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

An Embedded FPGA Architecture for Real-Time Model Predictive Control

Hussain I. Khajanchi * **Joseph N. Bruno** *
Ambrose A. Adegbege *

* *Department of Electrical and Computer Engineering, The College of New Jersey, 2000 Pennington Road, Ewing, NJ 08628 (e-mail: khajanh1@tcnj.edu, brunoj6@tcnj.edu adegbega@tcnj.edu).*

Abstract: We develop a custom hardware architecture for real-time implementation of embedded Model Predictive Control (MPC) on a Field Programmable Gate Array (FPGA). We propose a novel modular framework that allows for easy and rapid prototyping of control with capability for analog-to-digital conversion, numerical scaling, and digital-to-analog conversion. We demonstrate the effectiveness of the proposed framework for real-time control on a quadruple water tank system.

Keywords: Embedded Model Predictive Control, Primal-Dual algorithm, Real-Time Control, Field Programmable Gate Array, Quadruple Tank System.

1. INTRODUCTION

Model Predictive Control (MPC) is a class of advanced control algorithms in which the output control trajectory is optimized for a linear or non-linear model of the plant (Borrelli et al., 2017). MPC has inherent constraint handling capability such that at every time step when a new input is received, the controller solves a constrained optimization problem online. The optimization solver typically calculates a new set of control moves over a control horizon to optimize the predicted system state trajectory, while constraining the output space to account for latent constraints.

As the problem scope increases, the computational overhead needed to solve the optimization problem becomes increasingly intransigent. This can be unrealistic for real-time systems, as controllers designed for fast processes need to deliver control outputs quickly. To leverage the inherent flexibility and robustness of MPC while retaining a fast and efficient computing architecture, many optimization algorithms have been developed with speed in mind (Wang and Boyd, 2009; Richter et al., 2011; Patrinos and Bemporad, 2013; Jerez et al., 2014).

To increase the computational dexterity of MPC, there are many fast and simple solver algorithms tailored for the underlying Quadratic Program (QP). These algorithms include active-set methods (Borrelli et al., 2010), interior point methods (Wang and Boyd, 2009), gradient projection methods (Patrinos and Bemporad, 2013), and operator splitting methods (O'Donoghue et al., 2013). We consider a primal-dual algorithm proposed in (Blanchard and Adegbege, 2017) which solves an associated saddle point problem by updating the primal and the dual variables iteratively in a Gauss-Seidel fashion (Adegbege and Nelson, 2016) until the solver converges. This algorithm lends itself to efficient matrix-vector multiplications and projections which can be optimized for hardware implementation (Levenson et al., 2017; Sabo and Adegbege, 2018).

The contribution of this work is the implementation of the primal-dual MPC algorithm using a custom hardware architecture embedded within a Field Programmable Gate Array (FPGA). Specifically, a novel modular framework is developed that allows for treating signal conversion, signal scaling and the solver circuit as individual blocks which are interconnected via Register Transfer Level (RTL) abstraction of the dataflow. The proposed framework is attractive in that it is scalable and any solver algorithm can be incorporated into the implementation.

The rest of the paper is structured as follows. In section 2, we provide background information on the primal-dual algorithm we employed for solving the MPC underlying quadratic programming problem. In section 3, we discuss the hardware architecture both for implementing the primal-dual algorithm and for acceleration on an FPGA. Finally in section 4, we discuss experimental results arising from the real-time control of a quadruple water tank system using the proposed implementation architecture.

2. PROBLEM SET UP

We consider the problem of linear MPC for a nonlinear system described by the following discrete-time state space equation:

$$x_\xi[k+1] = f(x_\xi[k], u[k]) \tag{1a}$$
$$y_\xi[k] = h(x_\xi[k]) \tag{1b}$$

where $x_\xi \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the control input vector, and $y_\xi \in \mathbb{R}^m$ is the output vector. For linear MPC control design, we obtain a linearized version of (1) using truncated Taylor's series expansion of $f(\cdot)$ and $h(\cdot)$ about a suitable operating point as:

$$x[k+1] = Ax[k] + Bu[k]) \tag{2a}$$
$$y[k] = Cx[k]) \tag{2b}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the control input vector, and $y \in \mathbb{R}^m$ is the output vector. We assume that the pair (A, B) is controllable, the pair (A, C) is observable and that the output matrix C is full row ranked.

To account for the mismatch between the actual system model (1) and its linearized model (2) and to incorporate integral action for offset-free steady state behavior (Maeder et al., 2009), we construct an observer of the form:

$$\begin{aligned}\begin{bmatrix}\hat{x}[k+1]\\ \hat{d}[k+1]\end{bmatrix} = \begin{bmatrix}A & 0\\ 0 & I\end{bmatrix}\begin{bmatrix}\hat{x}[k]\\ \hat{d}[k]\end{bmatrix} + \begin{bmatrix}B\\ 0\end{bmatrix}u[k]\\ + \begin{bmatrix}L_x\\ L_d\end{bmatrix}(C\hat{x} + \hat{d} - y[k])\end{aligned} \tag{3}$$

where $\hat{x}$ and $\hat{d}$ are the estimated state and disturbance respectively, and Lx and L_d are appropriately chosen observer gains to ensure stability and proper operation of the observer. At steady state, the observer must satisfy:

$$\begin{bmatrix}A - I & B\\ C & 0\end{bmatrix}\begin{bmatrix}\hat{x}_\infty\\ u_\infty\end{bmatrix} = \begin{bmatrix}0\\ r_\infty - \hat{d}_\infty\end{bmatrix} \tag{4}$$

where r_∞ is the steady value of the reference input or set-point $r[k]$, $\hat{x}_\infty$ and $\hat{d}_\infty$ are the steady state values of the observed state and disturbance respectively, and u_∞ is the steady state control input. Note that (4) is invertible if the plant has no integrators and C is full row ranked, and so for any r_∞ and $\hat{d}_\infty$, there exist x_∞ and u_∞. We will exploit this parameterization in the MPC design.

2.1 Model Predictive Control Formulation

We formulate the MPC controller as (Maeder et al., 2009):

$$\min_{u_0 \cdots u_{N-1}} \frac{1}{2}(||x_N - \overline{x}_t||_p^2 + \sum_i^{N-1} ||x_i - \overline{x}_t||_Q^2 + ||u_i - \overline{u}_t||_R^2)$$

subject to

$$\begin{aligned}&x_{i+1} = Ax_i + Bu_i,\ d_{i+1} = d_i, i = 0, \cdots, N-1\\ &\underline{x} \le x_i \le \overline{x},\ i = 1, \cdots, N\\ &\underline{u} \le u_i \le \overline{u},\ i = 0, \cdots, N-1\\ &x_0 = \hat{x}[k],\ d_0 = \hat{d}[k], r_0 = r[k]\\ &\begin{bmatrix}A - I & B\\ C & 0\end{bmatrix}\begin{bmatrix}\overline{x}_t\\ \overline{u}_t\end{bmatrix} = \begin{bmatrix}0\\ r_0 - d_0\end{bmatrix}\end{aligned}$$

where $N \in \mathbb{R}$ is the prediction horizon, $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are weighting matrices, and $P \in \mathbb{R}^{n\times n}$ is the solution of an associated discrete Riccati equation. The vectors $\overline{x}_t$ and $\overline{u}_t$ are the steady-state target to which the state and the input are stirred by the MPC. The above MPC can be reformulated as a Quadratic Programming (QP) problem that must be solved at every time instant. By defining an augmented variable $z_i = [u_{i-1}^T\ x_i^T]^T,\ i = 1, \cdots, N$, the QP problem can be expressed as:

$$\min_z \frac{1}{2}z^T H z + z^T q \tag{5a}$$

$$\text{subject to } Ez = e,\ z \in \mathbb{Z} \tag{5b}$$

where $H \in \mathbb{R}^{(m+n)N\times(m+n)N}$, $E \in \mathbb{R}^{nN\times(m+n)N}$ are fixed problem data that depend on system matrices A and B, MPC matrices Q, R and P, and the prediction horizon N. The vectors $q \in \mathbb{R}^{(m+n)N}$ and $e \in \mathbb{R}^{nN}$, typically time-dependent, are input vectors to the QP and $z \in \mathbb{R}^{(m+n)N}$ is the variable to be optimized. The matrices H and E are typically large and sparse, and can be exploited for computational efficiency. The set $\mathbb{Z}$ holds the simple bound constraints of the system, while $Ez = e$ is the equality condition containing the complicating constraints. We refer readers to (Blanchard and Adegbege, 2017) for full description of matrices H and E. In a real-time control configuration, the input vectors q and e are sampled and/or estimated at every sample time to produce a new solution for the QP (5).

2.2 Primal-Dual Solver for Quadratic Programming

The primal-dual solver adopted here follows that of (Blanchard and Adegbege, 2017) which combines the Bound-Constrained Lagrangian (BCL) method (Nocedal and Wright, 2006) with the Successive Over-Relaxation method (SOR) for saddle-point problems (Benzi et al., 2005), to develop an iterative primal descent and dual ascent algorithm for QP problem (5).

The bound-constrained Lagrangian approach involves absorbing the equality constraints into the quasi-Lagrangian

$$L(z, \lambda) = \frac{1}{2}z^T H z + z^T q + \lambda^T(Ez - e), \tag{6}$$

and then enforcing the inequality constraint in the subproblem:

$$\min_z L(z, \lambda), \tag{7a}$$

$$\text{subject to } z \in \mathbb{Z}. \tag{7b}$$

An efficient method for dealing with subproblem (7) is the gradient projection method (Nocedal and Wright, 2006). Considering that the Karush Kuhn Tucker (KKT) optimality condition for z to be a solution of (7) is given by:

$$z - \mathbb{P}\left(z - \nabla_z L(z, \lambda)\right) = 0, \tag{8a}$$

$$\mathbb{P}_i(g) = \begin{cases}\underline{z}_i & if\ g_i \le \underline{z}_i,\\ g_i & if\ \underline{z}_i < g_i < \overline{z}_i,\\ \overline{z}_i & if\ g_i \ge \overline{z}_i.\end{cases} \tag{8b}$$

So, for fixed λ, an iterative procedure for updating z based on the gradient projection method can be constructed as:

$$z^{k+1} = \mathbb{P}\left(z^k - \nabla_z L(z^k, \lambda)\right). \tag{9}$$

To incorporate an update strategy for λ into (9), we employ the SOR strategy for the saddle-point problem:

$$\min_{z\in\mathbb{Z}} \max_\lambda L(z, \lambda) = \frac{1}{2}z^T H z + z^T q + \lambda^T(Ez - e). \tag{10}$$

The augmented or KKT system associated with (10) can be expressed as:

$$\begin{bmatrix}H & E^T\\ -E & 0\end{bmatrix}\begin{bmatrix}z\\ \lambda\end{bmatrix} = \begin{bmatrix}-q\\ -e\end{bmatrix}. \tag{11}$$

The SOR method involves the introduction of conditioning matrices D and W, relaxation factors α and ω, and the application of an appropriate matrix-splitting scheme to (11) to arrive at the following iterative procedure (Blanchard and Adegbege, 2017):

$$z^{k+1} = \mathbb{P}\left(z^k - \alpha D^{-1}(Hz^k + E^T\lambda^k + q)\right), \tag{12a}$$

$$\lambda^{k+1} = \lambda^k + \omega W^{-1}(Ez^{k+1} - e). \tag{12b}$$

The convergence properties and guidelines for choosing the conditioning matrices and the relaxation factors are discussed in (Blanchard and Adegbege, 2017). Observe

that the primal variable z and the dual variable λ are updated in a Gauss-Seidel manner. A gradient descent step is taken in the steepest descent direction followed by projection onto the feasible set $\mathbb{Z}$, and then a gradient ascent step in λ using the most recent update of z.

3. HARDWARE ARCHITECTURE FOR IMPLEMENTATION

For real-time control, we develop a custom hardware architecture for efficient control implementation and prototyping. The development is modularized to enable plug and play capability for different solver circuits and to allow for handling analog-to-digital conversion, numerical scaling, and digital-to-analog conversion in one framework.

3.1 Hardware Development Environment

Field Programmable Gate Arrays (FPGA) have become the tool of choice for both academia and industry for hardware prototyping. They are typically comprised of an integrated circuit containing configurable logic blocks, which can be programmed to execute any logical function through a Register-Transfer Language (RTL) like Verilog or VHDL. Modern FPGAs also contain add-on ICs such as memory components and multipliers. The heterogeneous computing nature of the FPGA allows the control designer a high degree of flexibility and system performance. FPGAs are noted for their inherently parallel computing framework; this can be exploited to accelerate the computation of matrix-based algorithms.

In this work, we target a Digilent Nexys 4 DDR Board hosting a Xilinx Artix-7 XC7A100T1CSG324C FPGA. This board is chosen for its accessibility and its ease of use, and has low cost relative to the amount of features. The FPGA itself contains 101,440 logic cells, 240 DSP multiplier slices, 300 I/O pins and a running master clock of 100 MHz. To implement the control algorithms specified in section 2, we used an integrated design of pure Verilog RTL and Vivado High-Level Synthesis (HLS)-based modules.

To interface with the physical system, we integrate a Digilent PMOD AD1 and DA2 for analog-to-digital (A/D) and digital-to-analog (D/A) conversions. The PMOD AD1 can support positive voltages in the 0-3.3V range with a 12-bit resolution. Similarly, the PMOD DA2 can reconstruct analog voltages in the 0-3.3V range with a 12-bit resolution. The PMOD AD1 contains two Sallen-Key filters for anti-aliasing and the PMOD DA2 contains external filters to prevent high-frequency noise in the analog output. We adopt a sampling frequency of 95.1 kHz as an arbitrary sampling rate; We deem this sampling rate sufficient for our application.

3.2 General Control Architecture

The FPGA implementation requires a significant amount of architecture design and experimentation to ensure proper data flow and functional operation. The general control architecture is shown in Fig. 1. There are several key elements to note in the diagram:

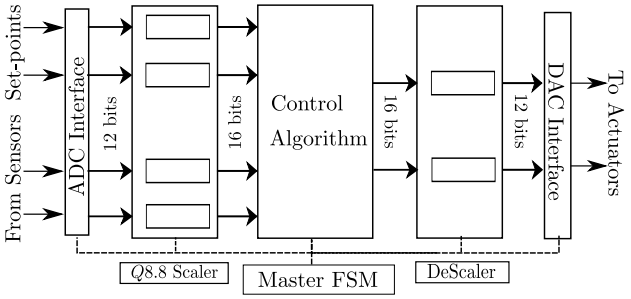

Fig. 1. The General Control Architecture for Embedded MPC

(1) PMOD AD1 Controller: This serves as the external interface to the physical system. This module uses a SPI-like interface to control an Analog Devices AD7674A A/D chip on board a Digilent PMOD AD1 peripheral. The sampling rate is parameterized and set at 95.1 kHz, and samples both the external setpoint voltage as well as the sensor signals.
(2) Fixed-Point Scaler: This converts the 12-bit sample into a Q8.8 Fixed-Point format suitable for the HLS-based control module. To accomplish this, the sample is zero-padded to 16-bits and then scaled by a constant. The scaling accounts for the discrepancy in between the raw Q8.8 interpretation and the actual constituent voltage.
(3) Control Algorithm: This module contains an RTL implementation of the MPC control algorithm. The module is such that any algorithm or solver circuit can be implemented without changing the external interfacing logic.
(4) De-scaler: This module takes the Q8.8 output of the control algorithm and converts it back into the raw 12-bit FPGA sample format for D/A processing.
(5) PMOD DA2 Controller: This module controls a Digilent PMOD DA2, which hosts two Texas Instruments DAC121S101 chips. The D/A chips can be controlled using the same SPI-like interface as the PMOD DA1. The two 12-bit control outputs are received and then shifted out serially at every serial clock cycle to generate the analog output control signal. This module is activated by a DAC-ENABLE signal from the Master FSM to ensure that valid control outputs are being converted into analog signals and to prevent further data contention.
(6) Master Controller: This module serves as the central control unit of the FPGA. The controller enables each module per a Finite State Machine (FSM) that ensures timely data delivery and adequate buffering to prevent data contention and ensure functional correctness. The controller issues enable signals to each of the aforementioned modules in a sequential manner - it keeps track of which module is complete and what data is ready/valid, and then issues the enable for the next downstream module once this information is received by the controller.

3.3 Design Process and Considerations

Traditionally, FPGAs are programmed using a Hardware Descriptive Language (HDL) which describes the behavioral specifications of the circuit to be implemented. This

approach offers the FPGA Engineer the greatest level of design control while abstracting the underlying circuit synthesis - however this approach tends to be costly in terms of labour hours and design verification. In the recent years, High-Level Synthesis (HLS) has emerged as an alternative to traditional HDL design. HLS offers the designer much higher levels of abstraction through C/C++ hardware synthesis - instead of writing the HDL by hand, the HLS compiler automatically maps the designer's C/C++ specification into a logical circuit directed through user-based pragmas. These pragmas allow the user to specify synthesis directives such as latency, resource usage, and parallelism/pipelining.

While HLS offers a significant speedup in design time, there are some constraints that the designer must consider. For one, the user has limited visibility into the resulting synthesized design. This can be an issue for resource-constrained systems in which the HLS-based design is not completely optimized for resource-usage. Moreover, hierarchical system design proved to be difficult to manage as it is difficult to analyze the dataflow from one module to another and to debug the system issues. HLS-based hardware also has poor performance for I/O logic - thereby making ADC/DAC interfacing with HLS difficult. Additionally, the C/C++ implementation needs to be written in a very specific, low-level manner - typical software constructs such as dynamic memory allocation and recursion are not supported for hardware synthesis.

To mitigate the drawbacks of HLS while exploiting the advantages, we adopt a mixed approach to implementing our hardware. We use RTL where it performs best - for system level I/O and dataflow control and we use Verilog HDL for the ADC/DAC interface modules. Hardware sythesis of the HDL-modules is carried out within Vivado environment after extensive verification and testbenching. A similar process is followed for the Master FSM development.

In order to synthesize the controller module, we first verify closed-loop functionality with MATLAB/Simulink. Using MATLAB's Fixed-Point Designer, we analyze the effects of fixed-point quantization and evaluate the resulting performance. The control algorithm is then ported into a C++ implementation which is converted into an RTL using Vivado's C-RTL co-simulation tool. The generated RTL is then exported to the main Vivado development environment as an Intellectual-Property (IP) module where it could be integrated with the rest of the FPGA RTL modules (such as the ADC controller, master FSM, etc.) The integrated system is tested end-to-end with SystemVerilog testbenches for full functional verification and timing closure.

3.4 MPC Controller Architecture

While the HDL generation for a standard control algorithm (e.g. PID) follows a typical MATLAB-to-HLS design paradigm, we need a special design approach for the MPC architecture since internal dataflow can be complicated. To address this, we employ a mixed-model approach where we synthesize the state observer and primal dual independently, and then handle dataflow with a custom RTL. The overall control architecture is shown in Fig. 2.

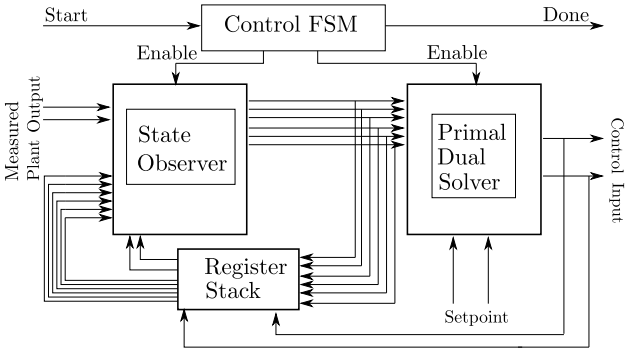

Fig. 2. High Level Control Architecture Block Diagram

The major components are elaborated as follows:

(1) MPC FSM: This module acts as a simple controller that manages the dataflow between the state observer, the Primal-Dual QP solver and the real-time control interface. The controller waits for an MPC-START signal from the bigger master controller, which then triggers the state observer to run. When the state observer asserts the STATE-OBSERVER-DONE signal, the controller enables the Register Stack to save the new values and to feed it on to the QP solver. The controller then allows the QP solver to run - when this is finished the MPC controller asserts an MPC-DONE signal, which is read by the external Master FSM.
(2) State Observer: This is implemented in HLS with the same algorithm described in (3). This module reads the fixed-point sensor voltages delivered by the ADC and scaling logic, and is triggered by an enable signal asserted by the MPC FSM.
(3) Primal-Dual QP Solver: This is implemented in HLS and exported to the larger design as a synthesized RTL IP. This module is triggered to run by the MPC FSM, and delivers the next control outputs which sent to the de-scaling logic and the DAC.
(4) Register Stack: In order to handle the feedback of the state observer values, we implemented a sequential register stack that stores the newest computed values of the state observer. On every MPC run, the register stack provides the state observer values and the control outputs computed in the previous iteration. This module is triggered simultaneously by the State Observer and the MPC FSM.

4. CASE STUDY TO QUADRUPLE TANK SYSTEM

The Quanser Quadruple Tank Water System provides a Multiple-Input Multiple-Output (MIMO) plant system test-bed for control design, testing and verification. The system consists of four water tanks, where the water is pumped in by two pumps at the bottom of the system. The pumps are driven by input voltages and feed water into the bottom two tanks as well as the two respective diagonal tanks as shown in Fig. 3. Water level is measured through pressure sensors located at the bottom of each tank. The range of the output sensor voltage is 0-4.2V, which corresponds to 0-25cm of water height in the tank. All of the interfacing is handled through the Quanser VoltPAQ 2-Channel Linear Amplifier System, which applies 3 times

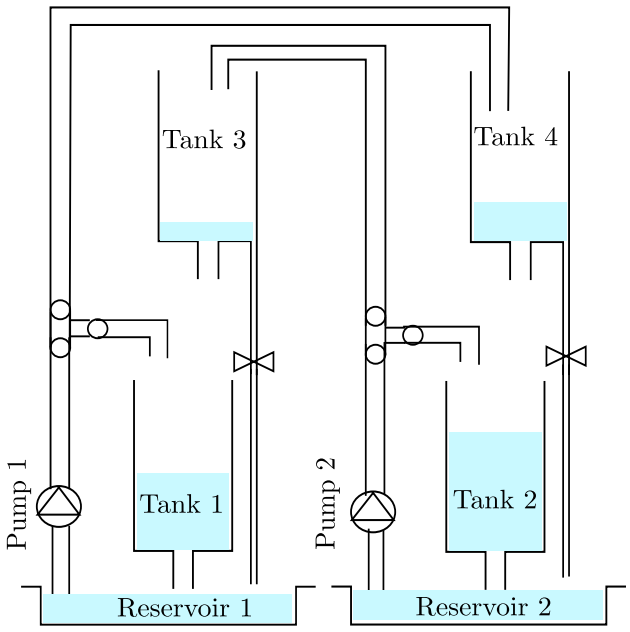

Fig. 3. Quanser Quadruple Tank Configuration

linear amplification to the input control signals as well as delivering the sensor voltages to the controller.

Plant Discretization For digital control implementation, we discretized the continuous time state-space derivation of the the quadruple tank system using Euler method with a sampling time of 5s. The resulting state-space matrices are given by:

$$A = e^{A_c T}, \tag{13}$$

$$B = \left(\int_{\tau=0}^{T} e^{A_c \tau} d\tau \right) B_c \tag{14}$$

$$C = C_c \tag{15}$$

where A_c B_c and C_c are the linearized continuous-time state-space matrices for the tank system about the operating heights of (10, 14, 2.6, 0.76) cm, and T is the sampling time (we refer readers to Johansson (2000) for detailed derivation).

4.1 Experimental Setup

In order to demonstrate the viability of our proposed control architecture, we embed the primal dual solver on an FPGA and close the loop with the Quanser Quadruple tank system. The sensor voltages are measured with RCA-to-differential voltage plugs from the Quanser VoltPAQ system to the ports of the PMOD AD1 on the FPGA. The outputs of the PMOD DA2 are wired to external signal conditioning circuits, and then connected to the VoltPAQ system using RCA connectors. The setpoints are generated using a National Instruments VirtualBench, which is readily available for experimentation. The setpoints, tank responses, and control inputs are all captured using Rigol DS0422 Oscilloscopes that allow for longer data collection and higher resolution than the NI VirtualBench. Each run of the experiment is set at 140 seconds, which allows for full exhibition of the plant and controller dynamics. To investigate the effect of the tank coupling on the control performance, we fixed one of the setpoints at 1.3 Volts (corresponding to a height of 7.74 cm in the tank) and the second as a square-wave signal with a period of 60s and a 50% duty cycle. The amplitude of the square wave is 1.8V and drops to 0V for the second half of the period.

4.2 Experiment Result and Discussion

To benchmark the MPC implementation, we first implement a decoupled Proportional-Integral (PI) controller operating independently on each of the tanks. Both PI controllers are housed on the same FPGA and shared the same resources (i.e the PMOD AD1 and the DA2) and are synchronized to deliver control outputs at the same time. The PI controllers are tuned using the Internal Model Control (IMC) strategy and then implemented in digital form. We do not include antiwindup mechanism in the PI implementation, except that the integrator values are capped to prevent excessive accumulation. For the MPC, the constraints on input voltages are explicitly accounted for during formulation. The input constraints arise because the FPGA cannot output higher than 3.3V and lower than 0V. We set the prediction horizon of the MPC controller to N = 10 as this was the maximum possible prediction horizon we could implement without exceeding the FPGAs resources.

Figs. 4 and 5 shows both the plant and the control responses for step changes in setpoints. As shown, the MPC implementation shows a superior tracking performance as compared to the PI and operates closely to the constraint limits. On the other hand, the PI exhibited significant steady-state offset. This is not surprising as the PI controllers were not designed to handle input constraints and do not include any anti-windup mechanism. Note that aggressive control actions can be minimized by penalizing control moves in the MPC formulation. While it is not our intention to compare the two classes of controllers, the developed modularized framework provides for easy and straightforward implementation of different controllers for purposes of testing and for comparisons.

5. CONCLUSION

In this work we have demonstrated the viability of MPC for real-time control. We implemented a primal-dual QP algorithm in a real-time control framework on an FPGA, and benchmarked its performance with that of a PI implementation. Future work includes optimizing the MPC formulation to exploit the sparsity in the QP matrices and to optimize resource usage on the FPGA. We also aim to reuse the real-time framework to experiment with several different MPC formulations and measure relative real-time performance. We can measure the relative resource usage with respect to the formulation, providing a holistic treatise on MPC formulations and their underlying implementation latency/chip usage.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the TCNJ School of Engineering and the New Jersey Space Grant Consortium (NJSGC) for their continued support.

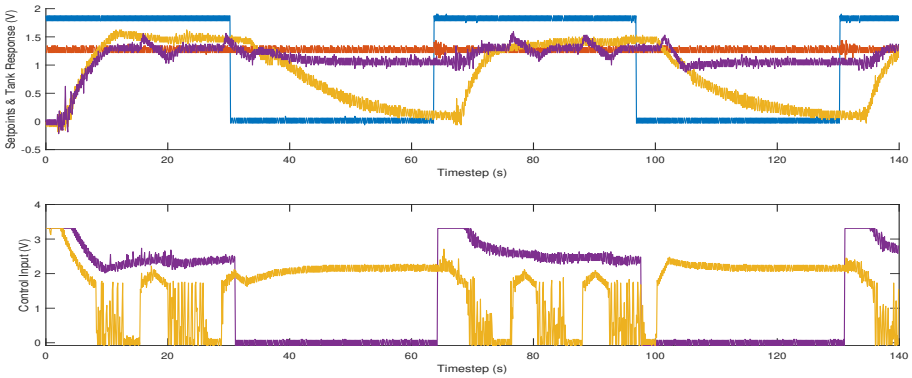

Fig. 4. PI Control Results: Setpoints (Tank 1-Orange and Tank 2-Blue), Responses (Tank 1-Purple, Tank 2-Yellow)

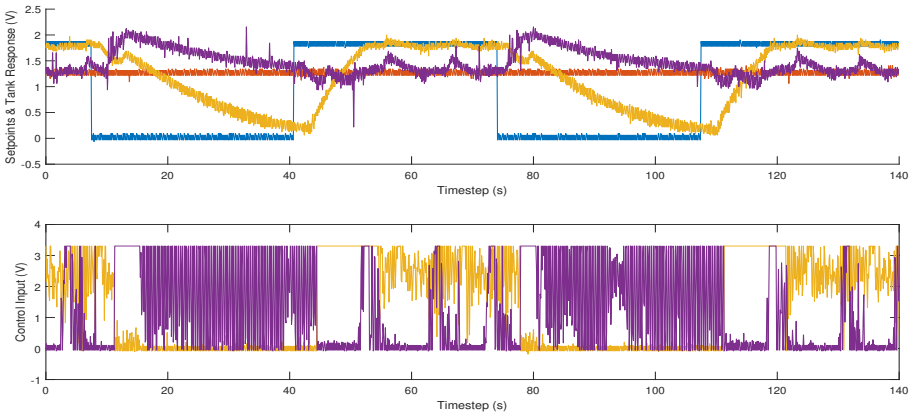

Fig. 5. MPC Control Results: Setpoints (Tank 1-Orange and Tank 2-Blue), Responses (Tank 1-Purple, Tank 2-Yellow)

REFERENCES

Adegbege, A.A. and Nelson, Z.E. (2016). A Gauss–Seidel type solver for the fast computation of input-constrained control systems. *Systems & Control Letters*, 97, 132–138.

Benzi, M., Golub, G.H., and Liesen, J. (2005). Numerical solution of saddle point problems. *Acta numerica*, 14, 1–137.

Blanchard, H.A. and Adegbege, A.A. (2017). An SOR-like method for fast model predictive control. *IFAC-PapersOnLine*, 50(1), 14418–14423.

Borrelli, F., Baotić, M., Pekar, J., and Stewart, G. (2010). On the computation of linear model predictive control laws. *Automatica*, 46(6), 1035–1041.

Borrelli, F., Bemporad, A., and Morari, M. (2017). *Predictive control for linear and hybrid systems*. Cambridge University Press.

Jerez, J.L., Goulart, P.J., Richter, S., Constantinides, G.A., Kerrigan, E.C., and Morari, M. (2014). Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, 59(12), 3238–3251.

Johansson, K.H. (2000). The quadruple-tank process: a multivariable laboratory process with an adjustable zero. *IEEE Transactions on Control Systems Technology*, 8(3), 456–465.

Levenson, R.M., Nelson, Z.E., and Adegbege, A.A. (2017). Programmable logic controller for embedded implementation of input-constrained systems. *IFAC-PapersOnLine*, 50(1), 14412–14417.

Maeder, U., Borrelli, F., and Morari, M. (2009). Linear offset-free model predictive control. *Automatica*, 45(10), 2214–2222.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

O'Donoghue, B., Stathopoulos, G., and Boyd, S. (2013). A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 21(6), 2432–2442.

Patrinos, P. and Bemporad, A. (2013). An accelerated dual gradient-projection algorithm for embedded linear model predictive control. *IEEE Transactions on Automatic Control*, 59(1), 18–33.

Richter, S., Jones, C.N., and Morari, M. (2011). Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6), 1391–1403.

Sabo, J.R. and Adegbege, A.A. (2018). A primal-dual architecture for embedded implementation of linear model predictive control. In *2018 IEEE Conference on Decision and Control (CDC)*, 1827–1832. IEEE.

Wang, Y. and Boyd, S. (2009). Fast model predictive control using online optimization. *IEEE Transactions on control systems technology*, 18(2), 267–278.