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

Abstract— **This paper presents a model-based PIL simulation framework targeting multi-core multi-application FPGA-based embedded platforms. The process from model-based simulations to implementing on the platform requires a target-specified code generation, compile and execution. The presented framework is able to automatically go through this process and perform the PIL simulation starting from a model-based environment in particular Simulink. It is also able to consider the multi-application nature of the target platform, executing the PIL simulation without interfering other applications. We validate the functionality of the PIL framework by testing a control systems application, using various PIL configurations.**

I. Introduction

The new developments of digital processors and the emerge of FPGA based embedded platforms enable digital control systems researchers to design and implement faster and more complex digital control algorithms [1]. However, using such platforms for implementation of control systems include some concerns. First, since traditional model-based simulations is usually not sufficient to capture control dynamics, it is essential to include platform implementation in rapid prototyping. Second, the time-critical nature of control systems should be considered in the implementation, specially in multi-application and multi-core scenario where the control application is not the only application implemented on the platform [2]. In this case, platform implemenation of control systems may be influenced by other applications on the platform. Therefore, it is essential to guarantee that the execution characteristics of control systems on the platform is independent from other applications.

The process from model-based simulation to platform implementation consists of platform-specific code generation for the control application, upload and execution of the code on the platform and inspecting the platform results [3]. The need for a framework which can automate this process seems essential to enable the rapid prototyping of the model-based control design including platform implementation.

The simulations which include platform implementation are hardware-in-the-loop (HIL) and processor-in-the-loop (PIL). In HIL the designed controller is excecuted on the embedded platform while the plant is modeled by a real-time simulation environment. Despite the realism benefit of HIL, its implementation is constrained by real-time requirements on the execution of the plant model. Also, the modifications in components and parameters in real-time simulator are time-consuming. Similar to HIL, PIL is also a simulation where the designed controller is executed on the embedded platform. The difference here is that the PIL is not real-time and the plant is simulated either on the host PC or on the platform. Using PIL enables the designer to verify the functionality of control code on the platform and measure the execution time and memory usage of the control algorithm [4]. This data can be used in temporal analysis of the control algorithm and time scheduling in the final implementation.

PIL simulation is widely used in the various control applications. In [5], a PIL simulation framework is presented to test the designed spacecraft control. The control code is generated manually and is not automated. it then executed on VxWorks. Similarly, [6] presents a PIL framework for unmanned air vehicle control where the controller is manually implemented on Texas Instrument boards. Similarly, [7] proposed a LabView based modeling environment from where a FPGA footprint for the controller is generated. An overview of the testing and debugging in model-based system engineering environments is provided in [8]. It demonstrated automated code generation and execution on a simple Raspberry Pi board through Simulink [9] standard support. However, the setup is fairly simple and cannot be used for FPGA-based embedded platforms. An interesting PIL framework is presented in [3] and [4] where the controller is implemented on a DSP board. It does not provide multi-application support though. The work presented in [10] suffers from similar drawbacks. In summary, state-of-the-art PIL frameworks do not provide support for multi-core and multi-application technologies and code generation is mainly performed for low-capacity hardware modules like Raspberry Pi and hand-written code is more commonly used. Therefore, additional development is necessary for supporting model-based simulations performed on FPGA-based embedded platforms.

In this paper a PIL framework for model-based simulations targeting multi-core platforms is presented. This new PIL framework can:

- Automatically generates the target specific code from Simulink model-based development environment.
- Uploads the code on the FPGA-based embedded platform and executes PIL simulation and can monitor platform outputs online.
- Communicate with a composable embedded platform which guarantees interference free execution of the simulation in multi-application and multi-core scenarios. It also allows the PIL user to decide on application scheduling and resource utilization on the platform.

The paper is organized as follows. Section II defines the targeted composable multi-core embedded platform. Section III describes the example motion system and the control application. Section IV describe possible PIL configurations for control application. Section V describes the developed PIL framework, automatic target-specific code generation and different possible configurations for PIL simulation on the platform. Finally, in Section VI the results of PIL

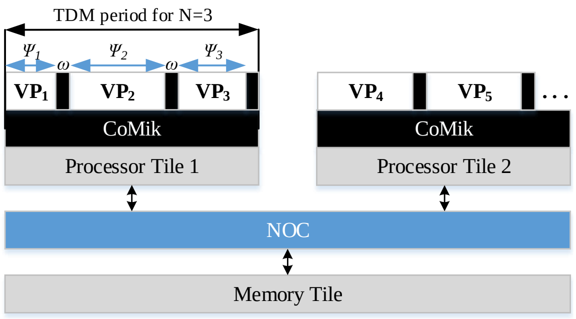


Fig. 1. Predictable embedded platform under consideration

simulations for two of the state-of-the-art controllers applied on the motion system is presented to validate and illustrate the capabilities of the framework.

II. Composable Multi-core Platform

The embedded platform targeted in this paper is Comp-SOC [11]. The architecture of this platform is tile-based which offers the configuration of a number of memory and processor tiles, and their interconnection 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 the 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 utilize 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 shows an example TDM table with 3 partition slots on the first processor tile, running periodically and sequentially.

III. Control Application

This section defines an example control application to be implemented on the embedded platform. We consider a dual rotary fourth-order single-input-multiple-output motion system [12]. Using dynamic equations, the mathematical model for this system can be represented by a state-space model. 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 is adopted by the experiments of [13] as follows:

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

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

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

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 in control design is to transform state-space into discrete-time. by defining equally distanced time instances t_k the discrete equivalent of system states is 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. Now the control task is to design $u[k]$ which makes $y[k]$ to follow $r[k]$.

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 $\Gamma = \int_0^h e^{As} B ds$.

B. Control Design and Implementation

The controller we chose for this application is a 2-DOF feedback-feedforward architecture. It 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 feedback (K) is a state-feedback controller aiming to stabilize the system. The design technique for K is linear-quadratic regulator(LQR) (It 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 (or approximately equal) to G_{CL}^{-1} it makes the trasfer function from reference $r[k]$ to the output $y[k]$ equal to 1. It means that the output perfectly follows the reference.

The design and Model-in-the-loop (MIL) validation of the designed controller can be performed using the presented block diagram in Fig. 2 by a model-based simulation environment in particular Simulink. At the MIL simulation, the

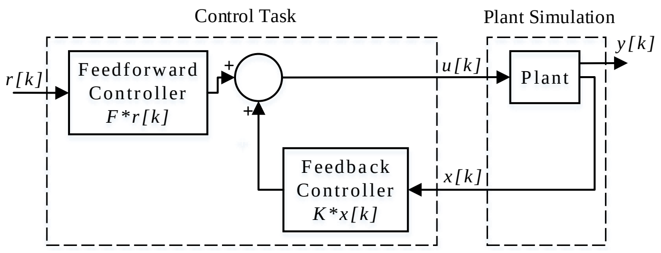
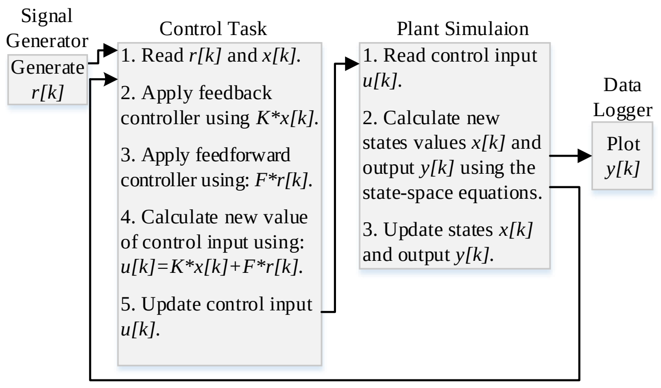


Fig. 2. 2-DOF feedback-feedforward tracking control structure



Signal Generator
Generate r[k]
Control Task
1. Read r[k] and x[k].
2. Apply feedback controller using K*x[k].
3. Apply feedforward controller using: F*r[k].
4. Calculate new value of control input using: u[k]=K*x[k]+F*r[k].
5. Update control input u[k].
Plant Simulaion
1. Read control input u[k].
2. Calculate new states values x[k] and output y[k] using the state-space equations.
3. Update states x[k] and output y[k].
Data Logger
Plot y[k]

Fig. 3. Pseudo code of the control application

correctness of the control algorithm is verified given certain assumptions (e.g., periodicity and time of executions) on the executions. After the MIL verification of control design, the PIL simulation is performed where as a part of the simulation is executed on the target platform (e.g., on the platform described in Section II). We consider two relevant PIL configurations which are described in the following.

IV. PIL Configurations

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 a part of the model-based simulation process, the signal generator and data logging are performed in the same model-based development environment – Simulink. This means that Simulink is the responsible of providing the time instance t_k and reference value in each step $r[k]$ and plots $y[k]$. For the rest of the simulation blocks, there are two possible PIL configurations which are PIL-control task only (PIL-C) and PIL-control task and plant simulation (PIL-CP).

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

In this configuration only the control task (C) is uploaded and executed on the target platform and the plant simulation (P) stays in Simulink environment as demonstrated in Fig. 4.

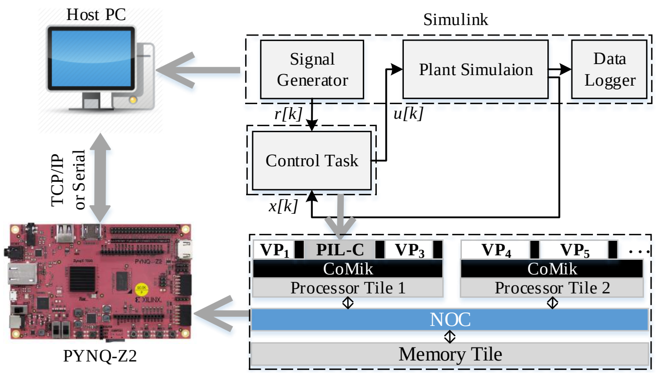


Fig. 4. PIL-C configuration. The embedded platform is implemented on a PYNQ-Z2 FPGA board. The host PC which runs Simulink communicate with the board through TCP/IP or serial connection sending $r[k]$ and $x[k]$, and receiving the resulted $u[k]$.

This PIL configuration is beneficial since it helps the designer to study about execution times and constraints of implementing a control task on the target platform. Since the target platform is executing a multi-application TDM scheduling, the PIL-C code is uploaded and executed only on its specific virtual platform (VP) to avoid (mutual) interference with other applications that may be running on the platform. Referring to Fig. 4, during the simulation, Simulink provides the 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 by giving the new control input to the plant simulation and computing 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 uploaded and executed on the target platform as demonstrated in Fig. 5. The benefit of this PIL configuration is simulating the plant on the targeted platform which is one step closer to the reality, since in the next design step (which is HIL) the plant will no longer be in Simulink environment. Similar to PIL-C, the PIL-CP code is uploaded and executed only on its specific virtual platform (VP) to avoid (mutual) interference with other applications that may be running on the platform. Referring to Fig. 5, Simulink provides the 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 composable from other applications 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.

A. *Code-generation*

The targeted platform described in Section II is implemented on PYNQ-Z2 FPGA board [14] (See Fig. 4). It

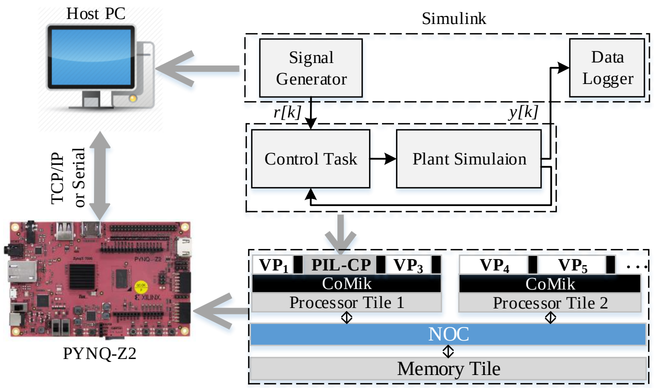
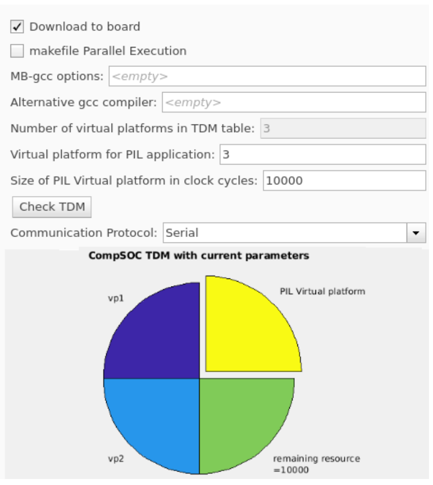


Fig. 5. PIL-CP configuration. Both control task and plant simulation are simulated on the platform. The host PC which runs Simulink communicate with the board through TCP/IP or serial connection sending $r[k]$, and receiving the resulted $y[k]$.


☑ Download to board
☐ makefile Parallel Execution
MB-gcc options: <empty>
Alternative gcc compiler: <empty>
Number of virtual platforms in TDM table: 3
Virtual platform for PIL application: 3
Size of PIL Virtual platform in clock cycles: 10000
Communication Protocol: Serial

Check TDM
CompSOC TDM with current parameters
vp1
PIL Virtual platform
vp2
remaining resource =10000

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

has a 650MHz dual-core 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, in the first step, we defined the platform described in Section II as a new target hardware in ”hardware specification” in Simulink specifications. Through this, we handed platform properties, such as Microblaze as the processor, data types, endianness, and largest atomic size to be considered in code-generation.

The code generation in this PIL framework is performed automatically through Mathworks embedded coder toolbox [15]. For this purpose a target-specific toolchain is created which can be chosen in simulation preference as the code generator. The process for generating code is trivial. The designer first decides which part of the simulation is to be executed on the platform. This part is then encapsulated in a subsystem. By choosing the created toolchain,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

Now that the platform part of the simulation is divided and the corresponding code is gereated, next 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: Starting the simulation is compiling the code to an executable output to be uploaded on the platform. Since the target platform has MicroBlaze processors, the suitable compiler is MB-GCC. The created toolchain provides the compiler library and address it in Simulink. The compiled output is an executable ’.ELF’ file. The next step is to upload the executable on the platform.

Code Upload: ComspSOC platform is a composable platform running multiple applications at the same time. In the upload procedure 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 can define 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 options, Simulink uploads the executable on the corresponding virtual platform using the TCP/IP connection.

Simulation: The final step is to run the 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 channel. 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 communications. While TCP/IP provides higher bandwidth, the designer can opt for serial communication to use a single connection for both communication and power supply of 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 sampling period in control design is $10ms$. The reference signal $r(t)$ is a $2sin(\pi t)$. The plant is simulated using Eq. 4 in a $100\mu s$ sampling period to mimic the motion system behavior. The designed TDM scheduling for the platform has 3 virtual platforms (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, referring to Fig. 9

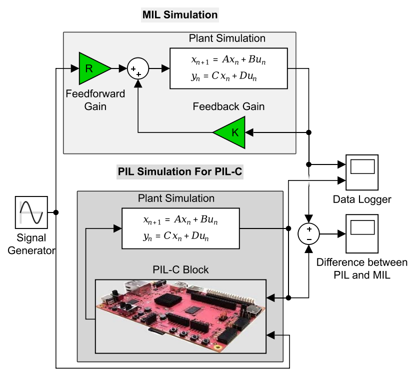


Fig. 7. Block diagram for PIL-C simulation created in Simulink Environment.

TABLE I
EXECUTION TIMES AND MEMORY USAGES FOR 4 SEC 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

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 numeric equivalent.

Execution time and memory: The constructed framework is able to report the memory usage and execution times. TABLE. I represents the measured values through the simulations. In this table, average 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 simulating for 4 seconds. The execution time and memory usage for PIL-CP is higher than PIL-C since the platform needs to simulate the plant which runs in 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 supports model-based environments in particular Simulink. It also enables the designer to implement and schedule the embedded platform on the FPGA within the same model-based environment. The composable embedded platform enables multi-applications scenarios where PIL simulation is executed next to other applications running on the platform without any interference. The predictability of the platform enables measuring execution time of the PIL simulation. 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. The work can be extended by adding ability to make the simulations real-time and also to include the real physical system to the loop enabling HIL simulations.

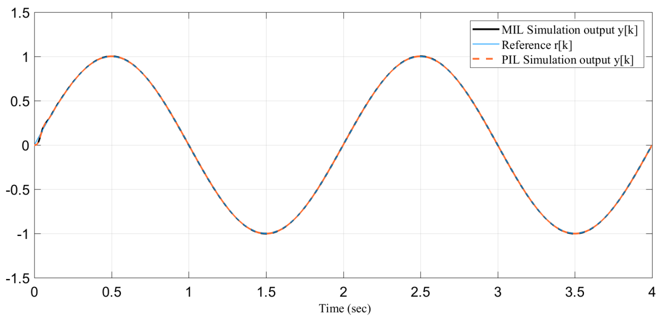


Fig. 8. PIL simulation output for PIL-C configuration. The MIL and PIL results are identical in this configuration

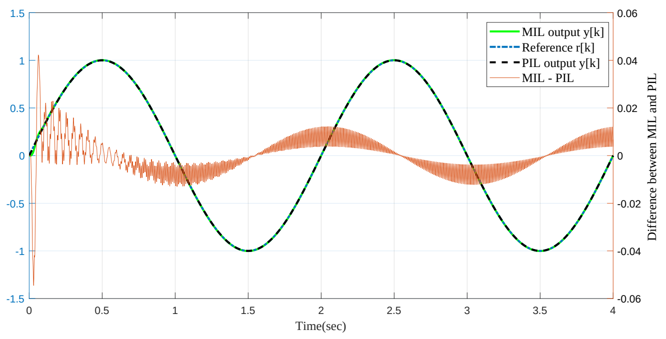


Fig. 9. PIL simulation output for PIL-CP configuration. The left hand scale is for simulation outputs and the right hand scale is for the difference between MIL and PIL simulations.

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

[3] 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.

[4] 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.

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

[6] 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*. IEEE, 2016.

[7] 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*. IEEE, 2016.

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

[9] "Simulink." [Online]. Available: https://www.mathworks.com/products/simulink.html

[10] H. J. You, K. Kim, and D. Jung, "A real-time simulator for processor-in-the-loop simulation of small satellites," in *ICCAIS*. IEEE, 2018.

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

[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. TU/e, 2003, dCT 2003.059.

[14] [Online]. Available: http://www.tul.com.tw/ProductsPYNQ-Z2.html

[15] "Embedded coder." [Online]. Available: https://nl.mathworks.com/products/embedded-coder.html