Received 30 September 2021; revised 24 November 2021; accepted 1 December 2021. Date of publication 7 December 2021; date of current version 29 December 2021. The review of this article was arranged by Associate Editor Luca Solero.

Digital Object Identifier 10.1109/OJIA.2021.3133477

Real-Time Implementation of Long-Horizon Direct Model Predictive Control on an Embedded System

EYKE LIEGMANN [1] (Student Member, IEEE), PETROS KARAMANAKOS [2] (Senior Member, IEEE), AND RALPH KENNEL [1] (Senior Member, IEEE)

[1]Chair of Electrical Drive Systems and Power Electronics, Technical University of Munich, 80333, Munich, Germany
[2]Faculty of Information Technology and Communication Sciences, Tampere University, FI-33101 Tampere, Finland

CORRESPONDING AUTHOR: EYKE LIEGMANN (e-mail: eyke.liegmann@tum.de)

This work was supported in part by the Bundesministerium für Bildung und Forschung (BMBF, German Federal Ministry of Education and Research) through the grant KI-Power (16ME0146K).

ABSTRACT This paper deals with the real-time implementation of a long-horizon finite control set model predictive control (FCS-MPC) algorithm on an embedded system. The targeted application is a medium-voltage drive system which means that operation at a very low switching frequency is needed so that the switching power losses are kept relatively low. However, a small sampling interval is required to achieve a fine granularity of switching, and thus ensure superior system performance. This renders the real-time implementation of the controller challenging. To facilitate this, a high level synthesis (HLS) tool, which synthesizes C++ code into VHDL, is employed to enable a higher level of abstraction and faster prototype development of the real-time solver of the long-horizon FCS-MPC problem, namely the sphere decoder. Experimental results based on a small-scale prototype, consisting of a three-level neutral point clamped (NPC) inverter and an induction machine, confirm that the algorithm can be executed in real time within the targeted control period of 25 μs.

INDEX TERMS Field programmable gate array, multilevel converters, model predictive control, variable speed drive.

I. INTRODUCTION

The main control methods for drive systems are the field oriented control (FOC) [1], and direct torque control (DTC) [2]. The former employs linear controllers to generate the modulating signal which is subsequently fed into an explicit modulator stage. Due to the linear control principle, however, decoupling of the d- and q-axes is not fully achieved during transients, a phenomenon which becomes even more prominent as the switching frequency decreases. As a result, the bandwidth of FOC needs to be reduced, resulting in slower transient responses. On the other hand, DTC can achieve excellent dynamic performance owing to its direct control principle, i.e., a dedicated modulator does not exist, but rather the switching signals are directly applied to the power converter. However, high current distortions can be produced at steady-state operation since the current is indirectly controlled by keeping the electromagnetic torque and flux magnitude within given bounds.

As a promising alternative to the aforementioned control methods, model predictive control (MPC) [3] has gained considerable attention in the field of power electronics in the past decade [4]–[9]. As the name suggests, MPC uses a model of the plant to predict its dynamics. Based on this prediction, the optimal control sequence is chosen on the basis of a cost function that captures and quantifies the desired system behavior. The most popular and widely used—at least in academia—MPC-based variant is the so-called finite control set MPC (FCS-MPC), i.e., a direct MPC strategy where the aim is for the controlled variables to track their reference values by directly manipulating the converter switches [10].

Employing FCS-MPC with long prediction horizons significantly reduces the current distortion and notable improvements under steady-state operating conditions can be achieved [11]. However, practical realization of real-time MPC with long prediction horizons is not trivial. This is due to the significant computational challenges arising from the high number of candidate switching sequences since the latter grows exponentially with the number of the horizon steps [12].

This gives rise to the question of a suitable embedded calculation platform that can handle the pronounced computational complexity within the typically very small sampling intervals encountered in power electronics applications, which are in the range of a few tens of microseconds. While the list of candidates is long, including graphic processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs), microcontroller units (MCUs), and their combinations, FPGAs seem to be the most promising control platform owing to their ability to perform calculations in a highly pipelined and parallelized manner [9]. In particular, FPGA-based system-on-chips (SoCs), that combine the advantages of an FPGA with multiple processor cores in one silicon chip, are frequently used in both academia [13], [14] and industry [15].

To further mitigate the computational cost of long-horizon FCS-MPC a dedicated branch-and-bound method—named sphere decoder—tailored to the needs of power electronic systems was discussed in [16]. In previous publications, the sphere decoder was implemented on dSPACE systems [17]–[19], with the limitations that either the sampling interval is long ($>100\ \mu$s) [17], [18], which reduces the switching granularity [10], or a two-level inverter is considered [19], implying that the number of the candidate solutions, and thus the complexity of the control problem, are relatively small.

In contrast to these works, implementations of sphere decoder on an FPGA were presented in [20]–[22]. The tailored VHDL implementation in [20] allowed for a sampling interval of 25 μs for a three-level neutral point clamped (NPC) inverter with an *RL* load, while in [21], a two-level inverter connected to an *RL* load via an *LC* filter is investigated. Both approaches, however, require significant effort in the implementation stage, making its adaptation to other problems a fairly laborious task. In [22], an induction machine (IM) is used in an FPGA-in-the-loop simulation, which was not verified experimentally, meaning that second-order effects of a real-world setup, such as measurement and observer noise, dc-link voltage ripple, dead-time of the inverter switches, saturation of the machine's magnetic material as well as space-harmonics due to its construction, parameters variations, etc., are neglected in the performance verification.

Motivated by the above, this paper presents an efficient real-time implementation of the sphere decoding algorithm, employed to solve the long-horizon FCS-MPC problem for a three-level NPC inverter driving an IM, as shown in Fig. 1. To this aim, optimization techniques of the high level synthesis (HLS) tool from Xilinx are utilized, which facilitate a rapid implementation on the SoC. By adopting this alternative strategy the targeted control frequency of 40 kHz is well

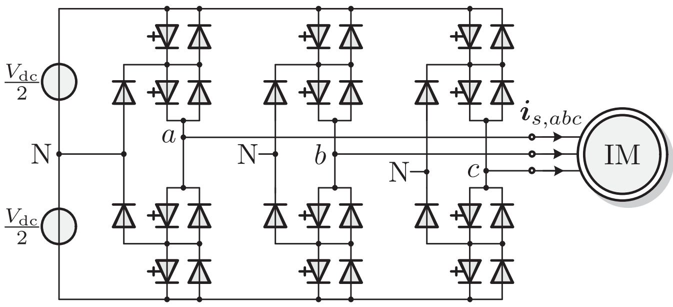


FIGURE 1. Three-level three-phase NPC voltage source inverter driving an induction motor.

achieved during both steady-state and transient, as verified by the presented experimental results acquired on a small-scale prototype. Hence, as demonstrated in this paper, the proposed implementation allows for a high granularity of switching, which, in turn, results in favorable system performance.[1]

This paper is structured as follows. Section II introduces the mathematical model of the case study used in this paper. Section III derives the control problem underlying FCS-MPC as a truncated integer least-squares (ILS) one. The discussed MPC strategy and details on its proposed real-time implementation on an FPGA are presented and analyzed in Sections IV, and V, respectively. In Section VI, the performance of the algorithm is assessed experimentally. Conclusions are drawn in Section VII.

II. CONTROL MODEL

The examined problem relates to the control of the stator current of an IM connected to a three-level NPC inverter, as depicted in Fig. 1. The dc-link voltage V_{dc} is assumed to be constant, while the neutral point potential N is assumed to be fixed and equal to zero. Based on the above, the mathematical model of the chosen case study is derived in the sequel.

Using the stator current $\boldsymbol{i}_s$ and rotor flux $\boldsymbol{\psi}_r$ in the $\alpha\beta$-plane as state, the dynamics of the IM are described by [24][2]

$$\frac{\mathrm{d}\boldsymbol{i}_s}{\mathrm{d}t} = -\frac{1}{\tau_s}\boldsymbol{i}_s + \left(\frac{1}{\tau_r}\boldsymbol{I} - \omega_r \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\right)\frac{X_m}{D}\boldsymbol{\psi}_r + \frac{X_r}{D}\boldsymbol{v}_s \tag{1a}$$

$$\frac{\mathrm{d}\boldsymbol{\psi}_r}{\mathrm{d}t} = \frac{X_m}{\tau_r}\boldsymbol{i}_s - \frac{1}{\tau_r}\boldsymbol{\psi}_r + \omega_r \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\boldsymbol{\psi}_r\,, \tag{1b}$$

where $X_s = X_{ls} + X_m$, $X_r = X_{lr} + X_m$, and X_m are the stator, rotor and mutual reactances, respectively, with X_{ls} being the stator leakage reactance and X_{lr} that of the rotor. Moreover,

[1]Some preliminary results are presented in [14], [23].

[2]Variables $\boldsymbol{\xi}_{abc} = [\xi_a\ \xi_b\ \xi_c]^T$ in the three-phase (abc) frame are transformed into variables $\boldsymbol{\xi}_{\alpha\beta} = [\xi_\alpha\ \xi_\beta]^T$ in the stationary, orthogonal ($\alpha\beta$) frame by employing a suitable transformation matrix $\boldsymbol{K}$, i.e., $\boldsymbol{\xi}_{abc} = \boldsymbol{K}\boldsymbol{\xi}_{\alpha\beta}$. In the remainder of the paper, variables in the *abc*-frame are denoted by the corresponding subscript, whereas the subscript is omitted for those in the $\alpha\beta$-frame.

$\tau_s = X_r D/(R_s X_r^2 + R_r X_m^2)$ is the transient stator time constant, while $\tau_r = X_r/R_r$ the transient time constant of the rotor winding, where R_s and R_r stand for the stator and rotor resistance, respectively. Finally, the constant D is defined as $D = X_s X_r - X_m^2$, and $\boldsymbol{I}$ is the identity matrix of appropriate dimensions. Note that the rotor angular speed ω_r is considered to be a time-varying parameter.

Given that the inverter phase voltage can assume three discrete values, namely, $-V_{\mathrm{dc}}/2$, 0, and $V_{\mathrm{dc}}/2$, depending on the position of its switches, the three-phase switch position is modeled as $\boldsymbol{u}_{abc} = [u_a\ u_b\ u_c]^T$, with $u_a, u_b, u_c \in \mathcal{U} = \{-1, 0, 1\}$ being the single-phase switch positions. Moreover, by choosing the stator current $\boldsymbol{i}_s$ and rotor flux $\boldsymbol{\psi}_r$ in the $\alpha\beta$ coordinate system as state variables, i.e., $\boldsymbol{x} = [i_{s\alpha}\ i_{s\beta}\ \psi_{r\alpha}\ \psi_{r\beta}]^T$, the stator current $\boldsymbol{y} = [i_{s\alpha}\ i_{s\beta}]^T$ as the system output, and the three-phase switch position $\boldsymbol{u}_{abc}$ as the system input, the following continuous-time state-space model is derived as

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{F}\,\boldsymbol{x}(t) + \boldsymbol{G}\,\boldsymbol{u}_{abc}(t) \tag{2a}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\,\boldsymbol{x}(t)\,, \tag{2b}$$

where the matrices $\boldsymbol{F}$, $\boldsymbol{G}$, and $\boldsymbol{C}$ are given in the appendix.

By using exact discretization, the discrete-time state-space model of the drive becomes

$$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}_{abc}(k) \tag{3a}$$

$$\boldsymbol{y}(k) = \boldsymbol{C}\boldsymbol{x}(k) \tag{3b}$$

where

$$\boldsymbol{A} = \mathbf{e}^{\boldsymbol{F}T_s}$$

$$\boldsymbol{B} = \int_0^{T_s} \mathbf{e}^{\boldsymbol{F}\tau}\mathrm{d}\tau\,\boldsymbol{G}$$

with $\mathbf{e}$ being the matrix exponential, T_s the sampling interval, and $k \in \mathbb{N}$.

III. CONTROL PROBLEM FORMULATION

The control objective is to track the stator current reference $\boldsymbol{i}_s^*$ while operating the drive at a low switching frequency so that the switching power losses are kept low. These goals are mapped into a scalar via the cost function J, defined as

$$J(k) = \sum_{\ell=k}^{k+N-1} \|\boldsymbol{i}_s^*(\ell+1) - \boldsymbol{i}_s(\ell+1)\|_2^2 + \lambda_u \|\Delta\boldsymbol{u}_{abc}(\ell)\|_2^2\,, \tag{4}$$

where N denotes the prediction horizon steps and the term $\Delta\boldsymbol{u}_{abc}(\ell) = \boldsymbol{u}_{abc}(\ell) - \boldsymbol{u}_{abc}(\ell-1)$ penalizes the switching effort. The weighting factor $\lambda_u > 0$ is introduced to enable operation on the trade-off between the tracking and control effort, thus enabling the adjustment of the switching frequency. Function (4) needs to be minimized in real time to find the sequence of switch positions

$$\boldsymbol{U}(k) = \left[\boldsymbol{u}_{abc}^T(k)\quad \boldsymbol{u}_{abc}^T(k+1)\ \ldots\ \boldsymbol{u}_{abc}^T(k+N-1)\right]^T \in \mathbb{U}\,, \tag{5}$$

with $\mathbb{U} \in \boldsymbol{\mathcal{U}}^N \subset \mathbb{Z}^{3N}$ and $\boldsymbol{\mathcal{U}} = \mathcal{U}^3$, that results in the desired optimal system behavior.

By denoting the stator current sequence over the prediction horizon as $\boldsymbol{I}_s(k) = [\boldsymbol{i}_s^T(k+1)\ \ldots\ \boldsymbol{i}_s^T(k+N)]^T$ and the stator current reference trajectory correspondingly as $\boldsymbol{I}_s^*(k)$, (4) can be written in vector form as

$$\begin{aligned} J(k) = \ & ||\boldsymbol{\Gamma}\boldsymbol{x}(k) + \boldsymbol{\Upsilon}\boldsymbol{U}(k) - \boldsymbol{I}_s^*(k)||_2^2 \\ & + \lambda_u ||\boldsymbol{S}\boldsymbol{U}(k) - \boldsymbol{E}\boldsymbol{u}_{abc}(k-1)||_2^2, \end{aligned} \tag{6}$$

with the matrices $\boldsymbol{\Gamma}$, $\boldsymbol{\Upsilon}$, $\boldsymbol{S}$, and $\boldsymbol{E}$ being defined in the appendix. After some algebraic manipulations, described in detail in [16], (6) becomes[3]

$$J(k) \approx \left(\boldsymbol{U}(k) + \boldsymbol{H}^{-1}\boldsymbol{\Theta}(k)\right)^T \boldsymbol{H}\left(\boldsymbol{U}(k) + \boldsymbol{H}^{-1}\boldsymbol{\Theta}(k)\right), \tag{7}$$

where the Hessian matrix $\boldsymbol{H}$ and the time-varying term $\boldsymbol{\Theta}(k)$ are defined as

$$\boldsymbol{H} = \boldsymbol{\Upsilon}^T\boldsymbol{\Upsilon} + \lambda_u \boldsymbol{S}^T\boldsymbol{S} \quad \text{and} \tag{8}$$

$$\boldsymbol{\Theta}(k) = \boldsymbol{\Upsilon}^T\left(\boldsymbol{\Gamma}\boldsymbol{x}(k) - \boldsymbol{I}_s^*(k)\right) - \lambda_u \boldsymbol{S}^T\boldsymbol{E}\boldsymbol{u}_{abc}(k-1)\,. \tag{9}$$

As shown in [16], function (7) can be written such that the associated optimization problem is a truncated integer least-squares (ILS) one, i.e.,

$$\begin{aligned} &\text{minimize} && ||\bar{\boldsymbol{U}}_{\mathrm{unc}}(k) - \boldsymbol{V}\boldsymbol{U}(k)||_2^2 \\ &\text{subject to} && \boldsymbol{U}(k) \in \mathbb{U}\,. \end{aligned} \tag{10}$$

The nonsingular, lower triangular matrix $\boldsymbol{V} \in \mathbb{R}^{3N\times 3N}$ is system dependent and is known as the lattice generator matrix that generates the discrete space (i.e., lattice) in which the solution lies. It is calculated using the Cholesky decomposition $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{H}$. Moreover, $\bar{\boldsymbol{U}}_{\mathrm{unc}}(k) = \boldsymbol{V}\boldsymbol{U}_{\mathrm{unc}}(k)$, where $\boldsymbol{U}_{\mathrm{unc}}(k)$ is given by

$$\boldsymbol{U}_{\mathrm{unc}}(k) = -\boldsymbol{H}^{-1}\boldsymbol{\Theta}(k)\,, \tag{11}$$

and it is the unconstrained solution, i.e., the solution that minimizes (7) when relaxing the feasible set from $\mathbb{U}$ to $\mathbb{R}^{3N}$. The optimal integer solution $\boldsymbol{U}_{\mathrm{opt}}(k)$ of (10) represents the lattice point with the shortest Euclidean distance to $\bar{\boldsymbol{U}}_{\mathrm{unc}}(k)$. The latter can be found in a computationally efficient manner by employing a branch-and-bound algorithm, called *sphere decoder*, as explained in the next section.

IV. SPHERE DECODING ALGORITHM

In the following the sphere decoding algorithm as discussed in [16] is presented along with refinements that relate to its real-time implementation.

A. PRINCIPLE OF THE SPHERE DECODER

According to its principle, a $3N$-dimensional hypersphere of radius ρ centered at the unconstrained solution $\bar{\boldsymbol{U}}_{\mathrm{unc}}(k)$ is computed. By doing so, only the candidate solutions inside the sphere have to be evaluated. The goal of the optimizer is

[3]Note that there is a constant term in (7) which is omitted, since it is independent of $\boldsymbol{U}(k)$ and therefore does not affect the optimal solution.

to tighten the radius ρ incrementally until only the optimal solution remains inside the sphere, whereas all other candidate solutions are excluded since they constitute suboptimal options. The initial radius ρ_{ini} should be chosen such that the resulting sphere is as small as possible for the majority of lattice points to be excluded a priori, but not too small so that at least one lattice point is enclosed. To this end, two initial radii are computed based on [25]

$$\rho(k) = ||\overline{\boldsymbol{U}}_{\text{unc}}(k) - \boldsymbol{V}\boldsymbol{U}(k)||_2\,. \tag{12}$$

More specifically, the first option for the initial radius $\rho_{\text{bab}}(k)$ is calculated based on the so-called Babai estimate, i.e., by rounding the unconstrained solution to its nearest integer

$$\boldsymbol{U}_{\text{bab}}(k) = \lfloor \boldsymbol{U}_{\text{unc}}(k) \rceil \in \mathbb{U}\,. \tag{13}$$

The second option, namely the radius $\rho_{\text{edu}}(k)$, is computed based on the educated guess $\boldsymbol{U}_{\text{edu}}(k)$. The latter stems from the previously calculated optimal solution $\boldsymbol{U}_{\text{opt}}(k-1)$ shifted by one time-step forward and repeating the last entry, as introduced in (40) in [16], i.e.,

$$\boldsymbol{U}_{\text{edu}}(k) = \begin{bmatrix} \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} & \dots & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \dots & \dots & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 \\ \boldsymbol{0}_{3\times3} & \dots & \dots & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 \end{bmatrix} \boldsymbol{U}_{\text{opt}}(k-1)\,. \tag{14}$$

Note that $\boldsymbol{I}$ and $\boldsymbol{0}$ in (14) are the identity and zero matrices the dimensions of which are indicated by the corresponding subscripts. Based on (12)–(14), the initial radius is chosen as the minimum of the two options, i.e.,

$$\rho_{\text{ini}}(k) = \min\{\rho_{\text{bab}}(k), \rho_{\text{edu}}(k)\}, \tag{15}$$

After having computed the initial radius $\rho_{\text{ini}}(k)$, the optimization process proceeds by evaluating the lattice points inside the sphere to extract the optimal solution. The pseudocode of the search process is presented in Algorithm 1. As shown, the sphere decoder traverses a search tree of $3N$ levels to find the optimal solution $\boldsymbol{U}_{\text{opt}}(k)$. To this aim, each node of the tree (i.e., a new *single-phase* switch position) is explored and the corresponding intermediate distance between the node explored, i.e., the thus far assembled sequence of switch positions, and $\boldsymbol{U}_{\text{unc}}(k)$ of appropriate dimensions is computed (line 7 in Algorithm 1).

If this distance—calculated in Algorithm 2—is smaller than the most recently updated upper bound ρ^2—initially being $\rho^2_{\text{ini}}(k)$—then the algorithm proceeds by examining the node at the next level of the tree, otherwise the remaining branch is pruned. This procedure is repeated until the bottom level (i.e., leaf node) of the tree is reached; at this point a tentative solution $\boldsymbol{U}_{\text{opt}}(k)$ is found and the sphere is tightened (lines 9–13 in Algorithm 1). Lines 21 to 27 in Algorithm 1 enable the sphere decoder to backtrack and visit unexplored nodes of the search tree. Finally, to guarantee optimality, all other possible

Algorithm 1: Sphere Decoding Algorithm.

```
 1: function SphDec($\rho^2, \overline{\boldsymbol{U}}_{\text{unc}}$)
 2:   $j = 1$                                   ▷search level
 3:   set each element in $\mathbf{sp}[3N] = -1$
 4:   set each element in $\boldsymbol{d}[3N] = 0$
 5:   for maximum number of iterations do
 6:     $u_j = \text{sp}_j$
 7:     $d'^2 = $ DistCalc($\boldsymbol{U}, \text{sp}_j, d_j^2$)
 8:     if $d'^2 \le \rho^2$ then
 9:       if $j = 3N$ then                     ▷leaf node
10:         BetterSolutionFound = true
11:         $\boldsymbol{U}_{\text{opt}} = \boldsymbol{U}$
12:         $\rho^2 = d'^2$
13:         $\text{sp}_j$++
14:       else                                 ▷not a leaf node
15:         $j$++                              ▷increase level
16:         $d_j^2 = d'^2$
17:       end if
18:     else                         ▷prune/explore sibling node
19:       $\text{sp}_j$++
20:     end if
21:     for $q = 3N$ downto 2 do
22:       if $\text{sp}_q > 1$ then                 ▷backtracking
23:         $\text{sp}_q = -1$
24:         $j = q - 1$                        ▷decrease level
25:         $\text{sp}_j$++
26:       end if
27:     end for
28:     if $\text{sp}_1 > 1$ then               ▷search completed
29:       OptimalSolutionFound = true
30:       break
31:     end if
32:   end for
33:   return $\boldsymbol{U}_{\text{opt}}$
34: end Function
```

paths from the higher levels of the tree towards the bottom one are being traversed until no better solution than the tentative one exists. Once all nodes have been explored or pruned, a certificate of optimality is provided (line 29 in Algorithm 1), meaning that the optimal solution has been found.

Lastly, after the sphere decoder terminates, the calculated sequence of switch positions is stored for reuse in the next control cycle to calculate the switching effort $\Delta\boldsymbol{u}_{abc}$ and the educated guess $\boldsymbol{U}_{\text{edu}}$ based on (13). According to the receding horizon principle, the computed three-phase switch position that corresponds to step k, i.e., $\boldsymbol{u}_{\text{opt}}(k)$, is sent to the inverter. For more details on the functionality of the sphere decoder the interested reader is referred to [16] and [26].

B. STOPPING CRITERION: MAXIMUM NUMBER OF VISITED NODES

Looking into the provided pseudocodes in more detail, Algorithm 1 is identical to the one presented in [20] with two proposed modifications to ensure hard real-time compliance and reduction of the computational complexity. First, due to the non-deterministic number of nodes the sphere decoder has to visit, a maximum number of iterations of the for loop is imposed in line 5 in Algorithm 1. This concept is similar to that adopted in [25], where the stopping criterion is based on the number of floating point operations (flops). In this work, an upper limit of 130 nodes ensures that the execution time of the sphere decoder does not violate the hard real-time requirement of the control algorithm operating at 40 kHz. The aforementioned value for the upper limit of nodes is chosen based on the *simulated* execution time of the implemented sphere decoding algorithm in the targeted control platform. During the offline tests the upper limit of nodes can be identified under different scenarios, since this depends on several factors, such as the system parameters, the available computational power, and the available execution time, i.e., the control frequency.

As can be understood, the implementation of the upper limit of nodes guarantees the termination of the search process within the available time. Specifically, if the process terminates before reaching the upper bound, optimality is ensured. If, on the other hand, the algorithm does not conclude to a solution before reaching this upper bound, the tentative solution with the lowest associated cost is used. This can be either the initial guess, or a better solution found in line 10 of Algorithm 1, implying that a possibly suboptimal choice is applied to the inverter.

However, it should be pointed out that the degree of suboptimality depends not only on upper limit of nodes, but also on the targeted switching frequency. This is due to the fact that the switching frequency changes by varying λ_u. Since, the entries of the lattice generator matrix $\boldsymbol{V}$ depend on λ_u, it is implied that changes in λ_u (and thus the switching frequency) affect the shape of the transformed search space. Specifically, a more orthogonal search space (i.e., bigger values of λ_u) enables a more effective search process. On the contrary, smaller values of λ_u (i.e., operation at higher switching frequencies) result in a search space that is very skewed, and thus a slower search process. This is discussed in greater detail in Sections V-C and VI-B. Finally, the upper limit, combined with the prediction horizon N also affect the (sub)optimality of the method. Since the minimum number of to-be-visited nodes is $9N$ (at least three sibling nodes per tree level), and the search tree grows exponentially with an increasing N, the probability for suboptimality also increases as N increases.

C. PARALLEL EVALUATION OF SIBLING NODES

The second proposed modification relates to the distance calculation of each node, see Algorithm 2. One important observation of the search process is that once a node of a particular level has been visited, all its sibling nodes will have to be evaluated as well. However, due to the depth-first search order, in most cases, the sibling nodes are visited at a later point in the search process, e.g., after backtracking from a lower level. Fig. 2 illustrates such a scenario, where the shaded nodes have been evaluated by the algorithm and the thick path indicates the currently explored node.

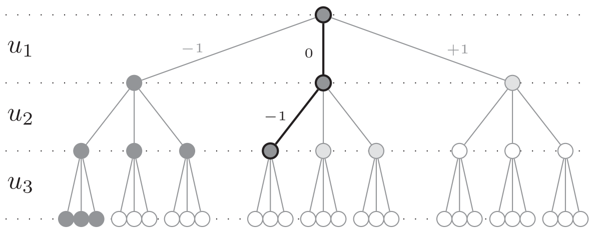


FIGURE 2. Snapshot of a search tree being explored. Nodes that are shaded have been visited by the algorithm, whereas unshaded nodes (i.e., the empty circles) have not been visited thus far. The light gray nodes have not been visited, but their intermediate radii have been calculated and stored in the look-up table Δ.

Algorithm 2: Distance Calculation.

```
1:  function DistCalc($\boldsymbol{U}$, $j$, $d_j^2$)
2:    $\boldsymbol{\Delta} \in \mathbb{R}^{3N\times 2}$        ▷static to store between calls
3:    if $u_j = -1$ then
4:      $\delta_{\text{const}} = \bar{u}_{\text{unc},j} - \boldsymbol{V}_{(j,1:(j-1))}\boldsymbol{U}_{1:(j-1)}$
5:      $\delta_j^2|_{u_j=-1} = (\delta_{\text{const}} + v_{(j,j)})^2 + d_j^2$
6:      $\delta_j^2|_{u_j=0} = \delta_{\text{const}}^2 + d_j^2$
7:      $\delta_j^2|_{u_j=+1} = (\delta_{\text{const}} - v_{(j,j)})^2 + d_j^2$
8:      return $\delta_j^2|_{u_j=-1}$
9:    else if $u_j = 0$ then
10:     return $\delta_j^2|_{u_j=0}$
11:   else                                  ▷$u_j = +1$
12:     return $\delta_j^2|_{u_j=+1}$
13:   end if
14: end Function
```

In line 4 of Algorithm 2, the term δ_{const} is introduced, due to the fact that the preceding switch positions $\boldsymbol{U}_{1:(j-1)}$ are identical for all three sibling nodes, e.g., $u_1 = 0$ in the illustration of Fig. 2. For this reason, δ_{const} is calculated only once when a level is visited for the first time, i.e., when $u_j = -1$. Lines 3 and 23 in Algorithm 1 ensure that the first visited node on each of the $j = 1, 2, \ldots, 3N$ levels is the node corresponding to $u_j = -1$.

Exploiting the ability of FPGAs to parallelize calculations allows the simultaneous computation of the intermediate (squared) radii in lines 5-7 of Algorithm 2 without increasing the execution time. Note that the result for $u_j = -1$ is immediately returned, while the other two radii are stored in the static matrix $\boldsymbol{\Delta} \in \mathbb{R}^{3N\times 2}$, with $3N$ resembling the number of levels in the search tree. This reduces every call of `DistCalc` for $u_j \neq -1$ to a simple look-up table, thus reducing its total execution time by two-thirds. Due to the fact that the radius calculation is the step with the highest computational complexity in Algorithm 1, this modification also significantly decreases the total execution time of the sphere decoder. The pre-calculated radii for not (yet) visited nodes are depicted in light gray in Fig. 2.

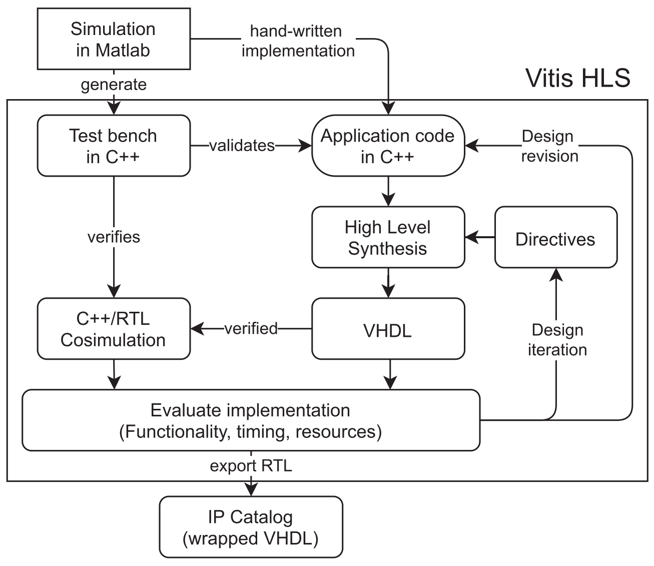


FIGURE 3. **Flow chart of implementation steps using Vitis HLS.**

V. FPGA IMPLEMENTATION

This section covers the details of the implementation of the sphere decoding algorithm on an FPGA. Specifically, Section V-A focuses on the high-level synthesis design flow and motivates the use of the HLS tool in general. This is followed by a brief description of the tool chain and the necessary steps to synthesize a control algorithm for an FPGA, as depicted in Fig. 3. In addition, some important aspects of the HLS tool are introduced, aiming to support other engineers in their first steps with the HLS tool. Following, Section V-B describes how the synthesized intellectual property (IP) core is embedded into the existing FPGA project. Moreover, important aspects that relate to the control system in question are presented. Finally, the FPGA resources and timing of the implemented design are analyzed in Section V-C.

A. HIGH-LEVEL SYNTHESIS

Instead of implementing the algorithm by writing VHDL code, the Vitis HLS tool by Xilinx is used to synthesize C++ into VHDL. The motivation for this is that a high-level language, i.e., C++, allows focusing on the functionality of the algorithm and eases the implementation process. Thanks to the higher level of abstraction, less code needs to be written, functional verification is fast and, therefore, errors can be detected at an early design stage [27]. Since the conversion to VHDL is automated and reproducible, the overall implementation time can be drastically reduced [28].

The main steps of the implementation flow are depicted in Fig. 3. The starting point of the discussed procedure is a Matlab simulation from which the test vectors and expected results for the test bench can be generated. In addition, a closed-loop simulation can be easily set up in C++ with the help of the Embedded Coder toolbox from Mathworks. In doing so, Simulink models or m-scripts can be synthesized into C or C++ in an automated manner.

With the Matlab simulation and test bench in place, the next step is to write the application code, i.e., the control algorithm, in C++ and validate its functionality using the test bench. Once the functional test is successful, the application code can be synthesized to VHDL. HLS generates several reports that give detailed information about the expected timing and resource usage of the implemented design, taking into consideration the Xilinx chip family and the clock frequency that is used. In addition, the initial test bench is also used for a functional test of the synthesized VHDL code in the C++/RTL cosimulation. The reports, in combination with the results of the cosimulation, allow the developer to make a more educated decision about the next step, namely whether to iterate the design, or proceed to the next step. In case of the latter, then exporting the algorithm as an IP core and integrating it into the existing FPGA design follows. These steps are summarized in Fig. 3.

In the following, the behavior of the synthesizer is described in more detail, since it is the crucial part of the design flow and its behavior can be modified using directives, i.e., pragmas. The goal is to introduce the key concepts that are relevant for control applications.

For instance, one important design decision is how the interfaces of the function are implemented in hardware, which can be adjusted using `#pragma HLS interface <mode> port=<name>`, with `<name>` referring to the variable name and `<mode>` to the interface type. With `ap_none`, the interface can be mapped to data ports in the FPGA and with `s_axilite` to the processor via an advanced extensible interface (AXI) lite interface.

Another important aspect is the optimization directives regarding loops, especially when implementing matrix manipulations (which, in most cases, are realized as nested for-loops). The pipelining directive reduces the initiation interval of a loop by allowing the concurrent execution of operations, i.e., each loop iteration does not have to be fully completed before the next iteration call can start. This means that a pipelined function or loop can process new inputs at every n^{th} clock cycle, with $n \in \mathbb{N}^{+}$.

A relevant procedure to pipelining loops is the implementation of arrays. The default behavior of HLS is to realize them using block memory (BRAM). However, BRAM has at most two access ports, and as a result only two entries of the array can be read or written within one clock cycle. To avoid this bottleneck, e.g., when accessing multiple entries of a matrix within a pipelined loop, the `#pragma HLS ARRAY_PARTITION` can be useful to force the synthesizer to partition the array into smaller blocks or individual registers.

Further, it is worth mentioning that HLS offers the possibility to synthesize floating-point variables and all associated mathematical operations. This is handy for lightweight algorithms, or quick test runs to move quickly from simulation to hardware. However, it is necessary to understand that floating-point operations require more resources, and, more

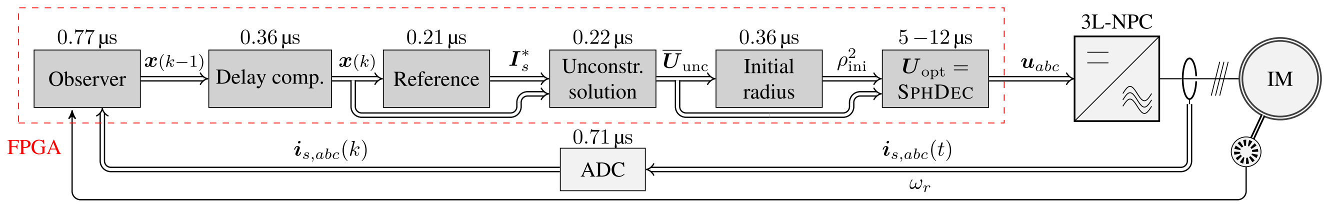


FIGURE 4. Block diagram of the experimental setup including the execution time of each block.

importantly, clock cycles than their fixed-point counterparts. Considering this, the fixed-point library of HLS, included in `ap_fixed.h`, can be utilized. This offers a simple interface to define new data types of arbitrary bit length and fractional bits. For example, `ap_fixed<12, 4>` creates a signed 12-bit data type with 4 integer bits (including the sign-bit) and 8 fractional bits. In addition, HLS guarantees a bit-accurate C++ simulation model, thus it can accurately match the behavior of the synthesized VHDL code. Hence, by employing such an approach, the transition from floating-point to fixed-point variables can be greatly simplified.

Finally, on a more general note, there is a trade-off between FPGA resources and the execution time of the algorithm. However, due to the small time constants in power electronics and the hard real-time requirements in control applications, the execution time is often the constraint that is more difficult to fulfill. Therefore, the aforementioned loop pipelining and fixed-point library are useful concepts that facilitate—and eventually enable—the reduction of the execution time and, thus, realization of computational complex algorithms on an FPGA.

B. CONTROL SYSTEM

The sphere decoder is implemented on the open-source control platform UltraZohm[4] [14], [29] which utilizes a Xilinx Zynq UltraScale+ 9EG as processing unit. The UltraScale+ combines a fairly large FPGA and multiple ARM processors, also referred to as processing system (PS), on the same silicon chip allowing an optimized distribution of tasks on the heterogeneous platform. To achieve the sampling interval $T_s = 25\ \mu$s and horizon length of $N = 3$ steps, the current control loop is implemented exclusively on the FPGA, while the ARM cores are utilized for initialization, supervision, and data logging. A block diagram of the FPGA implementation of the current control loop is shown in Fig. 4.

The FPGA blocks are initialized by the PS via the AXI. Since the system parameters and relevant matrices, e.g., $\boldsymbol{\Gamma}$ and $\boldsymbol{\Upsilon}$ (defined in the appendix), are assumed to be time invariant they are computed on the PS whenever necessary, e.g., when λ_u is updated. The matrices can be updated online, while the drive is operating in closed-loop. Since, however, the calculation of the matrices takes longer than one control cycle, the update rate is approximately one order of magnitude slower than the control frequency. At each time step the stator currents $\boldsymbol{i}_{s,abc}(t)$ are sampled and the rotor speed ω_r is calculated from the feedback of the incremental encoder. Based on these measurements, the state $\boldsymbol{x}(k-1)$ is observed, and the time delay introduced by the digital nature of the controller is compensated for using (3), so that $\boldsymbol{x}(k)$ is acquired. With the current state $\boldsymbol{x}(k)$, the reference trajectory $\boldsymbol{I}_s^*(k)$, i.e., the values of the reference current within the prediction horizon, is computed. With this information, the unconstrained solution $\overline{\boldsymbol{U}}_{\text{unc}}(k)$ is calculated on the FPGA with (10) and (12). Lastly, the squared initial radius $\rho_{\text{ini}}^2(k)$ for the sphere decoder is determined based on (13), and by subsequently evaluating (16).

C. RESOURCE AND TIMING ANALYSIS

Table I summarizes the resource utilization, timing, and arithmetic logic for each block of the FPGA implementation depicted in Fig. 4. Since the calculation blocks of the unconstrained solution, initial radius, and sphere decoding share the system-dependent matrices, they are implemented as one IP core and are therefore lumped together in Table I as "FCS-MPC algorithm". In addition, the total resources required to implement the design, as well as the utilization of the total resources available on the chip are stated, suggesting that it would also be possible to run the implementation on a smaller FPGA with less resources. The timing utilization in % (see the last row of Table I) refers to the execution time of the total control period of 25 μs, indicating that either the control period, the prediction horizon, or both could be further increased. Note that the observer, delay compensation, and reference calculation blocks are implemented in *floating-point* arithmetic, while the remaining part of the predictive controller is realized in *fixed-point* logic. This choice is based on the trade-off between improved timing, and resources usage for fixed point and ease of implementation for floating-point logic.

Fig. 5 shows the histogram for the number of visited nodes required for the sphere decoder to find the optimal solution during steady-state operation at a switching frequency of 300 Hz. In 85% of the occurrences, the sphere decoder requires only the minimum number of possible nodes, i.e., 27, to guarantee the optimal solution. For further analysis, the mean, 99th percentile and maximum number of nodes are highlighted in Fig. 5. It is important to note that the maximum number of nodes (i.e. 93) is less than the maximum number of allowed search iterations, i.e., 130, introduced to ensure the hard real-time compliance. This implies, that the optimal

[4]For more technical details, please refer to www.ultrazohm.com

TABLE I. **FPGA Resource Utilization of Look-Up Tables (LUTs), Flip-Flops (FFs), Block Memory (BRAM), and Digital Signal Processing (DSP) Slices for a Xilinx Zynq UltraScale+ 9EG Device [29]**

IP Core	DSP	LUT	FF	BRAM	Timing	Arithmetic
Flux observer	50	5,892	5,079	0	0.77 µs	floating point
Delay compensation	20	3,275	5,505	3	0.36 µs	floating point
Reference calculation	31	3,354	2,976	1	0.21 µs	floating point
FCS-MPC algorithm	192	36,571	19,377	3	4.0–18.1 µs	fixed point
Utilization in total	293	49,092	32,937	7	5.4–19.5 µs	
Utilization in %	11.6%	17.9%	6.0%	0.8%	22–78 %	

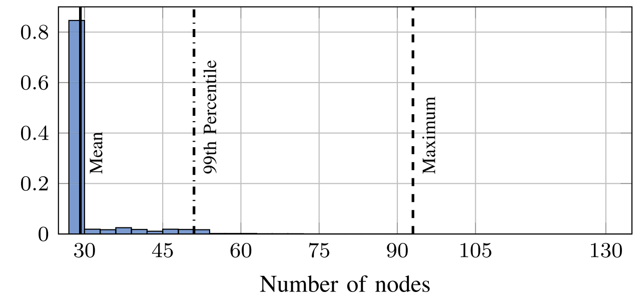


FIGURE 5. **Histogram of visited nodes at 300 Hz switching frequency.**

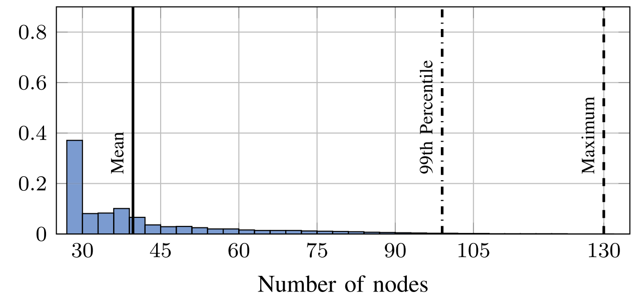


FIGURE 7. **Histogram of visited nodes at 1200 Hz switching frequency.**

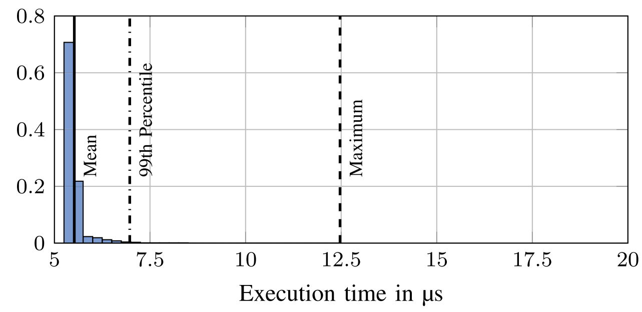


FIGURE 6. **Histogram of execution time of pre-calculations and the FCS-MPC algorithm at 300 Hz switching frequency.**

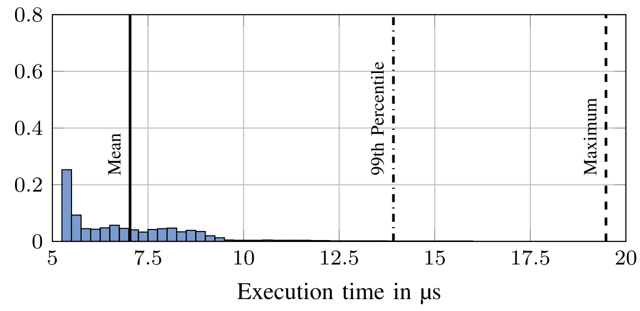


FIGURE 8. **Histogram of execution time of pre-calculations and the FCS-MPC algorithm at 1200 Hz switching frequency.**

solution is always found in steady-state operation and the proposed modifications do not introduce any suboptimality.

Fig. 6 displays the corresponding histogram of the execution time for the entire control algorithm, including all blocks that are listed in Table I during steady-state operation at the same switching frequency as before, i.e., 300 Hz. As can be observed, the execution time varies between 5.4 and 12.5 μs due to the heuristic (i.e., non-deterministic) nature of the sphere decoder tree search. Regardless of this, the mean execution time is 5.52 μs, while the 99th percentile is 6.97 μs, i.e., significantly less than the chosen T_s of 25 μs.

A second scenario is investigated in similar fashion in Figs. 7 and 8 at the switching frequency of 1200 Hz. In this case, the weighting factor λ_u is chosen smaller, and therefore the penalization of the changes in the switch positions is smaller. Because of this, the generated search space is more skewed, as explained in [16] and [30]. This implies that the

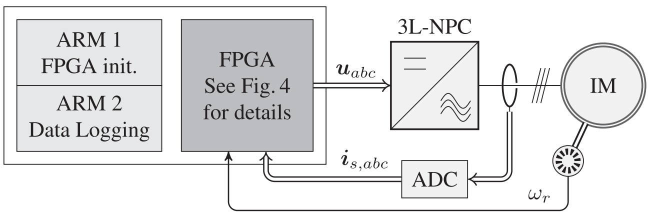
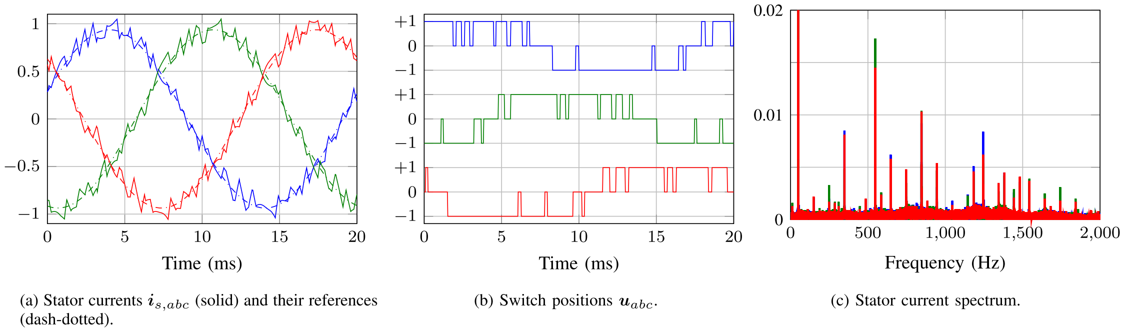


FIGURE 9. **Block diagram of experimental setup.**

TABLE II. **Rated Values of the Induction Machine**

Parameter	SI
rated voltage	380 V
rated current	5 A
rated torque	9 N m
rated speed	2870 rpm



(a) Stator currents $i_{s,abc}$ (solid) and their references (dash-dotted).
(b) Switch positions u_{abc}.
(c) Stator current spectrum.

FIGURE 10. Experimental results during steady-state operation with horizon $N = 3$ and $f_{sw} = 300$ Hz.

TABLE III. Parameters of the Drive System

Parameter	SI	p.u.
dc-link voltage V_{dc}	560 V	1.8
stator resistance R_s	2.1 Ω	0.049
rotor resistance R_r	2.2 Ω	0.052
mutual inductance L_m	340 mH	2.44
stator leakage inductance L_{ls}	10.1 mH	0.072
rotor leakage inductance L_{lr}	10.1 mH	0.072

search process is less efficient, leading to a less effective pruning of the search tree branches, and thus to higher execution times. This effect is clearly visible in both histograms, as they are more spread out. In only 37% of the cases, the sphere decoder requires visiting the minimum number of nodes, i.e., 27, to guarantee the optimal solution, as depicted in Fig. 7. It is important to note that in 0.043% of the cases, the number of visited nodes reaches the maximum number of allowed search iterations. As a result, the algorithm terminates before finding the certificate of optimality, thus introducing the *possibility* of applying a suboptimal solution.

A similar trend can be observed for the execution time in Fig. 8, where the mean execution time increases to 7.03 μs, while the 99th percentile is 13.92 μs. The maximum lies at 19.48 μs, i.e., still less than the chosen T_s of 25 μs, guaranteeing hard real-time capabilities despite searching the maximum number of nodes. Hence, the effectiveness of the chosen upper bound of 130 nodes is clearly demonstrated in this figure.

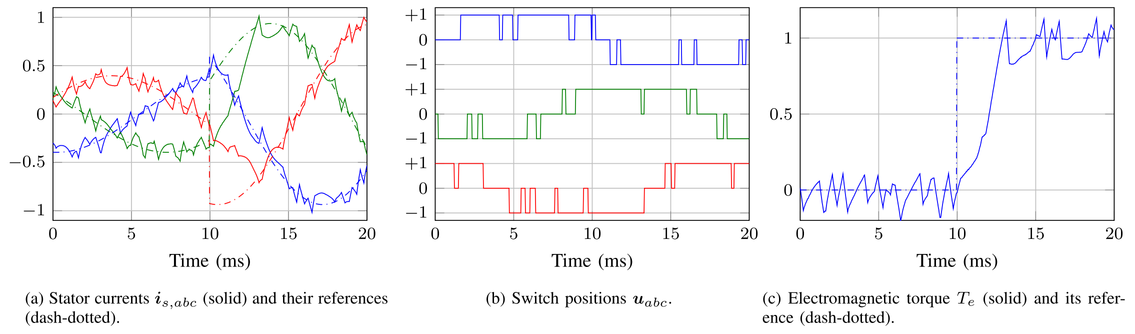
VI. EXPERIMENTAL RESULTS

For the presented results, a low-voltage (LV) drive shown in Fig. 1 is considered and both steady-state and transient performance at rated speed are investigated. A block diagram of the experimental setup is provided in Fig. 9, while the rated parameters as well as those of the drive system are listed in Tables II and III, respectively. Note that the per unit (p.u.) value of the total leakage inductance of the IM is 0.142. The dc-link voltage is realized by two independent power supplies; one across each of the top and bottom capacitors ensuring a fixed neutral point potential with minimum fluctuations. The load machine operates in constant-speed mode, while both inverters (i.e., the one driving the controlled machine and the one the load machine) share the same dc-link voltage, so that only the losses are supplied by the power supplies. Regarding the controller parameters, as mentioned before, the sampling interval is $T_s = 25$ μs and a three-step prediction horizon ($N = 3$) is realized. Finally, all results are shown in the p.u. system.

A. TIME-DOMAIN ANALYSIS

The weighting factor λ_u is tuned such that an average device switching $f_{sw} = 300$ Hz results in all cases examined in this subsection. The steady-state performance of the controller is shown in Fig. 10. For the depicted scenario, the torque reference is kept constant and equal to 1 p.u. As can be seen in Fig. 10(a), the implemented MPC algorithm achieves a good current reference tracking, resulting in a stator current total harmonic distortion (THD) of 7.1%. The corresponding current spectrum is depicted in Fig. 10(c). Considering the relatively low value of the total leakage reactance and the low switching frequency, such a THD value is reasonably good. This performance is achieved thanks to the three-step horizon that enables the algorithm to make better educated decisions. Moreover, the adopted low sampling interval results in a high granularity of switching, since the ratio of sampling-to-switching frequency is greater than 100. As discussed in [10], such a high ratio leads to favorable operation of the power electronic system when controlled by FCS-MPC, as also verified in this work. Finally, it is noteworthy that, as mentioned in the previous section, the sphere decoder always finds the optimal solution despite the upper bound on the nodes allowed to be explored. Hence, the depicted behavior is indeed the optimal.

With regard to the dynamic behavior of the drive, this is shown in Figs. 11 and 12. Specifically, Fig. 11 shows the drive behavior in a step-up change in the torque reference (translated into current reference change), while Fig. 12 depicts the dynamic performance when the torque reference is changed in a step-down manner. As can be seen, thanks to the inherent fast dynamics of direct control, MPC exhibits excellent behavior during transients with as short a settling time as possible, during both examined scenarios. Nonetheless, it is worth mentioning that in some cases the maximum number

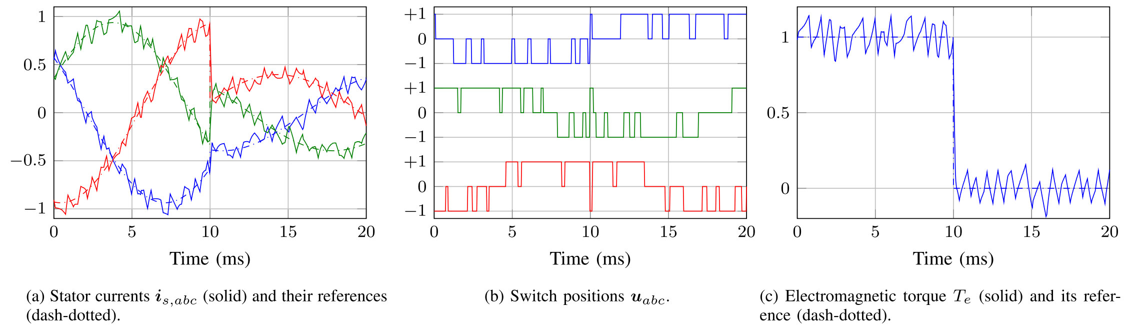


(a) Stator currents $\boldsymbol{i}_{s,abc}$ (solid) and their references (dash-dotted).

(b) Switch positions $\boldsymbol{u}_{abc}$.

(c) Electromagnetic torque T_e (solid) and its reference (dash-dotted).

FIGURE 11. Experimental results for a torque reference step-up change from 0 to 1 p.u.



(a) Stator currents $\boldsymbol{i}_{s,abc}$ (solid) and their references (dash-dotted).

(b) Switch positions $\boldsymbol{u}_{abc}$.

(c) Electromagnetic torque T_e (solid) and its reference (dash-dotted).

FIGURE 12. Experimental results for a torque reference step-down change from 1 to 0 p.u..

of allowed nodes was reached, meaning that suboptimal solutions were occasionally applied during the transient phenomena. However, this did not detract from the controller since it still managed to exhibit very fast responses. If such occasional suboptimality was to pose any problems, then alternative and more elaborate techniques could be adopted, such as the one described in [31].

B. TRADE-OFF CURVE AND OPTIMALITY

To further assess the performance of the controller and provide a deeper insight, the weighting factor λ_u is varied over a wide range of possible values, while keeping the nominal torque reference equal to 1 p.u. In doing so, a different switching frequency results for each value of λ_u, and, consequently, a different value for the current THD. By collecting all these individual experiments, the trade-off curve depicted in Fig. 13 results. Moreover, for comparison purposes, the same line of experiments was performed for FCS-MPC with one-step ($N = 1$) prediction horizon, and the results are shown in the same figure. From this figure, it is evident that the increased prediction horizon results in an improved performance. For example, at a switching frequency of $f_{\mathrm{sw}} = 350$ Hz the current THD produced with one-step FCS-MPC is 8.9%, whereas that of the three-step FCS-MPC is 6.3%. Hence, extending the prediction horizon, the current THD can be reduced by about 30%, implying significant reduction in the copper and iron—and eventually thermal—losses.

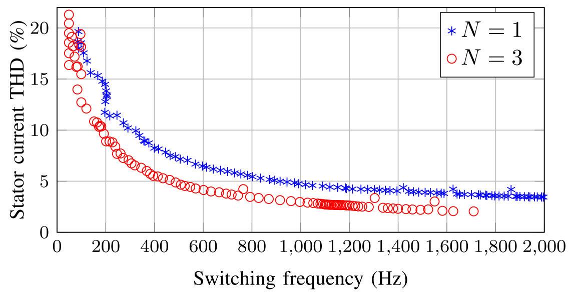


FIGURE 13. Trade-off between the stator current THD and the switching frequency for one-step ($N = 1$), and three-step ($N = 3$) FCS-MPC with sampling interval $T_s = 25$ µs. The individual experiments are indicated by (blue) asterisks and (red) circles for FCS-MPC with $N = 1$ and $N = 3$, respectively.

As mentioned in Section V-C, when the weighting factor changes the search space of the underlying optimization problem is affected. Specifically, a smaller λ_u results in a more skewed search space, and therefore less effective search process. This effect is visualized in Fig. 14, where the number of cases (i.e., individual experiments) is shown for which the maximum number of searched nodes has been reached. As can be observed, optimality is guaranteed for switching frequencies of up to 1.1 kHz, whereas for higher frequencies suboptimal solutions might be realized. However, it needs to be stressed that even though the upper limit on the allowable nodes has been reached in the said occasions, this does not

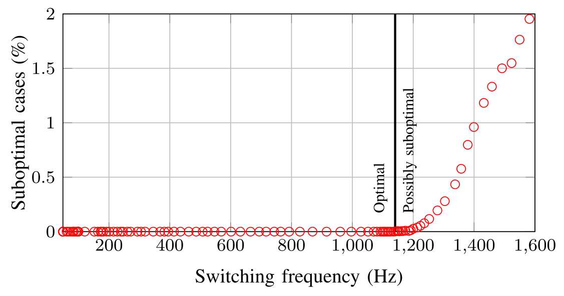


FIGURE 14. Possibility of a suboptimal solution for the achievable range of switching frequencies.

necessarily mean that a suboptimal solution is implemented. It is likely that the controller concludes to the optimal solution, but there is merely not enough time to obtain the certificate of optimality.

VII. CONCLUSION

This paper presented a computationally efficient real-time implementation of long-horizon FCS-MPC on an FPGA. Modifications in the sphere decoder that significantly reduce the computation time were presented. Experimental results showed the effectiveness of the algorithm based on a test case of a three-level NPC inverter driving an IM. As shown, thanks to the proposed refinements, and for the considered system, a prediction horizon of three steps with a maximum execution time of 20 μs were achieved while guaranteeing optimality during steady-state operation for switching frequencies lower than 1 kHz, i.e., frequencies that are of interest when medium-voltage drives are of interest. Hence, an as low control frequency as 40 kHz could be easily achieved, ensuring high granularity of switching, and, thus, favorable system performance.

APPENDIX

The system matrices in (2) are

$$\boldsymbol{C}=\begin{bmatrix}1&0\\0&1\\0&0\\0&0\end{bmatrix}^T,\ \boldsymbol{F}=\begin{bmatrix}-\frac{1}{\tau_s}&0&\frac{X_m}{\tau_r D}&\omega_r\frac{X_m}{D}\\0&-\frac{1}{\tau_s}&-\omega_r\frac{X_m}{D}&\frac{X_m}{\tau_r D}\\\frac{X_m}{\tau_r}&0&-\frac{1}{\tau_r}&-\omega_r\\0&\frac{X_m}{\tau_r}&\omega_r&-\frac{1}{\tau_r}\end{bmatrix},$$

$$\boldsymbol{G}=\frac{X_r}{D}\frac{V_{dc}}{2}\begin{bmatrix}1&0\\0&1\\0&0\\0&0\end{bmatrix}\boldsymbol{K},\ \boldsymbol{K}=\frac{2}{3}\begin{bmatrix}1&-\frac{1}{2}&-\frac{1}{2}\\0&\frac{\sqrt{3}}{2}&-\frac{\sqrt{3}}{2}\end{bmatrix}.$$

The matrices in function (7) are

$$\boldsymbol{\Gamma}=\begin{bmatrix}\boldsymbol{CA}\\\boldsymbol{CA}^2\\\vdots\\\boldsymbol{CA}^N\end{bmatrix},\ \boldsymbol{\Upsilon}=\begin{bmatrix}\boldsymbol{CB}&\mathbf{0}_{2\times2}&\dots&\mathbf{0}_{2\times2}\\\boldsymbol{CAB}&\boldsymbol{CB}&\dots&\mathbf{0}_{2\times2}\\\vdots&\vdots&&\vdots\\\boldsymbol{CA}^{N-1}\boldsymbol{B}&\boldsymbol{CA}^{N-2}\boldsymbol{B}&\dots&\boldsymbol{CB}\end{bmatrix}$$

$$\boldsymbol{E}=\begin{bmatrix}\boldsymbol{I}_3\\\mathbf{0}_{3\times3}\\\vdots\\\mathbf{0}_{3\times3}\end{bmatrix},\ \boldsymbol{S}=\begin{bmatrix}\boldsymbol{I}_3&\mathbf{0}_{3\times3}&\dots&\mathbf{0}_{3\times3}\\-\boldsymbol{I}_3&\boldsymbol{I}_3&\dots&\mathbf{0}_{3\times3}\\\mathbf{0}_{3\times3}&-\boldsymbol{I}_3&\dots&\mathbf{0}_{3\times3}\\\vdots&\vdots&&\vdots\\\mathbf{0}_{3\times3}&\mathbf{0}_{3\times3}&\dots&\boldsymbol{I}_3\end{bmatrix}.$$

REFERENCES

[1] D. W. Novotny and T. A. Lipo, *Vector Control and Dynamics of Ac Drives*. Oxford, U.K.: Oxford Univ. Press, 1996.

[2] I. Takahashi and T. Noguchi, "A new quick-response and high-efficiency control strategy of an induction motor," *IEEE Trans. Ind. Appl.*, vol. IA-22, no. 5, pp. 820–827, Sep. 1986.

[3] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Madison, WI: Nob Hill, 2009.

[4] J. Rodríguez *et al.*, "Predictive current control of a voltage source inverter," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 495–503, Feb. 2007.

[5] P. Cortés, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodríguez, "Predictive control in power electronics and drives," *IEEE Trans. Ind. Electron.*, vol. 55, no. 12, pp. 4312–4324, Dec. 2008.

[6] J. Rodríguez *et al.*, "State of the art of finite control set model predictive control in power electronics," *IEEE Trans. Ind. Informat.*, vol. 9, no. 2, pp. 1003–1016, May 2013.

[7] S. Kouro, M. A. Perez, J. Rodríguez, A. M. Llor, and H. A. Young, "Model predictive control: MPC's role in the evolution of power electronics," *IEEE Ind. Electron. Mag.*, vol. 9, no. 4, pp. 8–21, Dec. 2015.

[8] S. Vazquez, J. Rodríguez, M. Rivera, L. G. Franquelo, and M. Norambuena, "Model predictive control for power converters and drives: Advances and trends," *IEEE Trans. Ind. Electron.*, vol. 64, no. 2, pp. 935–947, Feb. 2017.

[9] P. Karamanakos, E. Liegmann, T. Geyer, and R. Kennel, "Model predictive control of power electronic systems: Methods, results, and challenges," *IEEE Open J. Ind. Appl.*, vol. 1, pp. 95–114, Nov. 2020.

[10] P. Karamanakos and T. Geyer, "Guidelines for the design of finite control set model predictive controllers," *IEEE Trans. Power Electron.*, vol. 35, no. 7, pp. 7434–7450, Jul. 2020.

[11] T. Geyer, P. Karamanakos, and R. Kennel, "On the benefit of long-horizon direct model predictive control for drives with *LC* filters," in *Proc. IEEE Energy Convers. Congr. Expo.*, Pittsburgh, PA, USA, 2014, pp. 3520–3527.

[12] P. Karamanakos, T. Geyer, N. Oikonomou, F. D. Kieferndorf, and S. Manias, "Direct model predictive control: A review of strategies that achieve long prediction intervals for power electronics," *IEEE Ind. Electron. Mag.*, vol. 8, no. 1, pp. 32–43, Mar. 2014.

[13] A. Galassini, G. Lo Calzo, A. Formentini, C. Gerada, P. Zanchetta, and A. Costabeber, "uCube: Control platform for power electronics," in *Proc. IEEE Workshop Elect. Mach. Des. Control Diagn.*, Nottingham, U.K., 2017, pp. 216–221.

[14] E. Liegmann, T. Schindler, P. Karamanakos, A. Dietz, and R. Kennel, "UltraZohm-An open-source rapid control prototyping platform for power electronic systems," in *Proc. Int. Aegean Conf. Elect. Mach. Power Electron. Int. Conf. Optim. Elect. Electron. Equip"*, Brasov, Romania, Sep. 2021, pp. 445–450.

[15] A. Rueetschi, P. Syrpas, B. Flak, K. Tomzik, and P. Steimer, "Heterogeneous control platform design for power conversion systems," *IEEE Trans. Ind. Informat.*, pp. 1–11, 2021, Early Access, doi: 10.1109/TII.2021.3104285.

[16] T. Geyer and D. E. Quevedo, "Multistep finite control set model predictive control for power electronics," *IEEE Trans. Power Electron.*, vol. 29, no. 12, pp. 6836–6846, Dec. 2014.

[17] R. Baidya, R. P. Aguilera, P. Acuña, S. Vasquez, and H. d. T. Mouton, "Multistep model predictive control for cascaded h-bridge inverters-formulation and analysis," *IEEE Trans. Power Electron.*, vol. 33, no. 1, pp. 876–886, Jan. 2018.

[18] P. Acuña, C. Rojas, R. Baidya, R. P. Aguilera, and J. Fletcher, "On the impact of transients on multistep model predictive control for medium-voltage drives," *IEEE Trans. Power Electron.*, vol. 34, no. 9, pp. 8342–8355, Sep. 2019.

[19] A. Andersson and T. Thiringer, "Assessment of an improved finite control set model predictive current controller for automotive propulsion applications," *IEEE Trans. Ind. Electron.*, vol. 67, no. 1, pp. 91–100, Jan. 2020.

[20] M. Dorfling, H. Mouton, T. Geyer, and P. Karamanakos, "Long-horizon finite-control-set model predictive control with non-recursive sphere decoding on an FPGA," *IEEE Trans. Power Electron.*, vol. 35, no. 7, pp. 7520–7531, Jul. 2020.

[21] E. Zafra, S. Vazquez, A. Marquez Alcaide, L. G. Franquelo, J. I. Leon, and E. Perez Martin, "K-best sphere decoding algorithm for long prediction horizon FCS-MPC," *IEEE Trans. Ind. Electron.*, pp. 1–11, 2021, *Early Access*, doi: 10.1109/TIE.2021.3104600.

[22] S. A. Bin Khalid, E. Liegmann, P. Karamanakos, and R. Kennel, "High-level synthesis of a long horizon model predictive control algorithm for an FPGA," in *Proc. Int. Exhib. Conf. Power Electron. Intell. Motion, Renew. Energy Energy Manag.*, 2020, pp. 1544–1551.

[23] E. Liegmann, P. Karamanakos, and R. Kennel, "Implementation of a long-horizon model predictive control algorithm on an embedded system," in *Proc. Eur. Power Electron. Conf.*, Ghent, Belgium, 2021, pp. P.1–P.10.

[24] J. Holtz, "The representation of ac machine dynamics by complex signal flow graphs," *IEEE Trans. Ind. Electron.*, vol. 42, no. 3, pp. 263–271, Jun. 1995.

[25] P. Karamanakos, T. Geyer, and R. Kennel, "Suboptimal search strategies with bounded computational complexity to solve long-horizon direct model predictive control problems," in *Proc. IEEE Energy Convers. Congr. Expo.*, Montreal, QC, Canada, 2015, pp. 334–341.

[26] T. Geyer, *Model Predictive Control of High Power Converters and Industrial Drives*. Hoboken, NJ: Wiley, 2016.

[27] J. Caba, F., Rincón, J. Barba, J. De La Torre, J. Dondo, and J. C. López, "Towards test-driven development for FPGA-based modules across abstraction levels," *IEEE Access*, vol. 9, pp. 31 581–31 594, Feb. 2021.

[28] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Mar. 2011.

[29] S. Wendel *et al.*, "UltraZohm-A powerful real-time computation platform for MPC and multi-level inverters," in *Proc. WorkshopPred. Control Elect. Drives Power Electron.*, 2019, pp. 1–6.

[30] P. Karamanakos, T. Geyer, and R. Kennel, "A computationally efficient model predictive control strategy for linear systems with integer inputs," *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 4, pp. 1463–1471, Jul. 2016.

[31] P. Karamanakos, T. Geyer, and R. P. Aguilera, "Long-horizon direct model predictive control: Modified sphere decoding for transient operation," *IEEE Trans. Ind. Appl.*, vol. 54, no. 6, pp. 6060–6070, Nov./Dec. 2018.

EYKE LIEGMANN (Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical power engineering from RWTH Aachen University, Aachen, Germany, in 2013 and 2016, respectively. He is currently working toward the Dr.-Ing. degree with the Technical University of Munich, Munich, Germany.

In 2014, he was an intern with ABB Corporate Research, Västerås, Sweden. From 2015 to 2016, he was with the Institute for Automation of Complex Power Systems, RWTH Aachen University. In 2018, he was Visiting Ph.D. Student with Stellenbosch University, Stellenbosch, South Africa. In 2019, he visited Tampere University, Tampere, Finland. His research interests include model predictive control of electrical drive systems and the real-time implementation of control algorithms on embedded systems.

Mr. Liegmann was the recipient of the Best Exhibition Award at the 2016 Power and Energy Student Summit and the 2019 Best Student Paper Award at the 2019 IEEE International Symposium on Predictive Control of Electrical Drives and Power Electronics.

PETROS KARAMANAKOS (Senior Member, IEEE) received the Diploma and the Ph.D. degrees in electrical and computer engineering from the National Technical University of Athens (NTUA), Athens, Greece, in 2007 and 2013, respectively.

From 2010 to 2011, he was with the ABB Corporate Research Center, Baden-Dättwil, Switzerland, where he worked on model predictive control strategies for medium-voltage drives. From 2013 to 2016, he was a Postdoc Research Associate with the Chair of Electrical Drive Systems and Power Electronics, Technische Universität München, Munich, Germany. Since September 2016, he has been an Assistant Professor with the Faculty of Information Technology and Communication Sciences, Tampere University, Tampere, Finland. His main research interests include the intersection of optimal control, mathematical programming and power electronics, including model predictive control and optimal modulation for power electronic converters and AC variable speed drives.

Dr. Karamanakos was the recipient of the 2014 Third Best Paper Award of the IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS and two Prize Paper Awards at conferences. He is an Associate Editor for the IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS and IEEE OPEN JOURNAL OF INDUSTRY APPLICATIONS.

RALPH KENNEL (Senior Member, IEEE) received the Diploma and Dr.-Ing. (Ph.D.) degrees from the University of Kaiserslautern, Kaiserslautern, Germany, in 1979 and 1984, respectively, and the Doctoral degree (honoris causa) from Universitatea Stefan cel Mare, Suceava, Romania, in 2018.

From 1983 to 1999, he worked on several positions with Robert BOSCH GmbH, Germany. Until 1997, he was responsible for the development of servo drives. He was one of the main supporters of VECON and SERCOS interface, two multicompany development projects for a microcontroller and a digital interface especially dedicated to servo drives. Furthermore, he actively took part in the definition and release of new standards with respect to CE marking for servo drives. Between 1997 and 1999, he was responsible for Advanced and Product Development of Fractional Horsepower Motors in automotive applications. His main activity was preparing the introduction of brushless drive concepts to the automotive market. From 1994 to 1999, he was appointed as a Visiting Professor with Newcastle University, Newcastle-upon-Tyne, U.K. From 1999 to 2008, he was a Professor of electrical machines and drives with Wuppertal University, Wuppertal, Germany. Since 2008, he has been a Professor of electrical drive systems and power electronics with Technische Universtät München, Munich, Germany. His main research interests include sensorless control of ac drives, predictive control of power electronics, and hardware-in-the-loop systems.

Dr. Kennel is a Fellow of IET (former IEE) and a Chartered Engineer in the U.K. Within IEEE, he is Treasurer of the Germany Section and also a Distinguished Lecturer of the Power Electronics Society (IEEE-PELS). Dr. Kennel was the recipient of the 2013 the Harry Owen Distinguished Service Award from IEEE-PELS, the EPE Association Distinguished Service Award in 2015, and the 2019 EPE Outstanding Achievement Award. Dr. Kennel was appointed as an Extraordinary Professor with the University of Stellenbosch (South Africa) from 2016 to 2019 and as a Visiting Professor with the Haixi Institute, Chinese Academy of Sciences from 2016 to 2021.