**Preprints of the 21st IFAC World Congress (Virtual)**
**Berlin, Germany, July 12-17, 2020**

# FPGA Implementation Framework for Low Latency Nonlinear Model Predictive Control

**Vaishali Patne** [*] **Deepak Ingole** [**] **Dayaram Sonawane** [*]

[*] *College of Engineering Pune, Shivajinagar Pune 411005, India (e-mail: {pva18.instru, dns.instru}@coep.ac.in).*
[**] *Gustave Eiffel University, University of Lyon, ENTPE, LICIT, F-69518 Lyon, France (e-mail: deepak.ingole@ifsttar.fr).*

**Abstract:** Embedded implementation of real-time Nonlinear Model Predictive Control (NMPC) is extremely challenging and complex. This paper presents a framework for implementation of NMPC on Field Programmable Gate Array (FPGA). We show the step-by-step procedure of FPGA implementation framework design of NMPC for a case study of 2D-crane system. In the implementation, we used *GRAMPC* software to construct NMPC and subsequently generate an FPGA specific low-level C/C++ code of the optimization solver. Generated C/C++ code is optimized for memory, speed, and resource utilization by the customized approach of applying pipelining and directives using Xilinx Vivado HLS toolchain. The NMPC is implemented on a Xilinx's ZYNQ-7000 SoC ZC706 FPGA board. The detailed analysis of the controller computational complexity in terms of memory, resource utilization, clock, and power consumption is presented. The performance of implemented NMPC is verified through Hardware-in-the-Loop (HIL) co-simulation using system generator tool. The presented results show the feasibility of FPGA-based GRAMPC framework for ultra-fast applications of NMPC.

*Keywords:* Nonlinear MPC, gradient method, FPGA, real-time control, HIL co-simulation.

## 1. INTRODUCTION

Nonlinear Model Predictive Control (NMPC) is getting a lot of attention from industries and researchers as it is an extremely effective control strategy for Multi-Input Multi-Output (MIMO) systems. It has been used effectively for nonlinear industrial processes like hovercraft (Adhau et al., 2019), aerial robot control (Kocer et al., 2019), steering wheel of autonomous vehicle (Quirynen et al., 2018), missile guidance (Bhattacharjee et al., 2020), and many more.

To increase the applicability of NMPC to real-time applications with resource limited hardware, there is a need for fast, lightweight, and hardware-specific algorithms to solve the nonlinear Optimal Control Problem (OCP) in sub-milliseconds time. Authors in Vukov et al. (2013), implemented NMPC for nonlinear system with 9 states on 3 GHz CPU achieving sampling time of 4 ms while authors in Liniger et al. (2017) obtained sampling time of 20 ms on 1.7 GHz system. Authors in Lekić et al. (2020) were able to achieve 500 $\mu$s on ARM Cortex-M7 (STM32F746-216 MHz) hardware for DC-DC converters.

Many software environments are available to optimize the nonlinear systems. The toolkits like *ACADO* (Houska et al., 2013), *VIATOC* (Kalmari et al., 2015) provides a feature of auto-code generation with low-level C code. The hardware implementation of the code is however restricted to OS-based embedded platforms, due to dependency on library files (Adhau et al., 2019). Various algorithms employed in NMPC are presented in Gros et al. (2016), while various methods for embedded optimization are given in Ferreau et al. (2017). The challenge is to choose the most suitable platform satisfying the requirements such as performance, available on-chip resources (memory, I/Os, communications, speed, *etc.*) power consumption, reconfigurability, portability, *etc.*

One of the attractive choices is to use FPGAs for the implementation of optimization algorithms. FPGA chips have very specific technical characteristics that enable them to execute complex algorithms faster than traditional solutions. In addition to the possibility of architecture reconfiguration, the hardware's parallel architecture and deterministic nature make it an ultimate solution for reducing round-trip latency and increasing the speed of the optimization algorithms.

Work in Zanelli et al. (2018) applied NMPC for the control of a quadrotor using low-power Xilinx ZYNQ SoC (System-on-Chip) with a sampling time of 10 ms using SQP (Sequential Quadratic Programming) with Real-Time Iteration (RTI) method. Authors in Guo et al. (2019) applied Particle Swarm Optimization (PSO) for intelligent vehicle, achieving 5 ms sampling time on FPGA prototype board. The survey of parallel implementations of NMPC is presented in Abughalieh and Alawneh (2019).

The embedded implementation of NMPC is a challenging and time consuming task. Authors in Kapernick and Graichen (2014), developed a *GRAMPC Software* for NMPC implementation in C/C++ using gradient-based method. Further, authors in Englert et al. (2019) have extended this framework to solve the nonlinear system with the state as well as input constraints. They also considered equality and inequality constraints along with terminal constraints. The main advantage of the framework is that the code is written in plain C/C++. Also, the GRAMPC gives code flexibility to the user, to make the necessary changes from the application perspective. GRAMPC is successfully applied in the control electric vehicle on CPU (Wu et al., 2018). However, the implementation of the GRAMPC framework on embedded hardware if difficult to find. With GRAMPC, one can implement NMPC on embedded platforms like ARM and other microcontrollers. However, the challenge is to implement the same on FPGAs because FPGA configuration needs hardware-specific High-Level Synthesis (HLS) C/C++ codes and does not support built-in functions of C/C++. This paper presents the step-by-step procedure of the FPGA implementation framework of NMPC using the 2D-crane system as the case study. We use the GRAMPC to construct nonlinear MPC in C/C++ environment. Subsequently, FPGA specific C/C++ HLS code of the GRAMPC is generated through our developed C++ wrapper. Further, the generated HLS code is optimized for memory, speed, and resource utilization. The FPGA-based GRAMPC algorithm is then implemented on a Xilinx's ZC706 − 7000 series FPGA via *bitstream* and tested via HIL co-simulations. Using this framework, the NMPC algorithm development time will be reduced significantly and it will be used for nonlinear control problems arise in embedded applications.

The main features of the developed framework include its use for the FPGA-based NMPC, optimal control, and state/parameter estimation using Nonlinear Moving Horizon Estimation (NMHE), supports the flexible splitting of the algorithmic workload and memory storage between software and hardware for trading-off the computational resource usage against performance, and also, can be easily implemented on different FPGA boards like Xilinx's ZYNQ7000 SoC ZC702, ZedBoard, Microzed, *etc.*

This paper is organized as follows. The structure of the GRAMPC framework is explained in Section 2. Section 3 discusses the hardware (FPGA) and the steps used for the real-time implementation of NMPC. Section 4 presents a case study of two-dimensional crane system and the HIL results are discussed in Section 5. The concluding remarks are drawn in Section 6.

## 2. GRAMPC FRAMEWORK

In this section, structure of the original GRAMPC software is described along with the formulation and optimization algorithm.

### *2.1 Nonlinear Model Predictive Control*

NMPC is an advanced control technique. Its most important advantage is its ability to handle constraints systematically. At each sampling instant, using the current state of the system as an initial state, the finite horizon OCP is solved to obtain a sequence of optimal controls. The first control action is applied to the system and the whole procedure is repeated in each sampling instant (Borrelli et al., 2017, Chapter 12). This is computationally expensive, restricting the use of NMPC to processes with relatively slow dynamics.

For real-time applications, the NMPC demands a fast optimization algorithm to calculate optimal control actions within one sample time. Also, the code needs to be hardware compatible and should be able to fit on resources available on selected hardware.

To address this issue, Kapernick and Graichen (2014) proposed the GRAMPC framework using first-order gradient method with augmented Lagrangian technique which has less computational load as compared to second-order methods. Thus it is possible to achieve the sampling rate of milliseconds or even microseconds (Englert et al., 2019).

### *2.2 NMPC Framework*

GRAMPC (v2.1) is a software for solving continuous nonlinear systems, subject to linear or nonlinear constraints on states and inputs. The general structure of the GRAMPC is as given in Fig. 1.

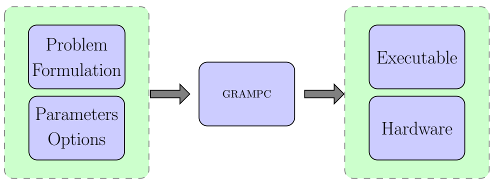

Fig. 1. General structure of GRAMPC.

The model of the plant is defined in the C function template file along with the dynamics of the system, constraints, and the cost function to be minimized. Initialization of the states, variables, and parameters is in a separate file. The OCP for the plant is solved repeatedly over the prediction horizon ($T$). The MPC is run in a close loop over the simulation time or for a single iteration. The executable file of the C code can be used to run the model on a hardware platform using MATLAB/Simulink.

### *2.3 NMPC Problem Formulation*

The problem formulation used in the GRAMPC framework is given as follows:

$$\begin{aligned}
&\min_{u,p,T} J(u,p,T,x_0) = V(x(T),p,T) \ + && \text{(1a)}\\
&\qquad \int_0^T l(x(t),u(t),p(t))\ dt, &&\\
&\text{s.t. } \ x(0) = x_0, && \text{(1b)}\\
&M\dot{x}(t) = f(x(t),u(t),p,t_0+t), && \text{(1c)}\\
&g(x(t),u(t),p,t) = 0, && \text{(1d)}\\
&g_T(x(T),p,T) = 0, && \text{(1e)}\\
&h(x(t),u(t),p,t) \leq 0, && \text{(1f)}\\
&h_T(x(T),p,T) \leq 0, && \text{(1g)}\\
&u(t) \in [u_{\min}, u_{\max}], && \text{(1h)}\\
&p \in [p_{\min}, p_{\max}], && \text{(1i)}\\
&T \in [T_{\min}, T_{\max}]. && \text{(1j)}
\end{aligned}$$

The cost function (1a) is minimized with reference to $u, p$, and $T$ as optimization variables. The system dynamics (1c) consider constant mass matrix $M$. The state vector at time $t$ is denoted by $x \in \mathbb{R}^{n_x}$, control inputs by $u \in \mathbb{R}^{n_u}$, and parameters by $p \in \mathbb{R}^{n_p}$. The terminal as well as general equality ($g_T$ and $g$) and inequality ($h_T$ and $h$) constraints are also included. The MPC solves OCP in (1) over prediction horizon $T \in \mathbb{R}$ generating new states, used as initial states in the next sampling instant.

### 2.4 Optimization Algorithm

The GRAMPC uses low-level plain C code without dependency on external libraries. It uses the indirect approach to optimize first and then discretize. The Euler method is used to solve the differential equations. The augmented Lagrangian method provides a sub-optimal sure solution at fast speed with a less computational cost. Also, it can handle both, equality as well as inequality constraints. For solving the inner minimization, the Projected Gradient Method (PGM) is used which also updates the multipliers along with the penalty parameters. The outer loop is terminated when all constraints are met and the inner loop is converged.

The augmented Lagrangian method is the combination of the penalty method (Nocedal and Wright, 2006) and the Lagrange method (Arora et al., 1991). In the penalty method, the value of the penalty parameter $\mu$ is usually large. This can cause ill-conditioning of the optimization problem. When combined with the Lagrangian method, smaller values of $\mu$ can be used. In the Augmented Lagrangian algorithm, $\mathcal{L}_\mathcal{A}$ is minimized successively, updating the value of $\lambda$ and $\mu$ every iteration.

$$\min \mathcal{L}_\mathcal{A}(x, \lambda, \mu) = \underbrace{f(x) + \lambda^T g(x)}_{\text{Lagrangian}} + \underbrace{\frac{\mu_k}{2}||g(x)||_2^2}_{\text{augmentation}}, \quad (2a)$$

$\lambda$ and $\mu$ is updated by,

$$\lambda_i \leftarrow \lambda_i + \mu_k g(x_k), \quad (2b)$$
$$\mu_k \leftarrow \rho \mu_k, \quad (2c)$$

where $x_k$ is solution of unconstrained problem at $k^{\text{th}}$ step and $\rho$ is penalty increase factor (e.g., factor of 1.5 or 10). This method can be easily extended to handle inequality constraints using slack variables. However, adding inequality constraints increases the computational burden. The next section explains the FPGA wrapper developed for implementation for the algorithm.

## 3. FPGA IMPLEMENTATION FRAMEWORK

This section discusses our main contribution, the FPGA framework to implement GRAMPC-based NMPC. Fig. 2 shows the detailed process to implement the controller on the FPGA.

### 3.1 Steps for Implementation of FPGA-based GRAMPC

*Code generation:* The available code can be compiled for microcontrollers with minor changes. However, for FPGA, standard C functions are not directly synthesizable. In this work, we developed C wrapper to transform the original GRAMPC C code to the synthesizable HLS code for the Xilinx FPGA board.

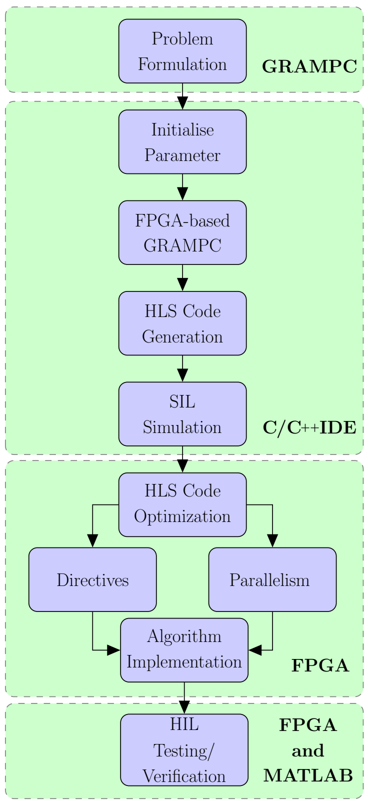

Fig. 2. Steps in design and implementation framework of NMPC on FPGA.

The developed wrapper statically defines the initialization parameters to avoid dynamic memory allocation. Library functions of C are replaced with inline functions while making necessary changes in some of the other functions. The wrapper eliminates all generalized functions by transforming the code into problem specific version, *e.g.*, having a choice of integrator at run time does not work with FPGA the way it works with microcontrollers. Thus the code executes with pre-defined options.

*SIL simulation:* The developed C code for FPGA-based GRAMPC is then executed in closed-loop. It is observed that the controller tracks the states successfully while following the constraints. The Software-in-the-loop (SIL) simulation is performed for various initial conditions using C code. The generated values of states and controls are stored in a text file for each iteration. These are then plotted for verification of results. We also analyze the complexity of the controller for different values of the horizon as well as the penalty matrices at this stage.

*Synthesis:* The code is synthesized using Vivado HLS (2018.3) when the code syntax is checked and the hierarchy of the design is analyzed. At this stage, the design is transformed into register transfer logic (RTL) and then to gate-level representation. It is verified that the changes made in the code from the FPGA perspective do not affect its performance.

*Code optimization:* Once the code is synthesized, we get the utilization of the FPGA resources (Block RAMs (BRAMs), DSP blocks, Flip-flops (FFs), and Look-up-Tables (LUTs)). Based on this information, we select the

FPGA board to be used. Vivado HLS provides various *attributes, pragmas*, and *directives* for improved performance, reduce latency or the resources used while achieving better data throughput *e.g.*,

- Function inline: *INLINE* helps in simplification of architectural hierarchy of the design.
- Loop manipulation: *LOOP_FLATTEN* and *LOOP _MERGE* are used for better optimization of the loop-body. *UNROLL* pragma is used to partially unrolling the loops based on timing as well as memory utilized.
- Coding style: use of proper HDL coding style helps in effective hardware design. Also, specifying necessary bit width can result in smaller and faster hardware.

Pipelining is also applied to loops (*PIPELINE*) wherever possible along with *ARRAY_MAP*, *ALLOCATION*, *etc.*, for the further optimization of the code. We get other performance evaluation parameters like achieved clock and power utilization at this stage. The code is now ready for the target board.

*Deploy on the FPGA:* We have used *System Generator* with MATLAB/Simulink to test and verify the developed FPGA implementation framework.

- Simulink design: various building blocks are provided in the system generator for Digital Signal Processing (DSP), *e.g.*, *black box* allows the import of the RTL generated in Vivado HLS into Simulink. This helps in designing directly in a flexible high-level system environment.
- Export RTL: after optimization and synthesis, the code is exported to MATLAB/Simulink for HIL co-simulation by using the Export RTL feature of the Vivado HLS.
- HIL co-simulation: the system generator (2018.3) has an option to choose the appropriate hardware. The Simulink design is then tested using simulation. The design is then compiled using HIL with JTAG interface. Once the *bitstream file* is created, the generated subsystem is added to the design. Use of subsystem executes the plant in the MATLAB/Simulink while the solver is on the FPGA hardware.

### *3.2 FPGA Board Used*

FPGAs are well known for high processing power and reconfigurable designs. Depending upon the problem size and required resources (obtained after synthesis step), we can select the appropriate FPGA board. In this work, we consider Xilinx's *ZC*706 evaluation board with the ZYNQ7000 XC7Z045 SoC processor. The SoC contains ARM as well as FPGA on board.

Table 1. Resources available on Xilinx's ZC706 FPGA.

| Resource | Available | Size (bits) | Memory (Mb) |
|---|---|---|---|
| BRAM | 1090 | 18000 | 20.09 |
| DSP | 900 | 48 | 0.0432 |
| FF | 437200 | 1 | 0.4372 |
| LUT | 218600 | 64 | 13.99 |

The FPGA is reconfigurable with the capability of parallel processing. This makes the platform ideal for the embedded implementation of MPC. If the required resources are less, cheaper options like Zedboard or MicroZed can also be used. Various resources available on the Xilinx's ZC706 and their total memory occupation are given in Table 1. The next section gives the details of 2D-crane as a case study for verification of the FPGA wrapper developed.

Table 2. States and inputs of 2D-crane system.

| | Variable | Parameter | Description | Unit |
|---|---|---|---|---|
| **States** | $x_1$ | r | Cart position | m |
| | $x_2$ | $\dot{r}$ | Cart velocity | m/s |
| | $x_3$ | $l$ | Rope length | m |
| | $x_4$ | $\dot{l}$ | Rope velocity | m/s$^2$ |
| | $x_5$ | $\theta$ | Angle with vertical | rad |
| | $x_6$ | $\dot{\theta}$ | Angular velocity | rad/s |
| **Inputs** | $u_1$ | $a_c$ | Acceleration of cart | m/s$^2$ |
| | $u_2$ | $a_r$ | Acceleration of rope | m/s$^2$ |

## 4. CASE STUDY: 2D-CRANE

The algorithm explained in Section 2 is verified using a case study of 2D-crane system. Fig. 3 shows the schematic

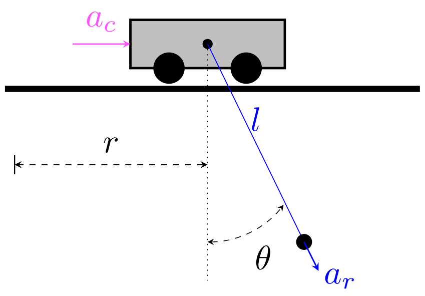

Fig. 3. Free body diagram of the 2D-crane.

of the 2D-crane system. The crane comprised of a cart, moving on the beam with a load attached to it using a rope of length $l$. We follow the model and notation presented in Kapernick and Graichen (2013). The states and inputs of the system are given in Table 2. The nonlinear model dynamics is given by,

$$\ddot{\theta} = \frac{-1}{l}(g\ \sin(\theta) + a_c\ \cos(\theta) + 2\ \dot{l}\ \dot{\theta}), \tag{3a}$$

where $g$ is the gravitational acceleration. The controls are bounded between $-2$ to $+2$ with sampling time of 2 ms. The controller aims to drive the crane from initial to final value without swinging of the pendulum, keeping the value of $\theta$ close to zero. The initial $(x_0, u_0)$ and the final values $(x_f, u_f)$ of the states and inputs are given by,

$$x_0 = [-2, 0, 2, 0, 0, 0]^\top, \tag{4a}$$
$$x_f = [2, 0, 2, 0, 0, 0]^\top, \tag{4b}$$
$$u_0 = [0, 0]^\top, \tag{4c}$$
$$u_f = [0, 0]^\top. \tag{4d}$$

For this case study, only integral cost or the Lagrange term is used, which is given by,

$$l(\tilde{x}, u) = \frac{1}{2}(\Delta x^T Q \Delta x + \Delta u^T R \Delta u), \tag{5a}$$
$$\Delta x = x - x_f, \tag{5b}$$
$$\Delta u = u - u_f, \tag{5c}$$

where $x_f$ and $u_f$ are the desired final values of states and control inputs, respectively. The $Q$ and $R$ are positive definite matrices given by,

$$Q = \mathrm{diag}(1,\ 2,\ 2,\ 1,\ 1,\ 4), \tag{6a}$$
$$R = \mathrm{diag}(0.05,\ 0.05). \tag{6b}$$

The nonlinear inequality constraint is given by,

$$l\cos(\theta) - 0.2(r + l\sin(\theta))^2 - 1.25 \leq 0. \tag{7}$$

Once the model, parameters, and constraints are defined, the steps given in Section 3.1 are followed to generate the FPGA ready code. The FPGA implementation and its closed-loop performance are discussed in the next section.

## 5. HIL CO-SIMULATION RESULTS

The performance evaluation of the FPGA-based GRAMPC is discussed in this section. The C HLS code is optimized for the 2D-crane model. It is then implemented on the Xilinx's ZC706 FPGA board using system generator. The NMPC HLS code is executed with sampling time $(Ts) = 2$ ms and horizon $(T) = 2$ s. The values of the penalty matrices, $Q$ and $R$, are given by (6) whereas the initial and final condition of all states is given in (4). The system is simulated over 12.5 s.

### 5.1 *Closed-loop Performance*

The main objective of the control of 2D-crane is to achieve the reference value while following the nonlinear constraints as well as bounds on the controls. Fig. 4 shows

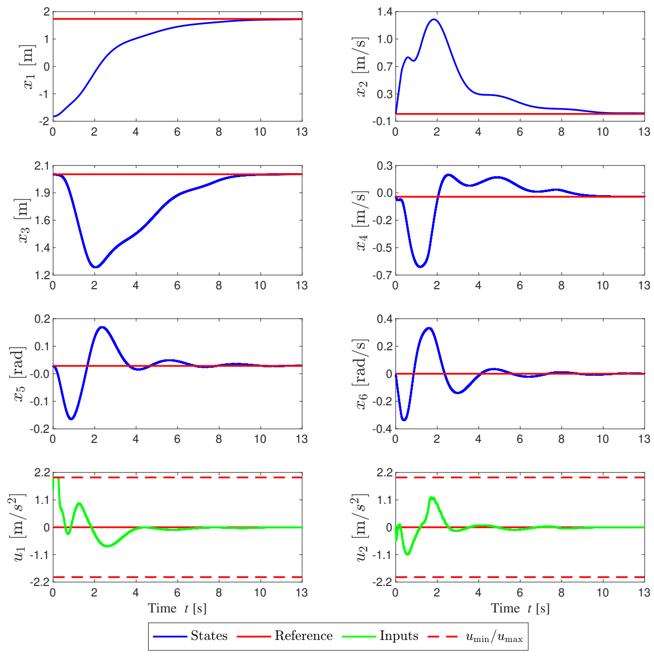

Fig. 4. Response of FPGA-based GRAMPC with HIL co-simulation for closed-loop control of 2D-crane.

the closed-loop response of the 2D-crane system to the FPGA-based NMPC. It can be seen that the FPGA-based NMPC performs as expected to track the desired reference values of all the states within the imposed constraints and bounds. The new cart position is reached without much variation in the pendulum angle thus avoiding the swinging of the pendulum.

### 5.2 *FPGA Resource Utilization*

One of the main challenges in FPGA implementation of NMPC is to keep resources within available limits. Even though the design fits on the Xilinx's ZC706 board, further reduction in the resources is attempted to make NMPC code lightweight. To reduce memory footprints, we used 32 bit single-precision (`float`) number format in the algorithm. Further, we use HLS directives like function inlining, array partitioning, *etc.* The comparison of the original and optimized resource utilization is shown in the Table 3. It can be seen that with the proposed code optimization we can reduce resources by a significant amount, which is important for large problems.

Table 3. Resource utilization of FPGA-based NMPC with and without HLS code optimization.

| **FPGA-based GRAMPC** | **LUTs** | **FFs** | **DSPs** | **BRAM** | **Total (Mb)** |
|---|---|---|---|---|---|
| Original | 141734 | 99921 | 884 | 108 | 9.39 |
| Optimized | 100929 | 67794 | 670 | 75 | 6.69 |
| Saving (%) | 28.79% | 32.15% | 24.21% | 30.55% | 28.75% |

Other key parameters of FPGA design are the clocks achieved and the power consumption. Table 4 shows these parameters for the proposed FPGA-based NMPC with original and optimized code. It can be seen that the achieved clock is sufficient for systems with sampling time in the range of nanosecond. Though there is less difference in clock achieved between before and after optimization, there is reduction in power utilization after the code optimization. This shows that we can effectively use proposed NMPC framework for fast control applications.

### 5.3 *Low Latency*

The time interval between the input and response is referred to as latency. Using parallel architecture of FPGA it is possible to process independent instructions at the same time resulting in ultra-fast latency. Occurrences of random events in FPGA are very limited delivering predictable and repeatable processing latency. As the zero latency is achieved, the framework is suitable for real-time applications requiring very fast sampling times.

## 6. CONCLUSION

In this paper, we present the FPGA implementation framework for low latency NMPC. The detailed step-by-step procedure of FPGA-based NMPC (GRAMPC) design and implementation is presented considering the case study of the 2D-crane system. In the proposed framework, we developed a wrapper on the top of the original

Table 4. Clock and power analysis of FPGA-based GRAMPC for the closed-loop control of 2D-crane.

| **Resource** | **Original** | **Optimized** |
|---|---|---|
| Targeted clock (ns) | 10 | 10 |
| Achieved clock (ns) | 0.201 | 0.207 |
| Total power (W) | 3.178 | 3.167 |

GRAMPC algorithm to generate FPGA specific HLS code of the NMPC. The generated HLS code is further optimized for resource utilization, speed, and memory, using the pragmas and directives provided by Xilinx Vivado HLS toolchain. We analyzed the performance of the FPGA-based NMPC using the Xilinx's ZC706 − 7000 series FPGA development board with the HIL co-simulation.

The analysis shows that the FPGA-based NMPC can be used for real-time applications owing to its fast speed and little development time. This will open new avenues for the use of FPGA for the control of nonlinear systems by customizing the available GRAMPC framework for small and medium systems. The developed framework is an excellent choice for low-latency, ultra-fast real-time applications such as robotics, flight control, and nonlinear systems involving complex calculations.

The future scope of this work is to propose modifications in the source code of the GRAMPC framework to make it directly implementable on the FPGA hardware. Also, the possibility of using the heterogeneous configuration, for the larger systems demanding higher memory footprints.

## ACKNOWLEDGEMENTS

We gratefully acknowledge the contribution of the Department of Science and Technology, Govt. of India, under Women Scientist Scheme (WOS-A/ET-120/2018). Funding in the form of the APJ Abdul Kalam Memorial International Travel Award, from the Automatic Control and Dynamic Optimization Society (ACDOS), is gratefully acknowledged. Deepak Ingole would like to thank for a financial contribution from the ERC under the EU Horizon 2020 research and innovation program (grant agreement no. 646592 MAGnUM project). We would like to thank the GRAMPC development team for their helpful discussions and suggestions. The goal of this work is to develop an FPGA-based implementation framework for NMPC using original GRAMPC and whole credit/copyright of the methods and algorithms is of the GRAMPC team.

## REFERENCES

Abughalieh, K.M. and Alawneh, S.G. (2019). A survey of parallel implementations for model predictive control. *IEEE Access*, 7, 34348–34360.

Adhau, S., Patil, S., Ingole, D., and Sonawane, D. (2019). Implementation and Analysis of Nonlinear Model Predictive Controller on Embedded Systems for Real-Time Applications. In *ECC*, 3359–3364.

Arora, J., Chahande, A., and Paeng, J. (1991). Multiplier Methods for Engineering Optimization. *International Journal for Numerical Methods in Engineering*, 32(7), 1485–1525.

Bhattacharjee, D., Chakravarthy, A., and Subbarao, K. (2020). Nonlinear model predictive control based missile guidance for target interception. *AIAA Scitech 2020 Forum*, 0865–0884.

Borrelli, F., Bemporad, A., and Morari, M. (2017). *Predictive Control for Linear and Hybrid Systems:Textbook.* Cambridge University Press.

Englert, T., Völz, A., Mesmer, F., Rhein, S., and Graichen, K. (2019). A Software Framework for Embedded Nonlinear Model Predictive Control Using a Gradient-Based Augmented Lagrangian Approach (GRAMPC). *Optimization and Engineering*, 769–809.

Ferreau, H.J., Almér, S., Verschueren, R., Diehl, M., Frick, D., Domahidi, A., Jerez, J.L., Stathopoulos, G., and Jones, C. (2017). Embedded Optimization Methods for Industrial Automatic Control. *IFAC-PapersOnLine*, 50(1), 13194–13209.

Gros, S., Zanon, M., Quirynen, R., Bemporad, A., and Diehl, M. (2016). From Linear to Nonlinear MPC: Bridging the Gap Via the Real-Time Iteration. *International Journal of Control*, 1–19.

Guo, H., Liu, F., Xu, F., Chen, H., Cao, D., and Ji, Y. (2019). Nonlinear Model Predictive Lateral Stability Control of Active Chassis for Intelligent Vehicles and Its FPGA Implementation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2–13.

Houska, B., Ferreau, H., Vukov, M., and Quirynen, R. (2013). ACADO Toolkit User's Manual. http://www.acadotoolkit.org.

Kalmari, J., Backman, J., and Visala, A. (2015). A Toolkit for Nonlinear Model Predictive Control Using Gradient Projection and Code Generation. *Control Engineering Practice*, 39, 56–66.

Kapernick, B. and Graichen, K. (2013). Model Predictive Control of an Overhead Crane Using Constraint Substitution. In *American Control Conference*, 3973–3978.

Kapernick, B. and Graichen, K. (2014). The Gradient Based Nonlinear Model Predictive Control Software GRAMPC. *European Control Conference*, 1170–1175.

Kocer, B.B., Tiryaki, M.E., Pratama, M., Tjahjowidodo, T., and Seet, G.G.L. (2019). Aerial Robot Control in Close Proximity to Ceiling: A Force Estimation-based Nonlinear MPC. *IEEE International Conference on Intelligent Robots and Systems*, 2813–2819.

Lekić, A., Hermans, B., Jovičić, N., and Patrinos, P. (2020). Microsecond Nonlinear Model Predictive Control for DC-DC Converters. *International Journal of Circuit Theory and Applications*, 48(3), 406–419.

Liniger, A., Domahidi, A., and Morari, M. (2017). Optimization-Based Autonomous Racing of 1:43 Scale RC Cars. *arXiv e-prints*, 628–647.

Nocedal, J. and Wright, S. (2006). *Penalty and Augmented Lagrangian Methods*, 497–528. Springer New York.

Quirynen, R., Berntorp, K., and Di Cairano, S. (2018). Embedded Optimization Algorithms for Steering in Autonomous Vehicles Based On Nonlinear Model Predictive Control. In *American Control Conference (ACC)*, 3251–3256. IEEE.

Vukov, M., Domahidi, A., Ferreau, H.J., Morari, M., and Diehl, M. (2013). Auto-Generated Algorithms for Nonlinear Model Predictive Control on Long and on Short Horizons. In *52nd IEEE Conference on Decision and Control*, 5113–5118.

Wu, Li, Du, Ding, Li, Yang, and Lu (2018). Fast Velocity Trajectory Planning and Control Algorithm of Intelligent 4WD Electric Vehicle for Energy Saving Using Time-Based MPC. *Recent Advancements on Electrified, Low Emission and Intelligent Vehicle-Systems, IET*, 153–159.

Zanelli, A., Horn, G., Frison, G., and Diehl, M. (2018). Nonlinear Model Predictive Control of A Human-Sized Quadrotor. In *European Control Conference (ECC)*, 1542–1547. IEEE.