Hardware and Software Co-Design for Accelerating Model Predictive Control on Heterogeneous Compute Systems

Marco Guerreiro, Marco Groß, and Steven Liu
Chair of Control Systems, RPTU University Kaiserslautern-Landau, Kaiserslautern, Germany
Corresponding author: marco.guerreiro@rptu.de

***Abstract*—The next-generation power system requires converters capable of a dynamic response while meeting multiple control objectives and constraints. These requirements can be systematically handled with model predictive control (MPC), a powerful optimization-based technique. This paper discusses a strategy to co-design an MPC algorithm on heterogeneous compute systems consisting of a processor and a field-programmable array (FPGA). The strategy is to use the processor to execute operations that have a high degree of complexity and data dependency, and the FPGA for operations with low data dependency and possibilities of parallelism. We use high-level synthesis (HLS) to aid the FPGA implementation and show that the strategy is able to synthesize a real-time capable controller for a grid-connected three-phase inverter.**

***Index Terms*—power converters, control, FPGA, high-level synthesis.**

I. Introduction

Power converters are key elements in the transition to the next generation power system. Modern converters have evolved to complex systems, designed to cope with various grid requirements such as processing bidirectional power flow, operation with medium- and high-voltage dc, and grid-forming operation [1], [2]. Controlling such converters presents a substantial challenge, since they require control systems capable of handling multiple control objectives while meeting physical constraints of the converter. One approach that can systematically handle these requirements is model predictive control (MPC), an optimization-based technique that predicts the behavior of the plant and optimizes control actions based on control objectives and system constraints [3].

Although MPC is an established control technique, it is not yet commonly adopted for power converter control. Fully leveraging the benefits of MPC requires a long prediction horizon, which is difficult to achieve in power converters due to their sub-millisecond real-time constraints and the computational complexity of the MPC algorithm [4]. Moreover, most of the tools available for embedded optimization are aimed at large-scale systems and are not ideally suited for the small- to medium-sized problems in power converter control.

To overcome these issues, many of the existing realizations of MPC controllers for power electronics use a high-performance digital signal processor (DSP) or a field-programmable gate array (FPGA), depending on the nature of the computations required by the MPC algorithm [5]. DSPs are better suited for algorithms that are sequential in nature, while FPGAs are used for algorithms with operations that can be parallelized. In practice, however, many MPC algorithms present a mixed structure and it is difficult to fully leverage the advantages of a DSP- or an FPGA-based architecture. Nowadays, there are accessible heterogeneous compute systems that integrate a high-performance processor and an FPGA on the same chip, providing a platform with sequential and parallel processing capabilities that are well suited for the realization of MPC algorithms [6], [7]. The main challenges in using these platforms is efficiently partitioning the MPC algorithm between the processor and FPGA, i.e. the hardware and software co-design, which usually requires expertise in hardware description languages (HDL) and fixed-point data representation.

This paper discusses the co-design of an MPC algorithm on a heterogeneous compute platform consisting of a processor and an FPGA. The algorithm is partitioned between the two according to its data dependency, complexity, and possibility of parallelism. Steps of the algorithm with a high degree of data dependency but few operations are executed by the processor, due to its higher clock and dedicated instructions for floating-point operations. Steps of the algorithm with a lower degree of data dependency and larger number of operations are executed by the FPGA to leverage its parallelism. High-level synthesis (HLS) is used for the FPGA implementation, in order to avoid using HDL and explicit manipulation of fixed-point data. HLS is a tool that allows converting algorithms written in high-level languages such as C and C++ to HDL code, while also abstracting the data representation [8], [9]. We apply the method to synthesize an MPC controller for a grid-connected three-phase inverter, and show that the resulting controller is feasible with an execution time well below the real-time requirement.

The rest of this paper is organized as follows. Section II presents the MPC formulation and algorithm. Section III discusses the design of the algorithm on the heterogeneous compute system, and Section IV discusses its actual realization on a real platform. The validation and results of the implementation are presented in Section V, and Section VI concludes this paper.

This research was made possible by funding from the Carl-Zeiss-Stiftung.

 DOI 10.1109/IECON58223.2025.11221641

II. Model Predictive Control

This section discusses the MPC formulation used in this paper, and an active-set method to solve the resulting optimization problem. The formulation assumes a discrete-time system, and uses state augmentation to include integrators for the outputs. Only a brief overview is given here. For more details and discussion the reader is referred to [10].

A. Problem formulation

Assume a discrete-time linear, time-invariant system described by

$$x_{\mathrm{m}}(k+1) = A_{\mathrm{m}}x(k) + B_{\mathrm{m}}u(k) \tag{1a}$$
$$y(k) = C_{\mathrm{m}}x_{\mathrm{m}}(k) \tag{1b}$$

with $x_{\mathrm{m}} \in \mathbb{R}^n$, $u, y \in \mathbb{R}^q$, $A_{\mathrm{m}} \in \mathbb{R}^{n\times n}$, $B_{\mathrm{m}} \in \mathbb{R}^{n\times q}$, and $C \in \mathbb{R}^{q\times n}$. The system (1) is augmented with an integrator for each output, leading to the model

$$x(k+1) = Ax(k) + B\Delta u(k) \tag{2a}$$
$$y(k) = Cx(k) \tag{2b}$$

with $x = \begin{bmatrix}\Delta x_{\mathrm{m}}^{\intercal}(k) & y^{\intercal}(k)\end{bmatrix}^{\intercal}$, $\Delta x_{\mathrm{m}}(k) = x_{\mathrm{m}}(k) - x_{\mathrm{m}}(k-1)$, $\Delta u(k) = u(k) - u(k-1)$, and

$$A = \begin{bmatrix} A_{\mathrm{m}} & 0_{n\times q} \\ C_{\mathrm{m}}A_{\mathrm{m}} & I_q \end{bmatrix} \tag{3a}$$
$$B = \begin{bmatrix} B_{\mathrm{m}} \\ C_{\mathrm{m}}B_{\mathrm{m}} \end{bmatrix} \tag{3b}$$
$$C = \begin{bmatrix} 0_{q\times n} & I_q \end{bmatrix} \tag{3c}$$

where I is the identity matrix.

The problem of controlling system (2) can be cast in an optimization framework by defining an objective to be optimized for a certain time window (the prediction horizon). Solving the control problem then becomes a problem of finding the control sequence that minimizes this objective, subject to system constraints. The objective can be written in terms of a cost index stated in a quadratic form, so that it can be cast in a standard quadratic programming (QP) format. Usually, this cost contains the tracking error and additional control objectives. Here, the following cost is used [10]:

$$J = \sum_{i=k_i}^{k_i+N-1} \big(r(i+1) - y(i+1)\big)^2 + r_w\big(\Delta u(i)\big)^2 \tag{4}$$

with constraints

$$x_{\min} \le x_{\mathrm{m}} \le x_{\max} \tag{5a}$$
$$u_{\min} \le u \le u_{\max} \tag{5b}$$

where $r \in \mathbb{R}^q$ is the reference, N is the prediction horizon, k_i is the sampling instant, $r_w \in \mathbb{R}^{q\times q}$ is a diagonal matrix containing weighting factors that are used to tune the response of the controller, and $x_{\min}$, $x_{\max}$, $u_{\min}$ and $u_{\max}$ are the bounds for the states and the inputs. This cost and the constraints can be written as a function of the input, initial state and system matrices (also denoted as condensed formulation), and the optimization problem can be written as a standard QP problem of the form

$$\begin{aligned} \min_{\Delta U} \quad & J = \frac{1}{2}\Delta U^{\intercal}E\Delta U + \Delta U^{\intercal}F \\ \text{s.t.} \quad & M\Delta U \le \gamma \end{aligned} \tag{6}$$

with $\Delta U = \begin{bmatrix}\Delta u(k_i) & \cdots & \Delta u(k_i+N-1)\end{bmatrix}^{\intercal}$, $E = \Phi^{\intercal}\Phi + R_w$, $F = -\Phi^{\intercal}\big(R - Gx(k_i)\big)$. The matrix M and vector γ represent the inequality constraints, and thus their dimensions depends on the constraints of the system. The matrix R_w is given by $R_w = I_N \otimes r_w$, and the vector R is given by $R = \begin{bmatrix}r(k_i+1) & r(k_i+2) & \cdots & r(k_i+N)\end{bmatrix}^{\intercal}$. The matrices Φ and G result from the model and are defined as in [10].

Now the problem becomes that of solving (6), whose solution provides the optimal control sequence over the prediction horizon N. Following the receding horizon policy, the first sample of ΔU is applied to the system, and the process is repeated in the next sampling instant.

B. Active-set methods and Hildreth's QP procedure

There are different methods and algorithms to solve (6). For relatively small systems, such as those found in the control of power converters, active-set methods tend to converge faster and be less computationally expensive [11]. In this approach, problem (6) can be written as a problem of finding the Lagrange multipliers associated with the constraints that optimize an equivalent problem. Once found, these multipliers are used to determine the optimal control sequence. The equivalent problem is stated as [10]

$$\begin{aligned} \min_{\lambda} \quad & J = \frac{1}{2}\lambda^{\intercal}H\lambda + \lambda^{\intercal}K \\ \text{s.t.} \quad & \lambda \ge 0 \end{aligned} \tag{7}$$

where the vector $\lambda \in \mathbb{R}^p$ contains the Lagrange multipliers, $H = ME^{-1}M^{\intercal}$, and $K = \gamma + ME^{-1}F$. The dimension of the λ vector depends on the number of constraints in the problem. Although problem (7) is another QP problem, the fact that it contains only lower bounds (even though the original problem can have lower and upper bounds) can be exploited to find its solution. Hildreth's QP procedure is one such algorithm [12]. Its main goal is to optimize for each element λ_i of λ at a time:

$$\frac{\partial J}{\partial \lambda_i} = 0 \rightarrow \lambda_i = -\frac{1}{h_{ii}}\left(k_i + \sum_{j=1}^{i-1} h_{ij}\lambda_j + \sum_{j=i+1}^{p} h_{ij}\lambda_j\right) \tag{8}$$

where h_{ij} is the element on the i-th row and j-th column of matrix H, and k_i is the i-th element of vector K. In each step of the algorithm, λ_i is determined based on the most recent entries of the λ vector, and if (8) results in $\lambda_i < 0$, then λ_i is set to zero. One iteration of the algorithm is completed when all elements of λ have been updated, and the algorithm stops after l iterations according to some convergence criteria. Then, the optimal control sequence is found with

$$\Delta U = -E^{-1}F - E^{-1}M^{\intercal}\lambda \tag{9}$$

 DOI 10.1109/IECON58223.2025.11221641

Algorithm 1 Hildreth's QP procedure

Input: H, K, l, p
Output: λ

```
λ ← 0
for k = 0, ..., l − 1 do
    for i = 0, ..., p − 1 do
        acc ← 0
        λ_i ← 0
        for j = 0, ..., p − 1 do
            acc ← acc + h_ij λ_j
        λ_i ← −(1/h_ii) (k_i + acc)
        if λ_i < 0 then
            λ_i ← 0
```

The procedure for a fixed number of iterations is summarized in Algorithm 1. The algorithm requires the matrix H, vector K, number of iterations (l) and the size of λ (p).

III. Implementation of the MPC Algorithm in a Heterogeneous Compute System

This section discusses the MPC algorithm described in the previous section, and proposes a way to partition the algorithm in order to implement it in a heterogeneous system consisting of a central processing unit (CPU) and an FPGA.

To realize the MPC algorithm in an embedded system with constrained computational resources and hard real-time constraints, it is important to reduce the number of online computations as much as possible. An analysis of problems (6) and (7) shows that the matrices E, M, and H are static, and thus can be computed offline. Meanwhile, vectors F, γ, and K depend on the states at instant k_i (the initial conditions) and the reference signal, and thus need to be updated at every sampling instant. However, these vectors contain terms which can be precomputed offline. For example, the vector K is given by $K = \gamma + ME^{-1}F$, where the factor ME^{-1} can be computed offline, such that no matrix inversion is required online.

Algorithm 2 shows the steps for the realization of the MPC algorithm, focusing on the parameters that need to be updated at every sampling instant. (Static parameters have been omitted in Algorithm 2.) In the first step, the augmented state vector x is obtained from the measurements and previous samples of the state vector x_m. This vector is used in the next step of the algorithm to obtain the vectors F, γ, and K. At this point, the QP problem (7) is set up and can be solved. This is shown in step 3 of the algorithm, where the vector λ is found based on the new K vector. With the updated Lagrange multipliers, the optimal control sequence for the whole prediction horizon is obtained, from which the actual control output is computed.

All steps in Algorithm 2 are vector and matrix operations. Although they are well suited for FPGA implementation, a full FPGA implementation might lead to a poor performance because the steps of the algorithm have to be executed in sequence due to their data dependency. Moreover, multiple data sources increase the complexity of the FPGA implementation.

Algorithm 2 Model predictive control algorithm

Input: $x_\mathrm{m}(k)$, $x_\mathrm{m}(k-1)$, $u(k-1)$, $r(k)$
Output: $u(k)$

1: $x \leftarrow$ assemble_augmented_state($x_\mathrm{m}(k)$, $x_\mathrm{m}(k-1)$)
2: F, γ, $K \leftarrow$ update_vectors(x, $u(k-1)$, $r(k)$)
3: $\lambda \leftarrow$ solve_qp(K)
4: $\Delta U \leftarrow$ optimal_control_sequence(F, λ)
5: $u(k) \leftarrow \Delta U_{1:m} + u(k-1)$

The most computationally expensive step of Algorithm 2 is step 3, where the QP problem (7) is iteratively solved with Algorithm 1. Despite its iterative nature, Algorithm 1 is relatively simple because it requires only two inputs (one of which is static), produces one output, and the innermost *for* loop of the algorithm consists of multiply-and-accumulate operations. This structure presents well defined data dependencies that simplify the hardware implementation, as well as offering an opportunity for parallelism, making Algorithm 1 particularly suited for FPGA acceleration. The remaining steps of Algorithm 2 are better suited for a processor-based implementation because although they have a more complex data dependency structure, the computations are straightforward.

Based on this discussion, the partitioning of the MPC algorithm in a heterogeneous compute system consisting of a central processing unit (CPU) and an FPGA is shown in Fig. 1. Static matrices and other parameters are stored in memory for the CPU and the FPGA at compile time and synthesis time, respectively. At run time, the CPU executes the first two steps of Algorithm 2 to obtain the updated K vector. This vector is sent to the FPGA, which in turn executes Algorithm 1 to obtain the new λ vector. The new λ vector is returned to the CPU, which completes the execution of the MPC algorithm.

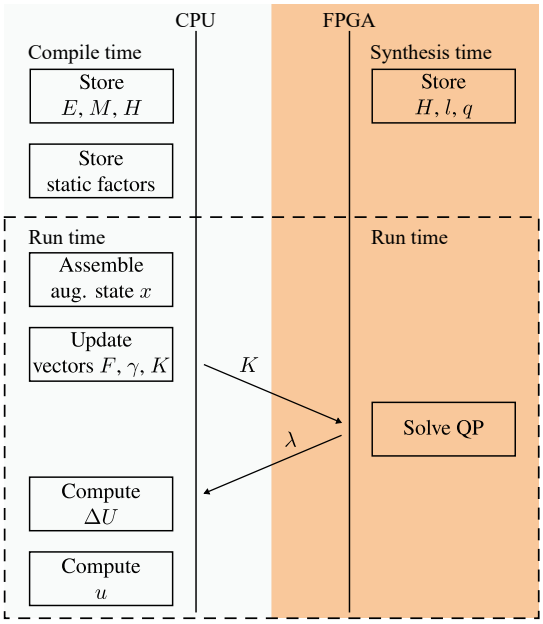

Fig. 1. Partitioning of the MPC algorithm in a heterogeneous compute system.

 DOI 10.1109/IECON58223.2025.11221641

IV. Realization of the MPC Algorithm with a Zynq-7020 System-on-a-Chip Device

This section discusses the realization of Algorithm 2 on a heterogeneous platform, the Zynq-7020 System-on-a-Chip (SoC) device [6], more specifically the XC7Z020-1CLG400C model. This SoC consists of a dual-core ARM A9 processor and an Artix-7-equivalent FPGA. Although the device contains a dual-core processor, only one core of the processor is used for execution of the algorithm, while the other core is used for external interfacing.

A. CPU implementation

As shown in Fig. 1, the CPU is responsible for executing steps 1, 2, 4 and 5 of Algorithm 2. These steps were implemented in the C programming language, using single-precision floating-point format. The code is executed from RAM, and the ARM A9 processor is clocked at 650 MHz. The C code was compiled with GCC using the "O2" optimization flag.

B. FPGA implementation with HLS

The FPGA is responsible for executing step 3 of Algorithm 2, namely the QP solver detailed in Algorithm 1. The algorithm was first implemented in C, and then the Vitis HLS tool was used to convert the C code to HDL. This results in logic block that can be implemented directly in the FPGA, and is denoted as an IP core.

The translation to HDL can be aided by introducing *pragma* directives in the C code that allow controlling the synthesis steps with respect to signal interfaces, the amount and type of hardware resources to use for specific computations, as well as other modifications, e.g. to facilitate interpretation of synthesis results. For further information, the reader is referred to [13].

As the MPC controller is partitioned between CPU and FPGA efficient data transfers must be ensured in order to keep the communication overhead minimal. In the Zynq device, this is achieved by using the Advanced eXtensible Interface (AXI) protocol paired with a Direct Memory Access (DMA) controller. The use of the AXI protocol as interface for the IP core is controlled using *pragma* directives. For the implementation of Algorithm 1, the ports for the parameters K and λ are defined as AXI streaming interfaces.

Algorithm 1 can also be reformulated to improve its execution. Consider the following expansion of (8):

$$\lambda_i = -\frac{k_i}{h_{ii}} - \sum_{j=1}^{i-1} \frac{h_{ij}}{h_{ii}} \lambda_j - \sum_{j=i+1}^{p} \frac{h_{ij}}{h_{ii}} \lambda_j \tag{10}$$

Since matrix H is static, it can be modified offline to reduce the required computation steps. First, every element on the main diagonal is replaced by its reciprocal. This makes the algorithm division-free and thereby significantly decreases computational effort. Next, in each row, all elements h_{ij} are multiplied with the new main diagonal element of this row, such that h_{ij} is now replaced by h_{ij}/h_{ii}. Denoting the elements of the modified matrix as h'_{ij}, this results in the following reduced formulation of (8):

$$\lambda_i = -k_i h'_{ii} - \sum_{j=1}^{i-1} h'_{ij} \lambda_j - \sum_{j=i+1}^{p} h'_{ij} \lambda_j \tag{11}$$

While the second step of the modification has a negligible effect on the execution time when implementing the QP solver on the CPU, the improvement obtained for the HLS generated IP core is significant. Synthesis is improved because it requires only two factors for each summand instead of three since $h_{ij} h'_{ii} \lambda_j$ is replaced by $h'_{ij} \lambda_j$. Another important advantage is achieved when choosing a fixed point implementation, which is a common method in HLS to reduce hardware consumption and execution time. If only the first modification step is applied, the order of magnitude of the main diagonal elements will often differ largely from the remaining elements. The second step has the effect of normalizing the matrix elements, so that the interval of values present in the matrix is minimized. Therefore, the bit width of the fixed point representation can be decreased, which again reduces hardware requirements and latency.

Further optimizations can be achieved using additional *pragma* directives and by writing the code in a manner that allows HLS to parallelize operations. The algorithm consists of three nested *for* loops: an outer loop for the multiple iterations, an intermediate loop to compute each element λ_i of the Lagrange multipliers, and an innermost loop to perform the multiply-and-accumulate operations of (11). Because Algorithm 1 demands each λ_i to be computed sequentially, only the innermost loop can be parallelized. The latency is minimized by first separating multiplication and accumulation in two individual loops. Both of the loops are unrolled using the appropriate *pragma* as there are no loop-carried dependencies between the operations. By default, HLS maps arrays to BRAM from which elements can only be read sequentially. In order to allow all elements to be accessed in parallel, array partitioning can be applied by adding the corresponding *pragma* directive. This instructs HLS to store array elements in registers instead of BRAM.

Finally, latency and hardware resources are lowered by choosing a larger clock period for the IP core as this allows more parallel operations to be scheduled in a single clock cycle and thus reduces the latency in clock cycles enough to compensate the higher duration of an individual cycle.

An advantage of implementing Algorithm 1 in the FPGA is that its execution time grows linearly with the number p of Lagrange multipliers, i.e. the number of constraints. If Algorithm 1 is executed sequentially, the execution time grows with $O(p^2)$ since (11) is repeated for each Lagrange multiplier. By fully parallelizing (11), the execution time grows with $O(p)$, since the execution time of the parallelized section no longer depends on the number of Lagrange multipliers. However, this come at the expense of increased FPGA logic resources required to parallelize more operations, which grows with $O(p)$.

 DOI 10.1109/IECON58223.2025.11221641

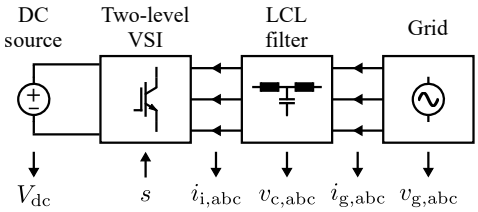

Fig. 2. Block diagram of grid-connected three-phase inverter.

TABLE I
PARAMETERS USED FOR THE SIMULATION OF THE THREE-PHASE INVERTER.

Parameter	Value	Parameter	Value
V_{dc}	650 V	L_i, R_i	3.4 mH, 0.1 Ω
V_{ac}	162.5 V	L_g, R_g	1.8 mH, 0.1 Ω
ω	$2\pi 50$ rad/s	C_f, R_c	20 µF, 0.06 Ω
f_s	7.5 kHz		

V. Results and Discussion

This section shows the results of the implementation of the MPC controller on the Zynq-7020 SoC for the control of a grid-connected three-phase inverter. The closed-loop response and real-time feasibility of the controller were verified using a processor-in-the-loop (PiL) scheme [14]. In this scheme, the controller is implemented in a real target, but the model of the system (including analog-to-digital converters (ADCs) and modulators) is simulated in a computer. The simulation and the controller exchange data through a communication channel, creating a closed-loop system. By measuring the time it takes to execute the controller on the real target, it is possible to verify the real-time capability of the controller. For simulation, the PLECS software was used.

For comparison purposes, the controller was implemented using three different approaches for the solver. The first approach consists of running Hildreth's algorithm as explained in Section III, using the CPU and FPGA of the Zynq platform, and is denoted as "Hild-Het." The second approach runs Hildreth's algorithm only on the CPU of the Zynq platform, and is denoted as "Hild-Hom." This allows comparing a heterogeneous and a homogeneous implementation of the same algorithm. A third implementation of the controller uses the Operator Splitting Quadratic Program (OSQP) solver, and runs only on the CPU of the Zynq platform. This allows comparing the proposed approach with a well-known solver used in embedded applications [15], [16]. For both Hildreth and OSQP algorithms, offline simulations were used to tune the parameters of each algorithm in order to reduce the number of iterations required by them. Then, for the closed-loop implementation, the algorithms were executed with a fixed number of iterations. For all cases, the C code was compiled for the Zynq platform using GCC and "O2" compiler optimization, and the clock of the processor was set to 650 MHz.

A block diagram of a three-phase inverter connected to an ac grid through an LCL filter is shown in Fig. 2, where $V_{dc} \in \mathbb{R}$ is the dc source, $s \in \mathbb{R}^3$ are the gate signals for the high-side switches of the inverter, $i_{i,abc} \in \mathbb{R}^3$ are the inverter-side currents, $v_{c,abc} \in \mathbb{R}^3$ are the filter capacitor voltages, $i_{g,abc} \in \mathbb{R}^3$ are the grid-side currents, and $v_{g,abc} \in \mathbb{R}^3$ is the grid-voltage. In this example, the objective is to control the grid-side currents, in order to control the power flow between the dc source and the ac grid. The model of the system in the dq frame is given by [17]

$$L_g \dot{i}_g = L_g W i_g - v_c + v_g \tag{12a}$$
$$L_i \dot{i}_i = L_i W i_i + v_c - u \tag{12b}$$
$$C_f \dot{v}_c = i_g - i_i + W v_c \tag{12c}$$
$$W = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix} \tag{12d}$$

where L_g, L_i, and C_f are the parameters of the LCL filter; $i_g, i_i, v_c \in \mathbb{R}^2$ are the grid-side current, inverter-side current, and filter capacitor voltage; $v_g \in \mathbb{R}^2$ and ω are the voltage and frequency of the ac grid; and $u \in \mathbb{R}^2$ is the control input to the space vector modulator of the converter. The parameters of the system are shown in Table I. In this problem, the inverter-side currents and the control inputs are constrained as follows:

$$\begin{bmatrix} -15 \\ -15 \end{bmatrix} \leq i_i \leq \begin{bmatrix} 15 \\ 15 \end{bmatrix} \tag{13a}$$
$$\begin{bmatrix} -328.4 \\ -181.7 \end{bmatrix} \leq u \leq \begin{bmatrix} 328.4 \\ 181.7 \end{bmatrix} \tag{13b}$$

This model is written in the QP form (6) by discretizing (12) using zero-order hold with a period of $T = 1/f_s = 133$ µs, the same as the switching frequency used for modulation of the gate signals of the inverter.

Fig. 3 shows the closed-loop response of the system with the MPC controller, using $N = 4$ and a weighting factor of $2 \cdot 10^{-5}$ for both inputs. The figure shows the control signals in the dq frame, and the grid-side currents, both in the dq and abc frames. The results shown were obtained

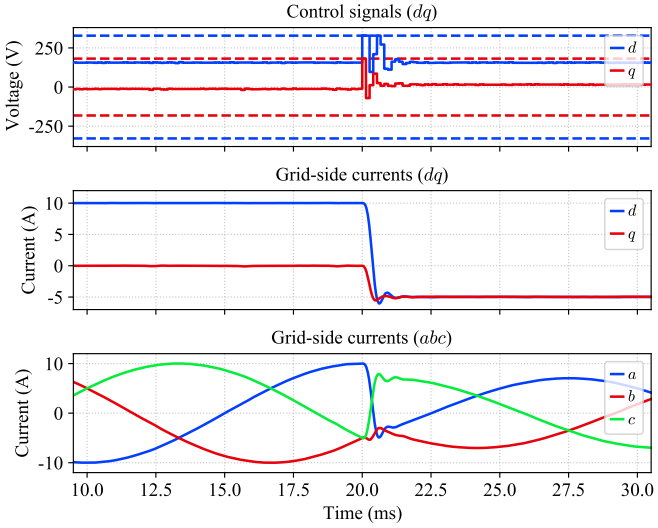

Fig. 3. Closed-loop validation of the MPC implementation for the three-phase inverter.

 DOI 10.1109/IECON58223.2025.11221641

TABLE II
IMPLEMENTATION RESULTS FOR THE THREE IMPLEMENTATIONS OF THE PREDICTIVE CONTROLLER.

Algorithm	Execution time	DSP slices utilization	Flip-flops utilization	Look-up tables utilization
Hild. het.	50.6 µs	60.0%	4.8%	25.5%
Hild. hom.	147.8 µs	-	-	-
OSQP hom.	125.3 µs	-	-	-

using the heterogeneous implementation of the algorithm, and demonstrate the response of the controller to a large change in the setpoint of the grid current. As the figure shows, the controller is able to drive the current to its new reference point within milliseconds, while keeping the control inputs inside their limits.

Table II shows the results of the execution time for the three different implementations, as well as the FPGA resource usage for the heterogeneous implementation of Hildreth's algorithm. The FPGA resources are as reported by the HLS tool, and the execution times are as measured in closed-loop. The execution time of the heterogeneous implementation is well below the real-time target of 133 µs, while its homogeneous implementation would not be feasible in real-time. The homogeneous implementation of the OSQP algorithm is just below the 133 µs real-time constraint, which would lead to high usage of the processor. As for the FPGA resource usage, the heterogeneous implementation of the Hildreth algorithm still leaves enough resources for implementation of additional logic that would be require to interface with a real converter, such as modulators for the gate signals and decoders for ADCs. For reference, the Hildreth implementation required 10 iterations, while OPSQ required 15 iterations.

VI. CONCLUSIONS

Advanced control strategies for modern power converters require increasingly complex embedded controllers for their implementation. Although the parallelism of FPGAs can be used to accelerate the implementation of such controllers, the expertise required to implement algorithms in HDL still presents a challenge for practitioners and researchers.

This paper discusses the implementation of an MPC controller on a heterogeneous compute platform. Using HLS, the core of the controller was implemented in the FPGA of the compute platform using with fixed-point data representation, while abstracting the need for proficiency in HDL. The real-time feasibility of the controller shows that HLS can be a valuable tool for their realization for power converter control.

REFERENCES

[1] B. K. Bose, "Power electronics, smart grid, and renewable energy systems," *Proc. IEEE*, vol. 105, no. 11, pp. 2011–2018, 2017.

[2] X.-K. Liu, Y.-W. Wang, Z.-W. Liu, and Y. Huang, "On the stability of distributed secondary control for DC microgrids with grid-forming and grid-feeding converters," *Automatica*, vol. 155, p. 111164, 2023.

[3] M. G. Forbes, R. S. Patwardhan, H. Hamadah, and R. B. Gopaluni, "Model predictive control in industry: Challenges and opportunities," *IFAC-PapersOnLine*, vol. 48, no. 8, pp. 531–538, 2015.

[4] S. Vazquez, J. I. Leon, L. G. Franquelo, J. Rodriguez, H. A. Young, A. Marquez, and P. Zanchetta, "Model predictive control: A review of its applications in power electronics," *IEEE Ind. Electron. Mag.*, vol. 8, no. 1, pp. 16–31, 2014.

[5] O. Gulbudak and E. Santi, "FPGA-based model predictive controller for direct matrix converter," *IEEE Trans. Ind. Electron.*, vol. 63, no. 7, pp. 4560–4570, 2016.

[6] M. Guerreiro, S. Kharade, P. dos Santos, and S. Liu, "Power electronics control with system-on-a-chip-based platforms," in *Proc. 2022 IEEE 20th Int. Power Electron. Motion Control Conf.*, 2022, pp. 353–359.

[7] S. Wendel et al., "UltraZohm - a powerful real-time computation platform for MPC and multi-level inverters," in *IEEE Int. Symp. Predictive Control Elect. Drives Power Electron.*, 2019, pp. 1–6.

[8] S. Lahti, M. Rintala, and T. D. Hämäläinen, "Leveraging modern C++ in high-level synthesis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 42, no. 4, pp. 1123–1132, 2023.

[9] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? a study on the state of high-level synthesis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 5, pp. 898–911, 2019.

[10] L. Wang, *Model Predictive Control System Design and Implementation Using MATLAB*. Springer, 2009.

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

[12] C. Hildreth, "A quadratic programming procedure," *Naval Research Logistics Quarterly*, vol. 4, no. 1, pp. 79–85, 1957.

[13] AMD, "Hls pragmas," https://docs.amd.com/r/en-US/ug1702-vitis-accelerated-reference/HLS-Pragmas, 2025, accessed: 2025-04-10.

[14] M. Guerreiro, W. Becker, P. dos Santos, and S. Liu, "An open processor-in-the-loop framework for power converter control," in *49th IEEE IECON*, 2023, pp. 1–6.

[15] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[16] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, "Embedded code generation using the OSQP solver," in *IEEE Conference on Decision and Control (CDC)*, 2017. [Online]. Available: https://doi.org/10.1109/CDC.2017.8263928

[17] P. Falkowski and A. Sikorski, "Finite control set model predictive control for grid-connected ac–dc converters with LCL filter," *IEEE Trans. Ind. Electron.*, vol. 65, no. 4, pp. 2844–2852, 2018.