# Composable Platform-Aware Embedded Control Systems on a Multi-Core Architecture

Juan Valencia, Dip Goswami and Kees Goossens
Electronic Systems Group, Eindhoven University of Technology
Eindhoven, The Netherlands
{j.valencia, d.goswami, k.g.w.goossens}@tue.nl

***Abstract*—In this work, we propose a design flow for efficient implementation of embedded feedback control systems targeted for multi-core platforms. We consider a composable tile-based architecture as an implementation platform and realise the proposed design flow onto one instance of this architecture. The proposed design flow implements the feedback loops in a data-driven fashion leading to *time-varying* sampling periods with short *average* sampling period. Our design flow is composed of two phases: (i) representing the timing behaviour imposed by the platform by a *finite* and *known* set of *sampling periods*, which is achieved exploiting the composability of the platform, and (ii) a linear matrix inequality (LMI) based platform-aware control algorithm that explicitly takes the derived platform timing characteristics and the shorter average sampling period into account. Our results show that the platform-aware implementation outperforms traditional control design flows (i.e., almost 2 times) in terms of quality of control (QoC).**

## I. Introduction

An efficient implementation of embedded control systems requires considering the trade-off between resource utilisation and quality of control (QoC) [1]. The resources can be computation (e.g., processor), communication (e.g., communication bus) and memory of the embedded implementation platforms. The QoC can be measured by different system parameters (e.g., *settling time*, *peak overshoot*) depending on high-level requirements. In general, a shorter sampling period translates into a higher QoC [2], [3], [4]. However, a shorter sampling period implies a higher requirement on resources (i.e., higher computation on processors, larger data on the bus). In this context, a cost-effective design requires sharing a resource among multiple applications [5]. Further, resource sharing generally introduces interferences between applications. For feedback control systems, such inter-application interference might cause undesirable delays in the loop which might degrade the QoC and potentially *destabilize* the system.

To mitigate inter-application interference, the platforms need to offer *composability* such that the applications are functionally and temporally independent of each other [6]. Using a time-division multiplex (TDM) based policy in the platform is one possible option to achieve composability. As an implementation platform, we consider an existing platform –Composable and Predictable System on Chip (CompSOC)– that uses a TDM based policy to achieve composability [7]. In this work, we study an efficient implementation of feedback control systems onto such composable platforms.

In traditional implementations of control systems, resources are allocated to the control tasks in such a way that they are executed *periodically* assuring *uniform* sampling periods.

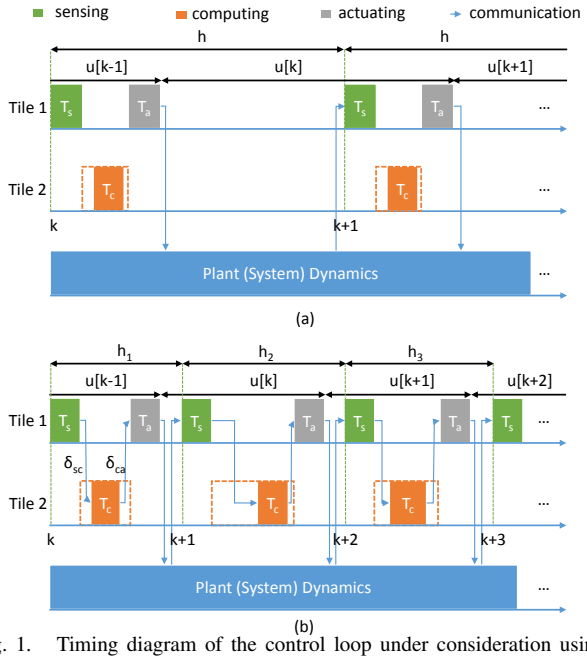


Fig. 1. Timing diagram of the control loop under consideration using two control design flows. (a) Baseline design flow, and (b) Platform-aware design flow.

Such periodic execution eases the control algorithm design and is addressed by the literature on *discrete-time/sampled-data* systems [8]. In this work, we show that such implementation (ensuring periodic sampling) might not be efficient in terms of resource utilisation and might be sub-optimal in terms of QoC for a given resource allocation. We refer to such implementation as *baseline design* (see Fig. 1 (a)).

As opposed to the baseline design, we propose a *data-driven* implementation of feedback control loops (see Fig. 1 (b)) allowing the control tasks to be executed as fast as possible. While such implementation is generally not suitable for sharing resources with other applications, the composability of the CompSOC platform allows for temporal isolation among co-existing applications and lets one application to run without interference in its allocated time slots. On the one hand, the proposed data-driven execution results in a shorter *average* sampling period. On the other hand, the control loops encounter variable sampling periods due to an occasional unavailability of resources (i.e., when other applications use them). We utilise the inherent predictable behaviour of the platform to represent the variation in sampling periods by a *finite and precisely known set of possible sampling periods*. Further, we propose a novel linear matrix inequality (LMI) based control law that utilises the

platform-specific timing information. We refer to the proposed design and implementation as *platform-aware design*. We show by experimentation that the QoC obtained using the proposed platform-aware approach is significantly higher than the one we achieve by the baseline design.

**Contributions:** The key novelty of our work is the design flow: First, we show that the platform timing behaviour results in a finite and known set of possible sampling periods. Next, we formulate the controller design as an LMI explicitly for these platform-derived sampling periods. In this process, we identify the sampling period that is closer to the average sampling period as a *nominal* sampling period. While occasional large sampling periods (due to the resource unavailability) can potentially destabilise the system, the proposed LMI-based design guarantees closed-loop stability for the platform-derived finite set of sampling periods. Thus, the design provides a flavour of an "average-case based control" design and implementation as opposed to traditional worst-case based design. Since the worst-case behaviour from a platform generally occurs only rarely and the system mostly runs near to the average-case behaviour, the overall efficiency of the proposed design is higher. We validate our claims by using a large experiment set and show that our approach provides a QoC that is almost 2 times higher than the traditional (i.e., baseline) design.

This article is organised as follows: Section II explains the feedback control loops that we consider. The composable platform and its timing model are described in Section III. Section IV describes mapping and execution of composable control applications onto the platform. Section V explains the proposed control law and the design flow. The experimental results and conclusions are presented in Sections VI and VII, respectively.

## II. Feedback control loops

A control application is responsible for regulating a continuous-time plant (i.e., a system or subsystem in the physical world) defined by:

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t) \end{aligned} \tag{1}$$

where $x(t)$ is the *state* of the plant, $y(t)$ is the output of the plant, and $u(t)$ is the control input that is computed by a control algorithm or controller. $A_c$, $B_c$, and $C_c$ are the so-called state, input, and output matrices, respectively. These matrices model the behaviour of the system dynamics.

We are concerned with the *regulation problem*. That is, the control objective is to design $u(t)$ such that $y(t) \rightarrow 0$ with time. The time it takes for the system output $y(t)$ to reach and stay in a closed region (e.g., $< 2\%$) around the reference value is the *settling time* of a feedback control loop. We define QoC (denoted by $\phi$) as inversely proportional to the settling time,

$$\phi = \frac{1}{\text{settling time}} \tag{2}$$

Thus, a higher QoC $\phi$ implies a shorter settling time.

**Input saturation:** In a real-life implementation, the control input is constrained by a maximum value (e.g., voltage, current limits of the power source). We consider that the input $|u(t)| \leq U_{MAX}$. That is, the maximum allowed input signal is $U_{MAX}$.

### A. Embedded implementations

Implementation of feedback control loops involves three consecutive operations: *sensing* ($T^s$ task), *computing* ($T^c$ task) and *actuating* ($T^a$ task). These tasks repeat, and the start and finish times of the *k*-th instance are given by $t_s(.)$ and $t_f(.)$, respectively. The execution times of $T^s_k$, $T^c_k$ and $T^a_k$ (the *k*-th task call) are given by,

$$\begin{aligned} e^s_k &= t_f(T^s_k) - t_s(T^s_k) \\ e^c_k &= t_f(T^c_k) - t_s(T^c_k) \\ e^a_k &= t_f(T^a_k) - t_s(T^a_k) \end{aligned} \tag{3}$$

The interval between two consecutive executions of sensing tasks $T^s_k$ and $T^s_{k+1}$ is known as the *sampling period* $h_k$:

$$h_k = t_s(T^s_{k+1}) - t_s(T^s_k) \tag{4}$$

Within each sampling period $h_k$, the control operations are executed sequentially (i.e., $T^s_k \rightarrow T^c_k \rightarrow T^a_k$). In addition, the time interval between the starting time of $T^s_k$ and finishing time of $T^a_k$ is known as the *sensor-to-actuator delay* $\tau_k$,

$$\tau_k = t_f(T^a_k) - t_s(T^s_k) \tag{5}$$

With the execution time description and definitions of the control tasks within a sampling period, it is then possible to derive implementation schemes for the control tasks.

**Data-driven implementation:** in the data-driven approach, the control tasks are executed right after the termination of their preceding tasks. That is,

$$\begin{aligned} t_s(T^c_k) &= t_f(T^s_k) + \epsilon \\ t_s(T^a_k) &= t_f(T^c_k) + \epsilon \\ t_s(T^s_{k+1}) &= t_f(T^a_k) + \epsilon \end{aligned} \tag{6}$$

where $\epsilon$ is time granularity supported by the embedded platform and considering that all the resources are allocated to the application. This means that there is no interruption between the executions. In this work, we take $\epsilon$ to be 1 clock cycle of the processor. Here, the sampling period $h_k$ can vary from one sampling period to the other, depending on the resource availability and the execution times of $T^s_k$, $T^c_k$ and $T^a_k$ (see Fig. 1 (b)). The advantage of such an implementation is a shorter *average* sampling period which can potentially be translated into higher QoC. However, the sampling period varies over time resulting in a *switched system*. Such switching behaviour can destabilise the overall closed-loop system [9] and therefore needs to be taken into account in the control design phase.

### B. Control task model

Considering the data-driven implementation approach, it is necessary to derive a control task model that explains the behaviour of system dynamics and control input as time evolves. The task partitioning can be done in different ways. To this end, we consider the task partitioning illustrated in Fig. 1 (b) an example to illustrate our approach. It should be noted that our approach is also applicable to another task partitioning. As can be seen, the sampling period $h_k = t_s(T^s_{k+1}) - t_s(T^s_k)$ varies from one cycle to the next one. As indicated in Eq. (5) and using Eq.(6), the sensor-to-actuator delay is defined as

$$\tau_k = t_f(T^a_k) - t_s(T^s_k) = t_s(T^s_{k+1}) - \epsilon - t_s(T^s_k) = h_k - \epsilon \tag{7}$$

As shown in Fig. 1 (b), the control input $u(t)$ is held until the next update, i.e., during $\tau_k$,

$$u(t) = u(t_k) = u[k],\ t_s(T_k^s) \leq t \leq t_f(T_k^a) \tag{8}$$

Using the model presented in [10], we have

$$x[k+1] = Ax[k] + B_0(\tau_k)u[k] + B_1(\tau_k)u[k-1] \tag{9}$$

where

$$\begin{aligned} A &= e^{A_c h_k} \\ B_0(h_k) &= \int_0^{h_k-\tau_k} e^{A_c s} ds \cdot B_c,\ B_1(h_k) = \int_{h_k-\tau_k}^{h_k} e^{A_c s} ds \cdot B_c \end{aligned}$$

In Eq. (9), we assume that $u[-1] = 0$ for $k = 0$. We define new system states $z[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$ with $z[0] = \begin{bmatrix} x[0] & 0 \end{bmatrix}^T$ obtaining the augmented higher order system

$$z[k+1] = A_{aug}(h_k)z[k] + B_{aug}(h_k)u[k] \tag{10}$$

where

$$\begin{aligned} A_{aug}(h_k) &= \begin{bmatrix} A & B_1(h_k) \\ 0 & 0 \end{bmatrix} \\ B_{aug}(h_k) &= \begin{bmatrix} B_0(h_k) \\ I \end{bmatrix} \end{aligned} \tag{11}$$

and $I$ is the identity matrix.

### C. Control law

The control input $u[k]$ is a *state feedback* controller of the following form,

$$u[k] = K_k z[k] \tag{12}$$

where $K_k$ is the state feedback gain at the $k$-th sample. The closed-loop system using Eq. (10) is given by,

$$z[k+1] = (A_{aug}(h_k) + B_{aug}(h_k)K_k)z[k] \tag{13}$$

**Switching behaviour:** with the control law (12), the closed-loop system keeps on switching as follows:

$$\begin{aligned} (A_{aug}(h_1) + B_{aug}(h_1)K_1) \rightarrow (A_{aug}(h_2) + B_{aug}(h_2)K_2) \\ \cdots \rightarrow (A_{aug}(h_3) + B_{aug}(h_3)K_3) \cdots \end{aligned} \tag{14}$$

The above switching behaviour can lead to system instability. Therefore, the design of $K_k$ must *guarantee* stability of the overall system, and provide a higher QoC at the same time.

## III. Composable multi-core platform

We consider a tile-based architecture that offers configuration with multi-processors (*processor tiles*), interconnections (*Network-on-Chip (NoC)*) and memories (*memory tiles*) within the same platform. An example architecture is shown in Fig. 2. Each processor tile is mainly composed of a MicroBlaze soft-core processor. The *monitor tile* is specialized for debugging purposes. The *memory tile* contains the external memory interface and controller, and the NoC provides interconnection between the tiles. The rest of the parameters in Fig. 2 are explained in the following.

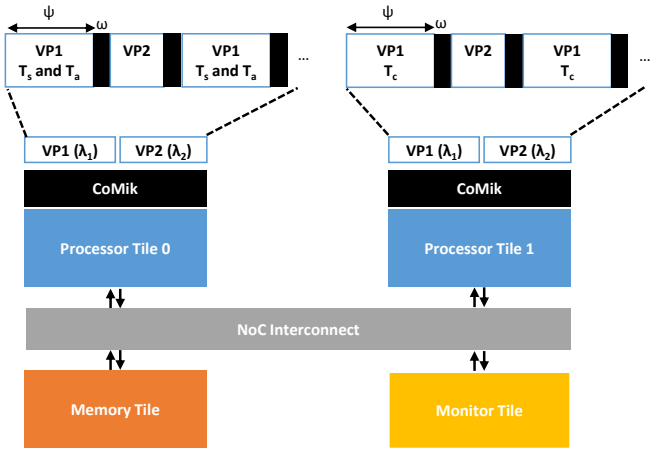


Fig. 2. High-level view of the CompSOC platform architecture under consideration.

### A. Virtual processors

In the above platform, composability is achieved by virtualising the processor resource. For this purpose, we use CoMik (Composable and Predictable Micro-kernel) that creates multiple virtual processors (VPs) that can be used as dedicated resources [11]. Each VP's utilisation of the underlying physical processor is allocated in a TDM manner. The VPs are cycle-accurately temporally isolated. That is, the activities on concurrent virtual processors do not affect each other's timing even by a cycle. In the following subsection, we describe how such cycle-accurate temporal isolation is achieved by CoMik.

### B. Cycle accurate temporal isolation

CoMik's TDM scheduling is regulated by a periodic interrupt that indicates a virtual processor *(state) context swap*. Fig. 3 illustrates how CoMik performs VPs context swap. In this example, an interrupt arrives at time I. This interrupt cannot be handled immediately since the processor is uninterruptible for a duration of U. This causes a jitter of time J. U is variable depending on the *critical region* or multi-cycle instruction and thus, J is also variable. Next, control is passed to CoMik's interrupt routine that performs the VPs context swap. The context of the previous VP is stored. This includes storing the state of the physical processor's registers etc on the stack. Subsequently, the next VP is scheduled. As shown in Fig. 3, the context swapping takes time T (i.e., transition). T can be variable due to variation in scheduling time. Clearly, the start of the next VP depends on J and T. To achieve cycle-accurate isolation, we split the TDM period into a fix duration CoMik slot of length $\omega$ cycles and partition (or VP) slot of length $\psi$ cycles. The CoMik slot starts at the time the context change interrupt is raised and lasts for a fixed duration such that

$$max(J) + max(T) + R \leq \omega \tag{15}$$

where R is 2 cycles to take into account the instruction fetch and decode stages of the pipeline, enabling the VP to start where it left off. A definitive upper bound $max(T)$ can be derived for the duration of the transition time. The jitter bound $max(J)$ is a design decision that restricts the maximum length of the partition-level critical region. This jitter bound should last minimally long enough to accommodate the processor's longest multi-cycle instruction. Increasing $max(J)$ also increases the necessary duration of the CoMik slot $\omega$. The

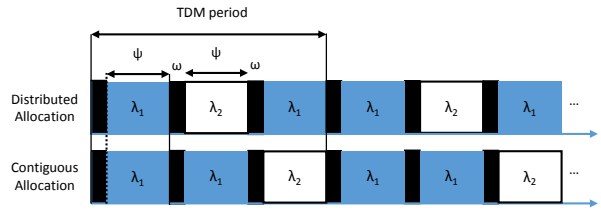


Fig. 4. Resource allocation example with $N = 3$ for $\Lambda = \{\lambda_1, \lambda_2\}$ (two applications): $S(\lambda_1) = 2$ and $S(\lambda_2) = 1$. The green blocks indicate the CoMik slots while the blue and white blocks indicate partitions slots for applications $\lambda_1$ and $\lambda_2$, respectively.

processor utilisation at the application-level is given by

$$\frac{\psi}{\psi + \omega} \tag{16}$$

which indicates that a longer $\omega$ is not desired to achieve higher utilisation. The trade-off is between a longer worst-case critical region and the decreasing CoMik slot overhead.

### C. Application scheduling

An application is executed in the allocated partition slot (or VP) and is paused every time a new CoMik slot starts. The execution is only resumed in the next partition slot assigned for the same application. In Fig. 4 this situation is illustrated by dividing a TDM period into three partitions and CoMik slots, where two partition slots are assigned for one application. The execution of an application is swapped between its partition slot, the next CoMik slot and possibly other application's partition slot. This further results in two *time domains*. The *global-time* or *wall-time* that counts for every single clock cycle of execution in the platform, and the *partition-time* that counts for every single clock cycle that has taken place within a partition slot. In the platform, we have specific timers for both time domains. This eases applications development with timing requirements.

### D. Platform timing model

With the above platform description, we consider a TDM table consisting of $N$ partition slots with $N \geq 1$. Each partition slot has a length of $\psi$ clock cycles and further, each CoMik slot has a length of $\omega$ clock cycles. The total length of a TDM table is given by $N \times (\psi + \omega)$, and with a set of applications $\Lambda$ where $0 < |\Lambda| \leq N$, the resource allocated to an application $\lambda \in \Lambda$ is given by the function $S(\lambda) : \Lambda \rightarrow \mathbb{N}$.

Fig. 4 shows the partition slots allocation in either distributed (i.e., slots can be separated by other partition or CoMik slots) or contiguous (i.e., one after the other separated by one CoMik slot) ways for two applications $\Lambda = \{\lambda_1, \lambda_2\}$ with $N = 3$. In this example, $S(\lambda_1) = 2$ and $S(\lambda_2) = 1$. That is, two partition slots are allocated to application $\lambda_1$ and one partition slot is allocated to application $\lambda_2$. Since each partition slot has $\psi$ clock cycles, $\lambda_1$ is executed on $S(\lambda_1) \cdot \psi$ clock cycles in each TDM period. A general expression of the allocated resource (as a fraction of the total resource) to an application is given by,

$$\frac{S(\lambda) \cdot \psi}{N \times (\psi + \omega)}, \ \forall \lambda \in \Lambda \tag{17}$$

In summary, the virtualisation capability of the platform enables the development and execution of applications by scheduling them into customisable partition slots. This will allow the application designer to take into account the timing properties of the platform (e.g., slot lengths and resource allocation) in order to independently develop the application on this platform and ensuring that it will not interfere with other applications. Such composable nature of the platform further allows multiple design teams to develop and verify their applications independently. Next, we describe how we design and implement our proposed platform-aware controller on this composable platform.

## IV. Composable Embedded Control Systems

Since the platform allows for independent development by functional and temporal separation, we focus only on the control application $\lambda_c \in \Lambda$. We consider a representative single-input single-output (SISO) plant dynamics for illustration. We implement the control system onto the platform with two synchronous processor tiles, one memory tile, one monitor tile, and the NoC (see Fig. 2). For the proposed data-driven implementation, we use the task partitioning illustrated in Fig. 1 (b). The tasks $T^a$ and $T^s$ are mapped onto one processor tile. The task $T^c$ is mapped onto another processor tile. As already described, the timing plays a crucial role in the design of the control law and next, we derive the timing behaviour experienced by the control loop.

### A. Timing properties of feedback loops

We are interested in characterising the exact timing behaviour the control application $\lambda_c$ will experience in the platform with the above implementation. Based on the platform properties, we obtain a finite set of possible sampling periods $h_k$ and subsequently, we utilise them in the design of the control law (detailed in the next section). Towards this, we first define the application execution time

$$e = e^s + \delta_{sc} + e^c + \delta_{ca} + e^a \tag{18}$$

where $e^s$, $e^a$ and $e^c$ are the execution times of $T^s$, $T^a$ and $T^c$, respectively. $\delta_{sc}$ and $\delta_{ca}$ are the delays (e.g., communication time over NoC) given by other operations in the *sensor-to-computing* and *computing-to-actuator* paths. Further, we consider the application execution time $e < \psi$. That is, we choose the partition slot $\psi$ to be longer than the application execution. The application runs within its allocated partition slots which are scheduled within a TDM period, and the resulting sampling periods depend on the resource allocation (i.e., $S(\lambda_c)$, distributed/contiguous allocation).

**Distributed resource allocation**: with distributed resource allocation (see Fig. 4), there is a maximum of $N + 1$ possible sampling periods independent of the number of slots allocated to the application. The possible sampling periods are

$$\begin{aligned} h_1 &= e \\ h_i &= e + (i - N + 1)\psi + (i - 1)\omega \end{aligned} \tag{19}$$

where $2 \leq i \leq N + 1$. As illustrated in Fig. 5, the addition of multiple slots (i.e., partition and CoMik slots) in between the execution of the application introduces timing cases that are related to the amount of partitions slots $N$. For instance, an application with distributed allocated resources and $N = 3$ might have $h_1 = e$, $h_2 = e + \omega$, $h_3 = e + 2\omega + \psi$, and $h_4 = e + 3\omega + 2\psi$ sampling period cases.

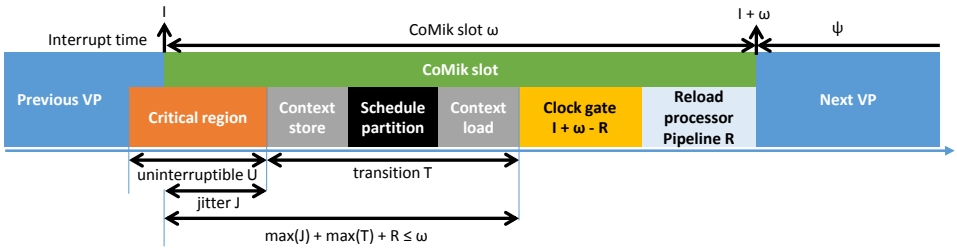


Fig. 3. CoMik cycle-accurate temporal isolation.

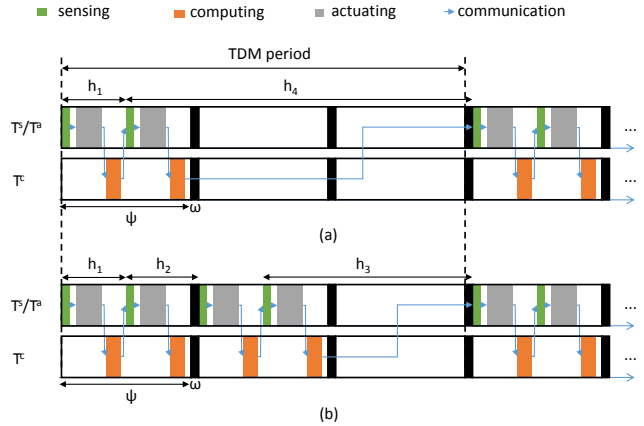


Fig. 5. The timing diagram of the control tasks with a distributed slot allocation for $N = 3$. (a) $S(\lambda_c) = 1$ with two possible sampling periods $h_1$ and $h_4$. (b) $S(\lambda_c) = 2$ with three possible sampling periods $h_1$, $h_2$, and $h_3$.

**Contiguous resource allocation**: contiguous resource allocation (see Fig. 4) results in a reduced subset of sampling periods of the distributed resource allocation. In the following, we derive the possible sampling periods for this case.

**Case I:** ($h_1 = e$): Fig. 6 (a) illustrates the scenario where the feedback loop is executed within a partition slot.

**Case II** ($h_2 = e + \omega$): For $S(\lambda_c) > 1$, the execution of the control loop might spread over two partition slots resulting in $h_2 = e+\omega$ due to the interruption by the CoMik slot. Fig. 6 (b) illustrates this scenario.

**Case III** ($h_3 = e + (N - S(\lambda_c))\psi + (N - S(\lambda_c) + 1)\omega$): Fig. 6 (c) illustrates the scenario when the execution of a feedback loop is spread over two TDM periods.

In this work, we consider a contiguous resource allocation. Therefore, as illustrated above, the sampling period $h_k$ of $\lambda_c$ switches between the elements set $H = \{h_1, h_2, h_3\}$. For a given platform configuration, the set $H$ is known at the design time.

### B. Average sampling period

With contiguous resource allocation described in Section IV-A, the ratio between the frequencies of occurrence $h_1$, $h_2$ and $h_3$ is approximately given by,

$$h_1 : h_2 : h_3 = \frac{S(\lambda_c)\psi}{e} - S(\lambda_c) : S(\lambda_c) - 1 : 1 \qquad (20)$$

That is, $S(\lambda_c)$ implies that the control loop can be executed $\frac{S(\lambda_c)\psi}{e}$ times in a TDM period of duration $N(\psi + \omega)$ cycles. Since $S(\lambda_c)$ partition slots are available, $h_2$ can occur a maximum of $S(\lambda_c) - 1$ times while $h_3$ occurs only once. Thus, the control loop runs with $h_1$ sampling period for $\frac{S(\lambda_c)\psi}{e} - S(\lambda_c)$ times. Thus, the average sampling period is given by,

$$h_{avg} = \frac{(\frac{S(\lambda_c)\psi}{e} - S(\lambda_c))h_1 + (S(\lambda_c) - 1)h_2 + h_3}{\frac{S(\lambda_c)\psi}{e}} \qquad (21)$$

From Equation (20), it is clear that $h_1$ occurs more frequently compared to $h_2$ and $h_3$. Since $h_1$ is significantly shorter than $h_2$ and $h_3$, $h_{avg}$ of the closed-loop system is closer to $h_1$. We consider $h_1$ as the *nominal* sampling period that we further use in the design of the platform-aware controller. We utilise the above platform-derived behaviour in the design of the controller.

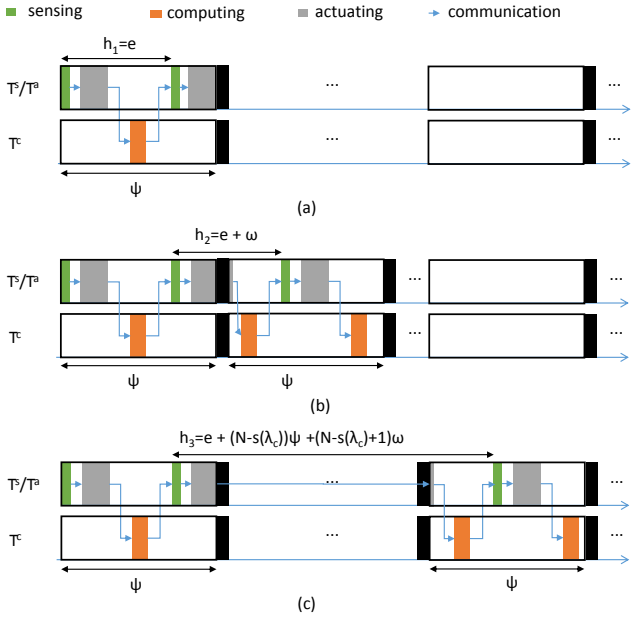


Fig. 6. The timing diagram of the control tasks with contiguous slot allocation. (a) Case I (b) Case II and (c) Case III.

## V. Platform-aware controller design

We have a system whose sampling period switches between elements of a known and finite set $H = \{h_1, h_2, h_3\}$ (derived from the analysis in Section IV-A). In this section, we present a platform-aware design method that utilises this platform-derived timing behaviour in the design of the controller to improve the QoC of the control application $\lambda_c \in \Lambda$. In view of the switched systems in Eq. (13), we consider three discrete-time switching (sub)systems,

$$z[k+1] \quad = \quad A_k z[k] \qquad (22)$$

where $A_k = (A_{aug}(h_k) + B_{aug}(h_k)K_k)$, $\forall h_k \in H$.

### A. Background: CQLF

$A_k$ are *stable* which implies that $z[k] \to 0$ as $k \to \infty$. Discrete-time LTI systems (22) are stable iff all eigenvalues of matrix $A_k$ lie inside the *unit* circle (or magnitude less than unity).

*Theorem 5.1: (Discrete-time Lyapunov equation [12])* Let $A_k \in \mathbb{R}^{n \times n}$. If there exist $P = P^T > 0$, $Q = Q^T > 0$ satisfying $A_i^T P A_i - P = -Q$, then $A_k$ is stable.

*Theorem 5.2: ([9], [13])* Consider $A_k$ to be discrete-time LTI systems of the form (22). $V(z) = z^T P z$ is the Common Quadratic Lyapunov Function (CQLF) of the systems $A_k$ if there exist $P = P^T > 0$, $Q = Q^T > 0$ and $P$ is the simultaneous solution of the discrete-time Lyapunov equations,

$$A_k^T P A_k - P = -Q < 0 \tag{23}$$

The existence of a CQLF is the necessary and the sufficient condition for the stability of the system with switching subsystems (22).

### B. LMI based design

As already mentioned, we choose $h_1$ as a *nominal sampling period* $h_n$. We design the controller gain $K_n$ corresponding to $h_n = h_1$ such that the closed-loop system $A_{cl,n}$ in Eq.(24) is stable with higher QoC.

$$z[k+1] = (A_{aug}(h_n) + B_{aug}(h_n)K_n)z[k] = A_{cl,n}z[k] \tag{24}$$

The design of $K_n$ can be done with a traditional design method such as Linear Quadratic Regulator (LQR) or a pole-placement technique [8]. The sampling period $h_k$ can be other than $h_n$. For $h_2, h_3 \in H$, we design feedback gains $K_2$ and $K_3$ such that the overall switching behaviour is stable (see Fig. 7).

*Theorem 5.3: (Design of $K_k$ for $k = 2, 3$ )*[1] Consider the subsystems shown in Eq. (22). If there exist $Y = Y^T > 0$ and $Z$ such that the following LMIs hold,

$$\begin{bmatrix} Y & YA_*^T + Z_k^T B_*^T \\ A_*Y + B_*Z_k & Y \end{bmatrix} > 0 \tag{25}$$

$$A_{cl,n}^{-1} Y - Y A_{cl,n}^T > 0 \tag{26}$$

where $A_* = A_{aug}(h_k)$, $B_* = B_{aug}(h_k)$ for $h_k = h_2, h_3$, then the systems in Eq. (22) have a CQLF with the following feedback gain

$$K_k = Z_k Y^{-1},\ k = 2, 3. \tag{27}$$

In summary, we apply gains $K_n$, $K_2$ and $K_3$ for the sampling periods $h_1$, $h_2$ and $h_3$ respectively. If the system runs with only nominal sampling period $h_n$, the closed-loop system provides a high QoC –which can be achieved by optimal design of $K_n$– with state-of-the-art design methods. Theorem 5.3 for designing $K_k$ guarantees that the closed-loop system will be stable in the presence of the switching between the $h_k, h_n \in H$. Since the system runs with $h_n$ more frequently, the QoC $\phi_{pa}$ only degrades by a small margin (which is shown in the experimental results).

[1]Proof omitted for space reasons.

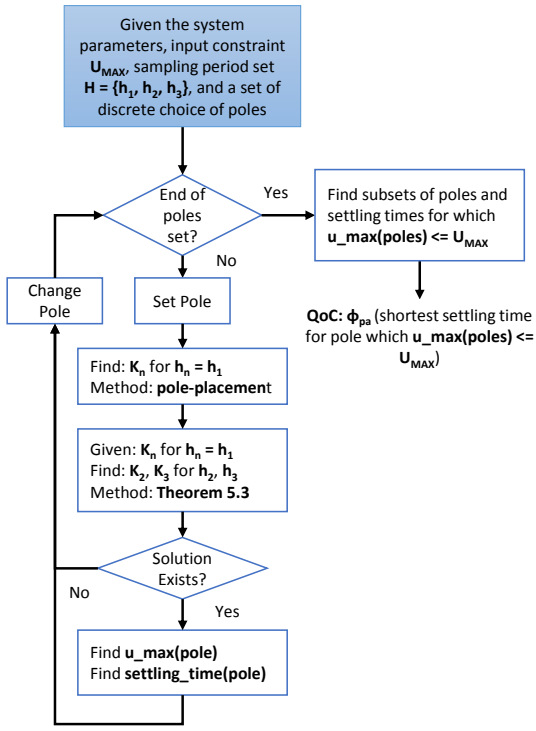


Fig. 7. Platform-aware design flow, where **u_max(.)** stands for the maximum control input for a certain set of poles.

### C. Platform-aware design flow

Since the QoC is dependent on the choice of poles that is used to design the gain $K_n$ for the nominal sampling period, we need to find the poles for which (i) $|u[k]| \leq U_{MAX}$, and (ii) solution exists for $K_2$ and $K_3$ in Theorem 5.3. Further, since we are interested in improving QoC, we choose the poles (among those which satisfy (i) and (ii)) for which we achieve the shortest settling time and the maximum QoC $\phi_{pa}$ as per Eq 2. Fig. 7 shows the control design flow for the proposed platform-aware approach. Toward this, we first discretize the design space for poles and obtain a set of possible optimal poles. For each of them, we apply the above design flow.

### D. Baseline design flow

The baseline design uses the periodic sampling period and the control gain $K$ is designed using a pole-placement technique. In this case, we choose the poles for which (i) $|u[k]| \leq U_{MAX}$ (ii) we achieve the shortest settling time and the maximum QoC $\phi_{bl}$ as per Eq 2. Overall design flow is illustrated in Fig. 8. Similar to the platform-aware design flow, we first discretise the design space for poles and for each of them, we apply the above design flow.

## VI. Experimental results

We illustrate the applicability of our proposed platform-aware design flow considering an automotive cruise control system [14]. The purpose of the system is to maintain a constant vehicle speed despite external disturbances. This is achieved by comparing the desired speed, and adjusting the engine throttle angle according to a control law. The continuous-time system model (according to Eq. 1) of this

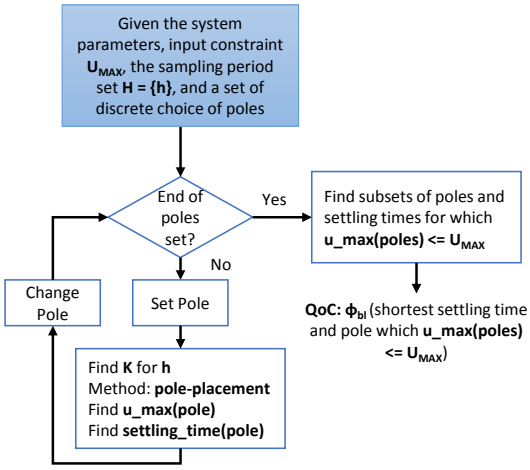


Fig. 8. Baseline design flow, where **u_max(.)** stands for the maximum control input for a certain set of poles.

cruise control system is given by,

$$\begin{aligned} \dot{x}(t) &= A_c \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + B_c u(t) \\ A_c &= \begin{bmatrix} 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 1.00 \\ -6.04 & -5.28 & -0.23 \end{bmatrix}, \; B_c = \begin{bmatrix} 0.00 \\ 0.00 \\ 2.47 \end{bmatrix} \\ y(t) &= C_c \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix}, \; C_c = [\; 1.00 \quad 0.00 \quad 0.00 \;] \end{aligned} \tag{28}$$

where we consider a control objective to drive the *vehicle's speed* $x_1(t)$ to 0 as fast as possible (i.e., short settling time and higher QoC $\phi$). The control tasks are implemented on the platform as described in Section IV. That is, two processor tiles were used to map the control tasks at a clock frequency of 120 MHz. The duration of one clock cycle is

$$\epsilon = \frac{1}{120 \text{ MHz}} \approx 8.3 \text{ ns} \tag{29}$$

Each TDM period was defined as $N = 10$ (i.e., 10 partition slots plus 10 CoMik slots), where partition and CoMik slots lengths were chosen in such a way, that $\frac{\psi}{\psi+\omega}$ gives above 90% resource utilisation within a TDM period:

$$\begin{aligned} \omega &= 4096 \text{ cycles} \approx 34.13 \; \mu\text{s} \\ \psi &= 10\omega = 40960 \text{ cycles} \approx 341.33 \; \mu\text{s} \end{aligned} \tag{30}$$

With the above system description and platform characteristics, we experimentally found that the execution of the control application is $e \approx 110$ $\mu$s. We simulated both the platform-aware and baseline design flows. For both design flows, we use an identical input signal saturation $U_{MAX} = 10^8$. As already described, we discretise the design space for poles. To keep the size of the design space reasonable, each pole value varied 0.05 units within the range of 0 and 1 discrete values. In total, this choice gave as 11931 poles combinations. These poles are used for the nominal sampling period in the platform-aware design, and the baseline design. We conducted three different experiments for both design flows.

- **Exp. 1** ($S(\lambda_c) = 1$): one partition slot is allocated for the control application which gives a 10% of partition slots usage within a TDM period.

TABLE I. SAMPLING PERIODS $h_k$, SETTLING TIME ($ST$), AND QOC ($\phi_{pa}$) OF $\lambda_c \in \Lambda$ FOR DIFFERENT $S(\lambda_c)$ ALLOCATIONS.

| $Exp.$ | $h_1$ | $h_2$ | $h_3$ | $ST$ | $\phi_{pa} = 1/ST$ |
|---|---|---|---|---|---|
| 1 | 110 $\mu$s | - | 3.53 ms | 28.8 ms | 34.72 1/s |
| 2 | 110 $\mu$s | 155.46 $\mu$s | 3.15 ms | 27.5 ms | 36.36 1/s |
| 3 | 110 $\mu$s | 155.46 $\mu$s | 2.03 ms | 26.8 ms | 37.31 1/s |

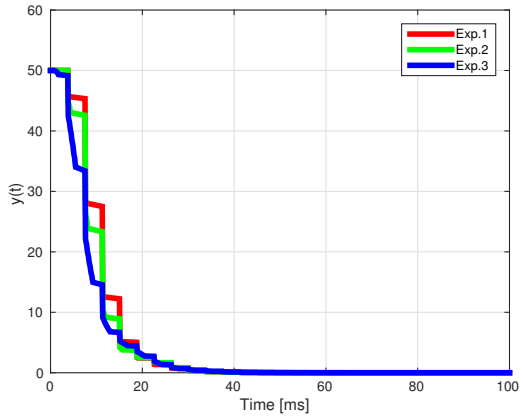


Fig. 9. Simulation of the cruise control system response ($y(t)$) for the three experiments conducted following the platform-aware design flow.

- **Exp. 2** ($S(\lambda_c) = 2$): two partition slots were allocated for the control application which gives a 20% of partition slots usage within a TDM period.
- **Exp. 3** ($S(\lambda_c) = 5$): five partition slots were allocated for the control application which gives a 50% of partition slots usage within a TDM period.

### A. Platform-aware design

As illustrated in Fig. 7, the simulation is started by defining the platform parameters (i.e., partition and CoMik slot durations, clock frequency, and TDM period length), the control input constraint $U_{MAX}$, the sampling period set H which is derived from the platform parameters and the resources allocated to the control application. In the platform-aware approach, we do not need a periodic sampling period and therefore the partition slots were assigned contiguously as shown in Fig. 11. Table I summarises the results obtained for the three different experiments. The system settling time varies depending on the allocated resource $S(\lambda_c)$. The resulting sampling periods due to the execution in the platform are shown and we also show the settling time and QoC $\phi_{pa}$ for the specific pole. For 10% resource, the settling time is longer (28.8ms) than the settling time with 20% and 50% resource allocation. Intuitively, a higher QoC is expected for a higher allocated resource. This is because a higher resource allocation $S(\lambda_c)$ implies a shorter average sampling period $h_{avg}$ and the proposed platform-aware approach exploits that knowledge in the choice of the nominal sampling period. Further, we simulated the system response which is plotted in Fig. 9, and show the system response in each experiment. It can be noticed that a remarkable performance improvement is achieved as the $h_{avg}$ gets shorter (e.g., see Exp.3). The optimal set of poles found in each experiment were [0.94 0.84 0.94 0.94], [0.94 0.89 0.94 0.94] and [0.94 0.89 0.94 0.94] for Exp.1, Exp.2 and Exp.3 respectively.

TABLE II. SAMPLING PERIODS $h_k$, SETTLING TIME ($ST$), AND QOC ($\phi_{bl}$) OF $\lambda_c \in \Lambda$ FOR DIFFERENT $S(\lambda_c)$ ALLOCATIONS.

| $Exp.$ | $h_1$ | $h_2$ | $h_3$ | $ST$ | $\phi_{bl} = 1/ST$ |
|---|---|---|---|---|---|
| 1 | 3.75 ms | - | - | 251.6 ms | 3.97 1/s |
| 2 | 1.87 ms | - | - | 125.8 ms | 7.94 1/s |
| 3 | 750.93 $\mu$s | - | - | 51.1 ms | 19.56 1/s |

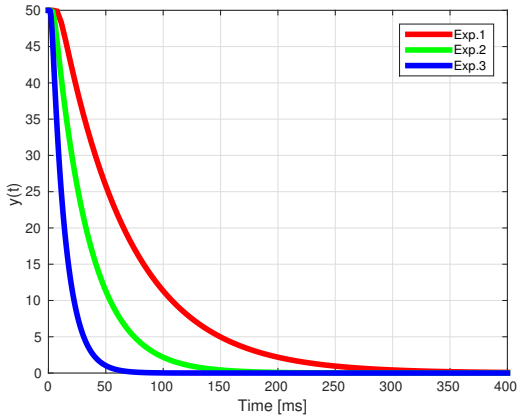


Fig. 10. Simulation of the cruise control system response ($y(t)$) for the three experiments conducted following the baseline design flow.

### B. Baseline design

In the baseline approach (see Fig. 8), we need periodic sampling. Thus, the partition slots were assigned in a distributed manner (see Fig. 11) in order to guarantee equal time intervals between two consecutive sensing operations. Table II summarises the results obtained for the three different experiments. The resulting sampling periods and the settling time due to the execution in the platform is shown. The system response for the three experiments is shown in Fig. 10, where the optimal poles found in each experiment are $[0.94\ 0.14\ 0.14\ 0.14]$, $[0.94\ 0.14\ 0.14\ 0.14]$ and $[0.94\ 0.14\ 0.24\ 0.49]$ in Exp.1, Exp.2 and Exp.3 respectively. It should be noticed that a shorter average sampling period (achieved by higher $S(\lambda_c)$) leads to a shorter settling time (i.e., higher QoC $\phi_{bl}$).

### C. Discussion

Intuitively, a higher resource allocation should provide a higher QoC $\phi$. In the platform-aware design the QoC $\phi_{pa}$ varies from $\phi_{pa} \approx 34$ in Exp.1 to $\phi_{pa} \approx 37$ in Exp.3. Similarly, in the baseline design the QoC increasingly varies from $\phi_{bl} \approx 3$ in Exp.1 to $\phi_{bl} \approx 19$ in Exp. 3. In all cases, the platform-aware design outperforms the baseline design. That is, from the results, registered in Tables I and II, the platform-aware yields a QoC $\phi_{pa} > (8.7 \times \phi_{bl})$ (in Exp. 1), $\phi_{pa} > (4.5 \times \phi_{bl})$ (in Exp. 2) and $\phi_{pa} > (1.9 \times \phi_{bl})$ (in Exp. 3). Clearly, the platform-aware design provides almost 2 times or more QoC compared to the baseline design. It is further notable from the results that the margin of QoC (i.e., $\phi_{pa} - \phi_{bl}$) between baseline and platform-aware is lower with higher resource allocation.

## VII. CONCLUSIONS

In this work, we presented a platform-aware design of feedback control loops considering a composable multi-core architecture as an implementation platform. The proposed method outperformed the traditional one by assuring short average sampling period. Our results further show how the resource allocation is reflected in the achievable QoC in the feedback control loops. Among the future extensions, we plan to exploit the periodicity of the platform-derived timing behaviour in design and implementation of the feedback control law.

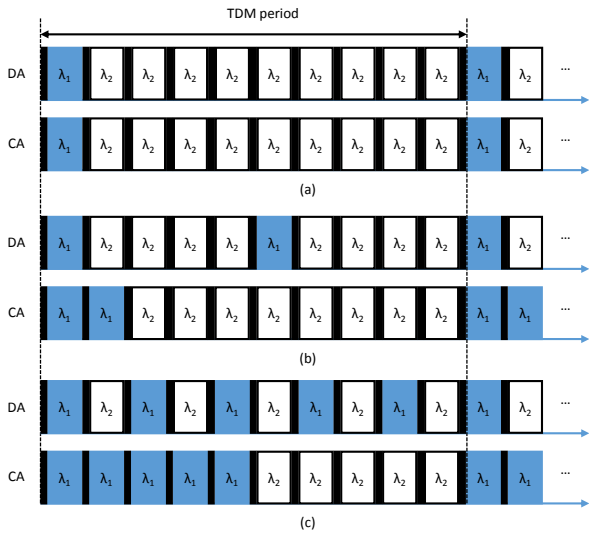


Fig. 11. Specific TDM partitioning for three different experiments.(a) Exp.1, (b) Exp.2, and (c) Exp.3. DA and CA stand for distributed and contiguous allocation, respectively.

## REFERENCES


[1] R. Castane, P. Marti, M. Velasco, and A. Cervin, "Resource Management for Control Tasks Based on the Transient Dynamics of Closed-Loop Systems," in *ECRTS*, 2006.

[2] A. Cervin and P. Alriksson, "Optimal On-Line Scheduling of Multiple Control Tasks: A Case Study," in *ECRTS*, 2006.

[3] E. Bini and A. Cervin, "Delay-Aware Period Assignment in Control Systems," in *RTSS*, 2008.

[4] A. Anta and P. Tabuada, "On the Benefits of Relaxing the Periodicity Assumption for Networked Control Systems over CAN," in *RTSS*, 2009.

[5] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Time-Triggered Implementations of Mixed-Criticality Automotive Software," in *DATE*, 2012.

[6] A. B. Nejad, "Composable Virtual Platforms for Mixed-Criticality Embedded Systems," Ph.D. dissertation, Delft University of Technology, November 2014.

[7] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. B. Nejad, A. Nelson, and S. Sinha, "Virtual Execution Platforms for Mixed-time-criticality Systems: The CompSOC Architecture and Design Flow," *SIGBED Rev.*, vol. 10, no. 3, pp. 23–34, Oct. 2013.

[8] R. C. Dorf and R. H. Bishop, *Modern Control Systems*. Addison Wesley, 1995.

[9] Z. Sun and S. S. Ge, *Stability Theory of Switched Dynamical Systems*. Springer, 2011.

[10] A. Bhave and B. Krogh, "Performance bounds on state-feedback controllers with network delay," in *CDC*, 2008.

[11] A. Nelson, A. Nejad, A. Molnos, M. Koedam, and K. Goossens, "CoMik: A predictable and cycle-accurately composable real-time microkernel," in *DATE*, 2014.

[12] W. Rugh, *Linear Systems Theory*. Prentice-Hall, N.J., 1996.

[13] O. Mason and S. R., "On common quadratic Lyapunov functions for stable discrete-time LTI systems," *IMA Journal of Applied Maths*, vol. 59, pp. 271–283, 2004.

[14] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing Signal Delay Constraints in Distributed Embedded Controllers," *Control Systems Technology, IEEE Transactions on*, vol. 22, no. 6, pp. 2337–2345, 2014.