp. 1–5.
[42] M.J. Pont, The Engineering of Reliable Embedded Systems, LPC1769, Lulu. com, 2015.
[43] M. Zimmer, D. Broman, C. Shaver, E.A. Lee, FlexPRET: A processor platform for mixed-criticality systems, in: IEEE 19th RTAS, IEEE, 2014, pp. 101–110.
[44] P. Derler, T.H. Feng, E.A. Lee, S. Matic, H.D. Patel, Y. Zheo, J. Zou, PTIDES: A programming model for distributed real-time embedded systems, Tech. Rep., University of Berkley, Dept. of Electrical Engineering and Computer Science, 2008.
[45] Z. Sun, S.S. Ge, Stability Theory of Switched Dynamical Systems, Springer Science & Business Media, 2011.
[46] O. Mason, R. Shorten, On common quadratic Lyapunov functions for stable discrete-time LTI systems, IMA J. Appl. Math. 69 (3) (2004) 271–283.
[47] J. Valencia, et al., Composable platform-aware embedded control systems on a multi-core architecture, in: DSD, 2015, pp. 502–509.
[48] S.K. Mitter, Filtering and stochastic control: A historical perspective, IEEE Control Syst. Mag. 16 (3) (1996) 67–76.
[49] Simulink. In: MATLAB & Simulink. https://www.mathworks.com/products/simulink.html.
[50] K.J. Åström, R.M. Murray, Feedback Systems, Princeton University Press, 2010.
[51] R.C. Dorf, R.H. Bishop, Modern Control Systems Solution Manual, Addison-Wesley, 1998.


Mojtaba Haghi is a Ph.D. student in the Electronic Systems group of the Department of Electrical Engineering at Eindhoven University of Technology (TU Eindhoven) since 2017. His research focuses on co-design and implementing control applications on resource-constrained embedded platforms. His research interests are platform-aware analysis, co-design, and implementation of feedback control systems on embedded platforms.

Shengru Yu received her master degree in Electrical Engineering at Eindhoven university of technology. During the thesis project, Shengru worked on finding a solution to the tradeoff problem between the quality of control(QoC) and resource usage for embedded control systems. Since 2021 she works as a design engineer at ASML, Veldhoven, the Netherlands. Her research interests are embedded control applications, embedded systems and platform scheduling.

Dip Goswami is an Assistant Professor in the Electronic Systems group of the Department of Electrical Engineering at Eindhoven University of Technology (TU/e). His research focuses on various design aspects of embedded control systems in resource-constrained domains such as automotive and robotics. Dip obtained his Ph.D. in Electrical and Computer Engineering from the National University of Singapore (NUS) in 2009. From 2010 to 2012, he was an Alexander von Humboldt Postdoctoral Fellow at TU Munich, Germany.

Kees Goossens is Full Professor of Real-time Embedded Systems in the Electronic Systems group of the Department of Electrical Engineering at Eindhoven University of Technology (TU/e). He focuses on composable (virtualized), predictable (real-time), low-power embedded systems, supporting multiple models of computation. At Topic Embedded Products, Goossens works on real-time dependable dynamic partial reconfiguration in FPGAs. Kees Goossens completed his Ph.D. at the University of Edinburgh in 1993. He become a senior principal research scientist at Philips and then NXP Semiconductors in 1995, a position he held for 14 years. In 2007, he was appointed Full Professor at Delft University of Technology (TU/Delft). In 2010, Goossens became Full Professor at Eindhoven University of Technology (TU/e). Since 2016, Goossens has been working as a System Architect for Topic Embedded Products.

Martijn Koedam received his master degree in Electrical Engineering at Technical University of Eindhoven. His work experience includes software development in the audio industry, design and implementation of a regression test framework for POS systems and evaluating wireless ticketing systems. Since 2011 he works as a researcher at the same university. His research interests include design, modeling, and simulation of embedded Systems-on-Chip, composable, predictable, real-time and mixed-criticality systems and execution models.

Andrew Nelson received his M.Sc. degree in Embedded Systems at Eindhoven University of Technology, Netherlands in 2009. After this, he moved to Delft University of Technology, Netherlands and received his Ph.D. there in 2014. He is currently employed as an assistant professor at Eindhoven University of Technology. His research interests include real-time(including mixed time-criticality) low-power multi-core embedded-systems and the timing analyses thereof.