Model-based Processor-in-the-loop Framework for Composable Multi-core platforms

Mojtaba Haghi, Martijn Koedam, Dip Goswami and Kees Goossens,
Electronic systems group, Eindhoven University of Technology (TU/e),
Eindhoven, The Netherlands,
Email: {s.m.haghi, m.l.p.j.koedam, d.goswami, k.g.w.goossens}@tue.nl.

***Abstract*—From model-based design to implementation on an embedded platform requires target-specific code generation, compilation, and execution. Processor-in-the-loop (PIL) simulation is an intermediate step meant for detailed testing and debugging in the development process. This paper presents a PIL simulation framework targeting multi-core FPGA-based embedded platforms. The presented framework allows for a fully automated process of performing PIL simulations on an FPGA-based embedded platform – CompSOC – starting from a Simulink model. The framework includes two PIL configurations – one configuration executes only the controller code on the target platform while other configuration executes both the controller and the plant code on the target platform. It considers scheduling of multiple applications and interference-free execution on the target platform under the PIL configurations. Further, the framework allows for logging various measurements of parameters such as execution time, memory usage and so on in the PIL configurations which can be used for testing and debugging purposes.**

***Keywords*-Model-based simulation; PIL simulation; Embedded control.**

I. Introduction

Embedded implementation of feedback control applications poses several challenges in terms of stability and performance [1]. A common assumption made in the controller design phase is that control tasks/software execute periodically, sequentially and in a jitter-free fashion. In widely used platforms such as Raspberry Pi, it is hard to achieve strictly periodic and jitter-free execution due to interference with other applications and system tasks sharing the platform. One way to handle such non-deterministic execution is to design robust controller for a range of execution jitter [2]. Often, such design robustness comes at the cost of performance which may not be acceptable in safety-critical domains. The alternative approach is to go for an iterative design approach to take the execution behavior into account [3]. This requires several design iterations to model the non-deterministic execution behavior and considering them in the controller design. Such platform-specific iterative design cycle is time-consuming and often, may not be feasible for many industrial scenarios.

The tailored embedded platforms for real-time applications are interesting targets for control applications [4]. These platforms offer properties such as determinism in execution times and composablity in multi-application scenarios (which guarantees interference free execution of applications). These properties guarantee periodic and jitter-free execution of the control applications. Such platform opens up potential for *nearly* iteration-free and performance-oriented design of embedded control systems [5].

The control design usually starts from model-in-the-loop (MIL) simulations in model-based simulation environments, whereas platform implementation requires platform-specific code generation for the control application, uploading and execution of the code on the platform, and logging and analysing the results obtained from the platform [6]. These steps are performed on the platforms using processor-in-the-loop or hardware-in-the-loop (HIL) simulations as essential intermediate steps.

In this paper, we present a PIL framework for model-based simulations targeting multi-core platforms. A PIL simulation is non real-time in nature which can have different configurations with different purposes. Under the PIL-C (PIL-control task only) configuration, the designed controller is executed on the embedded platform and the model of the physical system under study is simulated on the model-based environment (e.g., Simulink). This configuration enables the designer to verify the functional correctness of control code while executed on the platform. Under PIL-CP (PIL-control task and plant simulation) configuration, codes for both controller and plant model are executed on the embedded platform. This configuration is one step closer to HIL simulations as well as final implementation, since the model is no longer executed in simulation environment. It also enables the designer to verify correctness of the code in view of controller/model data exchange. Both configurations allow measuring relevant parameters such as execution time, memory usage and so on [7], which are used for temporal analysis of the control algorithm and platform scheduling in the final implementation.

Although PIL simulation is widely used in the various control applications, state-of-the-art PIL frameworks either require hand-written codes and manual implementation [8], (which is time consuming and error prone), or are designed for specific applications [6], [7], [9], [10] (which can not be generalized for other applications). Specific frameworks that automate platform implementation from a model-based control design only support low-capacity hardware modules like Raspberry Pi [11] which are not suitable for industry-scale control applications. Therefore, there is a need for a unified framework for PIL simulations targeting FPGA-based embedded platforms suitable for typical industrial control applications.

The presented PIL framework offers the following key features:

- Automatically generates the target-specific code for FPGA-based embedded platform CompSOC from any Simulink models environment (www.mathworks.com). The platform offers composability and deterministic execution behavior which in turn relevant for iteration-free design flows.
- Automatically uploads and executes PIL simulation codes

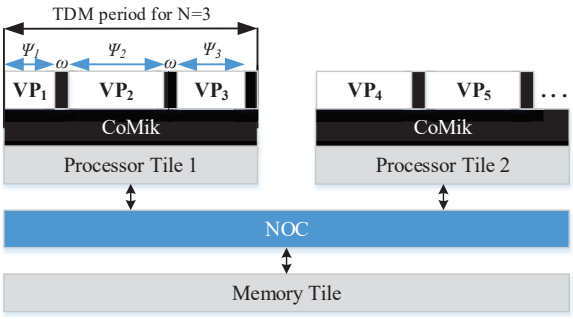

Fig. 1. Predictable embedded platform under consideration.

to the platform for both PIL-C and PIL-CP configurations.

- Enables the designer to program the embedded platform and decide on application scheduling and resource utilization within the same simulation environment.
- Reports measurement of parameters such as execution time, memory usage and so on for every PIL simulation.

The paper is organized as follows. In Section II the targeted embedded platform is demonstrated. In Section III the motion system and the control application under study is described. In Section IV possible PIL configurations for the control application are defined. In Section V the developed PIL framework and automatic target-specific code generation are presented. Finally, in Section VI the results of PIL-C and PIL-CP simulations of the designed controller is presented.

II. Composable Multi-core Platform

The embedded platform targeted in this paper is CompSOC [4]. The architecture of this platform is tile-based, which offers the configuration of a number of memory and processor tiles, and their interconnections through network-on-chip (NOC). Fig. 1 illustrates a possible architecture of the platform with two MicroBlaze soft-core processor tiles.

The platform is capable of composable execution of multi-application scenarios using partitioning on processor tiles, memory resources and interconnections. This guarantees an isolated and interference free implementation for each application regardless of presence of other applications. This is clearly beneficial for controllers as well as a plant model running on the platform. To do this, the platform uses a predictable and composable micro-kernel (CoMik) to create virtual processors (VPs) as processing resources. Each VP utilizes a portion of processing resource available on the underlying physical processors and their interconnections (i.e. NoC communications). A periodic time-division-multiplexing (TDM) policy is used on all processors and interconnections. This enables the platform to achieve real-time performance with cycle accurate time granularity.

To achieve this, the TDM is split into N partition slots with ψ_i clock cycles lengths, separated by CoMik slots with a fixed length of ω clock cycles. The CoMik slots are responsible of jitter-free context switching between VPs. Each application in one or more slots on (possibly) multiple processors interconnected by NOC connections. Applications are swapped in and out transparently and perfectly periodically by CoMik. Fig. 1 illustrates an example of a TDM table with 3 partition slots on the first processor tile, running periodically and sequentially.

III. Control Application

This section describes the motion system and the control application unser study. We consider a dual rotary fourth-order single-input-multiple-output motion system [12]. Defining θ_1 and θ_2 and their respective rotary speeds of ω_1 and ω_2 as the system states, the corresponding state-space model is adopted by the experiments of [13], and is as follows:

$$\begin{aligned} \dot{X}(t) &= AX(t) + BU(t), \\ Y(t) &= CX(t), \end{aligned} \tag{1}$$

where, $X(t) = [\theta_1, \theta_2, \omega_1, \omega_2]^t$ and,

$$\begin{aligned} A &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -7.08\times10^4 & 7.08\times10^4 & -1.1\times10^6 & 1.1\times10^6 \\ 7.08\times10^4 & -7.08\times10^4 & 1.1\times10^6 & -1.1\times10^6 \end{bmatrix}, \\ B &= \begin{bmatrix} 0 \\ 0 \\ 1.173\times10^4 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{2}$$

The control task here is to design control input $U(t)$ which makes θ_1 to follow a desired reference $r(t)$.

A. System Discretization

Since the target platform for implementing the controller is a digital system, the first step is to discretize the state-space model. By defining equally-distanced time instances t_k the discrete equivalent of system states are defined as:

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

Similarly, $y[k]$, $u[k]$ and $r[k]$ are defined.

Defining sampling period $h = t_k - t_{k-1}$, the discrete-time equivalent of system Eq. 1 is :

$$\begin{aligned} x[k+1] &= \phi x[k] + \Gamma u[k], \\ y[k] &= Cx[k], \end{aligned} \tag{4}$$

where $\phi = e^{Ah}$, and $\Gamma = \int_0^h e^{As} B ds$. The control task changes to design $u[k]$ which makes $y[k]$ follow $r[k]$.

B. Control Design and Implementation

The controller we have chosen is a 2-DOF feedback-feedforward controller, which is defined as:

$$u[k] = Kx[k] + Fr[k], \tag{5}$$

where K and F are feedback and feedforward controllers respectively. the block diagram representing the above control system is as shown in Fig. 2.

The state-feedback controller (K) aims to stabilize the system. The design technique for K is linear quadratic regulator(LQR) (which can be replaced by any state-of-the-art design techniques). The feedforward conrtoller is a closed-loop model inversion which guarantees accurate reference tracking. Referring to Fig. 2, we define closed-loop transfer function (which represents dynamic relation of the feedback+plant loop) as G_{CL}. Now if we design feedforward controller equal

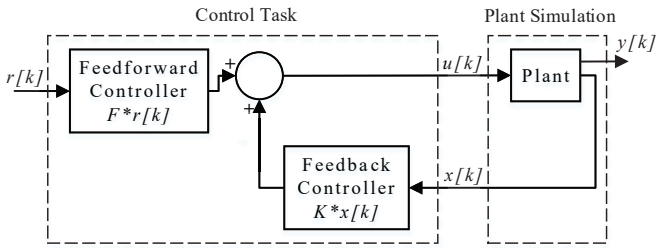

Fig. 2. 2-DOF feedback-feedforward tracking control structure.

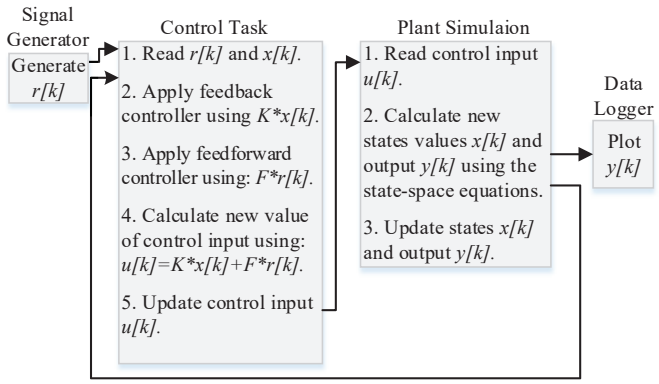

Fig. 3. Pseudo code of the control application.

(or approximately equal) to G_{CL}^{-1} it makes the transfer function from reference $r[k]$ to the output $y[k]$ equal to 1, which means perfect tracking of the reference.

The Model-in-the-loop (MIL) validation of the designed controller is performed using the presented block diagram in Fig. 2 by Simulink model-based simulation environment. In MIL simulation, the correctness of the control algorithm is verified given certain assumptions (e.g., periodicity and jitter-free execution times). The next step is hardware implementation using PIL simulations.

IV. PIL Configurations

The PIL simulation is performed by executing a part of the simulation on the target platform. Fig. 3 presents a pseudo code of the PIL simulation for the control structure shown in Fig. 2. The control application is divided into two tasks – control task (C) and the plant simulation (P). At each time instance t_k, the control task reads the current output $x[k]$ and reference $r[k]$ and and computes the next control input $u[k]$ by applying feedback and feedforward controllers. The plant simulation reads the current control input $u[k]$ and applies it to the system state-space Eq. 4 obtaining the resulted output $y[k]$ and states $x[k]$. This process is then performed periodically until the end of the simulation time.

Since the PIL simulation is governed by Simulink, the signal generator and data logger are performed in this environment. Simulink is responsible for providing the time instances t_k and reference values $r[k]$ and plotting the output $y[k]$. Based on the mapping of the rest of the blocks, we define two possible PIL configurations of PIL-C (PIL-control task only) and PIL-CP (PIL-control task and plant simulation).

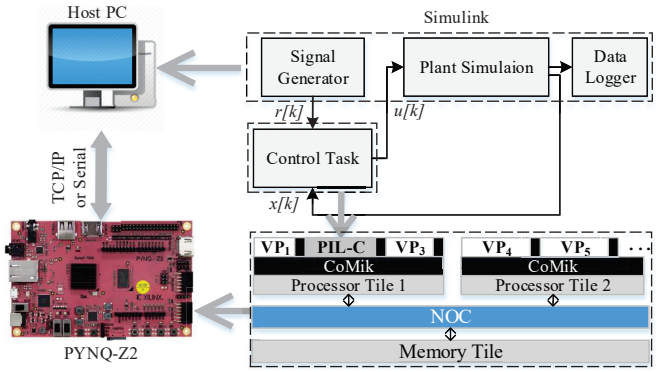

Fig. 4. PIL-C configuration. The embedded platform is implemented on a PYNQ-Z2 FPGA board. The host PC which runs Simulink communicate with the board through TCP/IP or serial connection sending $r[k]$ and $x[k]$, and receiving the resulted $u[k]$.

A. PIL-Control Task Only (PIL-C)

In this configuration only the control task (C) is executed on the target platform and the plant simulation (P) stays in Simulink environment as illustrated in Fig. 4. The PIL-C configuration allows to evaluate execution times and other constraints (e.g., memory) of implementing a control task on the target platform. Since the target platform executes a multi-application TDM policy, the PIL-C code is uploaded and executed only on its specific virtual platform (VP) to avoid (mutual) interference with other applications sharing the platform. Referring to Fig. 4, during the simulation, Simulink provides t_k, $r[k]$, and $x[k]$ and sends them to the virtual platform allocated to PIL-C and halts the simulation until the response of the platform is received. The platform then executes the generated code of the control task composable from other applications and sends back the new control input $u[k]$ to Simulink. Then Simulink resumes the simulation, gives the new control input to the plant simulation and computes the plant output $y[k]$ and states $x[k]$.

B. PIL-Control Task and Plant Simulation (PIL-CP)

In this configuration both control task (C) and plant simulation (P) are executed on the target platform as illustrated in Fig. 5. The benefit of this PIL configuration is simulating the plant on the targeted platform which is one step closer HIL, since this configuration evaluates the data exchange (in terms of data-types, precision and so on) between the control task and the plant. Similar to PIL-C, the PIL-CP code is uploaded and executed only on its specific virtual platform (VP). Referring to Fig. 5, Simulink provides t_k, and $r[k]$ and sends them to the virtual platform allocated to PIL-C and halts the simulation until the response of the platform is received. The platform then executes the generated code and gives back the new output value $y[k]$ to Simulink. Then Simulink continues the simulation by plotting the output and giving new values to the platform.

V. PIL framework

In this section we describe the development process of the PIL framework.

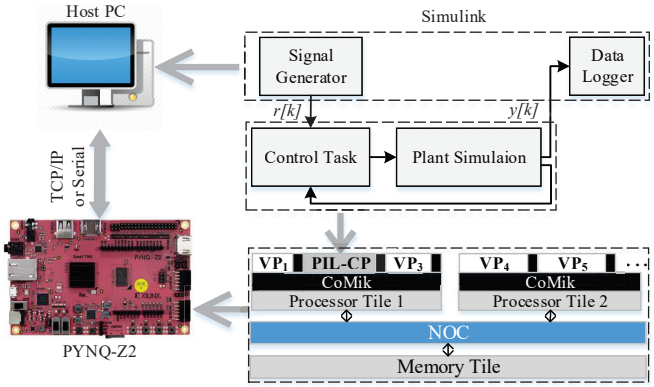

Fig. 5. PIL-CP configuration. Both control task and plant simulation are simulated on the platform. The host PC which runs Simulink communicate with the board through TCP/IP or serial connection sending $r[k]$, and receiving the resulted $y[k]$.

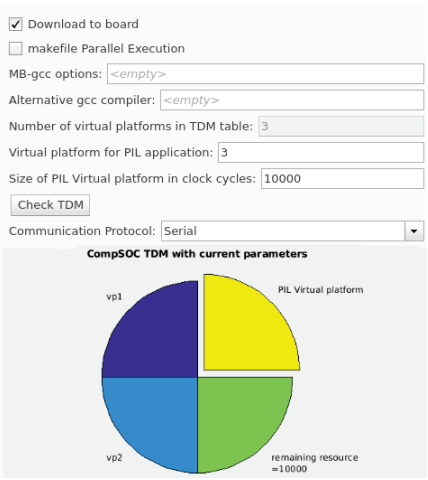

Fig. 6. The created menu in the tool-chain which enables the designer to define TDM variables as well as communication channel for the simulation.

A. Code-generation

The targeted platform described in Section II is implemented on PYNQ-Z2 FPGA board (www.tul.com.tw). The board has a 650MHz dual-core Arm Cortex-A9 processor with $512MB$ available memory. The board enables communication with the host PC using both serial and Ethernet connections. To enable target-specific code generation, we defined the platform described in Section II as a new target hardware in Simulink. Through this, we handed platform properties, such as Microblaze as the processor, data types, endianness, and largest atomic size to Simulink.

The code generation is performed by a developed tool-chain which adopts Mathworks embedded coder toolbox (www.mathworks.com) which automatically generates target-specific code. The process for generating code is trivial. The designer should decide which part of the simulation is to be executed on the platform. This part is then encapsulated in a subsystem. By choosing the developed tool-chain, generating code is to simply build the subsystem as a PIL block. The result is divided in two parts. First is a library with the corresponding generated codes for the subsystem. Second is a PIL block which is responsible for code upload and I/O exchange between Simulink and the platform through PIL simulations.

B. PIL simulation

The next step is to run the PIL simulation. Doing this is to normally simulate the model including the PIL block. The steps of the simulation is as follows.

Code Compilation: The simulation starts with building an executable to be uploaded to the platform. Since the target platform has MicroBlaze processors, the suitable compiler is MB-GCC. The created tool-chain provides the compiler library and address it in Simulink. The compiled output is an executable '.ELF' file.

Code Upload: CompSOC platform is a composable platform running multiple applications at the same time. In the code upload it should be considered that the controller must be uploaded on its specific virtual platform (VP). Fig. 6 represents the menu added to the simulation preferences where the designer defines the number of virtual platforms, TDM allocation and the size of the virtual platform allocated to the control application. The menu interactively plots the TDM wheel visualizing resource utilization of each VP and the remaining available processing units in TDM wheel to be allocated.

Considering these user-defined preferences, Simulink uploads the executable on the corresponding VP using the TCP/IP connection.

Simulation: The simulation is performed by the same procedure described in Section IV. The communication channel between Simulink and the platform can be either a serial or TCP/IP. The designer can choose one of them using the same menu in Fig. 6. Both communication channels are verified using Matlab defined benchmark tests resulting a bandwidth of 4500 bytes per second (B/s) for the serial and $18000(B/s)$ for the TCP/IP. While TCP/IP provides higher bandwidth, the designer can opt for serial communication to use a single connection for both communication and power supply to the board.

VI. Results

To validate the PIL framework, the designed controller discussed in Section III is simulated. Fig. 7 illustrates the PIL-C simulation in the Simulink environment. The considered sampling period for control design is $10ms$. The reference signal $r(t)$ is $2sin(\pi t)$. The plant is simulated using Eq. 4 in a $100\mu s$ sampling period to mimic the continuous behavior of the motion system. The designed TDM schedule for the platform has 3 VPs with equal size of $10ms$. For both PIL-C and PIL-CP configurations, only VP2 is allocated to the PIL simulation and the two rest were used for other applications. Fig. 8 and Fig. 9 are the result comparison between PIL and MIL simulations for PIL-C and PIL-CP respectively. For PIL-C the results of MIL and PIL are identical. For PIL-CP however, there is a difference between MIL and PIL. The reason could be the change in precision when the code generator replaces plant parameters by their

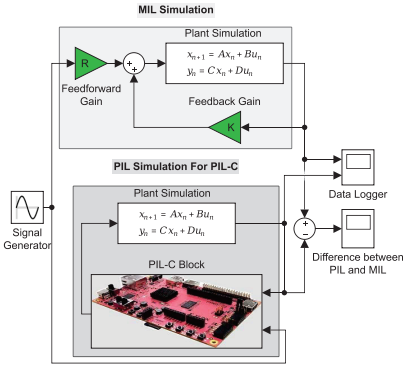

Fig. 7. Block diagram for PIL-C simulation created in Simulink Environment.

TABLE I
EXECUTION TIMES AND MEMORY USAGES FOR 4 SECONDS SIMULATION

PIL Configuration	PIL-C	PIL-CP
Avg. Execution Time in Each Step (ns)	600	140457
Total Execution Time (μs)	48003	11566663
Size of '.ELF' Executable $(KBytes)$	19	22

numeric equivalent.

Execution time and memory: The constructed framework is able to report the memory usage and execution times. TABLE. I represents the reported values for both simulations. Execution time in each step is the time spent on the platform to execute one step of the simulation. Total execution time is the total time spent on the platform to complete the simulation, considering simulation is lasted for 4 seconds. The execution times and memory usage for PIL-CP is higher than PIL-C, since the platform needs to simulate the plant which runs at a higher frequency and requires more computation in each step.

VII. Conclusion

In this paper, we proposed a model-based PIL simulation framework which targets composable multi-core platforms. The framework enables the designer to perform PIL simulations in Simulink model-based environment and schedule the embedded platform within the same environment. The composable embedded platform enables multi-applications scenarios where PIL simulation is executed next to other

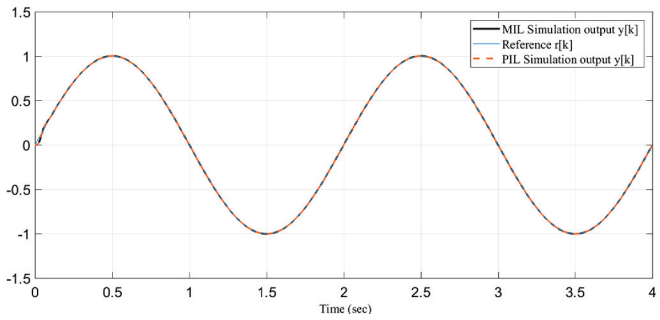

Fig. 8. PIL simulation output for PIL-C configuration. The MIL and PIL results are identical in this configuration.

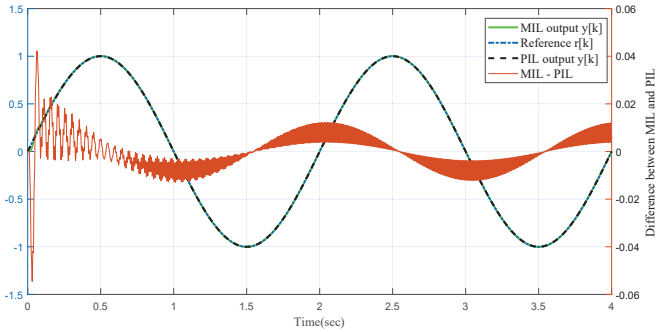

Fig. 9. PIL simulation output for PIL-CP configuration. The left hand scale is for simulation outputs and the right hand scale is for the difference between MIL and PIL simulations.

applications running on the platform without any interference. The deterministic executions on the platform enables accurate measurement of execution times for PIL simulations. This is beneficial for model-based validation of a wide range of control algorithms considering the hardware constraints. The results validates the functionality of the PIL framework for both configurations. The work can be extended by enabling the framework to perform HIL simulations as well as using multi-core nature of the platform, mapping different parts of the simulation different cores.

Acknowledgments

This work was partially supported by the H2020 project I-MECH (GA no.737453).

References

[1] D. Goswami, R. Schneider, A. Masrur, M. Lukasiewycz, S. Chakraborty, H. Voit, and A. Annaswamy, "Challenges in automotive cyber-physical systems design," in *SAMOS, 2012*.

[2] H. Yan *et al.*, "H_∞ output tracking control for networked systems with adaptively adjusted event-triggered scheme," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–9, 2018.

[3] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal analysis of timing effects on closed-loop properties of control software," in *RTSS*, 2014, pp. 53–62.

[4] K. Goossens *et al.*, "NOC-based multiprocessor architecture for mixed-time-criticality applications," *Handbook of Hardware/Software Codesign*, pp. 491–530, 2017.

[5] M. Haghi, F. Wenguang, D. Goswami, and K. Goossens, "Delay-based design of feedforward tracking control for predictable embedded platforms," in *ACC*, 2019.

[6] G. Francis, R. Burgos, P. Rodriguez, F. Wang, D. Boroyevich, R. Liu, and A. Monti, "Virtual prototyping of universal control architecture systems by means of processor-in-the-loop technology," in *APEC*, 2007.

[7] R. Liu, A. Monti, G. Francis, R. Burgos, F. Wang, and D. Boroyevich, "Implementing a processor-in-the-loop with a universal controller in the virtual test bed," in *PESC*, 2007.

[8] M. Ruba, N. Hunor, H. Hedesiu, and C. Martis, "FPGA-based processor-in-the-loop analysis of variable reluctance machine with speed control," in *AQTR*, 2016.

[9] M. Hu, G. Zeng, H. Yao, and Y. Tang, "Processor-in-the-loop demonstration of coordination control algorithms for distributed spacecraft," in *ICIA*, 2010, pp. 1008–1011.

[10] S. Lee, H. Bang, and D. Lee, "Predictive ground collision avoidance system for UAV applications: PGCAS design for fixed-wing UAVs and processor-in-the-loop simulation," in *ICUAS*, 2016.

[11] M. Różewicz and A. Piłat, "Study on controller embedding stage using model-based-design for a bike with CMG," in *MMAR*, 2018.

[12] W. Geelen *et al.*, "The impact of deadline misses on the control performance of high-end motion control systems," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 2, pp. 1218–1229, 2016.

[13] J. Boot, *Frequency response measurement in closed loop: brushing up our knowledge*, ser. DCT rapporten 2003.059. TU/e.