FPGA Implementation of Model Predictive Direct Current Control

Joël Vallone, Tobias Geyer, Eduardo Rath Rohr

Abstract—**An FPGA implementation of Model Predictive Direct Current Control (MPDCC) is reported in this paper. A central scheduler enumerates switching sequences and assigns these to several explorer units, which predict the system response and compute their associated cost. The proposed FPGA design is scalable, modular and requires little hardware resources. A five-level active neutral point clamped inverter with a medium-voltage induction machine is considered. The MPDCC scheme controls the machine's stator currents, the neutral point potential and the inverter's phase capacitor voltages at a low switching frequency.**

I. Introduction

Model predictive control (MPC) has received considerable attention from academic and industrial researchers in the past decades [1], [2], [3]. Despite its performance advantages for constrained multiple-input multiple-output systems, the requirement for solving an optimization problem in real time has traditionally restricted its application scope to systems with slow dynamics, such as process control plants.

In recent years, the development of fast optimization algorithms and the increasing availability of inexpensive and powerful computational platforms has opened the possibility of using MPC also for applications with fast dynamics. This includes power electronics systems, for which the optimization problem typically needs to be solved within several microseconds [4]. Due to their relatively low cost and high computational performance, field programmable gate arrays (FPGAs) are an attractive choice to serve as control hardware for such MPC algorithms [5].

In direct MPC, a common approach is to enumerate candidate switching sequences, to predict the system response and the corresponding value of a cost function, and to choose the switching sequence that minimizes this value. The cost function typically reflects the deviation of the controlled variables from their references and the switching effort. The FPGA implementation of such MPC schemes is reported in [6], [7], [8] for current controllers and in [9] for a torque and flux magnitude controller. A quasi centralized controller is described in [10] for a back-to-back converter system. To make the optimization problem computationally tractable, a prediction horizon of one step is widely adopted.

In an alternative direct MPC formulation, upper and lower bounds are imposed on the controlled variables and the cost function penalizes the switching effort [11]. For example, bounds can be imposed on the three-phase stator currents of an induction machine, giving rise to model predictive direct current control (MPDCC) [12]. Thanks to the use of extrapolation techniques, long prediction horizons with a modest computational burden can be achieved, yielding low current distortions at low switching frequencies [13]. This makes MPDCC particularly attractive for medium voltage (MV) applications, where the switching frequencies are typically limited to a few hundred Hertz.

In this paper, we propose an FPGA implementation of the MPDCC algorithm that is tailored to a small FPGA. Several computational units are operated in parallel to efficiently compute the cost of one candidate switching sequence. A central scheduler unit enumerates the candidate sequences and distributes them among the available computational units. The number of computational units can be adjusted according to the available resources on the FPGA. This approach was successfully implemented and tested on a Spartan 3 FPGA for a five-level active neutral point clamped (ANPC) inverter feeding an MV induction motor [14].

As such, the contribution of this paper is twofold. It is the first one to consider MPDCC for a five-level inverter drive system. More importantly, this is the first paper to report an *implementation* of MPDCC. The previous implementation of a related method, model predictive direct torque control (MPDTC), was done on a digital signal processor (DSP) [15].

The remainder of this article is structured as follows. After introducing the physical models of the five-level ANPC inverter and induction motor in Section II, Section III reviews and summarizes the MPDCC concept. Section IV describes the controller architecture and the choices made to ensure that the design fits into the FPGA. Section V and VI describe in more detail the major parts of the implemented solution. A comparison between the results obtained using hardware-in-the-loop and computer simulations is presented in VII. VIII concludes the paper.

II. Drive System Model

The vector $\boldsymbol{\xi}_{abc} = [\xi_a\ \xi_b\ \xi_c]^T$ in the three-phase abc system can be transformed to $\boldsymbol{\xi}_{\alpha\beta} = [\xi_\alpha\ \xi_\beta]^T$ in the stationary orthogonal $\alpha\beta$ coordinate system through $\boldsymbol{\xi}_{\alpha\beta} = \boldsymbol{K}\,\boldsymbol{\xi}_{abc}$ with

$$\boldsymbol{K} = \frac{2}{3}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix},\ \boldsymbol{K}^{-1} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2}. \end{bmatrix} \quad (1)$$

$\boldsymbol{K}^{-1}$ is the pseudo-inverse of $\boldsymbol{K}$. Throughout the paper, we adopt the per unit system.

A. Inverter Model

Consider the five-level ANPC inverter drive [14] depicted in Fig. 1. The inverter has four internal voltages—the neutral

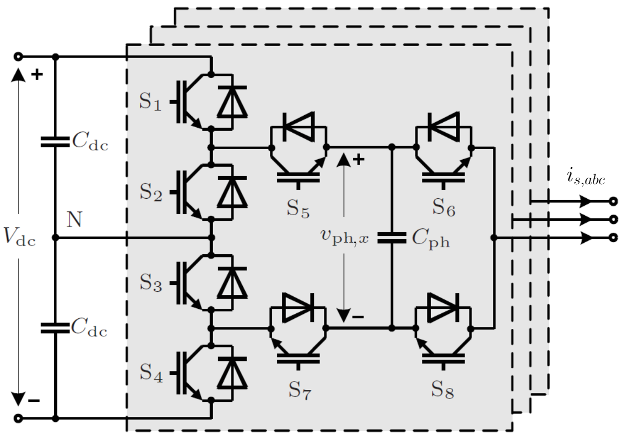


Fig. 1: Equivalent representation of the five-level active neutral point clamped (ANPC) voltage source inverter

point potential v_n and the three phase capacitor voltages $v_{\mathrm{ph},a}$, $v_{\mathrm{ph},b}$ and $v_{\mathrm{ph},c}$. These voltages form the inverter state vector $\boldsymbol{x}_{\mathrm{inv}} = [v_n \; v_{\mathrm{ph},a} \; v_{\mathrm{ph},b} \; v_{\mathrm{ph},c}]^T$. The continuous-time state-space equations are derived in [16].

Using the forward Euler discretization method, the discrete-time inverter model is obtained as

$$\boldsymbol{x}_{\mathrm{inv}}(k+1) = \boldsymbol{x}_{\mathrm{inv}}(k) + T_s \boldsymbol{B}_{\mathrm{inv}}(\boldsymbol{s}_{abc}(k))\boldsymbol{i}_{s,abc}(k)\,. \tag{2}$$

T_s denotes the sampling interval, $k \in \mathbb{N}$ is the discrete time step, $\boldsymbol{i}_{s,abc}$ is the three-phase stator current and $\boldsymbol{s}_{abc} = [s_a \; s_b \; s_c]^T$ is the three-phase switch position, where s_a, s_b, $s_c \in \{0, 1, \dots, 7\}$. The stator currents are assumed to be constant between the two successive discrete time instants kT_s and $(k+1)T_s$. $\boldsymbol{B}_{\mathrm{inv}}$ is a matrix that depends on the switch position $\boldsymbol{s}_{abc}$, see [16].

The inverter variables will be controlled to remain within acceptable bounds. For instance, for a minimum phase capacitor voltage $\underline{v}_{ph}$ and a maximum phase capacitor voltage $\overline{v}_{ph}$, we have $\underline{v}_{ph} \leq v_{ph,a}, v_{ph,b}, v_{ph,c} \leq \overline{v}_{ph}$.

B. Machine Model

When modelling the squirrel-cage induction machine in the stationary orthogonal ($\alpha\beta$) reference frame, we choose the stator current vector $\boldsymbol{i}_{s,\alpha\beta}$ and the rotor flux linkage vector $\boldsymbol{\psi}_{r,\alpha\beta}$ as the machine state vector $\boldsymbol{x}_m = [i_{s\alpha} \; i_{s\beta} \; \psi_{r\alpha} \; \psi_{r\beta}]^T$. The three-phase inverter voltage $\boldsymbol{v}_{abc}(\boldsymbol{s}_{abc})$ is the input to the machine model, which is a function of the dc-link voltage V_{dc}, the phase capacitor voltages $\boldsymbol{v}_{\mathrm{ph},abc}$ and the switch position $\boldsymbol{s}_{abc}$, see [16]. The rotor's angular velocity ω_r is a parameter.

Using the forward Euler discretization method, the state-space model

$$\boldsymbol{x}_m(k+1) = \boldsymbol{A}\boldsymbol{x}_m(k) + \boldsymbol{B}\boldsymbol{v}_{abc}(\boldsymbol{s}_{abc}(k)) \tag{3}$$

TABLE I: Machine parameters in the per unit system

Parameter	Symbol
Stator resistance	R_s
Rotor resistance	R_r
Stator leakage reactance	X_{ls}
Rotor leakage reactance	X_{lr}
Magnetizing reactance	X_m
Stator self reactance	$X_s = X_{ls} + X_m$
Rotor self reactance	$X_r = X_{lr} + X_m$
Determinant	$D = X_s X_r - X_m^2$
Transient stator time constant	$\tau_s = \frac{X_r D}{R_s X_r^2 + R_r X_m^2}$
Rotor time constant	$\tau_r = \frac{X_r}{R_r}$
Model correction factor	$p = 0.21096$

is obtained with the matrices

$$\boldsymbol{A} = \boldsymbol{I}_4 + T_s \begin{bmatrix} -\frac{1}{\tau_s} & 0 & \frac{X_m}{D\tau_r} + T_s p \omega_r^2 & \frac{X_m}{D}\omega_r \\ 0 & -\frac{1}{\tau_s} & -\frac{X_m}{D}\omega_r & \frac{X_m}{D\tau_r} + T_s p \omega_r^2 \\ \frac{X_m}{\tau_r} & 0 & -\frac{1}{\tau_r} & -\omega_r \\ 0 & \frac{X_m}{\tau_r} & \omega_r & -\frac{1}{\tau_r} \end{bmatrix} \tag{4}$$

$$\boldsymbol{B} = \begin{bmatrix} T_s \frac{X_r}{D} \boldsymbol{K} \\ \boldsymbol{0}_{2\times 2} \end{bmatrix}\,. \tag{5}$$

In (3), $\boldsymbol{v}_{abc}(\boldsymbol{s})$ is the three-phase converter output voltage when the switch position $\boldsymbol{s}$ is used.

The corresponding parameters are summarized in Table I. The correction term $T_s p \omega_r^2$ is added to compensate the error introduced by the forward Euler method. The term has been numerically derived by comparing the forward Euler with the exact Euler discretization. Forward Euler has been chosen as a discretization method because it provides sufficient accuracy and avoids the computationally expensive exponentiation of the system matrix.

III. Model Predictive Direct Current Control

MPDCC manipulates the inverter switch positions such that the three-phase stator currents of the machine are kept within symmetrical bounds around their references. Furthermore, the neutral point potential and the three phase capacitor voltages are kept within upper and lower bounds. A third control objective of MPDCC is the minimization of the switching frequency.

The latter objective is captured by a cost function that is minimized subject to the predicted evolution of the drive controller model and the upper and lower constraints, which are imposed on the controlled variables. This problem formulation gives rise to an integer optimization problem, which is solved in real time.

A. Controller Model

The state vector of the drive system is defined as the concatenation of the machine and inverter states $\boldsymbol{x} = [\boldsymbol{x}_m^T \; \boldsymbol{x}_{\mathrm{inv}}^T]^T$. The output vector $\boldsymbol{y} = [\boldsymbol{i}_{s,abc}^T \; v_n \; \boldsymbol{v}_{\mathrm{ph},abc}^T]^T$ constitutes the controlled variable. The input vector is the three-phase switch position $\boldsymbol{s}_{abc}$, which is the manipulated variable. To simplify the notation in the remainder of this article, we will use $\boldsymbol{s}$

instead of $\boldsymbol{s}_{abc}$. We treat the stator and rotor angular speeds ω_s and ω_r as time-varying parameters, which are assumed to be constant within the prediction horizon. The discrete-time dynamical equations of the controller model are given by (2) and (3).

B. Optimization Problem

Define the candidate switching sequence $\boldsymbol{S}(k) = [\boldsymbol{s}^T(k)\ \boldsymbol{s}^T(k+1)\ \dots\ \boldsymbol{s}^T(k+N-1)]^T$ of length N. Define the function $f_s(s_1, s_2, i_{s,abc})$ as the number of switching transitions from the switch position s_1 to s_2 when the machine stator currents are $i_{s,abc}$.

The optimization problem underlying MPDCC at time step k is

$$\begin{aligned} J_{\text{opt}} = & \min_{\boldsymbol{S}(k)} \frac{1}{NT_s} \sum_{l=k}^{k+N-1} f_s(\boldsymbol{s}(l-1), \boldsymbol{s}(l), \boldsymbol{i}_{s,abc}(l)) && \text{(6a)} \\ & + \lambda_n (v_n(k+N))^2 \\ \text{subj. to } & \boldsymbol{x}_m(l+1) = \boldsymbol{A}\boldsymbol{x}_m(l) + \boldsymbol{B}\boldsymbol{v}_{abc}(\boldsymbol{s}(l)) && \text{(6b)} \\ & \boldsymbol{x}_{\text{inv}}(l+1) = \boldsymbol{x}_{\text{inv}}(l) + T_s \boldsymbol{B}_{\text{inv}}(\boldsymbol{s}(l)) \boldsymbol{i}_{s,abc}(l) && \text{(6c)} \\ & \boldsymbol{y}(l) = \boldsymbol{C}\boldsymbol{x}(l) && \text{(6d)} \\ & |\boldsymbol{y}^*(l) - \boldsymbol{y}(l)| \le \boldsymbol{y}_{\text{bnd}} && \text{(6e)} \\ & \boldsymbol{s}(l) \in \boldsymbol{\mathcal{S}}(l) \subset \{0, 1, \dots, 7\}^3 && \text{(6f)} \\ & \forall l = k, \dots, k+N-1\,, \end{aligned}$$

where

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{K}^{-1} & \boldsymbol{0}_{3\times 6} \\ \boldsymbol{0}_{4\times 4} & \boldsymbol{I}_4 \end{bmatrix}. \tag{7}$$

The cost function captures the short-term switching frequency of the (candidate) switching sequence $\boldsymbol{S}(k)$ over the prediction horizon N. The short-term switching frequency is proportional to the sum of the switching transitions $f_s(\cdot)$ over the prediction horizon N divided by the length of the prediction interval in time. The second term in the cost function adds a terminal weight on the neutral point potential with the penalty λ_n. As shown in [17], this penalty reduces the likelihood of infeasibilities, in which no switching sequence $\boldsymbol{S}(k)$ exists that meets (8e). We refer to these infeasibilities as deadlocks.

The cost function is minimized subject to the following constraints. The first two equality constraints predict the dynamical evolution of the machine and inverter states. The third constraint provides the output vector $\boldsymbol{y}$, using the matrix $\boldsymbol{C}$. The output vector is constrained by symmetrical upper and lower bounds around the time-varying output reference $\boldsymbol{y}^*$. For example, symmetrical bounds are imposed on the phase a stator current ripple $|i^*_{sa}(l) - i_{sa}(l)| \le \delta_i$. Similar bounds are imposed on the phase b and c currents, the neutral point potential and the phase capacitor voltages. These bounds form the bound vector $\boldsymbol{y}_{\text{bnd}}$. The current bound width δ_i is a tuning parameter to adjust the total harmonic distortion (THD) of the stator current.

The fifth constraint restricts the set of switch positions $\boldsymbol{\mathcal{S}}(l)$ the inverter may switch to at time step l. This set follows from the inverter switching constraints, which are detailed in [16]. Switching is restricted to one phase level up or down. By adopting the sampling interval $T_s = 50\mu$s, the dynamic clamping constraints are always met and don't need to be explicitly imposed.

C. Main Concepts

To ensure that the optimization problem (6) is computationally tractable for long prediction horizons, heuristics are added that drastically reduce the computational burden without significantly affecting the closed-loop performance [12]. These techniques include the lazy evaluation, switching horizon and extrapolation.

1) Lazy Evaluation: If the output vector is within its bounds at time step k, we repeat the previous switch position and set $\boldsymbol{s}(k) := \boldsymbol{s}(k-1)$. This *lazy evaluation* heuristic reduces the switching frequency by delaying switching transitions until an output variable is about to violate its bound.

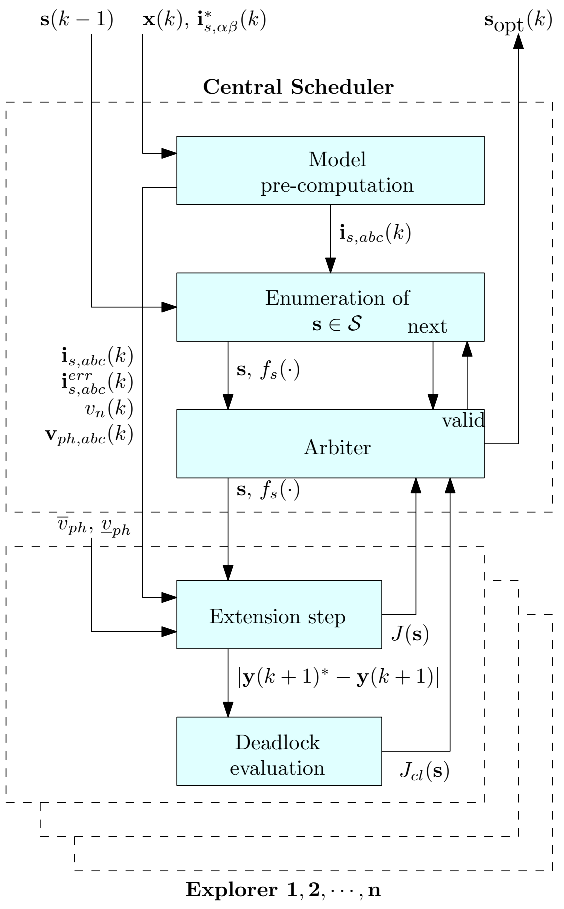
2) Switching Horizon: The switching horizon consists of "E" (extension) and "S" (switching) steps [11]. The *extension* step maintains the previous switch position until a bound is predicted to be violated. When a bound is hit, the controller proceeds to a *switching* step and enumerates the admissible switching transitions from the no longer feasible switch position to new switch positions. For the switching horizon "SE", for example, the controller first enumerates all valid switch positions $\boldsymbol{s}(k)$ and then extends the corresponding output trajectories until they hit a bound. The notion of the switching horizon drastically reduces the size of the solution space by limiting the number of switching transitions to be considered per candidate switching sequence.

3) Extrapolation with Interpolation: Instead of predicting the system evolution by incrementally evaluating the system model at each time step for a given switch position, we evaluate the system model at the time steps $k+1$ and $k+l$, with $l \gg 1$. The system response can then be obtained through linear interpolation [18]. This method is computationally simple and sufficiently accurate for the prediction horizons typically observed.

IV. Controller Architecture

The FPGA implementation of MPDCC was guided by the requirement that the design must not exceed the number of available logic blocks, multipliers and static RAM. The worst case execution time should be below 30μs, i.e. below 1200 clock cycles when assuming a clock frequency of 40 MHz. The controller architecture should be simple, scalable and modular to facilitate future design modifications and to take advantage of larger FPGAs. This section describes the design trade-offs and proposes an FPGA architecture that meets the requirements stated above.

To achieve a simple and scalable design, the controller architecture is based on dynamic scheduling. As shown in Fig. 2, the controller uses a central scheduler that samples the system state $\boldsymbol{x}(k)$, writes it into registers, enumerates the feasible switch positions and generates candidate switching sequences. The system state together with a switching sequence



$\mathbf{s}(k-1)$
$\mathbf{x}(k)$, $\mathbf{i}^*_{s,\alpha\beta}(k)$
$\mathbf{s}_{\text{opt}}(k)$
Central Scheduler
Model pre-computation
$\mathbf{i}_{s,abc}(k)$
Enumeration of $\mathbf{s} \in \mathcal{S}$
next
$\mathbf{i}_{s,abc}(k)$
$\mathbf{i}^{err}_{s,abc}(k)$
$v_n(k)$
$\mathbf{v}_{ph,abc}(k)$
$\mathbf{s}$, $f_s(\cdot)$
valid
Arbiter
$\mathbf{s}$, $f_s(\cdot)$
$\overline{v}_{ph}$, $\underline{v}_{ph}$
Extension step
$J(\mathbf{s})$
$|\mathbf{y}(k+1)^* - \mathbf{y}(k+1)|$
Deadlock evaluation
$J_{cl}(\mathbf{s})$
Explorer $\mathbf{1}, \mathbf{2}, \cdots, \mathbf{n}$

Fig. 2: FPGA controller architecture of the MPDCC algorithm. A central scheduler manages and distributes computation tasks to explorer units.

constitutes a node in the solution space. The scheduler assigns the exploration of nodes to explorer units.

The *explorer* units evaluate these nodes by computing the corresponding output trajectories through an extension step and evaluating the objective function. This information is returned to the scheduler, which stores the switching sequence of the node with the lowest cost J_{opt}. At the end of the enumeration and evaluation process, the scheduler returns the optimal switch position $\boldsymbol{s}_{\text{opt}}(k)$ as the solution.

In the following, we consider the switching horizon "SE". At time step k, the scheduler enumerates the feasible switch positions, and the explorer units perform the extension step. The controller architecture is designed to allow for longer switching horizons in the future.

A. Optimization Problem Revisited

When the heuristics described in Section III-C are applied, a new optimization problem arises. Let $L^*(\boldsymbol{s},\boldsymbol{x})$ be the maximum number of time steps such that the error of all monitored outputs $\boldsymbol{y}$ remains below the threshold defined by $\boldsymbol{y}_{\text{bnd}}$ when the initial state vector is given by $\boldsymbol{x}$ and the switch position $\boldsymbol{s}$ is held. That is,

$$\begin{aligned} L^*(\boldsymbol{s},\boldsymbol{x}) = &\max_{l\in\{0,1,2,\cdots\}} \; l && \text{(8a)}\\ \text{subj. to } & \boldsymbol{x}_m(l) = \boldsymbol{A}(l)\boldsymbol{x} + l\boldsymbol{B}\boldsymbol{v}_{abc}(\boldsymbol{s}) && \text{(8b)}\\ & \boldsymbol{x}_{\text{inv}}(l) = \boldsymbol{x}_{\text{inv}} + lT_s\boldsymbol{B}_{\text{inv}}(\boldsymbol{s})\boldsymbol{i}_{s,abc} && \text{(8c)}\\ & \boldsymbol{y}(l) = \boldsymbol{C}\boldsymbol{x}(l) && \text{(8d)}\\ & |\boldsymbol{y}^*(l) - \boldsymbol{y}(l)| \le \boldsymbol{y}_{\text{bnd}}\,, && \text{(8e)} \end{aligned}$$

with

$$\boldsymbol{A}(l) = \boldsymbol{I}_4 + lT_s \begin{bmatrix} -\frac{1}{\tau_s} & 0 & \frac{X_m}{D\tau_r} + l\tilde{a} & \frac{X_m}{D}\omega_r \\ 0 & -\frac{1}{\tau_s} & -\frac{X_m}{D}\omega_r & \frac{X_m}{D\tau_r} + l\tilde{a} \\ \frac{X_m}{\tau_r} & 0 & -\frac{1}{\tau_r} & -\omega_r \\ 0 & \frac{X_m}{\tau_r} & \omega_r & -\frac{1}{\tau_r} \end{bmatrix} \quad \text{(9a)}$$

$$\tilde{a} = T_s p \omega_r^2. \quad \text{(9b)}$$

Notice that if $L^*(\boldsymbol{s},\boldsymbol{x}) = 0$, then selecting the switch position $\boldsymbol{s}$ will result in a violation of the maximum admissible error in the next time step.

The new optimization problem to be solved online at time k is given by

$$\begin{aligned} \boldsymbol{s}_{opt}(k) = &\arg\min_{\boldsymbol{s}\in\boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))} \; J(\boldsymbol{s}(k-1),\boldsymbol{x}(k),\boldsymbol{s}) && \text{(10)}\\ & + \lambda_n \left(v_n(k + L^*(\boldsymbol{s},\boldsymbol{x}(k)))\right)^2, && \text{(11)} \end{aligned}$$

with

$$J(\cdot) = \begin{cases} \frac{f_s(\boldsymbol{s}(k-1),\boldsymbol{s},\boldsymbol{i}_{s,abc}(k))}{L^*(\boldsymbol{s},\boldsymbol{x}(k))}, & L^*(\boldsymbol{s},\boldsymbol{x}(k)) > 0 \\ +\infty, & L^*(\boldsymbol{s},\boldsymbol{x}(k)) = 0. \end{cases} \quad \text{(12)}$$

Notice from (8) and (9) that the state evolution is computed using forward Euler discretization with time step lT_s. A further simplification can be used to reduce even more the amount of computation required. For a constant $L \in \mathbb{N}$, consider the following linear interpolation:

$$\boldsymbol{x}(k+l) = \boldsymbol{x}(k) + \frac{l}{L}\left(\boldsymbol{x}(k+L) - \boldsymbol{x}(k)\right), \quad \text{(13)}$$

with the state vector at time $k + L$ computed using (8b) and (8c).

The following two sections describe the central scheduler and explorer modules in detail.

V. Central Scheduler

The central scheduler evaluates whether the previous switch position $\boldsymbol{s}(k-1)$ can be used again at time k with the resulting error of the monitored variables $\boldsymbol{y}(k+1)$ is still acceptable. That is, it checks if $|\boldsymbol{y}^*(k+1) - \boldsymbol{y}(k+1)| \le \boldsymbol{y}_{\text{bnd}}$ when $\boldsymbol{s}(k) = \boldsymbol{s}(k-1)$. If no error exceeds its maximum value, the switch position $\boldsymbol{s}(k-1)$ is kept at time k. Otherwise, if at least one of the monitored variables $\boldsymbol{y}(k+1)$ is predicted to violate the maximum error, the search for a new switch position is triggered. In this case, the central scheduler computes the set of candidate switch positions $\boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))$ taking into account the previous switch position $\boldsymbol{s}(k-1)$ and the current stator currents $\boldsymbol{i}_{s,abc}(k)$. It then delegates the evaluation of the objective function corresponding to each candidate solution to an available explorer unit. Finally, the scheduler collects the results of all candidate solutions and selects the one with the lowest cost.

A. Computation of Common Terms

In addition to coordinating the search for a new switch position, the scheduler also computes some terms that are common to all candidate solutions. The objective is to reduce the overall computation time required; this is achieved by eliminating the need to compute the same terms several times in each of the explorers. For instance, the natural response $\boldsymbol{A}(L)\boldsymbol{x}(k)$ and the future motor current reference $\boldsymbol{i}^*_{s,\alpha\beta}(k+L)$ do not depend on the switch position $\boldsymbol{s}(k)$.

The stator current and its reference at instant $k+L$ are computed in the $\alpha\beta$ reference frame:

$$\boldsymbol{i}_{s,\alpha\beta}(k+L) = [\boldsymbol{A}_1\ \boldsymbol{A}_2]\,\boldsymbol{x}(k) + \frac{X_r}{D}LT_s\boldsymbol{K}\boldsymbol{v}_{abc}(\boldsymbol{s}(k)) \tag{14a}$$

$$\boldsymbol{i}^*_{s,\alpha\beta}(k+L) = \boldsymbol{A}_0\boldsymbol{i}^*_{s,\alpha\beta}(k) \tag{14b}$$

where

$$\boldsymbol{A}_0 = \boldsymbol{I}_2 + \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\omega_s LT_s,\ \ \boldsymbol{A}_1 = \left(1 - \frac{1}{\tau_s}LT_s\right)\boldsymbol{I}_2,$$

$$\boldsymbol{A}_2 = \begin{bmatrix} \frac{1}{\tau_r} & \omega_r \\ -\omega_r & \frac{1}{\tau_r} \end{bmatrix}\frac{X_m}{D}LT_s + p(LT_s\omega_r)^2\boldsymbol{I}_2$$

The stator current reference $\boldsymbol{i}^*_{s,\alpha\beta}(k)$ is rotated forward with the stator frequency ω_s.

The term $[\boldsymbol{A}_1\ \boldsymbol{A}_2]\,\boldsymbol{x}(k)$ in (14a) and the current reference (14b) are computed by the central scheduler and passed as arguments to the explorers.

B. Deadlock Resolution

It is possible that for a given state vector $\boldsymbol{x}(k)$ and switch position $\boldsymbol{s}(k-1)$, all candidate solutions $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))$ result in a violation of the maximum error among the monitored variables in the next time step. That is, $L^*(\boldsymbol{s}, \boldsymbol{x}(k)) = 0$ for all $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))$. This situation is called a deadlock, and the central scheduler will select the switch position $\boldsymbol{s}(k)$ that results in the minimum error among the controlled variables at time step $k+1$. To be specific, define the objective function

$$J_{dl}(\boldsymbol{s}(k-1), \boldsymbol{x}(k), \boldsymbol{s}) = \max\{|\boldsymbol{y}^*(k+1) - \boldsymbol{y}(k+1)| - \boldsymbol{y}_{\text{bnd}}\} \tag{15}$$

where the maximization is taken among the elements of the vector $|\boldsymbol{y}^*(k+1) - \boldsymbol{y}(k+1)| - \boldsymbol{y}_{\text{bnd}}$. Notice that the deadlock resolution mechanism is only used when there is no switch position $\boldsymbol{s}$ in $\boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))$ such that $L^*(\boldsymbol{s}, \boldsymbol{x}(k)) > 0$. In this case, the switch position to be used is given by

$$\boldsymbol{s}(k) = \underset{\boldsymbol{s}\in\boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))}{\operatorname{argmin}}\ J_{dl}(\boldsymbol{s}(k-1), \boldsymbol{x}(k), \boldsymbol{s}). \tag{16}$$

A further improvement in the central scheduler monitors if a switch position $\boldsymbol{s}$ that results in $L^*(\boldsymbol{s}, \boldsymbol{x}(k)) > 0$ has been found "so far". If such sequence is known to exist, the central scheduler informs the explorers that there is no need to evaluate the deadlock cost J_{dl}, thus reducing the processing time required by the explorers.

C. Microarchitecture

Multiplications and additions are the operations that require the most FPGA resources in this design. In order to reduce the number of multipliers and adders used, the computational block depicted in Figure 3 is used throughout the design. The three-stage pipeline helps the FPGA timing constraints to be met. The multiplexers at the operands of the multiplier and adder are controlled by finite state machines (FSM), which in turn control the flow of the algorithm. The design implementation uses four of the computational blocks shown in Figure 3 and it can perform the computation of common terms described in Section V-A in 12 clock cycles.

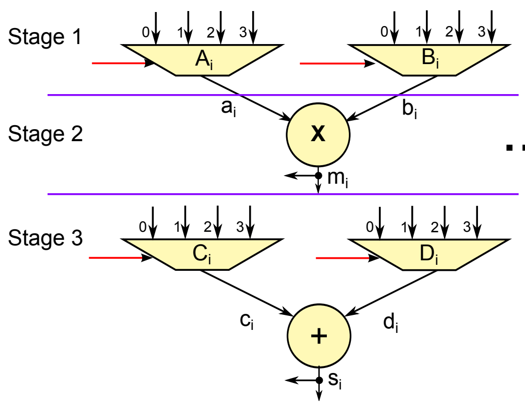


Fig. 3: Model pre-computation data-path. This pipelined data-path is divided into 3 stages. Stage 1: input multiplexers A_i and B_i for the multiplier; Stage 2: dedicated 16 bit integer multiplier with operands a_i and b_i; Stage 3: input multiplexers C_i, D_i feeding a 16 bit integer adder with operand c_i and d_i. This circuit can be replicated depending on the number of operations to perform simultaneously.

D. Summary of Central Scheduler Operation

The operation of the central scheduler is summarized in Algorithm 1, executed at each time step k.

Algorithm 1 Scheduler Operation

```
1: if $|\boldsymbol{y}^*(k+1) - \boldsymbol{y}(k+1)| \le \boldsymbol{y}_{\text{bnd}}$ when $\boldsymbol{s}(k) = \boldsymbol{s}(k-1)$
   then
2:     $\boldsymbol{s}(k) \leftarrow \boldsymbol{s}(k-1)$
3: else
4:     compute common terms as described in Section V-A
5:     create the set of candidate switch positions $\boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))$
6:     for $\boldsymbol{s}$ in $\boldsymbol{\mathcal{S}}(\boldsymbol{s}(k-1))$ do
7:         delegate computation of $J(\boldsymbol{s})$ and $J_{cl}(\boldsymbol{s})$ to an avail-
           able explorer
8:     end for
9:     if $\exists\ \boldsymbol{s}$ such that $L^*(\boldsymbol{s}) > 0$ then
10:        $\boldsymbol{s}(k) \leftarrow \text{argmin}\ J(\boldsymbol{s})$
11:    else
12:        $\boldsymbol{s}(k) \leftarrow \text{argmin}\ J_{dl}(\boldsymbol{s})$
13:    end if
14: end if
```

VI. Explorer Unit

The explorer units are responsible for evaluating the objective functions $J(\cdot)$ and $J_{dl}(\cdot)$ presented in (12) and (15). Several explorer units can be instantiated within the FPGA design. The design choice of how many explorer units to use is guided by the tradeoff between the available hardware resources and the total computation time. A higher number of explorer units can reduce the total time required to perform the model predictive direct current control algorithm at the expense of an increased number of multipliers and logic blocks to be used in the FPGA.

A. Linear Interpolation and Extrapolation

For a given candidate switch position $\boldsymbol{s}$ and state vector $\boldsymbol{x}(k)$, an explorer unit computes $L^*(\boldsymbol{s},\boldsymbol{x}(k))$, i.e., the maximum number of time steps that the switch position $\boldsymbol{s}$ can be applied to the system with initial conditions given by $\boldsymbol{x}(k)$ without violating the maximum error in any of the controlled variables $\boldsymbol{y}$.

In a first step, the explorer unit predicts the value of the error of the controlled values at time $k+L$. The design parameter $L \in \mathbb{N}$ determines the time instant corresponding to when the prediction is performed. For implementation, it is convenient to choose L as a power of two, as multiplications and divisions by L can be performed by bit shifting. Recall that the first three entries of the vector of monitored $\boldsymbol{y}$ are the machine currents in the abc frame. The error between the machine current reference and prediction is given by

$$\begin{aligned}\tilde{\boldsymbol{i}}_{s,abc}(k+L) =& \boldsymbol{i}^*_{s,abc}(k+L) - \boldsymbol{i}_{s,abc}(k+L) && \text{(17a)}\\ =& \boldsymbol{K}^{-1}\left(\boldsymbol{i}^*_{s,\alpha\beta}(k+L) - \boldsymbol{i}_{s,\alpha\beta}(k+L)\right) && \text{(17b)}\\ =& \boldsymbol{K}^{-1}\left(\boldsymbol{A}_0\boldsymbol{i}^*_{s,\alpha\beta}(k) - [\boldsymbol{A}_1\ \boldsymbol{A}_2]\,\boldsymbol{x}(k)\right)\\ &- \boldsymbol{K}^{-1}\left(\frac{X_r}{D}LT_s\boldsymbol{K}\boldsymbol{v}_{abc}(\boldsymbol{s})\right). && \text{(17c)}\end{aligned}$$

Recall that the explorer only has to compute the second parcel of (17c), since the first parcel does not depend on $\boldsymbol{s}$ and has already been computed by the central scheduler. Simmilarly, the error of the internal inverter voltages at time $k+L$ can be predicted using (8c).

$$\begin{aligned}\tilde{\boldsymbol{x}}_{inv}(k+L) =& \boldsymbol{x}^*_{inv} - \boldsymbol{x}_{inv}(k+L) && \text{(18a)}\\ =& \boldsymbol{x}^*_{inv} - \boldsymbol{x}_{\text{inv}}(k) - LT_s\boldsymbol{B}_{\text{inv}}(\boldsymbol{s})\boldsymbol{i}_{s,abc}(k). && \text{(18b)}\end{aligned}$$

Combining (17c) and (18b), the error of the controlled variables is given by

$$\tilde{\boldsymbol{y}}(k+L) = \begin{bmatrix}\tilde{\boldsymbol{i}}_{s,abc}(k+L)\\ \tilde{\boldsymbol{x}}_{inv}(k+L)\end{bmatrix}. \tag{19}$$

In a second step, the explorer unit uses linear interpolation to approximate the error at time $k+l$.

$$\tilde{\boldsymbol{y}}(k+l) = \tilde{\boldsymbol{y}}(k) + \frac{l}{L}(\tilde{\boldsymbol{y}}(k+l) - \tilde{\boldsymbol{y}}(k)) \tag{20}$$

Denote the j-th entry of a vector by the superscript (j). Equation (20) presents a set of seven equations affine in l that can be written as

$$\tilde{\boldsymbol{y}}^{(j)}(k+l) = a_j + lb_j \tag{21}$$

for some scalars a_j and b_j. The problem of finding $L^*(\boldsymbol{s},\boldsymbol{x}(k))$ can be expressed as

$$L^*(\boldsymbol{s},\boldsymbol{x}(k)) = \min\{l_1,\cdots,l_7\}, \tag{22}$$

where l_j is the largest nonnegative integer that satisfies

$$|a_j + l_jb_j| \le \boldsymbol{y}^{(j)}_{\text{bnd}}. \tag{23}$$

l_j can be explicitly computed as

$$l_j = \begin{cases} 0, & |a_j| > \boldsymbol{y}^{(j)}_{\text{bnd}}\\ +\infty, & |a_j| \le \boldsymbol{y}^{(j)}_{\text{bnd}} \text{ and } b_j = 0\\ \text{floor}\left(\frac{\boldsymbol{y}^{(j)}_{\text{bnd}}}{|b_j|}\right), & a_j = 0 \text{ and } b_j \ne 0\\ \text{floor}\left(\frac{\boldsymbol{y}^{(j)}_{\text{bnd}}+|a_j|}{|b_j|}\right), & |a_j| \le \boldsymbol{y}^{(j)}_{\text{bnd}} \text{ and } a_jb_j < 0\\ \text{floor}\left(\frac{\boldsymbol{y}^{(j)}_{\text{bnd}}-|a_j|}{|b_j|}\right), & |a_j| \le \boldsymbol{y}^{(j)}_{\text{bnd}} \text{ and } a_jb_j > 0,\end{cases} \tag{24}$$

where $\text{floor}(x)$ is the largest integer less than or equal to x.

B. Binary Search

The implementation of division operations in FPGAs is often avoided when possible since many clock cycles are required to perform the calculation. Notice that the explicit computation of l_j in (24) for the non-trivial cases requires a division by $|b_j|$. The implementation reported in this paper explores the fact that l_j is an integer and that upper and lower bounds ($\overline{L}$ and $\underline{L}$, respectively) can be imposed on l_j. For instance, using $\overline{L} = N$, will limit l_j to the prediction horizon of the MPC problem. Also, if a different candidate switch position $\tilde{\boldsymbol{s}}$ is known to have cost $J(\tilde{\boldsymbol{s}})$, then there exists $\underline{L}$ such that if $l_j < \underline{L}$, then $J(\boldsymbol{s}) > J(\tilde{\boldsymbol{s}})$. In these cases, an exact computation of l_j is not required, and the computation time can be shortened. For appropriate positive scalars $x_j, y_j > 0$, the problem can be reformulated as

$$\begin{aligned} l_j =& \max_{\underline{L}\le l\le\overline{L}}\ l && \text{(25a)}\\ \text{subj. to }& x_jl \le y_j. && \text{(25b)}\end{aligned}$$

Algorithm 2 is used to find l_j. Denote the m-th bit of l_j by $l_j[m]$, with $l_j[1]$ as the least significant bit. Let M be the smallest integer such that $M \ge \log_2(\overline{N})$. The auxiliary scalar variables a, b are used as bounds above and below l_j, respectively. The auxiliary scalar variable c is used to store the value of $\frac{\overline{L}x_j}{2^i}$ on the i-th iteration.

Figure 4 shows an example of execution of Algorithm 2.

VII. Results

The MPDCC controller was coded in VHDL for the switching horizon "SE", using the controller architecture outlined in the Section IV. Two aspects will be discussed in the following: the computation time versus the required hardware resources, and the closed-loop performance of the controller in terms of the current distortions and the average switching frequency.

Algorithm 2 Binary Search

```
1: $l_j \leftarrow 0$
2: $b \leftarrow 0$
3: $c \leftarrow \overline{L}x_j$
4: if $c \leq y_j$ then
5:     return indicating that $l_j = \overline{L}$
6: end if
7: $a \leftarrow \overline{L} - 1$
8: for $m = M, \cdots, 1$ do
9:     $c \leftarrow c/2$
10:    if $b + c < y_j$ then
11:        $b \leftarrow b + c$
12:        $l_j[m] \leftarrow 1$
13:    else
14:        $a[m] \leftarrow 0$
15:        if $a < \underline{L}$ then
16:            return indicating that $\nexists\, l \geq \underline{L}:\ x_j l \leq y_j$
17:        end if
18:    end if
19: end for
```

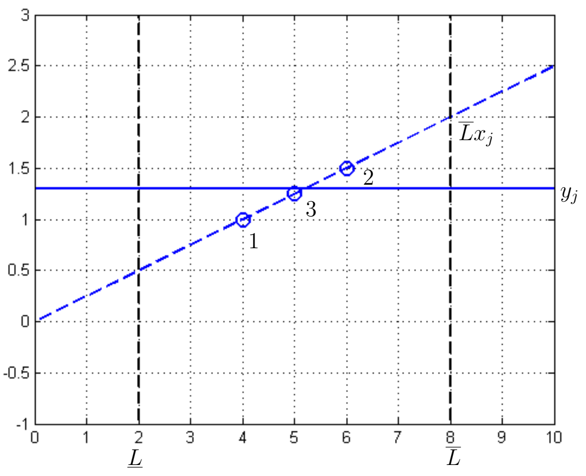


Fig. 4: Example of binary search execution with $\overline{L} = 8$, $\underline{L} = 2$, $x_j = 0.25$, and $y_j = 1.3$. The circles show the three steps used by the iterative search. After the 3rd step, it is found that $l_j = 5$.

A. Computation Time versus Hardware Resources

The number of clock cycles required to compute the optimal switch position can be derived using the simulation software package Modelsim HDL. For n explorer units and the switching horizon "SE", the number of clock cycles amounts, in the worst case, to $47 + 2'176/n$ cycles. In order to meet the requirement of computing the switch position in less than 1'200 clock cycles, $n = 2$ explorer units suffice, as can be seen in Table II. The use of two explorer units also represents an excellent trade-off between computation time and hardware usage. By adopting two (instead of one) explorer units, the worst case number of clock cycles is almost halved, while only 7'900 FPGA slices and 14 multipliers are required.

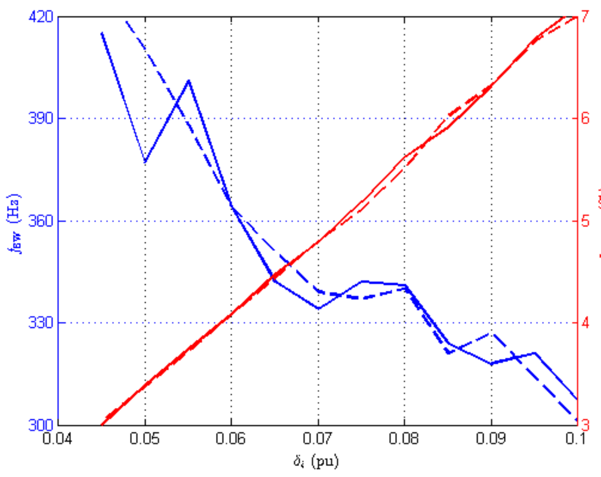


Fig. 5: Switching frequency and current THD as a function of the bound width δ_i imposed on the stator currents for the Matlab (dash-dotted line) and FPGA (solid line) implementation.

B. Controller Performance: Idealized Matlab versus FPGA Implementation

To assess the closed-loop control performance of the FPGA implementation, closed-loop simulations were run for various widths of the stator current bounds, δ_i. The resulting average switching frequency per semiconductor and the THD of the stator currents are shown in Figure 5 as solid lines. The neutral point potential and the phase capacitor voltages are kept well within their bounds. We also benchmarked the FPGA implementation with an idealized Matlab implementation of MPDCC. The latter uses a 64 bit floating point representation, the system model is discretized using exact Euler discretization, and the output trajectories are extended using the discrete-time system model (rather than extrapolation). The bounds on the output vector and the cost function parameters are the same as for the FPGA implementation.

A difference in the switching decisions was observed at 2.5% of the time steps. These differences are a result of the fairly coarse 16 bit fixed point representation on the FPGA and the linear extrapolation with interpolation method. An analysis revealed that the FPGA implementation sometimes suffers from an error of one time step on the predicted length of the switching sequence. Moreover, the value of the cost function also differs slightly due to numerical truncations. As can be observed in Figure 5, for the different bound widths δ_i, the switching frequencies and the current distortions are nevertheless similar for both implementations. The differences in the switching frequency and the current distortions are below 8% and 2%, respectively.

VIII. Conclusions

This paper proposed an implementation of MPDCC that is suitable for a low-cost FPGA. The proposed controller architecture is simple, scalable and modular, allowing for future extensions such as longer switching horizons. The performance results suggest that the MPDCC concept can be

TABLE II: Number of clock cycles and hardware usage as a function of the number of explorers units n

Number of explorer units	1	2	3	4	5	6
Number of clock cycles (worst case)	2'223	1'135	772	591	482	410
Speedup factor (of number of clock cycles)	1.00	1.96	2.88	3.76	4.61	5.42
Number of FPGA slices	4'700	7'900	11'200	14'400	17'700	20'900
Number of multipliers	10	14	18	22	26	30

sufficiently simplified to make it suitable for a low-cost FPGA, while providing a promising closed-loop performance.

References

[1] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—A survey. *Automatica*, 25(3):335–348, Mar. 1989.

[2] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Eng. Pract.*, 11(7):733–764, Jul. 2003.

[3] J. B. Rawlings and D. Q. Mayne. *Model predictive control: Theory and design*. Nob Hill Publ., Madison, WI, USA, 2009.

[4] J. Rodríguez, M. P. Kazmierkowski, J. R. Espinoza, P. Zanchetta, H. Abu-Rub, H. A. Young, and C. A. Rojas. State of the art of finite control set model predictive control in power electronics. *IEEE Trans. Ind. Informatics*, 9(2):1003–1016, May 2013.

[5] M.-W. Naouar, E. Monmasson, A. A. Naassani, I. Slama-Belkhodja, and N. Patin. FPGA-based current controllers for AC machine drives—a review. *IEEE Trans. Ind. Electron.*, 54(4):1907–1925, Aug. 2007.

[6] M.-W. Naouar, A. A. Naassani, E. Monmasson, and I. Slama-Belkhodja. FPGA-based predictive current controller for synchronous machine speed drive. *IEEE Trans. Power Electron.*, 23(4):2115–2126, Jul. 2008.

[7] T. J. Vyncke, S. Thielemans, and J. A. Melkebeek. Finite-set model-based predictive control for flying-capacitor converters: Cost function design and efficient FPGA implementation. *IEEE Trans. Ind. Informatics*, 9(2):1113–1121, May 2013.

[8] P. M. Sanchez, O. Machado, E. J. B. Peña, F. J. Rodríguez, and F. J. Meca. FPGA-based implementation of a predictive current controller for power converters. *IEEE Trans. Ind. Informatics*, 9(3):1312–1321, Aug. 2013.

[9] Z. Ma, S. Saeidi, and R. Kennel. FPGA implementation of model predictive control with constant switching frequency for PMSM drives. *IEEE Trans. Ind. Informatics*, 10(4):2055–2063, Nov. 2014.

[10] Z. Zhang, F. Wang, T. Sun, J. Rodríguez, and R. Kennel. FPGA-based experimental investigation of a quasi-centralized model predictive control for back-to-back converters. *IEEE Trans. Power Electron.*, 31(1):662–674, Jan. 2016.

[11] T. Geyer. Generalized model predictive direct torque control: Long prediction horizons and minimization of switching losses. In *Proc. IEEE Conf. Decision Control*, pages 6799–6804, Shanghai, China, Dec. 2009.

[12] T. Geyer. Model predictive direct current control: Formulation of the stator current bounds and the concept of the switching horizon. *IEEE Ind. Appl. Mag.*, 18(2):47–59, Mar./Apr. 2012.

[13] T. Geyer. A comparison of control and modulation schemes for medium-voltage drives: Emerging predictive control concepts versus PWM-based schemes. *IEEE Trans. Ind. Appl.*, 47(3):1380–1389, May/Jun. 2011.

[14] F. Kieferndorf, M. Basler, L.A. Serpa, J.-H. Fabian, A. Coccia, and G. Scheuer. A new medium voltage drive system based on ANPC-5L technology. In *Proc. IEEE Int. Conf. Ind. Technol.*, pages 605–611, Viña del Mar, Chile, Mar. 2010.

[15] G. Papafotiou, J. Kley, K. G. Papadopoulos, P. Bohren, and M. Morari. Model predictive direct torque control—Part II: Implementation and experimental evaluation. *IEEE Trans. Ind. Electron.*, 56(6):1906–1915, Jun. 2009.

[16] T. Geyer and S. Mastellone. Model predictive direct torque control of a five-level ANPC converter drive system. *IEEE Trans. Ind. Appl.*, 48(5):1565–1575, Sep./Oct. 2012.

[17] T. Burtscher and T. Geyer. Deadlock avoidance in model predictive direct torque control. *IEEE Trans. Ind. Appl.*, 49(5):2126–2135, Sep./Oct. 2013.

[18] Y. Zeinaly, T. Geyer, and B. Egardt. Trajectory extension methods for model predictive direct torque control. In *Proc. App. Power Electron. Conf. and Expo.*, pages 1667–1674, Fort Worth, TX, USA, Mar. 2011.