# Co-Design of a Controller and Its Digital Implementation: The MOBY-DIC2 Toolbox for Embedded Model Predictive Control

Alessandro Ravera, *Student Member, IEEE*, Alberto Oliveri, *Member, IEEE*, Matteo Lodi, *Member, IEEE*, Alberto Bemporad, *Fellow, IEEE*, W. P. M. H. Heemels, *Fellow, IEEE*, Eric C. Kerrigan, *Senior Member, IEEE*, and Marco Storace, *Senior Member, IEEE*

***Abstract*— Several software tools are available in the literature for the design and embedded implementation of linear model predictive control (MPC), both in its implicit and explicit (either exact or approximate) forms. Most of them generate C code for easy implementation on a microcontroller, and the others can convert the C code into hardware description language code for implementation on a field programmable gate array (FPGA). However, a unified tool allowing one to generate efficient embedded MPC for an FPGA, starting from the definition of the plant and its constraints, was still missing. The MOBY-DIC2 toolbox described in this brief bridges this gap. To illustrate its functionalities, the tool is exploited to embed the controller and observer for a real buck power converter in an FPGA. This implementation achieves a latency of about 30 $\mu$s with the implicit controller and 240 ns with the approximate explicit controller.**

***Index Terms*— Embedded model predictive control (MPC), field programmable gate array (FPGA), hardware–software co-design.**

## I. Introduction

LINEAR model predictive control (MPC) [1], [2] is a model-based approach for the regulation and tracking of linear systems with multiple inputs and outputs, subject to affine inequality constraints. The traditional (implicit) formulation requires solving a quadratic programming (QP) optimization problem at each sampling time, which can be time-consuming and then not applicable for systems with very fast dynamics [3]. For piecewise-affine (PWA) systems subject to mixed linear/logical constraints, the MPC optimization problem can be formulated as a mixed-integer QP [4]. Explicit MPC [5] moves most of the computation offline, and the online computation of the control action (both for linear and PWA systems) reduces to the evaluation of a PWA function, which significantly reduces the computational effort. Many approximate explicit control functions have also been proposed [6], [7], [8], [9], [10], which can contribute to further increase the controller update rate. The common drawback of all the explicit solutions is the rapid growth of the number of parameters as the dimension of the system increases, due to the so-called *curse of dimensionality* [11]. Some solutions have been proposed in the last decade for fast and efficient implementation of implicit [12], [13], [14] and explicit [15], [16], [17], [18] MPC on different platforms, including microcontrollers and field programmable gate arrays (FPGAs).

In the control literature, the implementation aspects and the corresponding support tools for creating real-time embedded controllers receive rather limited attention, while for practical applications these are of course crucial. In this brief, we present a second release of the MOBY-DIC[1] toolbox (briefly, MOBY-DIC2),[2] which extends the functionalities of the former version (MOBY-DIC1) [19]. This toolbox supports the co-design process of a controller and its digital implementation, enabling easy deployment of MPC-based control solutions for real-time applications. MOBY-DIC2 is a MATLAB toolbox based on object-oriented programming, which allows carrying out the *entire* MPC design flow, from the definition of system dynamics and constraints to the embedded architectures, including implicit/explicit, exact/approximate MPC, and supporting both the microcontroller and FPGA implementations of the controllers and observers. The main strength of MOBY-DIC2 is the automatic generation of efficient VHDL[3] code, by setting the level of parallelism, the number of bits for the fixed-point representation of inputs, outputs, and coefficients, and the scalings to automatically interface with analog-to-digital and digital-to-analog converters (ADCs and DACs). To demonstrate all the functionalities of MOBY-DIC2, embedded architectures implementing implicit, explicit, approximate MPC controllers, and Kalman filters are generated for a switching buck converter. Processor-in-the-loop (PIL) simulations and tests on a *real* converter prototype are provided, showing the strengths and capabilities of the tool.

## II. Survey on Available Tools

Several free and commercial tools are available to help the user in designing and/or implementing linear

Manuscript received 26 July 2022; revised 14 February 2023; accepted 3 March 2023. Recommended by Associate Editor U. V. Kalabic. *(Corresponding author: Alberto Oliveri.)*

Alessandro Ravera, Alberto Oliveri, Matteo Lodi, and Marco Storace are with the Department of Electrical, Electronic, Telecommunications Engineering and Naval Architecture, University of Genoa, 16145 Genoa, Italy (e-mail: alessandro.ravera@edu.unige.it; alberto.oliveri@unige.it; matteo.lodi@unige.it; marco.storace@unige.it).

Alberto Bemporad is with the IMT School for Advanced Studies Lucca, 55100 Lucca, Italy (e-mail: alberto.bemporad@imtlucca.it).

W. P. M. H. Heemels is with the Control Systems Technology Section, Department of Mechanical Engineering, Technische Universiteit Eindhoven, 5600 MB Eindhoven, The Netherlands (e-mail: m.heemels@tue.nl).

Eric C. Kerrigan is with the Department of Electrical and Electronic Engineering, Department of Aeronautics, Imperial College London, SW7 2AZ London, U.K. (e-mail: e.kerrigan@imperial.ac.uk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCST.2023.3254133.

Digital Object Identifier 10.1109/TCST.2023.3254133

[1] Acronym of the FP7 European Project "MOdel-Based sYnthesis of DIgital electronic Circuits for embedded control" FP7-ICT-2009-4 (2010–2012).

[2] Available online at https://github.com/COMPsys-UNIGE/MOBY-DIC

[3] Very high-speed integrated circuit hardware description language.

| | CVXGEN,FiOrdOs,OSQP FORCES*,qpOASES, ODYS QP*,QPGEN | PROTOIP | SPLIT | µAO-MPC | MPT3 | Hybrid Toolbox | ODYS QP Embedded MPC* | MOBY-DIC1 | MPC Toolbox* | MOBY-DIC2 |
|---|---|---|---|---|---|---|---|---|---|---|
| System and constraints | | | | X | X | X | X | X | X | X |
| Implicit MPC | | | X | X | X | X | X | | X | X |
| QP problem of implicit MPC | X | | X | X | X | X | X | | X | X |
| Explicit MPC | | | | | X | X | | X | X | X |
| Approximate explicit MPC | | | | | | | | X | X | X |
| MPC for PWA systems | | | | | X | X | | | | X (switched) |
| State estimation | | | | | | | X | | X | X |
| C code | X | X | X | X | | X | X | | X | X |
| VHDL code | | X | | | | | | X | | X |
| PIL simulations | | X | | | | | X | X | X | X |

Fig. 1. Functionalities of available tools. Commercial tools are marked with an asterisk. MPC toolbox is part of MATLAB.

implicit/explicit MPC. The main ones are summarized in Fig. 1, which lists all the steps necessary to generate and simulate an embedded MPC controller. Starting from the definition of the system and constraints, an implicit MPC controller can be designed, for linear or PWA systems. An explicit (either exact or approximate) solution can then be computed. When the system state is not completely measurable, a state observer must also be designed. C and VHDL code must then be created for implementation of the controller and observer on a microcontroller or an FPGA, respectively, or to perform PIL simulations involving a computer running the plant model (not in real-time) and an embedded device implementing the controller and/or the observer. We now briefly describe the tools listed in Fig. 1 from left to right.

The following tools generate C code that can be exploited for microcontroller implementation of QP solvers: CVXGEN [20], FiOrdOs [21], FORCES [21], qpOASES [22], QPGEN [23], ODYS QP Solver [24], and OSQP [25].

PROTOIP [26] provides a MATLAB interface to Xilinx Vitis high-level synthesis (HLS), allowing for the generation of an embedded circuit on a heterogeneous (microcontroller + FPGA) computing platform (in particular, Xilinx Zynq System on Chip) starting from an algorithm described in the C language. PROTOIP allows performing PIL simulations, using MATLAB and a Zynq board, and evaluating the circuit performances within MATLAB environment.

SPLIT [27] is a tool with a MATLAB interface for the generation of embedded implicit linear MPC, based on splitting algorithms, including the alternating direction method of multipliers (ADMM). Based on the matrices and constraints defining the QP optimization problem, a C code is generated for implementation on the microcontroller or, through PROTOIP [26], on heterogeneous computing platforms.

$\mu$AO-MPC [28] is a Python tool (with MATLAB interface) that allows generating C code for embedded implicit MPC starting from the definition of the system to regulate.

The hybrid toolbox [29] and multiparametric toolbox (MPT3) [30] are MATLAB tools that allow generating explicit MPC for linear and PWA systems. The hybrid toolbox also provides C files that can be deployed on the microcontroller.

ODYS Embedded MPC [31] uses a specific version of ODYS QP Solver [24] optimized for MPC applications and allows generating C code with industrial standards for very fast embedded implementation of implicit linear, linear time-varying, and nonlinear MPC and related state observers. PIL simulations can also be carried out.

MOBY-DIC1 allows designing, simulating, and implementing on FPGA explicit exact and approximate MPC, whereas the MATLAB MPC Toolbox [32] allows auto-generating C code from Embedded MATLAB functions for microcontroller implementation of both implicit and explicit (exact and approximate) MPC, based on the multiparametric solver described in [33]. State estimators can also be designed and implemented, and PIL simulations are available. However, the MPC toolbox does not support code generation for FPGA implementation. MOBY-DIC2 fills this gap, by extending the functionalities of MOBY-DIC1. It embeds a top-level interface for the following tools, whose invocation is transparent to the user: MPT3 for the design of explicit MPC, Xilinx Vitis HLS for the generation of VHDL code of the implicit MPC, and Xilinx model composer and system generator for the PIL simulations, including Simulink and an FPGA. MOBY-DIC2 is compatible with both MPT3 and MPC toolbox, since an MPC controller previously generated with one of these tools can be easily imported in MOBY-DIC2 for FPGA implementation or PIL simulation.

## III. Linear MPC

In this section, we briefly describe the formulation of MPC [1], [2] used within the MOBY-DIC2 toolbox for linear time-invariant (LTI) systems. We consider a dynamical system whose states, inputs, measurable parameters, unmeasurable disturbances, and outputs are denoted as $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, $p \in \mathbb{R}^{n_p}$, $d \in \mathbb{R}^{n_d}$, and $y \in \mathbb{R}^{n_y}$, respectively. The MPC controller aims at bringing the system state or output to a constant (regulation problem) or time-varying (tracking problem) reference $r$. All these variables are gathered in a vector $z = [x^T \ u^T \ p^T \ d^T \ y^T \ r^T]^T$. We use a subscript $k$ to indicate the value of a variable at a discrete time instant $kT_s$, where $T_s$ is the sampling period, and a subscript $k+i|k$ to indicate the predicted value of a variable at time $(k+i)T_s$, based on its value at time $kT_s$, $k \in \mathbb{N}$, $i \in \mathbb{N}$. Starting from measurements or estimations of $x_k$, $p_k$, $d_k$, and $r_k, \ldots, r_{k+N}$ (only for tracking problems), the MOBY-DIC2 toolbox allows generating linear MPC by solving the following optimization problem with a receding horizon approach:

$$\min_{\mathcal{E}_k} \quad e_{k+N|k}^T P e_{k+N|k} + \tag{1a}$$

$$+ \sum_{i=0}^{N-1} \left( e_{k+i|k}^T Q e_{k+i|k} + \epsilon_{k+i}^T R \epsilon_{k+i} \right) + \sum_{i=0}^{N_c} \rho \sigma_i^T \sigma_i$$

$$\text{s.t.} \quad x_{k+i+1|k} = A x_{k+i|k} + B u_{k+i} + E_x p_k + F_x d_k + G_x$$

$$(i = 0, \dots, N-1) \tag{1b}$$

$$y_{k+i|k} = Cx_{k+i|k} + Du_{k+i} + E_y p_k + F_y d_k + G_y$$

$$(i = 0, \dots, N) \tag{1c}$$

$$H_i z^T_{k+i|k} \leq K_i + \Omega_i \sigma_i \quad (i = 0, \dots, N_c) \tag{1d}$$

$$u_{k+i} = \Gamma_1 x_{k+i|k} + \Gamma_2 \quad (i = N_u, \dots, N-1). \tag{1e}$$

Within MOBY-DIC2, $p$ and $d$ are measured/estimated at every sampling instant and assumed to remain constant within the prediction horizon (i.e., $p_{k+i|k} = p_k$, and the same holds for $d$). In this optimization problem, expression (1a) contains the quadratic cost function to be minimized at time $kT_s$, where $\mathcal{E}_k = [u_k^T, \dots, u_{k+N_u-1}^T, x_{k|k}^T, \dots, x_{k+N|k}^T, \sigma_0^T, \dots, \sigma_{N_c}^T]^T$; $N$ is the prediction horizon, $N_u \leq N$ is the control horizon, $N_c \leq N$ is the time horizon where the constraints are imposed, $Q = Q^T$ and $P = P^T$ are the positive-semidefinite weight matrices, and $R = R^T$ is a positive definite weight matrix. The terms $\sigma_i$ $(i = 0, \dots, N_c)$ are slack vectors of soft constraints. We have $e_{k+i|k} := x_{k+i|k} - r_{k+i}$ for state regulation/tracking problems, whereas $e_{k+i|k} := y_{k+i|k} - r_{k+i}$ for output regulation/tracking. Finally, $\epsilon_{k+i} := u_{k+i} - r_u$, where $r_u$ is the input reference. For tracking problems, an augmented model is automatically generated to achieve offset-free tracking [34], in which $u_{k-1}$ becomes a system state with dynamics $u_k = u_{k-1} + \Delta u_k$, where $\Delta u_k$ is the new system input. Therefore, $r_u = 0$.

The affine time-invariant prediction model is given in (1b)–(1c), whereas inequality (1d) represents the affine inequality constraints at each time $(k+i)T_s$. Matrix $\Omega_i$ is diagonal and contains zeros or ones, indicating whether the corresponding constraint is *hard* or *soft*, respectively. Finally, (1e) fixes the input (control) for time instants outside the control horizon.

Within MOBY-DIC2 toolbox, MPC and state observers can also be designed and implemented for PWA systems, with a *switching* approach, i.e., a different controller/observer is designed for each system dynamics [35], [36].

## IV. MOBY-DIC2 Functionalities

Fig. 2 shows the main design flow of the MOBY-DIC2 toolbox for LTI systems. The green boxes represent the toolbox inputs, whereas the orange ones the outputs.

### A. Controller Design

The first step for MPC design is to define the prediction model (1b)–(1c) and the affine inequality constraints. The model can be represented either in discrete or continuous time. In the latter case, it is automatically converted into a discrete-time model with a sampling time $T_s$, specified by the user. If the controller latency is higher than $T_s$, the update time $T_{\text{ctrl}}$ of the controller can be set as a multiple of $T_s$. This means that only for the prediction phase, the LTI system is re-discretized with sampling time $T_{\text{ctrl}}$. Both hard and soft constraints can be imposed for any variable at any prediction instant. The MPC parameters appearing in problem (1) are then set within a MATLAB structure, and an object representing either an implicit or an explicit MPC controller is created. We remark that the controllers can also be created using MPT3 or MPC toolbox and then converted into MOBY-DIC2 objects.

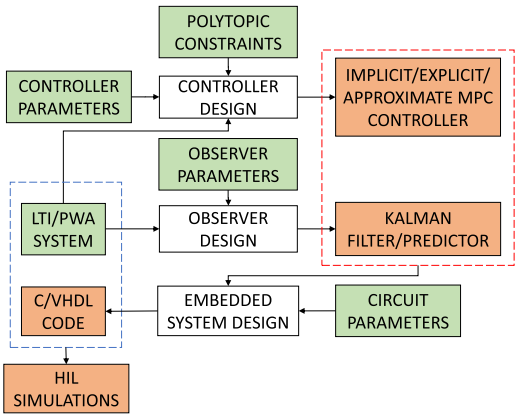



Fig. 2. MOBY-DIC2 project flow. Green boxes: toolbox inputs; orange boxes: outputs.

With the implicit controller, problem (1) is recast as a standard constrained QP with a sparse formulation, i.e.,

$$\min_{\mathcal{E}_k} \ \frac{1}{2}\mathcal{E}_k^T F \mathcal{E}_k + f^T \mathcal{E}_k$$
$$\text{s.t. } A_{\text{in}}\mathcal{E}_k \leq b_{\text{in}}, \quad A_{\text{eq}}\mathcal{E}_k = b_{\text{eq}}. \tag{2}$$

Additional details regarding the construction of the matrices can be found in [37]. MOBY-DIC2 solves problem (2) at each control interval $T_{\text{ctrl}}$ by resorting to either the MATLAB function `quadprog` or the ADMM algorithm [38]. As better described in the following sections, code generation is supported only for ADMM, whereas `quadprog` can be used for software simulation purposes.

The explicit MPC controller is generated by invoking the MPT3 toolbox, which solves problem (1) through multiparametric QP. Then, the control action $u_k$ is represented as $u_k = f_{\text{pwa}}(x_k, p_k, d_k, r_k, \dots, r_{k+N})$, where $f_{\text{pwa}}$ is a PWA function defined over a generic polytopic partition, like the one shown in Fig. 3 (left). MOBY-DIC2 allows also approximating the function $f_{\text{pwa}}$ with another PWA function defined over a regular simplicial domain partition [6] (Fig. 3, right), to obtain lower computational effort and allowing to decrease the system sampling time. An object representing an approximate MPC controller can be created by specifying the number of partitions for each domain dimension.

### B. Observer Design

Kalman filters and predictors can be easily created to estimate the system state and unmeasurable disturbance of an LTI system, by providing a process covariance matrix $Q_K$, a measurement covariance matrix $R_K$, and an observer sampling time $T_{\text{obs}}$, which, for the reasons explained in Section IV-A, can be a multiple of the system sampling time $T_s$. Summarizing, the relationship between the system sampling time $T_s$, observer sampling time $T_{\text{obs}}$, and controller sampling time $T_{\text{ctrl}}$ is $T_{\text{ctrl}} = \alpha T_{\text{obs}} = \alpha\beta T_s$, where $\alpha$ and $\beta$ are positive integer numbers. The Kalman observers are designed through the MATLAB function `kalman`, to obtain

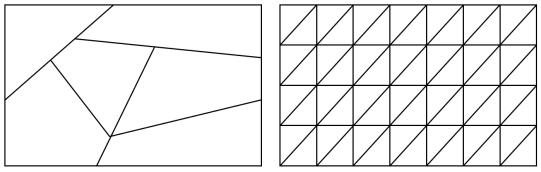


Fig. 3. Two-dimensional domain partitioned into generic polytopes (left) and regular simplices (right), with seven and four partitions along the first and second dimensions, respectively.

either a current (Kalman filter) or a delayed (Kalman predictor) estimator. In the first case, the estimation at time $k$ is also based on measurements at time $k$; in the second case, only measurements up to time $k-1$ are used.

### C. Embedded System Design

This functionality represents the main strength of the tool. The designed controller and observer can be set as internal properties of the LTI system object, to create a closed-loop system including the Kalman filter. An embedded system object can then be created, representing the circuit to be implemented in a microcontroller or FPGA. The block scheme of the embedded system is shown in Fig. 4; the green rectangle represents the hardware architecture generated by MOBY-DIC2, to be implemented in an FPGA or microcontroller (red rectangle). The controller and the observer can also be implemented individually. The circuit is thought to get the reference $r_k, \ldots, r_{k+N}$ (in case of tracking problems), the parameter $p_k$, and the outputs $y_k$ through ADCs in a given range, which can be set within MOBY-DIC2, typically [0 4095] for 12-bit ADCs. These values are converted into the correct range through linear scalings, performed by `SCALE` blocks, and provided to the controller and observer, as shown in the figure. The control output $u_k$ is scaled to the DAC range.

For implicit MPC, the ADMM algorithm [38] is exploited by the circuit. ADMM steps are simple to implement and computationally very cheap, which makes this algorithm particularly suitable for implementation in embedded devices with limited computing resources [12]. The main linear algebraic operation in each step is matrix–vector multiplication. In addition, there is no data dependence between the optimization variables in an iteration; this allows for the processing of each optimization variable in parallel, thus making ADMM particularly suitable for FPGA implementations [38], [39], [40]. Also, QPGEN, OSQP, and SPLIT tools use the ADMM algorithm for the generation of C code. For FPGA implementation, two architectures can be generated: a fast one, which exploits parallelism to perform matrix–vector products, thus achieving a low latency at the cost of more hardware resources, and a small one, with higher latency and lower resource utilization. A fixed-point representation of data is chosen, where the user can set the number of bits. Also, the number of bits of ADC and DAC can be arbitrarily set. The toolbox creates C++ code, which is automatically converted into VHDL code through Vitis HLS, without any user intervention. In this case, also the VHDL code of the observer is generated through Vitis HLS.

For explicit and approximate MPC, the circuit architectures described in [16] are implemented. Also in this case, for FPGA

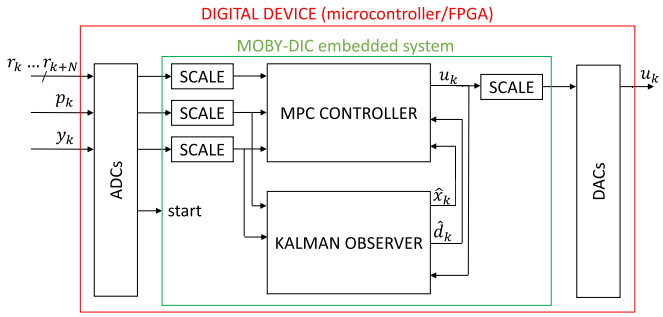



Fig. 4. Block scheme of the MOBY-DIC2 embedded system (green box) to be implemented on a digital device (red box).

implementation, a serial architecture and a parallel architecture are available, to trade off circuit size and latency. In this case, the code is generated directly in VHDL language.

C code can also be generated for microcontroller implementation of all kinds of controllers and observers, by exploiting the same algorithms used for the FPGA, with a floating point data representation. In this case, it is not possible to choose between serial (small) and parallel (fast) architectures.

### D. Simulations

The MOBY-DIC2 toolbox allows performing different kinds of simulations of the closed-loop systems including all types of controllers and observers. The simplest simulation runs directly in MATLAB. The plots of the time evolution of all the measured and estimated variables are automatically generated. A Simulink model can be created for interactive simulations, where references and parameters can be varied online. Also, the plant model can be replaced by a more accurate one, which is often useful (as in the example shown in this brief) since the prediction model used to design the MPC controller may be simpler than the plant model used for simulation. MOBY-DIC2 interfaces also with Xilinx Model Composer and System Generator to generate the Simulink models for PIL simulations, as the one shown in Fig. 5, where the embedded system (controller and observer) runs on an FPGA connected to the computer. The scaling blocks for adapting the signals to the ADC and DAC ranges are also generated. This allows evaluating the effects of delays and quantization errors due to ADCs and DACs and fixed-point representation of data. The snippet of code in Fig. 6 shows how the main MOBY-DIC2 functionalities can be invoked with a few commands.

## V. Case Study

### A. Experimental Results on a Buck Converter

The MOBY-DIC2 toolbox was used to generate different embedded systems (including different types of MPC controllers and a Kalman predictor) to be implemented on FPGA to regulate a real buck converter. A buck converter (see Fig. 7) is a switching DC–DC converter that allows transforming an input DC voltage $v_{\text{in}}$ to a lower DC output voltage $v_{\text{out}}$. The circuit is composed of an inductor with voltage $v$ and current $i$, a diode with voltage drop $v_d$, an MOS transistor operating as a switch, an output capacitor, and a load absorbing a current $i_{\text{out}}$. $R_L$ indicates the parasitic resistance of the inductor. The

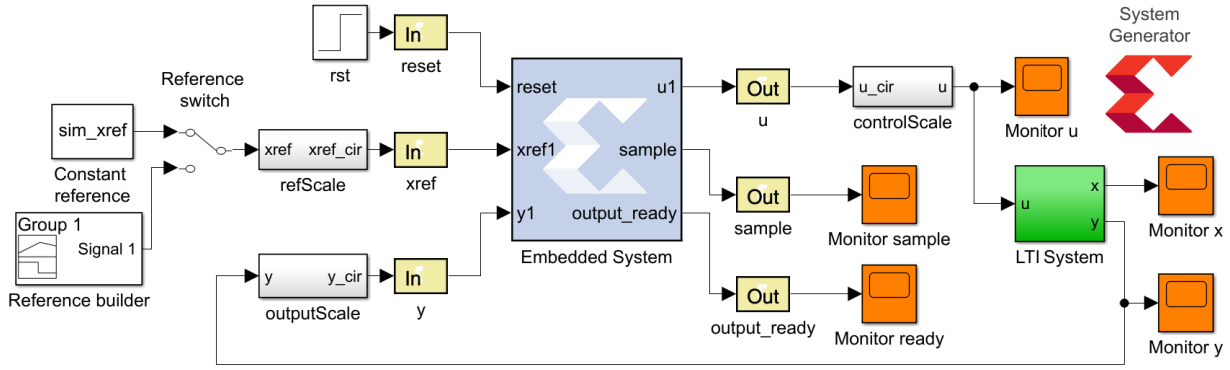



Fig. 5. Automatically generated Simulink model for PIL simulation.

```
% Create continuous time system
ctsys = ltiSys(nx, nu, ny, 'ct');
ctsys = ctsys.SetMatrices('A', A);
ctsys = ctsys.SetMatrices('B', B);
% Create constraints
constr = constraints(nx, nu, ny, Nc);
constr = constr.setConstraints('u', umin, umax, t);
constr = constr.setConstraints('x', xmin, xmax, t);
% Create implicit, explicit and approximate MPC objects
opts = struct('P', P, 'Q', Q, 'R', R, 'N', N, ...);
impl_ctrl = implicitMPCctrl(ctsys, Ts, constr, opts);
exp_ctrl = explicitMPCctrl(ctsys, Ts, constr, opts);
appr_ctrl = ApproxMPCctrl(exp_ctrl, appr_opts);
% Create Kalman filter
kal_filt = kalmanFilter(ctsys, Tobs, QK, RK);
% Associate controller and observer to system
ctsys = ctsys.setController(impl_ctrl);
ctsys = ctsys.setObserver(kal_filt);
% Generate Simulink model for simulation
ctsys.generateSimulinkModel();
% Create embedded system
es = embeddedSystem(ctsys, range);
% Generate C code
es.generateC();
% Generate VHDL code with circuit options
es.generateVHDL(vhdl_opts);
```

Fig. 6. Portion of the MOBY-DIC2 MATLAB code.

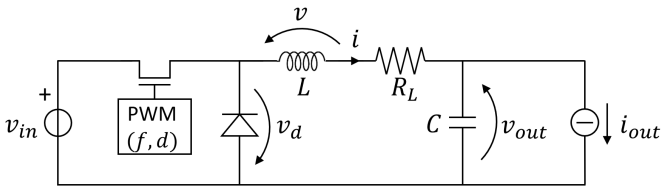



Fig. 7. Circuit model of a DC–DC buck converter.

MOS transistor is driven by a pulsewidth modulation (PWM) signal with frequency $f$ and duty cycle $\delta \in [0, 1]$.

The buck converter is a switching system, consisting of two states: when the PWM signal is high, the MOS conducts current (*on* phase); when the PWM signal is low, the MOS does not conduct (*off* phase). We assume that the converter always works in continuous conduction mode, i.e., with $i > 0$, with state equations as follows:

$$\begin{cases} \dfrac{di}{dt} = \dfrac{s(v_{\text{in}} + v_d) - v_d - R_L i - v_{\text{out}}}{L} \\ \dfrac{dv_{\text{out}}}{dt} = \dfrac{i - i_{\text{out}}}{C} \end{cases} \tag{3}$$

TABLE I
CONVERTER PARAMETERS AND CONSTRAINTS

| Name | Value | Name | Value | Name | Value |
|---|---|---|---|---|---|
| $f$ | 30 kHz | $C$ | 1320 µF | $v_{in}$ | 6 V |
| $L$ | 47 µH | $v_d$ | 0.9 V | $R_L$ | 400 mΩ |
| $\delta_{min}$ | 0.2 | $\delta_{max}$ | 0.8 | $i_{max}$ | 3 A |

where $s = 1$ in on phase and 0 in off phase. The goal of the controller is to set the duty cycle $\delta$ of the switch such that the average value of the output voltage (which is subject to some ripple) tracks a reference value $\bar{v}_{\text{out,ref}}$, by fulfilling hard constraints $i \le i_{\max}$ and $\delta_{\min} \le \delta \le \delta_{\max}$. The values of parameters and constraints are summarized in Table I.

The nonlinear model (3) can be used to simulate the converter, but is not suitable as a prediction model for linear MPC. To this aim, a linear averaged model can be derived [41], [42]. By considering an augmented system, the averaged linear model can be written in discrete-time version as follows:

$$\begin{bmatrix} \bar{\imath}_{k+1} \\ \bar{v}_{out,k+1} \\ \delta_k \end{bmatrix} = \begin{bmatrix} 1 + \dfrac{T_s R_L}{L} & -T_s/L & \dfrac{T_s(v_{\text{in}} + v_d)}{L} \\ \dfrac{T_s}{C} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{\imath}_k \\ \bar{v}_{out,k} \\ \delta_{k-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Delta\delta_k + \begin{bmatrix} 0 \\ -\dfrac{T_s}{C} \\ 0 \end{bmatrix} i_{out,k} + \begin{bmatrix} -\dfrac{T_s v_d}{L} \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

where $T_s = 1/f \approx 33.3\ \mu$s.

By assuming $v_{\text{in}}$ as a constant, this model is in the same form as model (1b)–(1c), where $x = [\bar{\imath}\ \bar{v}_{\text{out}}\ \delta]$, $u = \Delta\delta$, $p = i_{\text{out}}$, and $y = \bar{v}_{\text{out}}$, which can be assumed to be equal to $v_{\text{out}}$ (small-ripple approximation [42]) and then easily measured on the circuit. The average current $\bar{\imath}$ must be instead estimated through a Kalman filter.

The inductor current $i$ does not appear in model (4), and then the constraint on the inductor current must be reformulated in terms of the variables available in the prediction model. The inductor current is subject to a ripple, which can be evaluated, based on circuit equations, as $\delta(v_{\text{in}} - v_{\text{out}})/(2fL)$ [42]. The constraint $i \le i_{\max}$ can then be approximated as $\bar{\imath} + \delta(v_{\text{in}} - \bar{v}_{\text{out}})/(2fL) \le i_{\max}$. If $\delta$ is replaced by its maximum value $\delta_{\max}$, the constraint becomes more conservative, but can be written as in expression (1d).

TABLE II
CIRCUIT RESOURCE USAGE, LATENCY, AND POWER WITH ALL DIFFERENT CONTROLLERS

| | LUT | FF | BRAM | DSP | latency | power |
|---|---|---|---|---|---|---|
| explicit | 560 (1.05%) | 517 (0.49%) | 0 (0%) | 5 (2.27%) | 400 ns | 224 mW |
| approx. explicit | 881 (1.66%) | 680 (0.64%) | 9 (6.43%) | 6 (2.73%) | 240 ns | 227 mW |
| implicit (fast) | 11433 (21.49%) | 10341 (9.72%) | 1.50 (1.07%) | 99 (45%) | 29.34 µs | 496 mW |
| implicit (small) | 4836 (9.09%) | 10371 (9.75%) | 1.50 (1.07%) | 16 (7.27%) | 845.42 µs | 257 mW |

This state constraint is imposed only at the prediction instant $(k+1)T_s$, whereas input constraint $\delta_{\min} \leq \delta \leq \delta_{\max}$ is imposed only at the current instant $kT_s$.

The tracking problem can now be formulated as in (1), by setting $N = 5$, $N_u = 3$, $N_c = 1$, $\Gamma_1 = [0\ 0\ 1]$, $\Gamma_2 = 0$, $P = Q = \text{diag}(0, 2, 0)$, and $R = 1$.

Implicit, explicit, and approximate MPC controllers were designed with $T_{\text{ctrl}} = T_s$, i.e., with $\alpha = \beta = 1$. The PWA function defining the explicit controller is defined over a 5-D domain partitioned into six polytopes. The approximate controller was generated by partitioning the domain into regular simplices, with nine partitions along the state and reference dimensions and one partition along the parameter dimension.

The Kalman predictor for the estimation of $\bar{\imath}$ was designed with $T_{\text{obs}} = T_s$ starting from the averaged model (4), by setting a process covariance matrix $Q_K = \text{diag}(0.1, 0.1)$ and a measurement covariance $R_K = 0.1$.

### B. Circuit Design

Three embedded system objects were created, including the Kalman predictor and the three designed MPC controllers. For each of them, available circuits were generated and implemented on a Zynq-7000 XC7Z020-1CLG484C FPGA, with a clock frequency of 50 MHz.

Explicit controllers use a 12-bit fixed-point representation, implicit controllers use an 18-bit fixed-point representation, whereas 12 bits were set for ADC and DAC. Table II shows the number of used look-up tables (LUTs), flip-flops (FFs), block random access memories (BRAM), and digital signal processors (DSPs) for all the circuit architectures. The percent values are relative to the total available resources in the Zynq board. Also, the circuit latency and estimated average power are listed. As expected, the approximate explicit MPC controller has the lowest latency, about two orders of magnitude lower than the sampling time $T_s$, with a very low resource utilization. An even smaller, but slower, circuit is obtained with the explicit MPC controller. Due to the high level of parallelism, the latency of the fast implicit solution is still lower than $T_s$ at the cost of a much higher resource occupation (especially DSP) and power consumption. If parallelism is not exploited (see the last row of Table II), the latency is much higher than $T_s$ and the architecture cannot be exploited in real experiments. We remark that the latency value shown in the table is only related to the embedded system computation; a further delay of 1.46 µs must be added due to the ADC conversion of measurements.

### C. PIL Simulations

A Simulink model was generated with MOBY-DIC, combined with the Xilinx Model Composer and System Generator, to perform PIL simulations (see Fig. 5). The state-space model automatically generated by MOBY-DIC2 is the linear prediction model used for MPC design. We replaced it with the nonlinear model (3) to get a more accurate simulation of the converter. The Zynq board implementing the implicit fast architecture was connected to the PC, and the simulation results are shown in Fig. 8. The top panel of the figure shows the time evolution of $v_{\text{out}}$ (blue), the value of $\bar{v}_{\text{out}}$ estimated by the observer (red), and the reference output voltage (black, dashed). The middle panel shows $i$ (blue), the estimated value of $\bar{\imath}$ (red), and the bound $i_{\max}$ (gray). Note that the current ripple visible in the blue profile is not estimated by the observer, which is designed based on the averaged model. The black dashed line is the output current $i_{\text{out}}$. The bottom panel shows the evolution of $\delta$ (blue) and its bounds (gray).

The digital circuit correctly estimates $\bar{\imath}$ and $\bar{v}_{\text{out}}$ through the Kalman predictor and regulates $v_{\text{out}}$ to the variable reference $v_{\text{out,ref}}$, even if the parameter $i_{\text{out}}$ changes. At the beginning, the controller brings $v_{\text{out}}$ to 1.8 V in about 4 ms. After 7.5 ms, the reference voltage is set to 3.3 V (left black boxes), and after 12.5 ms, the output current $i_{\text{out}}$ is varied from 0.8 to 1.2 A (right black boxes). The constraints are always satisfied.

### D. Experiments

The FPGA implementing the fast implicit controller was connected to a real buck converter and we used the same scenario as for the PIL simulation. The constant input voltage is applied through a BREMI BRS 55 power supply, whereas the time-varying output current is set through a PEL 3031E DC electronic load. The output voltage and load current are acquired through ADCs, whereas the output voltage reference is changed from 1.8 to 3.3 V through a switch in the Zynq board and is, therefore, an internal digital signal. The controller's output $\delta$ is provided to a block described in VHDL code generating the PWM signal to be applied to the MOS of the buck converter. An LTS 6-NP Hall-effect probe was used to acquire the inductor current through an RIGOL DS1000Z oscilloscope, together with the output voltage $v_{\text{out}}$ and the PWM signal. Fig. 9 shows portions of the experimental results corresponding to the black rectangles in Fig. 8, i.e., during the changes in operating conditions. The experiment was repeated using two different control systems generated by MOBY-DIC2 and implemented on the FPGA: in one case, implicit MPC was used, and in the other, approximate explicit MPC was used. A Kalman predictor generated by MOBY-DIC2 was also used. The top panels show $v_{\text{out}}$ and $v_{\text{out,ref}}$, the middle panels $i$, $i_{\text{out}}$ and $i_{\max}$, and the bottom panels $\delta$, which is reconstructed in MATLAB starting from the measured PWM signal and its bounds. Note that in all the cases, the regulation is successful and the measured results are similar to the simulated ones.

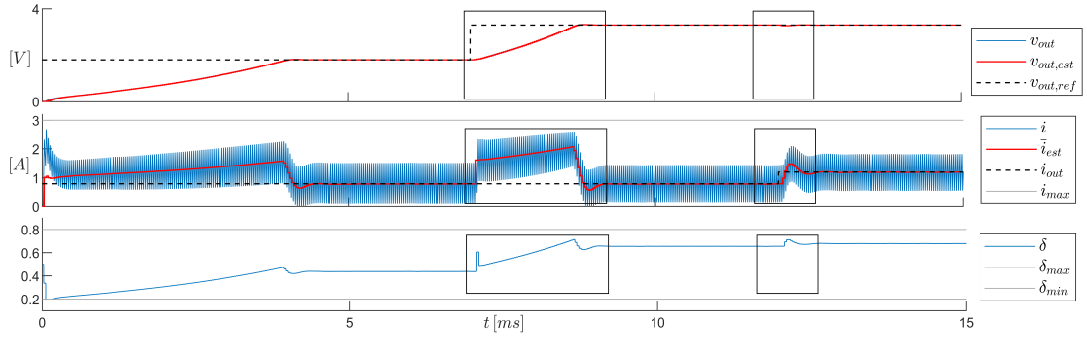
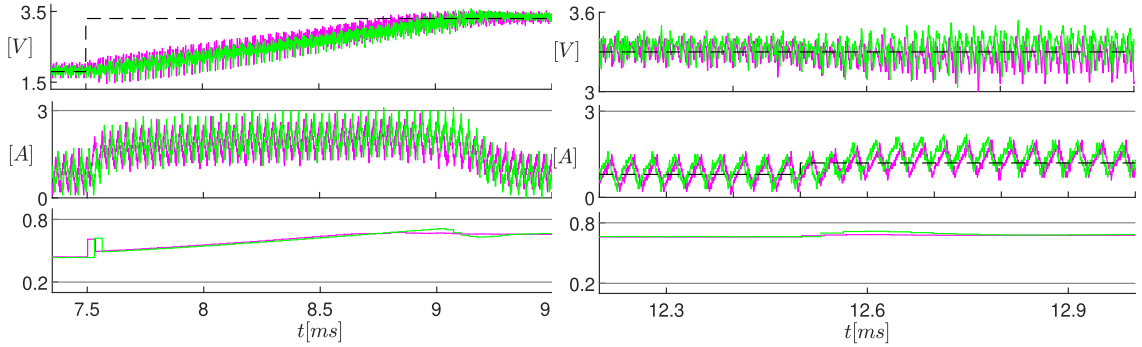



Fig. 8. PIL simulation results with implicit MPC: output voltage (top), inductor current (middle), and duty cycle (bottom).




(a) case 1 (b) case 2

Fig. 9. Experimental results with implicit (green) and approximate MPC (magenta). Top panels: output voltage $v_{\text{out}}$ and its reference $v_{\text{out,ref}}$ (black dashed); middle panels: inductor current $i$, output current $i_{\text{out}}$ (black dashed), and constraint $i_{\text{max}}$ (gray); bottom panels: duty cycle $\delta$ and its bounds (gray). Cases 1 (left panels) and 2 (right panels) correspond to the black rectangles in Fig. 8. (a) Case 1. (b) Case 2.

## VI. Conclusion

Controllers based on implicit, explicit, and approximate MPC, including a state observer, were designed and implemented on a Zynq-7000 FPGA for the regulation of a real switching buck converter. Explicit MPC controllers need limited hardware resources and achieve latencies of hundreds of nanoseconds, whereas the parallel implicit version requires much more resources (especially DSPs) with a latency of tens of microseconds. A smaller, yet too slow, circuit implementing implicit MPC has also been designed.

Without MOBY-DIC2, the same experiments would have required multiple tools (e.g., the MPC toolbox and PROTOIP), which are not guaranteed to be fully compatible and possibly require some manual modification to the generated code. MOBY-DIC2 allowed instead to carry out the whole design, simulation, and implementation process, by generating different control strategies and circuit architectures and comparing their performances. MOBY-DIC2 has proven to be an effective tool for control-circuit co-design and co-simulation.

Different QP solvers can possibly be added to MOBY-DIC2 for simulation and/or implementation purposes. The functionalities could also be extended to nonlinear MPC, which is possible due the fact that several circuit architectures are already available in the literature for the solution of nonlinear constrained optimization problems [43], [44], [45].

## Acknowledgment

The authors thank the consortium of European FP7 Project MOBY-DIC (ID: 248858) and Tomaso Poggi who actively worked on the toolbox.

## References

[1] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, Dec. 2104.

[2] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Madison, WI, USA: Nob Hill Publishing, 2017.

[3] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, Mar. 2010.

[4] F. Torrisi and A. Bemporad, "HYSDEL—A tool for generating computational hybrid models," *IEEE Trans. Contr. Syst. Technol.*, vol. 12, no. 2, pp. 235–249, Mar. 2004.

[5] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, Jan. 2002.

[6] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace, "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations," *IEEE Trans. Autom. Control*, vol. 56, no. 12, pp. 2883–2897, Dec. 2011.

[7] B. A. G. Genuit, W. P. M. H. Heemels, and L. Lu, "Approximation of explicit model predictive control using regular piecewise affine functions: An input-to-state stability approach," *IET Control Theory Appl.*, vol. 6, no. 8, pp. 1015–1028, May 2012.

[8] C. Wen, X. Ma, and B. E. Ydstie, "Analytical expression of explicit MPC solution via lattice piecewise-affine function," *Automatica*, vol. 45, no. 4, pp. 910–917, 2009.

[9] L. H. Cseko, M. Kvasnica, and B. Lantos, "Explicit MPC-based RBF neural network controller design with discrete-time actual Kalman filter for semiactive suspension," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 5, pp. 1736–1753, Sep. 2015.

[10] S. Chen et al., "Approximating explicit model predictive control using constrained neural networks," in *Proc. Annu. Amer. Control Conf. (ACC)*, Milwaukee, WI, USA, Jun. 2018, pp. 1520–1527.

[11] R. E. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[12] I. McInerney, G. A. Constantinides, and E. C. Kerrigan, "A survey of the implementation of linear model predictive control on FPGAs," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 381–387, 2018.

[13] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, "Predictive control using an FPGA with application to aircraft control," *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 3, pp. 1006–1017, May 2014.

[14] G. Cimini, D. Bernardini, S. Levijoki, and A. Bemporad, "Embedded model predictive control with certified real-time optimization for synchronous motors," *IEEE Trans. Control Syst. Technol.*, vol. 29, no. 2, pp. 893–900, Mar. 2021.

[15] F. Bayat, T. A. Johansen, and A. A. Jalali, "Combining truncated binary search tree and direct search for flexible piecewise function evaluation for explicit MPC in embedded microcontrollers," *IFAC Proc. Volumes*, vol. 44, no. 1, pp. 1332–1337, Jan. 2011.

[16] A. Oliveri and M. Storace, "Hardware-in-the-loop simulations of circuit architectures for the computation of exact and approximate explicit MPC control functions," in *Proc. IEEE Int. Conf. Circuits Syst.*, Seville, Spain, Dec. 2012, pp. 380–383.

[17] M. Gulan, G. Takacs, N. A. Nguyen, S. Olaru, P. Rodriguez-Ayerbe, and B. Rohal'-Ilkiv, "Efficient embedded model predictive vibration control via convex lifting," *IEEE Trans. Control Syst. Technol.*, vol. 27, no. 1, pp. 48–62, Jan. 2019.

[18] C. Jugade, D. Ingole, D. N. Sonawane, M. Kvasnica, and J. Gustafson, "A memory efficient FPGA implementation of offset-free explicit model predictive controller," *IEEE Trans. Control Syst. Technol.*, vol. 30, no. 6, pp. 2646–2657, Nov. 2022.

[19] A. Oliveri et al., "MOBY-DIC: A MATLAB toolbox for circuit-oriented design of explicit MPC," *IFAC Proc. Volumes*, vol. 45, no. 17, pp. 218–225, 2012.

[20] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, no. 1, pp. 1–27, Mar. 2012.

[21] C. N. Jones, A. Domahidi, M. Morari, S. Richter, F. Ullmann, and M. Zeilinger, "Fast predictive control: Real-time computation and certification," *IFAC Proc. Volumes*, vol. 45, no. 17, pp. 94–98, 2012.

[22] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Math. Program. Comput.*, vol. 6, no. 4, pp. 327–363, 2014.

[23] P. Giselsson and S. Boyd, "Diagonal scaling in Douglas–Rachford splitting and ADMM," in *Proc. IEEE Conf. Decis. Control*, Los Angeles, CA, USA, Dec. 2014, pp. 5033–5039.

[24] G. Cimini, A. Bemporad, and D. Bernardini. (Sep. 2017). *ODYS QP Solver*. [Online]. Available: https://odys.it/qp

[25] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Math. Program. Comput.*, vol. 12, no. 4, pp. 637–672, Dec. 2020.

[26] A. Suardi, E. C. Kerrigan, and G. A. Constantinides, "Fast FPGA prototyping toolbox for embedded optimization," in *Proc. Eur. Control Conf. (ECC)*, Linz, Austria, Jul. 2015, pp. 2589–2594.

[27] H. A. Shukla, B. Khusainov, E. C. Kerrigan, and C. N. Jones, "Software and hardware code generation for predictive control using splitting methods," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14386–14391, 2017.

[28] P. Zometa, M. Kögel, and R. Findeisen, "$\mu$AO-MPC: A free code generation tool for embedded real-time linear model predictive control," in *Proc. Amer. Control Conf.*, Washington, DC, USA, Jun. 2013, pp. 5320–5325.

[29] A. Bemporad. (Dec. 2004). *Hybrid Toolbox—User's Guide*. [Online]. Available: http://cse.lab.imtlucca.it/bemporad/hybrid/toolbox

[30] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *Proc. Eur. Control Conf. (ECC)*, Jul. 2013, pp. 502–510.

[31] ODYS Srl. (Oct. 2019). *ODYS Embedded MPC*. [Online]. Available: https://odys.it/embedded-mpc

[32] A. Bemporad, M. Morari, and N. Ricker, *Model Predictive Control Toolbox for MATLAB*. The Mathworks, Inc. Accessed: 2004. [Online]. Available: http://www.mathworks.com/help/mpc

[33] A. Bemporad, "A multiparametric quadratic programming algorithm with polyhedral computations based on nonnegative least squares," *IEEE Trans. Autom. Control*, vol. 60, no. 11, pp. 2892–2903, Nov. 2015.

[34] G. Pannocchia, "Offset-free tracking MPC: A tutorial review and comparison of different formulations," in *Proc. Eur. Control Conf. (ECC)*, Jul. 2015, pp. 527–532.

[35] S. Di Cairano, H. E. Tseng, D. Bernardini, and A. Bemporad, "Steering vehicle control by switched model predictive control," *IFAC Proc. Volumes*, vol. 43, no. 7, pp. 1–6, Jul. 2010.

[36] A. Oliveri, M. Lodi, and M. Storace, "Design and circuit implementation of approximate switched MPC," in *Proc. Eur. Conf. Circuit Theory* Desing, Dresden, Germany, Sep. 2013, pp. 1–4.

[37] J. M. Maciejowski, *Predictive Control for Linear and Hybrid Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 2007.

[38] T. V. Dang, K. V. Ling, and J. M. Maciejowski, "Embedded ADMM-based QP solver for MPC with polytopic constraints," in *Proc. Eur. Control Conf. (ECC)*, Linz, Austria, Jul. 2015, pp. 3446–3451.

[39] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Trans. Autom. Control*, vol. 59, no. 12, pp. 3238–3251, Dec. 2014.

[40] P. Zhang, J. Zambreno, and P. H. Jones, "An embedded scalable linear model predictive hardware-based controller using ADMM," in *Proc. Int. Conf. Appl.-Specif. Syst. Archit. Process. Proc.*, Seattle, WA, USA, Jul. 2017, pp. 176–183.

[41] R. D. Middlebrook and S. Cuk, "A general unified approach to modelling switching-converter power stages," in *Proc. IEEE Power Electron. Spec. Conf. (PESC)*, Cleveland, OH, USA, Jun. 1976, pp. 18–34.

[42] R. W. Erickson and D. Maksimovic, *Fundamentals of Power Electronics*. New York, NY, USA: Springer, 2007.

[43] F. Xu, H. Chen, X. Gong, and Q. Mei, "Fast nonlinear model predictive control on FPGA using particle swarm optimization," *IEEE Trans. Ind. Electron.*, vol. 63, no. 1, pp. 310–321, Jan. 2016.

[44] A. Raha, A. Chakrabarty, V. Raghunathan, and G. T. Buzzard, "Embedding approximate nonlinear model predictive control at ultrahigh speed and extremely low power," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 3, pp. 1092–1099, May 2020.

[45] D. Tavernini, M. Metzler, P. Gruber, and A. Sorniotti, "Explicit nonlinear model predictive control for electric vehicle traction control," *IEEE Trans. Control Syst. Technol.*, vol. 27, no. 4, pp. 1438–1451, Jul. 2019.


Open Access funding provided by 'Università degli Studi di Genova' within the CRUI CARE Agreement