Real-time FPGA Implementation of Direct MPC for Power Electronics

Bartolomeo Stellato and Paul J. Goulart

***Abstract*— Common approaches for direct model predictive control (MPC) for current reference tracking in power electronics suffer from the high computational complexity encountered when solving integer optimal control problems over long prediction horizons. Recently, an alternative method based on approximate dynamic programming showed that it is possible to reduce the computational burden enabling sampling times under $25\,\mu$s by shortening the MPC horizon to a very small number of stages while improving the overall controller performance. In this paper we implemented this new approach on a small size FPGA and validated it on a variable speed drive system with a three-level voltage source converter. Time measurements showed that only $5.76\,\mu$s are required to run our algorithm for horizon $N = 1$ and $17.27\,\mu$s for $N = 2$ while outperforming state of the art approaches with much longer horizons in terms of currents distortion and switching frequency. To the authors' knowledge, this is the first time direct MPC for current control has been implemented on an FPGA solving the integer optimization problem in real-time and achieving comparable performance to formulations with long prediction horizons.**

I. Introduction

Among the control strategies adopted in power electronics, model predictive control (MPC) [1] has recently gained popularity due to its various advantages [2]. MPC has been shown to outperform traditional control methods because of its ease in handling time-domain constraint specifications and its wide applicability to several power systems topologies and operating conditions compared to traditional approaches.

Direct (*finite control set*) MPC [3] tackles both the control and modulation problems in a single computational stage where the manipulated variables are the discrete switch positions applied to the converter without the need of a modulator. However, the resulting optimization problem is an integer program that is provably $\mathcal{NP}$-hard [4] to solve. In [5] and [6] the authors propose various means of reducing the computational burden of direct MPC by efficiently solving the optimization problem using the *sphere decoding* [7] algorithm. Although this method appears promising relating to previous work, the computation time required to perform the sphere decoding algorithm for long horizons (i.e. $N = 10$), is still far slower than the sampling time typically required, i.e. $T_s = 25\,\mu$s.

In a recent approach described in [8] and [9] the authors address the computational issues of direct MPC by casting the problem in the framework of approximate dynamic programming (ADP) [10]. The infinite horizon value function is approximated using the approach in [11] and [12] by solving a semidefinite program (SDP) [13] offline. In this way the controller horizon can be shorten by applying the estimated tail cost to the last stage while keeping good control performance. Closed loop simulations [8], [9] showed that with the ADP based approach, even short prediction horizons give better performance than the approach in [5] and [6] with much longer planning horizons.

In this work we implemented the ADP based approach on a small size Xilinx Zynq FPGA (xc7z020) in fixed-point arithmetic and verified its performance with hardware in the loop (HIL) tests in both steady state and transients. As a case study, we considered a variable-speed drive system consisting of a three-level neutral point clamped voltage source inverter connected to a medium-voltage induction machine. The plant is modeled as a linear system with a switched three-phase input with equal switching steps for all phases.

The results showed almost identical performance to the closed-loop simulations in [8] and [9] and very fast computation times allowing us to comfortably run our controller within the $25\,\mu$s sampling time.

The remainder of the paper is organized as follows. In Section II we describe the drive system case study. In Section III the direct MPC problem is derived. Section IV the physical model used. In Section V we briefly present closed-loop simulation results from [8], [9] to characterize the achievable performance of the hardware implementation. In Section VI we describe the hardware setup, the algorithm and all the FPGA implementation details. In Section VII HIL tests are performed in steady state and transients operation. Finally, we provide conclusions in Section VIII.

In this work we use normalized quantities by adopting the per unit system (pu). The time scale t is also normalized using the base angular velocity ω_b that in this case is $2\pi \cdot 50\,{}^{\mathrm{rad}}/_{\mathrm{s}}$.

II. Drive System Case Study

In this work we consider a variable speed drive system consisting of a three-level neutral point clamped (NPC) voltage source inverter driving a medium-voltage (MV) induction machine; see [9]. The total dc-link voltage V_{dc} is assumed to be constant and the neutral point potential N fixed.

We define the switch positions in the three-phase legs as integer input variables $\boldsymbol{u}_{sw} = \begin{bmatrix} u_a & u_b & u_c \end{bmatrix}^\top$ with $u_a, u_b, u_c \in \{-1, 0, 1\}$. The model of the drive and the induction motor can be described in terms of the stator currents $\boldsymbol{i}_s$ and the rotor fluxes $\boldsymbol{\psi}_r$ in the $\alpha\beta$ plane using the following discrete-time linear time invariant (LTI) system

$$\begin{aligned} \boldsymbol{x}_{ph}(k+1) &= \boldsymbol{A}_{ph}\boldsymbol{x}_{ph}(k) + \boldsymbol{B}_{ph}\boldsymbol{u}_{sw}(k) \\ \boldsymbol{y}_{ph}(k) &= \boldsymbol{C}_{ph}\boldsymbol{x}_{ph}(k), \end{aligned} \tag{1}$$

This work was supported by the People Programme (Marie Curie Actions) of the European Unions Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 607957 (TEMPO).

B. Stellato and P. J. Goulart are with the University of Oxford, Parks Road, Oxford, OX1 3PJ, U.K. {bartolomeo.stellato, paul.goulart}@eng.ox.ac.uk

where the state vector is $\boldsymbol{x}_{ph} = \begin{bmatrix} i_{s,\alpha} & i_{s,\beta} & \psi_{r,\alpha} & \psi_{r,\beta} \end{bmatrix}^\top$ and the output vector is taken as the stator current, i.e. $\boldsymbol{y}_{ph} = \boldsymbol{i}_s$. The sampling time is $T_s = 25\,\mu$s. See [9] for a detailed derivation.

III. Model Predictive Current Control

A. Problem Description

Our control scheme must address two conflicting objectives simultaneously. On the one hand, the distortion of the stator currents $\boldsymbol{i}_s$ corresponds to ripples in the torque of the motor that are the main source of mechanical stress on the load and the bearings. In order to reduce damage to the machine and the load, the distortion of stator currents must be kept as low as possible. On the other hand, high frequency switching of the inputs $\boldsymbol{u}_{sw}$ produces high power losses and stress on the physical devices. To reduce the energy needed and to preserve the lifespan of the components, we therefore should minimize the switching frequency of the integer inputs.

We measure the current distortion via the total harmonic distortion (THD) that can be minimized by reducing the ripples in the produced currents [9]. Given the ideal reference stator sinusoids $\boldsymbol{i}_s^*$ in the $\alpha\beta$ plane, the ripples can be obtained as the error signal: $\boldsymbol{e} = \boldsymbol{i}_s - \boldsymbol{i}_s^*$. Thus, minimizing the 2-norm of $\boldsymbol{e}$ will minimize the THD. We include the reference currents as two additional purely oscillating uncontrollable states $\boldsymbol{x}_{osc} = \boldsymbol{i}_s^*$ within our model dynamics.

In order to avoid the high computational and storage cost involved in computing online the switching frequency estimate defined as a finite impulse response (FIR) filter with a large time window, we approximate it using an infinite impulse response filter (IIR) modeled as an LTI system; see [9]. Let us define three binary phase inputs denoting the phase switch position changes:

$$\boldsymbol{p}(k) = \begin{bmatrix} p_a(k) & p_b(k) & p_c(k) \end{bmatrix}^\top \in \{0,1\}^3, \tag{2}$$

with $p_s(k) = \|u_s(k) - u_s(k-1)\|_1$, $s \in \{a,b,c\}$. We define the second order filter estimating the switching frequency as

$$\boldsymbol{x}_{flt}(k+1) = \boldsymbol{A}_{flt}\boldsymbol{x}_{flt}(k) + \boldsymbol{B}_{flt}\boldsymbol{p}(k)$$

where $\hat{f}_{sw} = \begin{bmatrix} 0 & 1 \end{bmatrix} \boldsymbol{x}_{flt}(k)$ is the estimated switching frequency. The poles of $\boldsymbol{A}_{flt}$ are defined as $a_1 = 1 - 1/r_1$ and $a_2 = 1 - 1/r_2$ with $r_1, r_2 >> 0$. We define the difference between the approximation $\hat{f}_{sw}(k)$ and the target frequency f_{sw}^* by $e_{sw}(k) := \hat{f}_{sw}(k) - f^*(k)$. Thus, minimizing the 2-norm of e_{sw} will bring the switching frequency estimate as close as possible to the desired one. We can augment the state space to include the filter dynamics and the target frequency by adding the states $\boldsymbol{x}_{sw} = \begin{bmatrix} \boldsymbol{x}_{flt}^\top & f_{sw}^* \end{bmatrix}^\top$ and matrices $\boldsymbol{A}_{sw} = \text{blkdiag}(\boldsymbol{A}_{sw}, 1)$, $\boldsymbol{B}_{sw} = \begin{bmatrix} \boldsymbol{B}_{flt}^\top & \boldsymbol{0}_{1\times 3}^\top \end{bmatrix}^\top$.

B. MPC Problem Formulation

Let us define the complete augmented state as

$$\boldsymbol{x}(k) \coloneqq \begin{bmatrix} \boldsymbol{x}_{ph}(k)^\top & \boldsymbol{x}_{osc}(k)^\top & \boldsymbol{x}_{sw}(k)^\top & \boldsymbol{u}_{sw}(k-1)^\top \end{bmatrix}^\top \tag{3}$$

with $\boldsymbol{x}(k) \in \mathbb{R}^9 \times \{-1,0,1\}^3$ and total state dimension $n_x = 12$. Vector $\boldsymbol{x}_{ph}$ represents the physical system from Section II, $\boldsymbol{x}_{osc}$ defines the oscillating states of the sinusoids to track introduced in Section III-A, $\boldsymbol{u}_{sw}(k-1)$ are additional states used to keep track of the switches positions at the previous stage and $\boldsymbol{x}_{sw}$ the states related to the switching filter from Section III-A.

The system inputs are defined as

$$\boldsymbol{u}(k) \coloneqq \begin{bmatrix} \boldsymbol{u}_{sw}(k)^\top & \boldsymbol{p}(k)^\top \end{bmatrix}^\top \in \mathbb{R}^{n_u},$$

where $\boldsymbol{u}_{sw}$ are the physical switches positions and $\boldsymbol{p}$ are the three binary inputs entering in the frequency filter from Section III-A. The input dimension is $n_u = 6$. To simplify the notation, let us define the matrices $\boldsymbol{G}$ and $\boldsymbol{T}$ to obtain $\boldsymbol{u}_{sw}(k)$ and $\boldsymbol{p}(k)$ from $\boldsymbol{u}(k)$ respectively: i.e. such that $\boldsymbol{u}_{sw}(k) = \boldsymbol{G}\boldsymbol{u}(k)$ and $\boldsymbol{p}(k) = \boldsymbol{T}\boldsymbol{u}(k)$. Similarly, to obtain $\boldsymbol{u}_{sw}(k-1)$ from $\boldsymbol{x}(k)$ we define matrix $\boldsymbol{W}$ so that $\boldsymbol{u}_{sw}(k-1) = \boldsymbol{W}\boldsymbol{x}(k)$.

The MPC problem with horizon $N \in \mathbb{N}$ can be written as

$$\begin{aligned}
\underset{\boldsymbol{u}(k)}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \gamma^k \ell(\boldsymbol{x}(k)) + \gamma^N V(\boldsymbol{x}(N)) && \text{(4a)}\\
\text{subject to} \quad & \boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) && \text{(4b)}\\
& \boldsymbol{x}(0) = \boldsymbol{x}_0 && \text{(4c)}\\
& \boldsymbol{x}(k) \in \mathcal{X},\ \boldsymbol{u}(k) \in \mathcal{U}(\boldsymbol{x}_0), && \text{(4d)}
\end{aligned}$$

where the stage cost is defined combining the THD and the switching frequency penalties

$$\ell(\boldsymbol{x}(k)) = \|\boldsymbol{C}\boldsymbol{x}(k)\|_2^2 = \|\boldsymbol{e}(k)\|_2^2 + \delta \|e_{sw}(k)\|_2^2.$$

The tail cost $V(\boldsymbol{x}(N))$ is an approximation of the infinite horizon tail and is defined as the quadratic function

$$V(\boldsymbol{x}(N)) = \boldsymbol{x}(N)^\top \boldsymbol{P}_0 \boldsymbol{x}(N) + 2\boldsymbol{q}_0^\top \boldsymbol{x}(N) + r_0, \tag{5}$$

where $\boldsymbol{P}_0 \in \mathbb{S}_x^n$ is positive semidefinite, $\boldsymbol{q}_0 \in \mathbb{R}^{n_x}$ and $r_0 \in \mathbb{R}$. We precompute the tail cost approximation using approximate dynamic programming (ADP) by solving a semidefinite program (SDP) [13] offline. For a comprehensive description of the ADP method employed refer to [9]. The matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ define the extended system dynamics and the output vector; they can be derived directly from the physical model (1) and from the considerations in Section III-A.

The input constraints set is denoted as

$$\begin{aligned}
\mathcal{U}(\boldsymbol{x}_0) = \{ & -\boldsymbol{T}\boldsymbol{u}(k) \le \boldsymbol{G}\boldsymbol{u}(k) - \boldsymbol{W}\boldsymbol{x}(k) \le \boldsymbol{T}\boldsymbol{u}(k), && \text{(6a)}\\
& \|\boldsymbol{T}\boldsymbol{u}(k)\|_\infty \le 1, && \text{(6b)}\\
& \boldsymbol{G}\boldsymbol{u}(k) \in \{-1,0,1\}^3\}, && \text{(6c)}
\end{aligned}$$

where constraint (6a) defines the relationship between $\boldsymbol{u}_{sw}$ and $\boldsymbol{p}$ from (2). Constraint (6b) together with (6a) defines the switching constraints $\|\boldsymbol{u}_{sw}(k) - \boldsymbol{u}_{sw}(k-1)\|_\infty \le 1$ imposed to avoid a shoot through in the inverter positions that could damage the components. Finally, (6c) enforces integrality of the switching positions.

Observe that the controller tuning parameters are δ, which defines the relative importance of the THD and f_{sw} components in the cost function, and r_1, r_2 that shape the switching frequency estimator.

C. Control Loop

The complete block diagram is shown in Figure 1. The desired torque T determines the currents $\boldsymbol{x}_{osc}$ by setting the

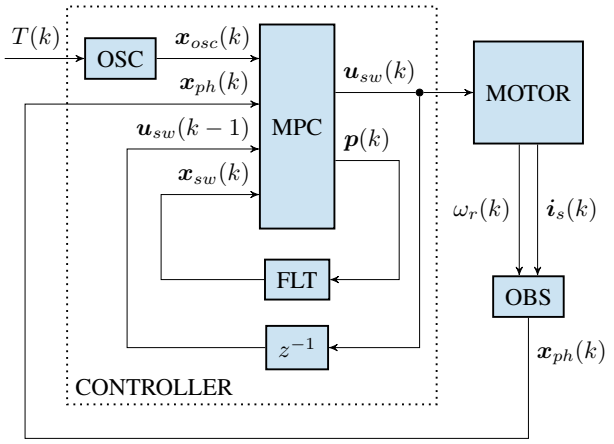


Fig. 1. Block diagram of the control loop. The controller within the dotted line receives the desired torque $T(k)$ and the current motor states $\boldsymbol{x}_{ph}(k)$ providing the switches position $\boldsymbol{u}_{sw}(k)$.

initial states of the oscillator OSC. The motor speed ω_r and the stator currents $\boldsymbol{i}_s$ are measured directly from the machine and used by the observer OBS providing the physical states of the motor $\boldsymbol{x}_{ph}$. The auxiliary inputs $\boldsymbol{p}$ are fed into the filter FLT estimating the switching frequency in $\boldsymbol{x}_{sw}$. The switches positions $\boldsymbol{u}_{sw}$ go through a one step delay and are exploited again by the MPC formulation.

Following a receding horizon control strategy, at each stage k problem (4) is solved obtaining the optimal sequence $\{\boldsymbol{u}^\star(k)\}_{k=0}^{N-1}$ from which only $\boldsymbol{u}^\star(0)$ is applied to the switches. At the next stage $k+1$, given new vectors $\boldsymbol{x}_{osc}(k), \boldsymbol{x}_{ph}(k), \boldsymbol{u}_{sw}(k-1)$ and $\boldsymbol{x}_{sw}(k)$ as in Figure 1 a new optimization problem is then solved providing an updated optimal switching sequence, and so on. The whole control algorithm, appearing within the dotted line, runs within $25\,\mu$s.

D. Optimization Problem in Vector Form

Since we consider short horizons, we adopt a condensed MPC formulation of problem (4) with only input variables producing a purely integer program. In this way all the possible discrete input combinations can be evaluated directly. With a sparse formulation including the continuous states within the variables, it would be necessary to solve a mixed-integer program requiring more complex computations.

Let us define the input sequence over the horizon N starting at time 0 as $\boldsymbol{U} = \left[\boldsymbol{u}^\top(0)\ \boldsymbol{u}^\top(1)\ \dots\ \boldsymbol{u}^\top(N-1)\right]^\top$, where we have dropped the time index from $\boldsymbol{U}$ to simplify the notation. With straightforward algebraic manipulations outlined in [9] it is possible to rewrite problem (4) as a parametric integer quadratic program in the initial state $\boldsymbol{x}_0$

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{U}^\top \boldsymbol{Q} \boldsymbol{U} + 2\boldsymbol{f}\left(\boldsymbol{x}_0\right)^\top \boldsymbol{U} \\ \text{subject to} \quad & \boldsymbol{A}_{ineq}\boldsymbol{U} \leq \boldsymbol{b}_{ineq}(\boldsymbol{x}_0) \\ & \mathcal{G}\boldsymbol{U} \in \{-1,0,1\}^{3N}. \end{aligned} \tag{7}$$

IV. Framework for Performance Evaluation

To benchmark our algorithm we consider a neutral point clamped voltage source inverter connected with a medium-voltage induction machine and a constant mechanical load. As a typical medium-voltage induction machine example, we use the same model as in [5]. We consider an idealized model with the semiconductors switching instantaneously. In addition, if not otherwise stated, all simulations were done at rated torque, nominal speed, fundamental frequency of $50\,\mathrm{Hz}$ and rated currents.

V. Achievable Performance in Steady State

We hereby briefly present the results of closed loop simulations in steady-state operation obtained in [8] and [9] to show the achievable performance of our hardware implementation in terms of THD and switching frequency. For comparison, the system was simulated also with the controller described in [5] (denoted as DMPC). Both controllers were tuned to obtain a switching frequency around $300\,\mathrm{Hz}$. For more details, please refer to [8], [9]. The results are presented in Table I.

TABLE I

Simulation Results from [8], [9] with ADP and with DMPC [6] at switching frequency $300\,\mathrm{Hz}$

	ADP		DMPC [6]	
	δ	THD [%]	λ_u	THD [%]
$N=1$	4	5.24	0.00235	5.44
$N=2$	5.1	5.13	0.00690	5.43
$N=3$	5.5	5.10	0.01350	5.39
$N=10$	10	4.80	0.10200	5.29

Our method, with a horizon of $N=1$ provides both a THD improvement over the DMPC formulation in [5] with $N=10$ and a drastically better numerical speed. This shows how choosing a meaningful cost function can provide good control performance without recourse to long horizons. Moreover, we also performed a comparison with longer horizons $N=2$, $N=3$ and $N=10$. Our method, with horizon $N=10$ would give an even greater reduction in THD to $4.80\,\%$.

VI. FPGA Implementation

A. Hardware Setup

We implemented the control algorithm on a Xilinx Zynq (xc7z020) [14], a low cost FPGA, running at approximately $150\,\mathrm{MHz}$ mounted on the Zedboard evaluation module [15]. The control algorithm was coded in C++ using the PROTOIP Toolbox [16]. The FPGA vendors tool Xilinx Vivado HLS [17] was used to convert the written code to VHDL defining the Programmable Logic connections.

B. Algorithm Description

We now present a detailed description of how the controller within the dotted lines in Figure 1 is implemented on the FPGA.

The updates in OSC and FLT were implemented as simple matrix multiplications. The solver for the integer problem (7) has been implemented with a simple exhaustive search algorithm for three reasons: first, the tail cost approximation provides good performance with very few horizon steps while considering a relatively small amount of input

combinations; second, the structure of the problem allows us to evaluate both the inequalities and cost function for multiple input sequences in parallel; third, the FPGA logic is particularly suited for highly pipelined and/or parallelized operations, which are at the core of exhaustive search.

To exploit the FPGA architecture, we implemented our algorithm in fixed-point arithmetic using custom data types defined in Vivado HLS [17]. In particular, we used 4 integer and 0 fractional bits to describe the integer inputs and 2 integer and 22 fractional bits to describe the states and the cost function values. This choice is given by the minimum number of bits necessary to describe these quantities from floating-point simulations in Section V. Note that the exhaustive search algorithm does not suffer from any accumulation of rounding error because it consists entirely of independent function evaluations, in contrast to iterative optimization algorithms [18].

Algorithm 1 Controller Algorithm

1: **function** COMPUTEMPCINPUT($T(k), \boldsymbol{x}_{ph}(k)$)
Data: $\boldsymbol{x}_{osc}(k-1)$, $\boldsymbol{x}_{sw}(k-1)$, $\boldsymbol{p}(k-1)$, $\boldsymbol{u}_{sw}(k-1)$
Parameters: $\boldsymbol{U}^{seq} \in \mathbb{Z}^{6\times 27^N}$, $J_{ub} \in \mathbb{R}$
Initialize: $\boldsymbol{J} \in \mathbb{R}^{27^N}$, $J_{min} \in \mathbb{R}$ and $i_{min} \in \mathbb{N}$
Execute Filter and Oscillator to Obtain Initial State:
2: **if** change in $T(k)$ **then**
3: $\boldsymbol{x}_{osc}(k) \leftarrow$ Reset to match torque [9]
4: **else**
5: $\boldsymbol{x}_{osc}(k) \leftarrow \boldsymbol{A}_{osc}\boldsymbol{x}_{osc}(k-1)$
6: **end if**
7: $\boldsymbol{x}_{sw}(k) \leftarrow \boldsymbol{A}_{sw}\boldsymbol{x}_{sw}(k-1) + \boldsymbol{B}_{sw}\boldsymbol{p}(k-1)$
8: $\boldsymbol{x}_0 \leftarrow \left[\boldsymbol{x}_{ph}(k)^\top \ \boldsymbol{x}_{osc}(k)^\top \ \boldsymbol{x}_{sw}(k)^\top \ \boldsymbol{u}_{sw}(k-1)^\top\right]^\top$
Precompute Vectors:
9: $\boldsymbol{f}(\boldsymbol{x}_0) \leftarrow$ Compute from $\boldsymbol{x}_0$ [9]
10: $\boldsymbol{b}_{ineq}(\boldsymbol{x}_0) \leftarrow$ Compute from $\boldsymbol{x}_0$ [9]
Loop 1 - Compute Cost Function Values:
11: **for** $i = 1, \dots, 27^N$ **do**
12: $\boldsymbol{u} \leftarrow \boldsymbol{U}^{seq}_{(:,i)}$
13: $\boldsymbol{u}_{(4:6)} \leftarrow \boldsymbol{p}(k) = \|\boldsymbol{u}_{(1:3)} - \boldsymbol{u_{sw}(k-1)}\|_1$
14: **if** $\boldsymbol{A}_{ineq}\boldsymbol{u} \leq \boldsymbol{b}_{ineq}(\boldsymbol{x}_0)$ **then**
15: $\boldsymbol{J}_{(i)} \leftarrow \boldsymbol{u}^\top \boldsymbol{Q}\boldsymbol{u} + 2\boldsymbol{f}(\boldsymbol{x}_0)^\top \boldsymbol{u}$
16: **else**
17: $\boldsymbol{J}_{(i)} \leftarrow J_{ub}$
18: **end if**
19: **end for**
Loop 2 - Find Minimum:
20: $J_{min} \leftarrow J_{ub}$, $i_{min} \leftarrow 1$
21: **for** $i = 1, \dots, 27^N$ **do**
22: **if** $\boldsymbol{J}_{(i)} \leq J_{min}$ **then**
23: $J_{min} \leftarrow \boldsymbol{J}_{(i)}$
24: $i_{min} \leftarrow i$
25: **end if**
26: **end for**
Return Results
27: $\boldsymbol{u}_{sw}(k) \leftarrow \boldsymbol{U}^{seq}_{(1:3,i_{min})}$ and $\boldsymbol{p}(k) \leftarrow \boldsymbol{U}^{seq}_{(4:6,i_{min})}$
28: **return** $\boldsymbol{u}_{sw}(k)$
29: **end function**

We provide pseudo-code for our method in Algorithm 1. From Figure 1, the controller receives the required torque $T(k)$ and the motor states $\boldsymbol{x}_{ph}(k)$ and returns switches $\boldsymbol{u}_{sw}(k)$.

From line 2 to line 8 the oscillator OSC and the filter FLT are updated to compute the new initial state $\boldsymbol{x}_0$ for the optimization algorithm. Note that, if there is a change in the required torque, the oscillator states $\boldsymbol{x}_{osc}(k)$ are reset to match the new $T(k)$. Line 9 and 10 precompute the vectors in problem (7) depending on $\boldsymbol{x}_0$.

The main loop iterating over all input combinations is split in two for parts: Loop 1 which is completely decoupled and can be parallelized and Loop 2 which can only be pipelined.

Loop 1 from line 11 to 19 computes the cost function values for every combination i and stores it into vector $\boldsymbol{J}$. All the possible input sequences combinations are saved in a static matrix $\boldsymbol{U}^{seq}$. For every loop cycle, sequence i is saved into variable $\boldsymbol{u}$. Then, in line 13, the value of $\boldsymbol{p}(k)$ is updated inside $\boldsymbol{u}$ with $\boldsymbol{u}_{sw}(k-1)$ according to (2). If $\boldsymbol{u}$ satisfies constraint $\boldsymbol{A}_{ineq}\boldsymbol{u} \leq \boldsymbol{b}_{ineq}(\boldsymbol{x}_0)$, then the cost function is stored in $\boldsymbol{J}_{(i)}$ (line 15). Otherwise, $\boldsymbol{J}_{(i)}$ is set to a high value J_{ub}. Note that, even though it would bring considerable speed improvements, we cannot precompute offline the quadratic part $\boldsymbol{u}^\top\boldsymbol{Q}\boldsymbol{u}$ of the cost and the left side of the inequality $\boldsymbol{A}_{ineq}\boldsymbol{u}$ because we need to update vector $\boldsymbol{u}$ in line 13 at each control cycle according to $\boldsymbol{u}_{sw}(k-1)$. Each iteration of this loop is independent from the others and can therefore be parallelized efficiently.

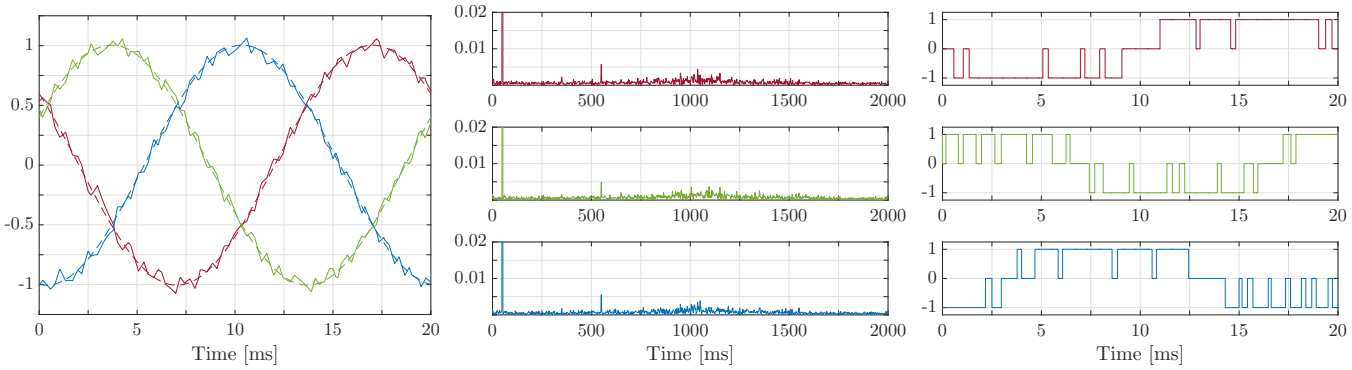
Loop 2 from line 20 to 26 is a simple loop iterating over the computed cost function values to find the minimum and save it into J_{min}. Every iteration depends sequentially on J_{min} which is accessed and can be modified at every i. Thus, in this form it is not possible to parallelize this loop which can be, however, pipelined.

C. Circuit Generation

In Vivado HLS [17] it is possible to specify directives to optimize the circuit synthesis according to the resources available on our board. Loop 1 and Loop 2 were pipelined and the preprocessing operations from line 2 to 10 parallelized. We generated the circuit for the algorithm 1 with horizons $N = 1$ and $N = 2$ at frequency 150 MHz (clock cycle of 7 ns). The resources usage and the timing estimates are displayed in Table II. Since timing constraints were met, there was no need to parallelize Loop 1 to reduce clock cycles. Note that for $N = 2$ we are using already 91 % of

TABLE II
RESOURCES USAGE AND TIMING ESTIMATES FOR IMPLEMENTATION ON THE XILINX ZYNQ FPGA (XC7Z020) RUNNING AT 150 MHz

		$N = 1$	$N = 2$
FPGA Resources	LUT	15127 (28 %)	31028 (58 %)
	FF	11156 (10 %)	20263 (19 %)
	BRAM	6 (2 %)	21 (7 %)
	DSP	89 (40 %)	201 (91 %)
Clock Cycles		371	1953
Delay		2.60 μs	13.67 μs



(a) Three-phase stator currents (solid lines) with their references (dashed lines).

(b) Stator current spectrum.

(c) Three-phase switching position inputs.

Fig. 2. Waveforms produced during HIL Tests by the direct model predictive controller at steady state operation, at full speed and rated torque. Horizon of $N = 1$ is used. The switching frequency is approximately 300 Hz and the current THD is 5.23 %.

the DSP multipliers. This is due to the limited amount of resources available on our chosen low cost hardware.

VII. Hardware In The Loop Tests

We performed hardware in the loop (HIL) experiments using the controller FPGA fixed-point implementation developed in Section VI and the machine model described in Section IV. The control loop was operated using the PROTOIP toolbox [16]: the plant model was simulated on a Macbook Pro 2.8 GHz Intel Core i7 with 16GB of RAM while the control algorithm was entirely executed on the Zedboard development board described in Section VI-A.

A. Steady State

The controller was benchmarked in HIL in steady-state operation to compare its performance to the achievable performance results obtained in Table I. We chose the same controller parameters as in [8], [9]. The weighting δ is chosen such that the switching frequency is around 300 Hz. The infinite horizon tail cost approximation SDP [9] is formulated using YALMIP [19] and solved offline using MOSEK [20].

The HIL tests for horizon $N = 1$ are shown in Figure 2 in the per unit system. The three-phase stator currents are displayed over a fundamental period in Figure 2a, the three spectra are shown in Figure 2b with THD of 5.23 % and the input sequences are plotted in Figure 2c.

From the experimental benchmarks with horizon $N = 1$ and $N = 2$ we obtained THD $= 5.23\,\%$ and THD $= 5.14\,\%$ respectively. As expected, these results are very close to the simulated ones from [8], [9]; see Table I. The slight difference ($\sim 0.01\,\%$) comes from the fixed-point implementation of the oscillator OSC and the filter FLT in Figure 1.

B. Transients

One of the main advantages of direct MPC is the fast transient response [5]. We performed HIL torque transients tests with the same tuning parameters as in the steady state benchmarks. At nominal speed, reference torque steps are imposed, see Figure 3b. These steps are translated into different current references to track, as shown in Figure 3a, while the computed inputs are shown in Figure 3c.

These behaviors match very closely the transient results in [8], [9]. In particular, the torque step from 1 to 0 in the per unit system presents an extremely short settling time of 0.35 ms similar to deadbeat control approaches [21]. This is achieved by inverting the voltage applied to the load. Switching from 0 to 1 torque produces much slower response time of approximately 3.5 ms because of the limited available voltage in the three-phase admissible switching positions.

As noted in [5], having a longer horizon in direct MPC or a better predictive behavior does not significantly improve the settling times. This is because the benefit of longer prediction obtainable by extending the horizon or adopting a powerful final stage costs is reduced by the saturation of the inputs during the transients.

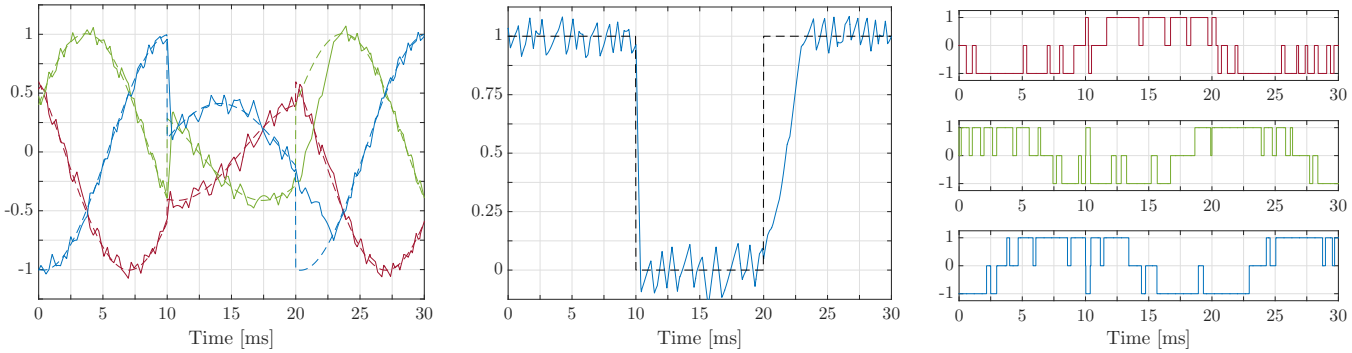
C. Execution Time

To show that the controller is able to run on cheap hardware within $T_s = 25\,\mu$s, we measured the time the FPGA took to execute Algorithm 1 for horizon $N = 1$ and $N = 2$. We compared our results to the time needed to solve the DMPC formulation in [6] for the same horizon lengths on a Macbook Pro with Intel Core i7 2.8 GHz and 16GB of RAM using the commercial integer program solver Gurobi Optimizer [22]. The results are shown in Figure 4.

The FPGA execution times are $5.76\,\mu$s and $17.27\,\mu$s for horizon $N = 1$ and $N = 2$ respectively. Note that they present a slight overhead of approximately $3.5\,\mu$s compared to the estimates in Table II since the measured times include the time needed to exchange the input-output data from the FPGA to the ARM processor through the RAM memory. Without the overhead, the estimated FPGA computing times obtained by the circuit generation are always exact, see [17].

Note that the time needed by the FPGA to compute the control algorithm is deterministic with zero variance. This makes our HIL implementation particularly suited for real-time applications. Furthermore, it is important to underline that the method we propose is the *only* method available that can produce integer optimal solutions to this problem achieving this performance in $25\,\mu$s sampling time.

The execution times needed by Gurobi Optimizer are $621.2 \pm 119.98\,\mu$s and $750.40 \pm 216.15\,\mu$s for horizons $N = 1$ and $N = 2$ respectively. The non-negligible standard deviation appears because of the branch-and-bound algorithm implemented in Gurobi. However, since we are considering



(a) Three-phase stator currents (solid lines) with their references (dashed lines).

(b) Actual (solid line) and reference (dashed line) torques.

(c) Three-phase switching position inputs.

Fig. 3. Reference torque steps produced by the direct model predictive controller in HIL tests with horizon $N = 1$.

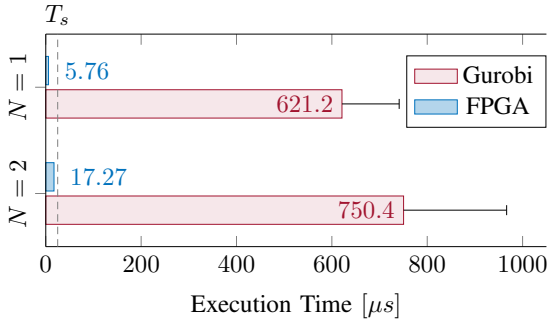


Fig. 4. Execution times required by the Xilinx Zynq FPGA (xc7z020) to execute our controller based on an ADP formulation (blue) and using Gurobi Optimizer [22] to solve the formulation in [5] on a Macbook Pro with Intel Core i7 2.8 GHz and 16GB of RAM

real-time applications, we are interested in the worst case number of visited nodes which is always the whole tree of combinations, i.e. 27^N. Note that, the DMPC formulation has been solved in [5] using a different branch-and-bound algorithm based on the sphere decoding algorithm [7], but the worst case number of visited nodes cannot be simplified because of the $\mathcal{NP}-$hardness of the problem.

VIII. Conclusions

In this work we implemented on the low size Xilinx Zynq FPGA (xc7z020) platform the ADP based approach in [8], [9] for efficient direct model predictive current in power converters.

Hardware in the loop (HIL) tests in steady state operation showed almost identical closed loop performance to the simulation results in [8], [9] in terms of total harmonic distortion (THD) and switching frequency. With our implementation of the ADP based method on low cost hardware and very small number of stages, we were able to outperform the direct MPC formulation in [5], [6] with long horizons. We also performed HIL transient simulations where we managed to obtain the same fast dynamic performance as the simulations in [8], [9].

In addition, we demonstrated that the implemented method from [8], [9] can run within the sampling time of $25\,\mu$s by measuring the execution time on the FPGA. Results show that only $5.76\,\mu$s and $17.27\,\mu$s are required to run our controller for horizons $N = 1$ and $N = 2$ respectively while obtaining good control performance.

Acknowledgment

We would like to thank Andrea Suardi and Bulat Khusainov for their suggestions regarding FPGA implementation.

References

[1] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design* . Nob Hill Publishing, LLC, 2014.

[2] P. Cortes, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodriguez, "Predictive Control in Power Electronics and Drives," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 12, pp. 4312–4324, Nov. 2008.

[3] T. Geyer, "Low Complexity Model Predictive Control in Power Electronics and Power Systems," Ph.D. dissertation, ETH Zürich, 2005.

[4] D. Bertsimas and R. Weismantel, *Optimization over integers*. Belmont, Massachussetts: Dynamic Ideas, 2005.

[5] T. Geyer and D. E. Quevedo, "Multistep Finite Control Set Model Predictive Control for Power Electronics," *IEEE Transactions on Power Electronics*, vol. 29, no. 12, pp. 6836–6846, 2014.

[6] ——, "Performance of Multistep Finite Control Set Model Predictive Control for Power Electronics," *IEEE Transactions on Power Electronics*, vol. 30, no. 3, pp. 1633–1644, 2015.

[7] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 2806–2818, Jul. 2005.

[8] B. Stellato and P. J. Goulart, "High-Speed Direct Model Predictive Control for Power Electronics," in *Control Conference (ECC), 2016 European*, Jul. 2016.

[9] ——, "High-Speed Finite Control Set Model Predictive Control for Power Electronics," *IEEE Transactions on Power Electronics (To appear)*, 2016.

[10] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, Massachusetts, 1996.

[11] D. P. de Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Operations Research*, 2003.

[12] Y. Wang, B. O'Donoghue, and S. Boyd, "Approximate dynamic programming via iterated Bellman inequalities," *International Journal of Robust and Nonlinear Control*, 2014.

[13] L. Vandenberghe and S. Boyd, "Semidefinite Programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.

[14] Xilinx, Inc., *Zynq-7000 All Programmable SoC Technical Reference Manual*.

[15] Avnet Inc., *Zedboard Hardware User's Guide*, 2nd ed.

[16] A. Suardi and E. C. Kerrigan, "Fast FPGA prototyping toolbox for embedded optimization," *European Control Conference (ECC)*, pp. 2589–2594, 2015.

[17] Xilinx, Inc., *Vivado Design Suite User Guide - High-Level Synthesis*.

[18] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer Science & Business Media, 2006.

[19] J. Löfberg, "YALMIP : A Toolbox for Modeling and Optimization in MATLAB," in *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[20] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 35).*, 2015.

[21] J. Rodriguez and P. Cortes, *Predictive control of power converters and electrical drives*. John Wiley & Sons, 2012, vol. 40.

[22] Gurobi Optimization, Inc., "Gurobi Optimizer Reference Manual," Tech. Rep., 2015.