2019 Fifth Indian Control Conference (ICC)
IIT Delhi, India, January 9-11, 2019

Implementation and Analysis of Offset-Free Explicit Model Predictive Controller on FPGA

Saket Adhau[1], Kiran Phalke[1], Apeksha Nalawade[1], Deepak Ingole[2], Sayli Patil[1] and Dayaram Sonawane[1]

Abstract— **It is well-known that the real-time implementation of Model Predictive Control (MPC) is cumbersome because of the huge burden of solving optimization problem on-line at each sample time. Due to this, MPC has been mainly restricted to processes with rather slow dynamics, such as the ones encountered in the oil and gas refineries. However, recent algorithmic advances (such as the explicit MPC) allowed usage of MPC to problems arising in the automotive or power electronics industry where the time scales are in the milli- to microsecond range. This paper focuses on the Field Programmable Gate Array (FPGA) implementation of offset-free explicit MPC and its detailed analysis for the position control of PMDC motor. We show the analysis of controller computational complexity in terms of memory, resource utilization, clock and power consumption. Along that effects of various tuning parameters on the number of regions is also presented with respect to the changing prediction horizon length. Finally, the performance of implemented offset-free explicit MPC is compared with the standard explicit MPC and Proportional-Integral controller for reference-tracking, constraints handling, and disturbance rejection. Results indicate that the performance of offset-free explicit MPC is superior but at the cost of increased memory footprint.**

I. Introduction

Model Predictive Control (MPC) is an advanced process control strategy that achieves optimal control input satisfying a set of constraints. Since 1980s oil and gas industries have made considerable contributions to the MPC theory and practices (see [1]). But, the primary reason, MPC flourished in these specific industries was the slow dynamic processes with sampling time in minutes to hours that used to be controlled by MPC. Considering, the modern applications of control theory like robotics, surgical instruments control, automobiles, electric and autonomous vehicles, where the processes are not static but the application changes as per the conditions around them. Thus, sampling time requirement is also high as compared to the traditional applications where MPC originated. Thus, to meet the needs of these modern day applications, researchers and engineers are taking efforts to implement MPC on an embedded platform for the real-time control [2], [3]. For implementing MPC on an embedded platform there are multiple ways. The authors in [4] have investigated the practical feasibility of MPC on $8\,\mathrm{bit}$ Arduino board with second-order model and sampling time of $20\,\mathrm{ms}$. MPC has been implemented on Field Programmable Gate Array (FPGA) by [5].

Based on the required computational complexity, an optimization problem in MPC can be solved using two ways namely on-line MPC and off-line or Explicit MPC (EMPC). On-line MPC is computationally burdening on the processor and require a longer time for solving quadratic problems. Therefore, it requires processors working at higher clock speeds. On the other hand, explicit MPC comprises of controllers data in the form of Look-Up Tables (LUTs) and point location algorithm which efficiently searches through LUTs to get an optimized solution for the control problem. Hence, it is computationally less complex and therefore less burdening on the processor [2], [6]. On the downside, memory requirement for EMPC is high as compared to on-line MPC as all the possible cases of control problem needs to be identified and stored in the actual memory of embedded platform. Despite the associated memory demand, its distinguish features have extended its application to several areas of engineering, an overview of recent applications of EMPC can be found in [7, Chapter 3].

Generally, disturbances enter in the system either through states or inputs. To improve the performance of the system affected by the disturbances and to achieve offset-free control objective, the system model is augmented with a disturbance model which is used to estimate and predict the mismatch between measured and predicted outputs. The state and disturbance estimates are used to initialize the MPC problem (see [8, Chapter 12]).

In this paper, we developed an FPGA-based offset-free EMPC solution for position control of a Permanent Magnet Direct Current (PMDC) motor. Explicit MPC is first constructed in MATLAB using Multi-Parametric Toolbox (MPT) and the low level C-code compatible for embedded platforms is exported and deployed on FPGA. The first part of result shows the analysis of explicit MPC with different settings of sampling time, prediction horizon, and the penalty on input and output. Considering the worst case scenario for memory requirement, we selected Xilinx's ZedBoard FPGA. It is a low-cost development board for the Xilinx Zynq-7000 SoC. The second part of the results, show the FPGA-based controller performance and the detailed analysis of its implementation. Further, the performance of designed offset-free explicit MPC is compared with the standard explicit MPC and Proportional-Integral (PI) controller for reference tracking and disturbance rejection. Moreover, to get an idea about memory usage with problem size and prediction horizon, the FPGA implementation of offset-free

[1] Department of Instrumentation and Control Engineering, College of Engineering Pune, Shivajinagar 411005, India {adhauss17.instru, phalkekp14.instru, nalawadeas14.instru, patilsd17.instru, dns.instru}@coep.ac.in
[2] University of Lyon, IFSTTAR, ENTPE, Lyon 69120, France deepak.ingole@ifsttar.fr

978-1-5386-6246-5/19/$31.00 ©2019 IEEE

EMPC is compared with standard EMPC and PI controller.

II. DC Motor Position Control

In this study, we show the analysis of embedded implementation of EMPC designed for the position control of a PMDC motor. This section describes the dynamical model of the PMDC motor.

A. DC Motor Modeling

For deriving a PMDC motor model, torque and electrical equations described in [9, Chapter 2] have been considered. The electrical circuit of the motor is shown in Fig. 1. It can be represented by a voltage source (V_a) across the coil of the armature. The electrical equivalent of armature coil can be denoted by an inductance (L_a) in series with a resistance (R_a) in series with an induced voltage or back electromotive force (emf) (V_m) which opposes the voltage source. A differential equations for the electrical circuit

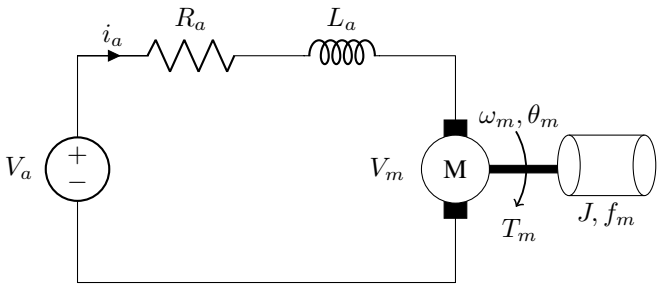

Fig. 1. Electrical equivalent circuit of PMDC motor.

shown in Fig. 1 can be derived by applying Kirchoff's voltage law. Considering angular velocity, armature current and the angular position as the states and armature voltage as the input, above continuous-time equations can be written in state space form as follows

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{1a}$$
$$y(t) = Cx(t) + Du(t), \tag{1b}$$

when the problem is of position control, system matrices A, B, C and D are given as

$$A = \begin{bmatrix} -\frac{f_m}{J} & \frac{k_m}{J} & 0 \\ -\frac{k_m}{L_a} & -\frac{R_a}{L_a} & 0 \\ 1 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{1}{L_a} \\ 0 \end{bmatrix},$$
$$C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, D = 0.$$

The nominal values and notations of system parameters considered in the model development are taken from the [10, Table I].

III. Offset-Free Explicit MPC

A. Plant Model

Consider a discrete-time version of the Linear-Time Invariant (LTI) system in (2),

$$x(t + T_s) = Ax(t) + Bu(t), \tag{2a}$$
$$y(t) = Cx(t) + Du(t), \tag{2b}$$

where $x(t) \in \mathbb{R}^n$ is the system state vector, $u(t) \in \mathbb{R}^l$ is the system input vector and $y(t) \in \mathbb{R}^m$ is the system output vector, moreover, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times l}$, $C \in \mathbb{R}^{m\times n}$ and $D \in \mathbb{R}^{m\times l}$ are system matrices with the assumption that pair (A, B) is stabilizable and (C, A) is detectable.

B. Disturbance Modeling

The objective is to design an offset-free explicit model predictive controller based on LTI model (2) in order to have measured output $y(t)$ track desired reference $r(t)$ with zero steady-state error in presence of plant-model mismatch and unmeasured disturbances. To achieve this objective, the plant model (2) is augmented with a disturbance vector $d(t) \in \mathbb{R}^p$ [8, Chapter 12] as shown below,

$$\underbrace{\begin{bmatrix} x(t+T_s) \\ d(t+T_s) \end{bmatrix}}_{x_e(t+T_s)} = \underbrace{\begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix}}_{A_e} \underbrace{\begin{bmatrix} x(t) \\ d(t) \end{bmatrix}}_{x_e(t)} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{B_e} u(t), \tag{3a}$$
$$y_e(t) = \underbrace{\begin{bmatrix} C & C_d \end{bmatrix}}_{C_e} \underbrace{\begin{bmatrix} x(t) \\ d(t) \end{bmatrix}}_{x_e(t)} + D_e u(t), \tag{3b}$$

where $B_d \in \mathbb{R}^{n\times p}$, $C_d \in \mathbb{R}^{m\times p}$ are the disturbance model matrices and dimensions of matrices 0 and I are $m \times l$ and $m \times m$ respectively. The subscript 'e' denotes the extended version of the combined state and disturbance.

C. State and Disturbance Estimation

Extended state x_e is estimated from the plant measurement by designing a Luenberger observer for augmented system (3) as follows,

$$\hat{x}_e(t + T_s) = A_e\hat{x}_e(t) + B_e u(t) + L_e(y(t) - \hat{y}_e(t)), \tag{4a}$$
$$\hat{y}_e(t) = C_e\hat{x}_e(t) + D_e u(t), \tag{4b}$$

where $L_e = \begin{bmatrix} L_x & L_d \end{bmatrix}^T$ is the filter gain matrices for the state (of dimension $n \times m$) and the disturbance (of dimension $p \times m$), respectively and can be obtain by pole placement.

D. MPC Formulation

Model predictive control is the form of control algorithm which uses a model describing the system and where the control action is obtained by solving a Constrained Finite Time Optimal Control (CFTOC) problem for the current state ($\hat{x}_e(t)$) of the plant at each sampling time (t). The sequence of optimal control inputs ($U^\star = \{u_0^\star, \ldots, u_{N-1}^\star\}$) is computed for a predicted evolution of the system model over a finite horizon (or prediction horizon (N)). However, only the first element of the control sequence ($u_0^\star$) is applied and the current state ($\hat{x}_e(t)$) of the system is then measured again at the next sampling time ($t + 1$).

Using LTI system model in (2) and disturbance observer

in (4) the MPC problem is designed as follows:

$$\min_{U} \sum_{k=0}^{N-1} \left(y_k - r_k\right)^T Q \left(y_k - r_k\right) + \Delta u_k^T R \Delta u_k \tag{5a}$$

$$\text{s.t. } x_{k+1} = Ax_k + Bu_k + B_d d_k, \tag{5b}$$

$$d_{k+1} = d_k, \tag{5c}$$

$$y_k = Cx_k + Du_k + C_d d_k, \tag{5d}$$

$$\Delta u_k = u_k - u_{k-1}, \tag{5e}$$

$$u_{\min} \leq u_k \leq u_{\max}, \tag{5f}$$

$$u_{-1} = 0, \tag{5g}$$

$$x_0 = \hat{x}(t), \tag{5h}$$

$$d_0 = \hat{d}(t), \tag{5i}$$

$$\forall k \in \{k = 0, \ldots, N-1\}, \tag{5j}$$

where $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{l \times l}$ are the weighting matrices, with condition $Q \succeq 0$ to be positive semi-definite, and $R \succ 0$ to be positive definite. We denote N as the prediction horizon, x_{k+1} as the vector of predicted states at sample time k, $U = \{u_0, \ldots, u_{N-1}\}$ as the sequence of control actions, r_k is the output reference trajectory to be tracked, and x_0, d_0, and u_{-1} are the given initial conditions. By solving (5) with a given initial conditions, the optimization yields open loop optimal input sequence $U^\star = \{u_0^\star, \ldots, u_{N-1}^\star\}$ from which only the first control action, i.e., $u_0^\star$ is applied to the plant and again a CFTOC problem (5) is solved at next sample time $k+1$.

E. Explicit MPC

In MPC, an optimization problem (5) needs to be solved at each sample time. Such an optimization problem can be formulated as a multi-parametric Quadratic Programming (mp-QP) problem as

$$\min_{U} \left\{ U^T H U + \tilde{x}_0^T F U \right\} + \tilde{x}_0^T Y \tilde{x}_0, \tag{6a}$$

$$\text{s.t. } GU \leq w + W\tilde{x}_0, \tag{6b}$$

where $\tilde{x}_0 = [\hat{x}_e^T(t) \;\; u^T(t-1) \;\; r^T(t)]^T$ is the vector of initial conditions and by denoting q as a number of inequalities, matrices $H \in \mathbb{R}^{l.N \times l.N}, F \in \mathbb{R}^{(n+p)\times(l.N)}, Y \in \mathbb{R}^{(n+p)\times(n+p)}, G \in \mathbb{R}^{q \times l.N}, w \in \mathbb{R}^q, W \in \mathbb{R}^{q \times n}$ can be obtain by weighting matrices Q and R. For the detailed description about the computation of the matrices H, F, Y, G, w, W see [6].

The optimal solution $U^\star$ is a Piecewise Affine (PWA) function of the initial condition, which can be calculated off-line by solving mp-QP problem [6]. This problem can be solved by using the freely available MATLAB based Multi-Parametric Toolbox (MPT) [11]. Once the mp-QP problem (6) is solved off-line and stored in the form of LUTs, in the on-line phase, explicit MPC uses the obtained optimal solution in a receding horizon fashion in which $U^\star = \kappa(\tilde{x}_0)$ is a continuous PWA function mapped over a polyhedral regions:

$$\kappa(\tilde{x}_0) = \begin{cases} F_1\tilde{x}_0 + g_1 & \text{if } \tilde{x}_0 \in \mathcal{R}_1 \\ \vdots & \\ F_M\tilde{x}_0 + g_M & \text{if } \tilde{x}_0 \in \mathcal{R}_M \end{cases} \tag{7}$$

where $\mathcal{R}_i = \{\tilde{x} \in \mathbb{R}^{(n+p)} \mid Z_i\tilde{x} \leq z_i\} \quad \forall\, i = 1, \ldots, M$ are the polyhedral regions and $F_i \in \mathbb{R}^{l \times (n+p)}$, $g_i \in \mathbb{R}^l$ are the linear gains. where the i^{th} gain is selected according to the set of linear inequalities $Z_i\tilde{x} \leq z_i$ that the state vector satisfies. Moreover, M denotes the total number of regions, h_i is a number of half-spaces in a polyhedral set and $Z_i \in \mathbb{R}^{h_i \times (n+p)}$, $z_i \in \mathbb{R}^{h_i}$ are the matrices which forms a polyhedral set. The advantage of explicit form of optimizer as shown in (7) is, computation of optimal control inputs reduces to a mere function evaluation. This task is division-free since, only addition and multiplication operations are required to evaluate $\kappa(\tilde{x}_0)$ for a specific value of $\tilde{x}_0$. In the on-line phase of explicit MPC, the task is to identify an index of the polyhedral region in which the current state $\tilde{x}_0$ lies. This problem is called point location problem which can be solved in sequential search algorithm (see [12, Algorithm 1]). In the next section, steps followed in the implement of the EMPC on a FPGA are described.

IV. FPGA Implementation of Explicit MPC: Methodology

A. Controller Design and SIL Verification

There are several state-of-the-art software tools available for the constriction and code generation of explicit MPC control laws, such as Multi-Parametric Toolbox (MPT), Parametric Optimization (POP) toolbox [13], and hybrid toolbox [14]. In this work, the explicit MPC controller in (5) is constructed using the MPT. Then, we did several closed-loop simulation to verify the performance of controller in terms of reference tracking, constraints and disturbance handling. In this step, we analyzed computational complexity of the offset-free EMPC with varying sampling time and penalty on the output and input, and the length of prediction horizon. This step is also called Software-In-the-Loop (SIL) verification where we can priori know the worst case run-time and the memory required to store control laws.

B. Code Generation and SIL Verification

The Zedboard support low-level C/C++ programming languages. Considering this, after SIL verification we exported a low-level C code of the sequential search algorithm. Along with the sequential search algorithm, this code generated vectors/matrices of controller i.e., Z_i, z_i, F_i, g_i which will stored on the on-chip memory of Zedboard. These vectors/matrices are exported with the single precision floating-point data format (32-bit).

C. FPGA Implementation Flow

In this step, we used Protoip (IP prototyping in FPGA hardware) [15] software tool which provides Tool Command Language (Tcl)-based functions accessible by both Xilinx

Vivado Design Suite and MATLAB. The main steps of the design flow can be performed via MATLAB command line interfacing. First, a design template is built using design parameters like inputs and outputs for the algorithm and a clock frequency etc. Then exported sequential search algorithm is deployed into the FPGA. After successful compilation, we can obtain the FPGA resource utilization in terms of BRAM, DSP blocks, FF, and LUTs. Estimated clock cycle achieved and the FPGA power utilization is also obtained at this stage. After, this the bitstream file is generated with for the Zedboard configuration.

D. HIL Verification

Once the FPGA is programmed with the EMPC algorithm, next step is to verify the controller performance with respect to various parameters like settling time, peak overshoot etc. The Hardware-In-the-Loop (HIL) co-simulation is performed using Protoip tool. First, the FPGA is turned on and the data packets consisting of initial condition, reference and previous control input are sent to the FPGA. For the first iteration, the logic written into FPGA calculates only first control action which then feed to the PMDC motor model present in the MATLAB.

For all these operations, MATLAB uses `FPGAclientMATLAB` function, which handles data reading and writing to the FPGA for a specified number of iterations. Finally, plotting the data points obtained from all the iterations, controller performance can be analyzed. At this stage, time required for the simulation for a specified number of iterations is also obtained, which is used to determine the sampling time achieved by the controller.

E. Resource Utilization and Memory Calculation

This section gives details about how the resource utilization of FPGA is combined in a single value so as to use it for further implementation analysis purpose. Generally, FPGA resources are given separately in terms of BRAM, DSP, FF and LUTs. In this paper, we are combining the data available from FPGA manufacturer's data-sheet about the actual memory occupied by the these 4 blocks to obtain a single number, that is memory required in kilobytes kB to store and perform a particular controller algorithm. A BRAM is characterized by a block of RAM each of width 18 kilobits kb, whereas each DSP slice is considered to have a 48-bits wide accumulator register for storage purpose, each FF is considered to store 1-bit information and each LUT is also characterized by a static RAM block each of size 64-bits (see [16]). All the four building blocks of FPGA, their total number of units available on the Zedboard and their unit memory occupation are represented in the Table I. Adding individual values will return a single representing total memory available for programmable logic implementation on Zedboard. Thus, it can be seen that the total memory available on the FPGA board including the memory to store data and also the one to perform computations is 1057 kB, which is approximately equal to 1 MB.

TABLE I
AVAILABLE RESOURCES ON A ZEDBOARD.

Resource	Available Size	(bits)	Total memory (kB)
BRAM	280	18000	627.2
DSP	220	48	1.3
FF	106400	1	13
LUT	53200	64	415.6

V. EXPLICIT MPC ANALYSIS: MATLAB RESULTS

This section presents the analysis of computational complexity (in terms of number of regions) of offset-free EMPC and compared with the standard EMPC for different settings as shown below:

- Sampling time, T_s; bounds: $0.001 \leq T_s \leq 0.1$.
- Horizon length, N; bounds: $2 \leq N \leq 8$.
- Output penalty matrix, Q; bounds: $1 \leq Q \leq 100$.
- Input penalty matrix, R; bounds: $0.01 \leq R \leq 1$.
- Number states, x; bounds: $3 \leq R \leq 6$.

The above parameters are tightly coupled with each other and shows an impact on the generation of number of regions in PWA controller, $\kappa(\tilde{x}_0)$. In the following, we show the effects of sampling time and penalty matrices with varying prediction horizon on the number of regions.

A. Case I: Effect of Sampling Time

First, the effect of changing sampling is presented for 0.1, 0.01 and 0.001 s, whereas the Q and R values for this result are kept as 1 and 1 respectively. Fig 2 shows the evaluation of number of regions for different prediction horizons for above settings. It can be seen that $T_s = 0.01$ s generates more number of regions in both controller.

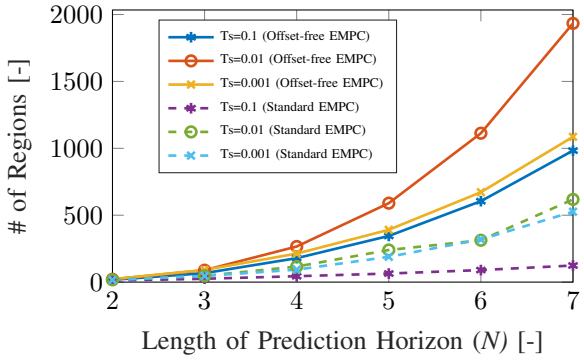

Fig. 2. Effect of sampling time on the number of regions for standard and offset-free EMPC.

B. Case II: Effect of Output Penalty

In this case, we show the effect of output penalty matrix with values of 1, 10 and 100 and sampling time was 0.1 s and R value was fixed at 1. Fig 3 shows the evaluation of number of regions for different prediction horizons. It can be seen that $Q = 10$ in standard EMPC and $Q = 100$ in offset-free EMPC generates more number of regions.

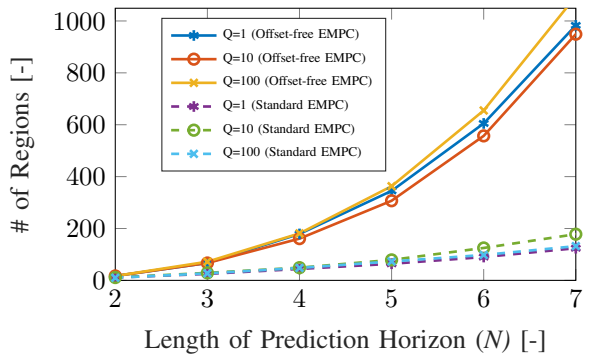

Fig. 3. Effect of output penalty on the number of regions for standard and offset-free EMPC.

C. Case III: Effect of Input Penalty

Finally, for the effect of input penalty on the number of regions is observed for the values 0.1, 0.01 and 0.001, and Q is set at 1 and the sampling time is again 0.1 s. All the effects are observed for EMPC generated for position control i.e., for 3 state model and then for 6 state model which is offset-free EMPC. Fig 4 shows the evaluation of number of regions for different prediction horizons. It can be seen that $R = 0.1$ in standard EMPC (for horizon up to 6) and $R = 1$ in offset-free EMPC (for horizon above 5) generates more number of regions.

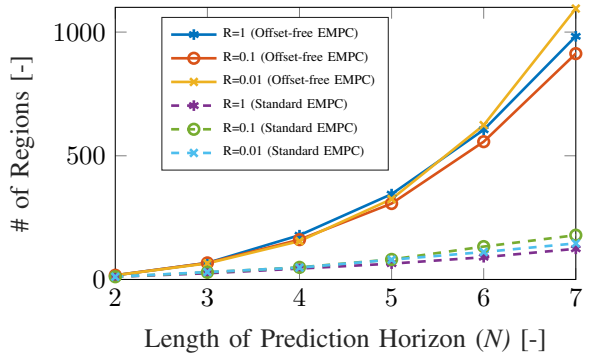

Fig. 4. Effect of input penalty on the number of regions for standard and offset-free EMPC.

VI. Explicit MPC Analysis: FPGA Results

After performing detailed analysis of controller complexity in MATLAB, we selected sampling time of 0.1 s, length of prediction horizon as 5, output penalty as 1, and input penalty as 0.01 for the FPGA implementation of offset-free EMPC and compared the results with standard EMPC and PI controller. In the following, we present the memory demand for standard EMCP with 3 states, and offset-free EMPC with 6 states with varying length of prediction horizon. The Table II summarize the total memory utilization for prediction horizon of 2 to 5. It can be seen that fewer the states or prediction horizon length, lesser is the memory footprint of the controller in the FPGA memory. Also, it shows how the power consumption of FPGA and the achieved clock cycle period varies for these implemented controllers.

TABLE II
Comparison of FPGA-based standard and offset-free EMPC for position control.

EMPC Controller	# of States	# of N	# of Regions	# of BRAM	Memory (kB)	Clock (ns)	Power (W)
Standard	3	2	13	2	36	0.597	1.772
Standard	3	3	45	17	62	0.684	1.761
Standard	3	4	91	34	100	0.668	1.775
Standard	3	5	173	66	172	0.356	1.787
Offset-free	6	2	17	17	62	0.532	1.760
Offset-free	6	3	72	66	172	0.454	1.805
Offset-free	6	4	181	130	316	0.658	1.857
Offset-free	6	5	325	258	603	0.612	1.967

VII. FPGA HIL Co-Simulation Results

In this section, the results obtained by performing HIL co-simulation of standard EMPC, offset-free EMPC, and PI controller are presented. The comparison is carried out on the basis of controller performance parameters and then with respect to their memory and resource utilization.

A. Controller Performance Comparison

The performance of FPGA-based offset-free EMPC is compared with the standard EMPC and PI controller. The objective of all controllers is to track desired reference of angular position. Both explicit controllers are designed with input constraints of $-18 \leq u \leq 18$, where as PI controller is designed with saturation limits on obtained control action. HIL co-simulation is carried out for 30 s with varying reference. Also, to test the performance of controller for disturbance rejection, we applied a disturbance in the output for the time 25 to 28 s. Fig. 5 shows the output response of all three controllers and corresponding control inputs. It can be seen that the performance of standard and offset-free explicit MPC is same when there is no disturbances. PI controller shows oscillations and takes more time to settle to reference. But, at the same time PI controller took less control input. Now, when we applied disturbance from time 25 to 28 s offset-free explicit MPC performance better and tries to reduce the effect of disturbances through control action. However, standard EMPC shows no effect of disturbance on the control action. PI controller gives worst performance for disturbance handling.

B. Controller Complexity and FPGA Analysis

Here we show the resource utilization of different controllers is shown. The main bottleneck in the implementation

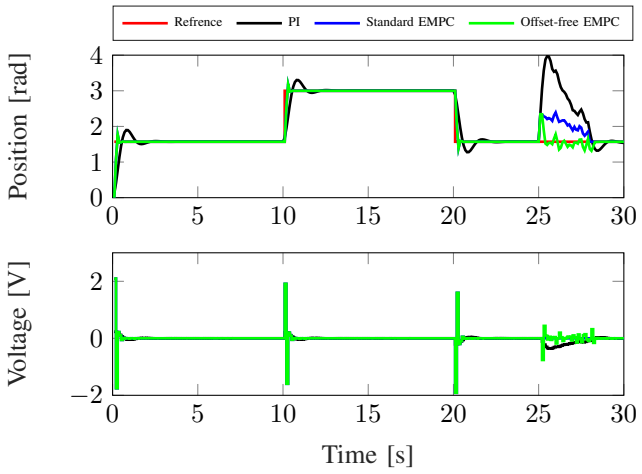

Fig. 5. Comparison of PI, standard EMPC and offset-free EMPC HIL co-simulation for position control of PMDC motor.

of offset-free EMPC is its on-chip memory demand. To store 325 regions (for $N = 5$) on a FPGA with single precision floating-point, i.e., `float` in C/C++ we used one of the features of FPGA that is memory `ALLOCATION` pragma [17] with horizontally store data in the BRAMs. Table III summarized the resource utilization of all three controllers designed for position control of DC motor. It can be seen that the memory demand of PI controller is least among all and it remain same for the control of any system. However, it is sub-optimal controller and lags in the overall performance. On the other hand standard EMPC took 82 kB of memory which means that we can still implement large prediction horizon controller as it has scope to store more data. But, the downside of this controller is that it will not be able to handle disturbance and will need sensors to measure all the states. The offset-free MPC took highest memory i.e., 603 kB. However, it is able to handle disturbances arising due to model mismatch or sensor faults.

TABLE III
FPGA RESOURCE UTILIZATION OF STANDARD EMPC, OFFSET-FREE EMPC AND PI CONTROLLER.

Controller	BRAM	DSP	FF	LUT	Total Memory (kB)
PI	2	24	4166	3543	33
Standard EMPC	21	14	3722	4388	82
Offset-free EMPC	258	14	3453	3157	603

VIII. CONCLUSIONS

This paper focuses on the FPGA implementation of disturbance modeling-based offset-free explicit MPC for the position control of DC motor. Further, we presented the detailed analysis of memory footprints required to implement standard and offset-free explicit MPC with different settings, which helped us in the selection of FPGA device. After that, we compared the HIL co-simulation results of three controllers i.e., PI controller, standard and offset-free explicit MPC with respect to settling time, overshoot and disturbance rejection ability. Moreover, the analysis of the resource utilization, clock time, and power consumption of each controller is presented. From the results, it is clearly seen that the offset-free explicit MPC gives the best performance among the compared controllers. However, the memory demand of offset-free explicit MPC is more and that is due to a the increased number of states. Memory reduction techniques could bring down the memory footprints, but not as-low-as PI controller, which is still a better choice for memory constrained embedded platforms.

ACKNOWLEDGMENT

We gratefully acknowledge the support from R & D center of the COEP. Deepak Ingole would like to thank for a financial contribution from the ERC under the European Unions Horizon 2020 research and innovation program (grant agreement no. 646592 MAGnUM project).

REFERENCES

[1] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.
[2] T. A. Johansen, "Toward dependable embedded model predictive control," *IEEE Systems Journal. DOI*, vol. 10, 2015.
[3] H. J. Ferreau, S. Almér, R. Verschueren, M. Diehl, D. Frick, A. Domahidi, J. L. Jerez, G. Stathopoulos, and C. Jones, "Embedded optimization methods for industrial automatic control," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13 194–13 209, 2017.
[4] G. Takács, P. Zometa, and R. Findeisen, "Embedded model predictive vibration control on low-end 8-bit microcontrollers via automatic code generation," in *23rd International Congress on Sound & Vibration*. IIAV, 2016, pp. 1–8.
[5] A. Gersnoviez, M. Brox, and I. Baturone, "High-speed and low-cost implementation of explicit model predictive controllers," *IEEE Transactions on Control Systems Technology*, 2017.
[6] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.
[7] D. Ingole, "Embedded implementation of explicit model predictive control," Ph.D. dissertation, IAM FCHPT STU in Bratislava, 2017.
[8] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
[9] R. Krishnan, *Electric motor drives: modeling, analysis, and control*. Prentice Hall, 2001.
[10] V. Sankardoss and P. Geethanjali, "Parameter estimation and speed control of a PMDC motor used in wheelchair," *Energy Procedia*, vol. 117, pp. 345–352, 2017.
[11] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *Proceedings of the European control conference*, no. EPFL-CONF-186265, 2013.
[12] D. Ingole, J. Holaza, B. Takács, and M. Kvasnica, "FPGA-based explicit model predictive control for closed-loop control of intravenous anesthesia," in *Proceedings of the 20th International Conference on Process Control*. IEEE, 2015, pp. 42–47.
[13] R. Oberdieck, N. A. Diangelakis, M. M. Papathanasiou, I. Nascu, and E. N. Pistikopoulos, "Pop–parametric optimization toolbox," *Industrial & Engineering Chemistry Research*, vol. 55, no. 33, pp. 8979–8991, 2016.
[14] A. Bemporad, "Hybrid toolbox–users guide," 2003.
[15] B. Khusainov, E. C. Kerrigan, A. Suardi, and G. A. Constantinides, "Nonlinear predictive control on a heterogeneous computing platform," in *IFAC World Congress 2017*, July 2017.
[16] *Zynq-7000 All Programmable SoC Data Sheet: Overview*, Zynq, Xilinx, June 2017.
[17] X. Inc., *SDx Pragma Reference Guide*, UG1253 (v2017.1) ed., June 2017.