Predictive control using an FPGA with application to aircraft control

Edward N. Hartley, Juan L. Jerez, *Student Member, IEEE,* Andrea Suardi, Jan M. Maciejowski, *Fellow, IEEE,* Eric C. Kerrigan, *Member, IEEE,* and George A. Constantinides, *Senior Member, IEEE*

***Abstract*—Alternative and more efficient computational methods can extend the applicability of MPC to systems with tight real-time requirements. This paper presents a "system-on-a-chip" MPC system, implemented on a field programmable gate array (FPGA), consisting of a sparse structure-exploiting primal dual interior point (PDIP) QP solver for MPC reference tracking and a fast gradient QP solver for steady-state target calculation. A parallel reduced precision iterative solver is used to accelerate the solution of the set of linear equations forming the computational bottleneck of the PDIP algorithm. A numerical study of the effect of reducing the number of iterations highlights the effectiveness of the approach. The system is demonstrated with an FPGA-in-the-loop testbench controlling a nonlinear simulation of a large airliner. This study considers many more manipulated inputs than any previous FPGA-based MPC implementation to date, yet the implementation comfortably fits into a mid-range FPGA, and the controller compares well in terms of solution quality and latency to state-of-the-art QP solvers running on a standard PC.**

***Index Terms*—Predictive control, Field programmable gate arrays, Optimization methods, Aerospace control**

I. Introduction

A strength of model predictive control (MPC) is its ability to systematically handle constraints [1]–[4]. At each sampling instant, the solution of a constrained finite-horizon optimal control problem is used as the basis for the control action applied to the plant. With a linear prediction model, a convex quadratic cost function and linear inequality constraints on the plant states and inputs, this can be posed as a convex quadratic program (QP). For MPC with offset-free reference tracking, it is commonplace to also calculate a steady state equilibrium target to which the MPC regulates [5]–[7]. For a system with redundant actuators, this requires the solution of an additional (albeit smaller) constrained optimisation problem.

Manuscript received December 12, 2012; revised May 01, 2013; accepted June 16, 2013. Manuscript received in final form June 27, 2013. This work was supported by EPSRC (Grants EP/G030308/1, EP/G031576/1 and EP/I012036/1) and the EU FP7 Project EMBOCON grant agreement number FP7-ICT-2009-4 248940, as well as industrial support from Xilinx, the Mathworks, and the European Space Agency.

E. N. Hartley and J. M. Maciejowski are with the Department of Engineering, Cambridge University, Trumpington Street, CB2 1PZ. United Kingdom.

J. L. Jerez, A. Suardi and G. A. Constantinides are with the Department of Electrical and Electronic Engineering at Imperial College London, SW7 2AZ. United Kingdom.

E. C. Kerrigan is with the Department of Aeronautics and the Department of Electrical and Electronic Engineering at Imperial College London, SW7 2AZ. United Kingdom.

For small predictive control problems, an attractive option is to pre-compute the explicit solution of the QP as a piecewise-affine function of the current state (and reference) using multi-parametric programming [8], [9]. For large predictive control problems, the computation and storage of the explicit solution is impractical, and the constrained optimisation problem must be solved online. For a broad range of control applications that could benefit from employing predictive control, the cost and power requirements of general purpose computing platforms necessary to meet hard real-time requirements are unfavourable, and custom circuits designed specifically for the predictive control application are an attractive alternative. In this paper, a field-programmable gate array (FPGA)-based implementation of an MPC controller for a large airliner is demonstrated. Tracking of roll, pitch and airspeed are achieved through the use of a steady state target calculator and MPC regulator [5], [6], each requiring the solution of a constrained QP. Whilst arguments for the use of MPC in flight control can be found in [10]–[12], the focus of the present paper is on the FPGA-based methodology for implementation of the optimisation scheme rather than tuning controller parameters or obtaining formal certificates of stability, for which a mature body of theory is already available [3], [13].

A. Background and Motivation

Field programmable gate arrays (FPGAs) are reconfigurable chips that can be customised for a specific application. This enables the implementation of complex algorithms exploiting wide parallellisation. For predictive control applications, FPGAs are easily embedded as a system component, and offer cycle-accurate timing guarantees. The low clock frequencies in comparison to high-performance microprocessors can translate to lower power consumption. Computational speed is regained through parallelisation and customisability. Transistor scaling is still improving the performance and reducing the power consumption and cost in each new technology generation.

There have been several previous FPGA implementations of QP solvers for predictive control. In [21], speeds comparable to MATLAB implementations were achieved for an aircraft example with four states, by using a high-level synthesis tool to convert a C-like description into a hardware description. A soft-core (sequential) processor implemented on the FPGA fabric was used in [20] to execute a C implementation of the QP solver and demonstrate the performance on a two-state drive-by-wire system. In [22], [23], a mixed software/hardware implementation is used where the core matrix computations

TABLE I
CHARACTERISTICS OF EXISTING FPGA-BASED QP SOLVER IMPLEMENTATIONS

Year	Ref.	Number format	Method	Linear Solver	QP Form	Design Entry	Implementation Architecture	Clock Frequency	QP size n_v	QP size n_c
2011	[14]	fixed	Mehrotra IP	Cholesky	D	AccelDSP	custom HW	20 MHz	3	6
2011	[15]	fixed/float	Hildreth	–	D	–	custom core	–	–	–
2011	[16]	float23	log-barrier IP	CG	D	VHDL	custom HW	70 MHz	12	24
2012	[17]	float32	Active set	Factorization	D	C/Verilog	HW/PowerPC	100 MHz	3	6
2012	[18]	float24	Active set	Chol-update	D	VHDL	custom HW	70 MHz	12	24
2012	[19]	float18	log-barrier IP	CG	D	VHDL	custom HW	70 MHz	16	32
2012	[20]	float32	Dual	Cholesky	D	C/C++	soft-core	150 MHz	3	6
2012	Present (MPC)	float32	PDIP	MINRES	S	VHDL	custom HW	250 MHz	377 (174)	408 (170)
2012	Present (tgtcalc)	fixed	FGM	N/A	D	HDL-Coder	custom HW	250 MHz	29	58

PDIP denotes "primal dual interior point", FGM denotes "fast gradient method", D and S denote dense and sparse formulations, respectively, whereas '–' indicates data not reported in publication, and N/A denotes that the field is not applicable. HW denotes hardware. "Soft-core" indicates vendor provided sequential soft processor, whilst "custom core" indicates a user-designed soft processor. Symbols n_v and n_c denote the number of decision variables and number of inequality constraints, respectively. For the present MPC, values for horizons of $N = 12$ (no brackets) and $N = 5$ (brackets) are shown.

are carried out in parallel custom hardware, whilst the remaining operations are implemented in a general purpose microprocessor. The performance was evaluated on two-state systems. In contrast, in [17] the linear solvers were implemented in software while custom accelerators were used for the remaining operations. A motor servo system with two states was used as a case study. The use of non-standard number representations was studied in [15] with a hybrid fixed-point floating-point architecture and a non-standard MPC formulation tested on a satellite example with six states. Whilst [14] presents a full fixed-point implementation, no analysis or guarantees are provided for handling the large dynamic range manifested in interior-point methods. Recently, active-set [18] and interior-point [16], [19] architectures were proposed using (very) reduced precision floating-point arithmetic and solving a condensed QP. Feasibility was demonstrated on an experimental setup with a 14th order SISO open-loop stable vibrating beam, with impressive computation times. Many online optimisation-based FPGA controllers have been for low-dimensional systems — a domain in which explicit MPC could also potentially be an option. In [24] a summary of FPGA-based MPC implementations up until 2010 is presented. Table I in the present work shows a survey of more recent developments, albeit highlighting little progress. A common trend is the use of dense QP formulations in contrast with the current trends in research for structure-exploiting optimisation algorithms for predictive control.

B. Summary of Contribution

The present paper considers a plant model with significantly more manipulated inputs than prior FPGA-based implementations. Furthermore, the sparse structure of the uncondensed optimisation problem is exploited and unlike prior FPGA-based implementations of MPC, the steady-state target calculator for offset free control is included on-chip as a separate constrained QP. The implementation is also capable of running reliably at higher clock rates than prior designs. Table II summarises key characteristics of the design.

1) MPC Regulator: The MPC regulator is based on a modification to a VHDL-based design, first presented in [24], [25] and applied in [26]. A primal-dual interior-point (PDIP) algorithm is used to solve the constrained QP. A parallel implementation of the minimum residual (MINRES) algorithm is used to solve the system of linear equations occuring at each iteration of the PDIP algorithm. Unlike many embedded microprocessors, an FPGA does not preclude standard double precision arithmetic. However, single precision arithmetic is used to reduce the number of hardware resources required, since this reduces the size, cost and power consumption of the FPGA device needed to realise the design.

The MINRES algorithm is known to be sensitive to numerical precision and conditioning, and a preconditioner is often needed to accelerate convergence and to improve the quality of the converged solution. In [24], [25] no preconditioning was used, whilst in [26] a scaling of the state space prediction matrices was demonstrated as an effective off-line preconditioner.

To demonstrate the trade-off between control performance and solution time, a numerical study is performed to investigate the compromise between the number of iterations of the inner MINRES algorithm, and the effect of off-line model scaling and on-line matrix preconditioning. All of these directly influence the total solution time and the solution quality, both in terms of fidelity of the computed control input with respect to that obtained from a standard QP solver, and in terms of the resulting closed loop control performance. Despite using single precision arithmetic, the proposed design using off-line scaling and on-line preconditioning, running on an FPGA with the circuit clocked at $250\,\mathrm{MHz}$, gives solution accuracies and times comparable to those of standard matrix factorisation-based algorithms using double precision arithmetic on a mid-range (circa 2012) laptop computer.

2) Target Calculator: The design for the MPC regulator exploits the sparse structure of the optimisation problem [27], [28]. This structure is not present in the steady state target calculation problem and therefore the design cannot be re-used without substantial modification. FPGA resource constraints make implementing a second floating point interior point solver an unrealistic option. Since the target calculation problem is dense and bound constrained, the steady state target calculator is realised using the fast gradient method (FGM) [29]. This was carried out using the Mathworks' HDL Coder [30], using fixed point arithmetic.

3) System Integration: The two QP solvers are each implemented as peripherals to a Xilinx MicroBlaze soft-core processor [31], which is used to handle data transfer between the two QP solvers and the outside world via UDP/IP over

TABLE II
KEY PARAMETERS CHARACTERISING MPC IMPLEMENTATION

(a) MPC parameters

Description	Value
Plant States	12
Plant Inputs	17
Disturbances	10
Horizon	12
Sampling period	0.2 s

(b) QP Solver characteristics

(MPC regulator)	
QP Algorithm	Primal-dual interior point
Arithmetic	32-bit IEEE float
(Target calculator)	
QP Algorithm	Fast gradient method
Arithmetic	Fixed point

(c) FPGA circuit parameters

Description	Value	Description	Value
Test board	Xilinx ML605	MPC clock	250 MHz
FPGA Model	Xilinx V6-LX240T	Target calc. clock	250 MHz
Synthesis tool	Xilinx ISE	MicroBlaze clock	100 MHz

100 Mbit/s Ethernet. In contrast to [17], [22], [23] where the custom circuit is used to accelerate key parts of the QP solver with the remainder implemented in software on a conventional processor, the present design uses the MicroBlaze solely to bridge communication. This can be replaced with a custom interface layer to suit the demands of a given application.

The resulting "system-on-a-chip" is verified using a test-bench system with the plant model (in this case a high-fidelity nonlinear model of the rigid body dynamics of a large airliner [11]), simulated faster-than-real-time in SIMULINK. The plant state estimates and output reference setpoints are communicated to the system running on the FPGA in the payload of a UDP packet at each sampling instant, with the FPGA returning control commands in the same manner.

C. Outline

The remainder of the paper is organised as follows: Section II summarises the MPC control problem; Section III introduces the case study application; Section IV describes the hardware implementation; Section V presents a numerical investigation of the trade-off between MINRES iterations, online and off-line preconditioning and control performance; Section VI describes the test bench; Section VII presents numerical results; and Section VIII concludes.

II. PREDICTIVE CONTROL PROBLEM

In this section, the constrained, linear predictive tracking control problem with quadratic cost is briefly summarised. The forms of the two quadratic programs to be solved on the FPGA at each sampling instant are described.

A. Predictive regulation control problem

Let $\overline{x} \in \mathbb{R}^{n_x}$ and $\overline{u} \in \mathbb{R}^{n_u}$ denote the state and controlled input of a nonlinear plant at an equilibrium trim point, where n_x is the number of states and n_u is the number of inputs. Defining the nonlinear plant state at time k as $x(k) \in \mathbb{R}^{n_x}$ and input $u(k) \in \mathbb{R}^{n_u}$ respectively, and letting $\delta x(k) \triangleq x(k) - \overline{x}$, and $\delta u(k) \triangleq u(k) - \overline{u}$, a linear prediction model of the form

$$\delta x(k+1) = A\delta x(k) + B\delta u(k) + B_d w(k) \tag{1}$$

can be obtained by linearisation of a nonlinear model around this $(\overline{x}, \overline{u})$, where $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$ and $B_d \in \mathbb{R}^{n_x \times n_w}$, and $w(k) \in \mathbb{R}^{n_w}$ is an unmeasured disturbance.

Letting $\delta x_s(r(k))$ and $\delta u_s(r(k))$ be state and input setpoints relative to the equilibrium trim point, N be the control horizon (equal to the prediction horizon), $\delta\hat{x}(k)$ be the current estimate of the linearised state, $\hat{w}(k)$ be an estimate of the unmeasured disturbance, δx_i and δu_i denote predicted state and input deviations from the trim point at a time instant i steps into the future, $\theta \triangleq [\delta x_0^T, \delta u_0^T, \cdots, \delta x_N^T]^T$, $\| \bullet \|_M^2$ be a shorthand for $\bullet^T M \bullet$, and $P \geq 0$, $Q \geq 0$, $R > 0$ be appropriately sized weighting matrices, then the finite-horizon optimal control problem to solve at each time step is

$$\min_{\theta} \|\delta x_N - \delta x_s\|_P^2 + \sum_{i=0}^{N-1} \left(\|\delta x_i - \delta x_s\|_Q^2 + \|\delta u_i - \delta u_s\|_R^2 \right) \tag{2a}$$

$$\text{subject to:} \quad \delta x_0 = \delta\hat{x}(k) \tag{2b}$$

$$\delta x_{i+1} = A\delta x_i + B\delta u_i + B_d\hat{w}(k),\ i = 0, \ldots, N-1 \tag{2c}$$

$$\delta u_{\min} \leq \delta u_i \leq \delta u_{\max}, \quad i = 0, \ldots, N-1. \tag{2d}$$

For simplicity, state constraints are neglected and with only input constraints considered, no further measure is needed to ensure that a feasible solution to the QP exists, hence no method to detect infeasibility is required in the QP solver.

Letting $\otimes$ denote the Kronecker product, $\oplus$ denote the matrix direct sum, $I_i \in \mathbb{R}^{i \times i}$ be an identity matrix, and $\mathbf{1}_i \in \mathbb{R}^{i \times 1}$ be a vector of ones, let

$$H \triangleq 2\left(I_N \otimes (Q \oplus R)\right) \oplus P \tag{3a}$$

$$G \triangleq \begin{bmatrix} I_N \otimes \begin{bmatrix} 0_{n_u \times n_x} & I_{n_u} \\ 0_{n_u \times n_x} & -I_{n_u} \end{bmatrix},\ 0_{2Nn_u \times n_x} \end{bmatrix} \tag{3b}$$

$$F \triangleq \begin{bmatrix} -I_{n_x} & & & & \\ A & B & -I_{n_x} & & \\ & & \vdots & \ddots & \\ & & & & -I_{n_x} \end{bmatrix} \tag{3c}$$

$$h \triangleq 2\left[-\mathbf{1}_N^T \otimes \begin{bmatrix} \delta x_s^T Q & \delta u_s^T R \end{bmatrix} \quad -\delta x_s^T P\right]^T \tag{3d}$$

$$g \triangleq \mathbf{1}_N \otimes \begin{bmatrix} \delta u_{\max}^T & -\delta u_{\min}^T \end{bmatrix}^T \tag{3e}$$

$$f \triangleq \left[-\hat{x}^T(k) \quad -(\mathbf{1}_N^T \otimes \hat{w}^T(k) B_d^T)\right]^T. \tag{3f}$$

The optimal control problem (OCP) (2) is posed as a parametric QP,

$$\min_{\theta} \tfrac{1}{2}\theta^T H\theta + h^T\theta \quad \text{subject to} \quad G\theta \leq g,\ F\theta = f \tag{4}$$

where vectors h and f change between problem instances.

B. Steady state target calculation

For nominal offset-free steady state tracking of reference setpoints it is common to use a target calculator [5]–[7] to calculate δx_s and δu_s that satisfy

$$A\delta x_s(k) + B\delta u_s(k) + B_w\hat{w}(k) = \delta x_s(k) \tag{5a}$$

$$C_r\delta x_s(k) = r(k) \tag{5b}$$

$$\text{and} \quad \delta u_{\min} \leq \delta u_s \leq \delta u_{\max}, \quad \delta x_{\min} \leq \delta x_s \leq \delta x_{\max} \tag{5c}$$

where $C_r \in \mathbb{R}^{n_r \times n_x}$, and $r(k) \in \mathbb{R}^{n_r}$ is a vector of n_r reference setpoints to be tracked without offset if, for some value r_∞, $r(k) \to r_\infty$ as $k \to \infty$. A feasible solution to (5) is not guaranteed to exist. Let $A_s \triangleq \begin{bmatrix}(A-I) & B\\ C_r & 0\end{bmatrix}$, $B_s \triangleq \begin{bmatrix}-B_w & 0\\ 0 & I\end{bmatrix}$, $\theta_s \triangleq \begin{bmatrix}\delta x_s\\ \delta u_s\end{bmatrix}$, $b_s \triangleq \begin{bmatrix}\hat{w}(k)\\ r(k)\end{bmatrix}$ and $W > 0$ be a weighting matrix. The solution of

$$\min_{\theta_s} \quad \tfrac{1}{2}\theta_s^T A_s^T W A_s \theta_s - b_s^T B_s^T W A_s \theta_s \tag{6}$$

subject to (5c) will find a solution satisfying the equality constraints if one exists and return a least-squares approximation if one does not. This is a dense QP with no equality constraints; however, it is not guaranteed that $A_s^T W A_s > 0$ and the solution of (6) may not be unique. Defining $\mathcal{Q} \triangleq Q \oplus R$, $A_s^\perp$ to be a matrix whose columns form an orthogonal basis of $\mathrm{Ker}(A_s)$ and $\mathcal{H}_s \triangleq A_s^T W A_s + A_s^\perp {A_s^\perp}^T \mathcal{Q} A_s^\perp {A_s^\perp}^T$, a (strictly convex) target calculation problem can now be posed as

$$\min_{\theta_s} \tfrac{1}{2}\theta_s^T \mathcal{H}_s \theta_s - b_s^T B_s^T W A_s \theta_s \tag{7}$$

subject to (5c). The optimal values $\delta x_s^*(r(k))$ and $\delta u_s^*(r(k))$ are then used as the setpoints in the regulation problem (2).

III. Case study — control of a large airliner

For this implementation, the control of the roll, pitch and airspeed of a nonlinear Simulink-based model [11], [32] of the rigid-body dynamics of a Boeing 747-200 with individually manipulable controls surfaces is considered.

A prediction model of the form (1) is obtained by linearisation of the nonlinear model about an equilibrium trim point for straight and level flight at an altitude of $600\,\mathrm{m}$ and an airspeed of $133\,\mathrm{ms}^{-1}$, discretised with a sample period of $T_s = 0.2\,\mathrm{s}$. The linearised model considers 14 states (roll rate, pitch rate, yaw rate, airspeed, angle of attack, sideslip angle, roll, pitch, yaw, altitude, and four engine power states). Yaw angle and altitude are neglected in the prediction model used for the predictive controller, since they do not affect the roll, pitch and airspeed (leaving 12 remaining states). The 17 inputs considered consist of four individually manipulable ailerons, left spoiler panels, right spoiler panels, four individually manipulable elevators, a stabiliser, upper and lower rudder, and four engines. The effects of landing gear and flaps are not considered as these substantially change the local linearisation. The disturbance input matrix B_d is selected to describe an input disturbance in discrete time that is equivalent to a constant disturbance to the rate of each of the first 10 states in continuous time. The cost function (2a) is chosen with the weights in Table III and the constraints on the inputs are summarised in Table IV.

The twelve states $\delta x(k)$ of the model (1) are assumed measurable, along with the two variables that were neglected in the prediction mode: the altitude, and the yaw angle. The disturbance $w(k)$ cannot be measured. As is standard practice in predictive control, an observer is therefore used to estimate $w(k)$. The observer includes a one step ahead prediction to allow the combined target calculator and predictive regulator a deadline of one sampling period for computation.

TABLE III
Cost function

Matrix	Value
Q	$\mathrm{diag}(7200, 1200, 1400, 8, 1200, 2400, 4800, 4800, 0.005, 0.005, 0.005, 0.005)$
R	$\mathrm{diag}(0.002, 0.002, 0.002, 0.002, 0.003, 0.003, 0.02, 0.02, 0.02, 0.02, 21, 0.05, 0.05, 3, 3, 3, 3)$
P	Solution to discrete-time algebraic Riccati equation

TABLE IV
Input constraints

	Input	Feasible region	Units
1,2	Right, left inboard aileron	$[-20, 20]$	deg
3,4	Right, left outboard aileron	$[-25, 15]$	deg
5,6	Right, left spoiler panel array	$[0, 45]$	deg
7,8	Right, left inboard elevator	$[-23, 17]$	deg
9,10	Right, left outboard elevator	$[-23, 17]$	deg
11	Stabiliser	$[-12, 3]$	deg
12,13	Upper, lower rudder	$[-25, 25]$	deg
14–17	Engines 1–4	$[0.94, 1.62]$	–

The target calculator is configured with $C_r = [\mathbf{e}_4, \mathbf{e}_7, \mathbf{e}_8]^T$, where $\mathbf{e}_i$ is the ith column of the 29×29 element identity matrix, in order that the references to be tracked are airspeed, roll, and pitch angle. The weighting matrix W is selected as an appropriately sized identity matrix.

IV. Hardware implementation

Whilst for software running on desktop-oriented platforms the use of IEEE double precision (64-bit) floating-point arithmetic is generally unquestioned, for embedded platforms single precision (32-bit) arithmetic is often preferred due to the super-linear relationship between word-length and silicon resources [33]. In addition, memory storage and bandwidth requirements — key issues for resource-constrained embedded applications — decrease with the number of bits used. Many applications, including predictive control, do not require the accuracy provided by double precision arithmetic if sufficient care is taken to formulate the problem in a numerically favourable way.

Factors different to those important for a software implementation can motivate the choice of algorithm for a custom hardware design. In sequential software, a smaller floating-point operation (FLOP) count leads to shorter algorithm runtimes. In hardware it is the ratio of parallelisable work to sequential work that determines the potential speed of an implementation. Furthermore, the proportion of different types of operations can also be an important factor. For example, multiplication and addition have lower latency and use fewer hardware resources than division or square root operations.

A. MPC QP Solver

This subsection describes the main architectural and algorithmic details for the design of the solver for problem (4).

1) Primal-dual Interior-point Algorithm: The present design uses a primal-dual interior-point algorithm [28] with an iterative linear solver at its core (Figure 1). Iterative linear solvers can be preferable over direct (factorisation-based)

methods in this context, despite the problem sizes being small in comparison to the problems for which iterative methods have been used historically. Firstly, matrix-vector multiplication accounts for most of the computation at each iteration, an operation offering multiple parallelisation and pipelining opportunities. Secondly, there are few division and square root operations compared to factorisation-based methods. Finally, these methods allow one to trade off accuracy for computational time by varying the number of iterations. Despite iterative methods being more sensitive to poor conditioning than direct methods, for the present application, with suitable problem scaling, a relatively small number of iterations can be sufficient to obtain adequate accuracies, as observed in Section V. Conversely, direct methods are more difficult to parallelise and pipeline, with many points at which all subsequent operations depend upon the solution of a single division operation (with a comparatively long latency).

Because there is no matrix factorisation to re-use, a simple primal-dual interior-point algorithm [28] is employed instead of Mehrotra's predictor-corrector algorithm [34], which is commonly found in software packages. Rather than checking a termination criterion, the number of interior-point iterations is fixed to 18 since this guarantees that a (possibly suboptimal) solution is available at a given time. A detailed investigation into the number of iterations needed by interior-point methods is not the subject of this paper, however, *a posteriori* testing indicates that 18 iterations is sufficient to achieve the accuracy of solution needed, and a crude bound on the duality measure at the final iteration indicates $\mu_{18} \geq 1.5^2 \cdot 0.35^{18} \approx 1.4 \times 10^{-8}$. Both this and its reciprocal are well within the dynamic range of single precision floating point arithmetic. In step 5 of the PDIP algorithm, a backtracking line search algorithm is used, reducing the step-length by a factor of 0.5 per iteration, over a maximum of 17 iterations before rounding to zero.

To accelerate the convergence of the iterative linear solver for Step 3 in the PDIP algorithm, a positive diagonal preconditioner [33], [35] is used whose elements (indexed with braced subscripts) are given by

$$M_{\{ii\}}(\mathcal{A}_k) \triangleq 1/\sqrt{\textstyle\sum_{j=1}^{Z} |\mathcal{A}_{k,\{ij\}}|}, \tag{8}$$

where Z is the number of columns of the matrix. The problem $M(\mathcal{A}_k)\mathcal{A}_k M(\mathcal{A}_k) w_k = M(\mathcal{A}_k) b_k$ is solved and the solution to the original problem is recovered by computing $z_k = M(\mathcal{A}_k) w_k$. Hardware resource usage motivates the use of a simple diagonal structure. Whilst the original motivation for the derivation of this preconditioner was to bound variables rather than to improve convergence [33], [35], it has a positive effect on the convergence of the iterative algorithms for this and other applications. The hardware implementation is outlined in Section IV-A3 and its effect on reducing the number of iterations required for convergence is illustrated in Sections V and VII.

2) Architecture of QP Solver: This design is based on the initial architecture described in detail in [24], [25], implemented using VHDL and Xilinx IP-cores for floating point arithmetic and RAM structures. The implementation is split into two distinct blocks: one block accelerates the computational bottleneck of the algorithm (solving the linear equations in Figure 1, step 3) implementing a parallel MINRES solver; the other block computes all the remaining operations.

1. **Initialization** $\theta_0 = 0.05$, $\nu_0 = 0.3 \cdot \mathbf{1}$, $\lambda_0 = 1.5 \cdot \mathbf{1}$, $s_0 = 1.5 \cdot \mathbf{1}$, $\sigma = 0.35$, $I_{IP} = 18$.
for $k = 0$ to $I_{IP} - 1$
2. **Linearization** $\Lambda_k = \mathrm{Diag}(\lambda_k)$, $S_k = \mathrm{Diag}(s_k)$

$$\hat{\mathcal{A}} \triangleq \begin{bmatrix} H & F^T \\ F & 0 \end{bmatrix}, \quad \Phi_k \triangleq G^T W_k^{-1} G, \quad W_k^{-1} \triangleq \Lambda_k S_k^{-1}$$

$$\mu_k \triangleq (\lambda_k^T s_k)/(Nl + 2p),$$

$$r_k^\theta \triangleq -\Phi_k \theta_k - h - F^T \nu_k - G^T(\lambda_k - \Lambda_k S_k^{-1} g + \sigma \mu_k s_k^{-1}),$$

$$r_k^\nu \triangleq -F\theta_k + f, \; \mathcal{A}_k = \hat{\mathcal{A}} + \begin{bmatrix} \Phi_k & 0 \\ 0 & 0 \end{bmatrix}, \; b_k = \begin{bmatrix} r_k^\theta \\ r_k^\nu \end{bmatrix}.$$

3. **Solve** $\mathcal{A}_k z_k = b_k$ for $z_k \triangleq \begin{bmatrix} \Delta\theta_k^T & \Delta\nu_k^T \end{bmatrix}^T$
4. $\Delta\lambda_k \triangleq \Lambda_k S_k^{-1}(G(\theta_k + \Delta\theta_k) - g) + \sigma\mu_k s_k^{-1}$
   $\Delta s_k \triangleq -s_k - (G(\theta_k + \Delta\theta_k) - g)$
5. **Line Search** $\alpha_k \triangleq \max_{(0,1]} \alpha : \begin{bmatrix} \lambda_k + \alpha\Delta\lambda_k \\ s_k + \alpha\Delta s_k \end{bmatrix} > 0.$
6. $(\theta_{k+1}, \nu_{k+1}, \lambda_{k+1}, s_{k+1}) =$
   $(\theta_k, \nu_k, \lambda_k, s_k) + \alpha_k(\Delta\theta_k, \Delta\nu_k, \Delta\lambda_k, \Delta s_k)$

end

Fig. 1. Primal-dual interior-point algorithm

The block that prepares the linear systems consists of a custom sequential machine with custom complex instructions that reduce instruction storage requirements and is close to 100% efficient, i.e. there are no cache misses or pipeline stalls, so a useful result is produced at every clock cycle. The original sequence of instructions has been modified for problems with only input bound constraints, significantly reducing the amount of computation for calculating Φ_k. In addition, more instructions have been added to calculate the preconditioner M and to recover the search direction z_k from the result for the preconditioned system given by the linear solver. Since very few elements in Φ_k are changing from iteration to iteration, the updating of the preconditioner M is not costly.

The parallel MINRES solver accelerates computations by performing dot-products in a parallel fashion. This block consists of a bank of multipliers that perform all the multiplications concurrently followed by an adder reduction tree that accumulates the results. Since matrix $\mathcal{A}_k$ can be rearranged into a banded form by interleaving the primal and dual variables [27], the size of the dot-products is equal to the size of the band $2V - 1$ where $V = 2n_x + n_u$ is the halfband of the matrix. A customised storage scheme is used to exploit the fact that many elements in $\mathcal{A}_k$ are constant or zero and there is repetition due to the multi-stage problem structure and time-invariance. This saves approximately 70% of memory storage requirements in comparison to a standard banded storage scheme [24]. Exploitation of this structure would not have been possible with direct methods, since the non-zero band in the factorised matrices becomes dense.

Another benefit of FPGA technology for real-time applications is the ability to provide cycle accurate computation time guarantees. Let I_{IP} and I_{MR} be the number

TABLE V
VALUES FOR c IN (9) FOR DIFFERENT IMPLEMENTATIONS.

Horizon	Online preconditioning		Horizon	Online preconditioning	
N	Yes	No	N	Yes	No
5	29	28	12	40	39

of PDIP and MINRES iterations, respectively. Let f_c denote the clock frequency (250 MHz in this implementation) and c relate the time spent by the sequential block to the time spent in a MINRES iteration (this varies with different implementations as described in Table V). Let $Z = N(2n_x+n_u)+2n_x$ denote the number of elements in b_k, and $P \triangleq \lceil (2Z + V + 12\lceil \log_2(2V-1)\rceil + 230)/Z \rceil$, then for the current design, computation time is given by

$$(I_{IP} \cdot P \cdot Z(I_{MR} + c))/f_c \text{ seconds.} \quad (9)$$

3) Online Preconditioning Implementation: Two options for implementing the online preconditioning can be considered. The first option is to compute the preconditioned matrix in the sequential block and store it in the linear solver. This requires no extra computational resources; however, it imposes an extra computational load on the sequential block, increasing overall latency. It also prohibits the use of the customised reduced storage scheme, since the non-zero elements that were previously constant between iterations are no longer constant in the preconditioned matrix.

The second option, adopted by the present implementation, only computes the preconditioner in the sequential block. The original matrix is stored in RAM in the linear solver component, and the preconditioner is applied by a bank of multipliers inserted at the output of the memory. This requires approximately three times as much computation; however, this is not on the critical path, i.e. memories storing the matrix can be read earlier, so there is no effect on execution speed. The reduced storage scheme is retained at the cost of a significant increase in the number of multipliers. There is a trade-off between the extra resources needed to implement this procedure and the amount of acceleration gained through a reduction in iteration count, as investigated in Section VII.

B. Target Calculator

The QP solver described in Section IV-A2 is designed to directly exploit the sparse structure of the problem (4). The target calculation problem (7) does not exhibit such a structure, thus the solver of Section IV-A2 cannot be applied directly. A separate QP solver is therefore required.

Matrix $\mathcal{H}_s \in \mathbb{R}^{n_s \times n_s}$, where $n_s = n_x + n_u$ is relatively small, positive definite by construction, and the constraints (5c) are simple bounds. The FGM algorithm [29] is considered, being division free and well-suited to fixed-point implementation (which uses significantly fewer FPGA resources).

A diagonal scaling matrix M_s is obtained in the same manner as (8). The scaled matrix $\overline{\mathcal{H}}_s \triangleq M_s \mathcal{H}_s M_s$ has its numerical values in the range $[-1, 1]$ and eigenvalues in the range $(0, 1]$ (proven in [33], [35]). The same scaling is applied to calculate $\overline{F}_s \triangleq M_s A_s^T W B_s b_s$ and $\overline{\theta}_q \triangleq M_s[\delta x_q^T, \delta u_q^T]^T$,

1. Initialisation
$\overline{\theta}_s^{(0)} = 0$, $y^{(0)} = 0$, $L = 1$, $\mu = \lambda_{\min}(\overline{H}_s)$,
$\beta = (1-\sqrt{\mu})/(1+\sqrt{\mu})$, $k_{\max} = I_{FG}$
for $k = 1$ **to** $k_{\max}$

2. Update
$\nabla J^{(k)} \leftarrow \overline{\mathcal{H}}_s y^{(k-1)} + \overline{f}_s$
$\overline{\theta}_s^{(k)} \leftarrow \max\{\min\{y^{(k-1)} - (1/L)\nabla J^{(k)}, \overline{\theta}_{\max}\}, \overline{\theta}_{\min}\}$
$\Delta\theta_s^{(k)} \leftarrow \theta_s^{(k)} - \theta_s^{(k-1)}, \quad y^{(k)} = \overline{\theta}_s^{(k)} + \beta\Delta\theta_s^{(k)}$

end for
3. Output $\overline{\theta}_s^* = \overline{\theta}_s^{(k_{\max})}$

Fig. 2. Fast Gradient Method (where $\text{eig}(\overline{\mathcal{H}}_s) \subseteq (0, 1]$)

$q \in \{\max, \min\}$. For the plant described in Section III, the minimum and maximum eigenvalues of $\mathcal{H}_s$ are $\lambda_{\min}(\mathcal{H}_s) \approx 0.002$ and $\lambda_{\max}(\mathcal{H}_s) \approx 5582$. After scaling, $\lambda_{\min}(\overline{\mathcal{H}}_s) \approx 2.79 \times 10^{-4}$ and $\lambda_{\max}(\overline{\mathcal{H}}_s) = 1.0$.

The implementation is prototyped at a register transfer level using SIMULINK and the MATLAB Fixed Point Toolbox, with control logic specified using M-code function blocks and programmatically converted to VHDL using Mathworks' HDL Coder R2012a. The target calculator circuit is implemented as four distinct subsystems. Subsystem #1 calculates $\overline{f}_s \triangleq \overline{F}_s b_s$ through multiplication of an $n_s \times n_b$ element matrix by n_b element vector (where $n_b = n_d + n_r$). Since this occurs only once per sample, this matrix-vector multiplication is carried out sequentially, with one multiplication and one addition happening simultaneously per clock cycle, to conserve FPGA resource usage. Subsystem #2 contains the implementation of the FGM (Figure 2). As with MINRES, the majority of the effort in this algorithm occurs in a matrix vector multiplication, which happens once per iteration. The matrix $\overline{\mathcal{H}}_s$ is stored with each column in a separate RAM, so that the multiplication of the elements of each row of $\overline{\mathcal{H}}_s$ with the elements of $y^{(k-1)}$ can be performed in parallel, with the sum of the elementwise products calculated using a pipelined tree reduction structure. This latter being conveniently generated by HDL Coder by using the "Sum of Elements" block configured to use a "Tree" structure, with register balancing. Subsystem #3 reverses the preconditioning, element-by-element in series, and subsystem #4 calculates $T_Q^{-1}Qx_s$, $T_R^{-1}Ru_s$ and $T_Q^{-1}Px_s$ — the input parameters to be used to construct h (3d) for the MPC controller circuit of Section IV-A2, but scaled by diagonal matrices T_Q and T_R to improve problem conditioning (Section V).

The fixed point data types (Table VI) are chosen to match the resources present in the particular FPGA device targeted. In particular, data types for multiplication operations are chosen with consideration of the DSP48E resources on the FPGA, each of which includes a 25×18 bit integer multiplier. Matrix values are represented with a 25 bit data type and vector variables with a 35 bit data type, meaning that two DSP48E units are required per vector element to enable a throughput of one element-wise vector product per clock cycle.

The length of the integer portions are chosen based on the maximum ranges expected based on the input constraints

TABLE VI
DATA TYPES USED WITHIN TARGET CALCULATOR

Subsystem	Variable	Dimension	Data type
1	b_s	13×1	sfix35_En21
1	$\overline{F}_s$	29×13	sfix25_En18
1/2	f_s	29×1	sfix35_En21
2	$\overline{\mathcal{H}}_s$	29×29	sfix25_En23
2	$y^{(k)}$	29×1	sfix35_En21
2	$\overline{\theta}_{\max}, \overline{\theta}_{\min}$	29×1	sfix35_En21
2/3	$\overline{\theta}_s^{(k)}$	29×1	sfix35_En21
3	$\mathrm{diag}(M_s)$	29×1	sfix25_En19
3/4	θ_s	29×1	sfix35_En21
4	L_s	41×29	sfix25_En16
4	$\begin{bmatrix} T_Q^{-1}Qx_s \\ T_R^{-1}Rx_s \\ T_Q^{-1}Px_s \end{bmatrix}$	41×1	sfix35_En21

TABLE VII
TARGET CALCULATOR TIMING

Subsystem	Clock cycles	Overlap
1	$n_b + n_s n_b + 10$	–
2	$n_s + I_{FG}(n_s + 33)$	$-n_s$
3	$n_s + 4$	$-n_s$
4	$n_s + (2n_x + n_u)n_s + 10$	$-n_s$
Total	$(I_{FG} + n_b + n_u + 2n_x)n_s + 33I_{FG} + n_b + 24$	

Overlap is the number of cycles where data is transferred from one stage to the next which must be subtracted to obtain overall timing.

from the case study (Section III). The number of clock cycles needed is shown by stage in Table VII, where I_{FG} denotes the number of FGM iterations. The time for each FGM iteration is insignificant compared to the interior point regulator, so $I_{FG} = 1000$ is chosen as a conservative value. A detailed analysis of the convergence properties of FGM using fixed point arithmetic can be found in [36].

V. OFFLINE PRE-SCALING FOR PDIP/MINRES

For each PDIP iteration, the convergence of the MINRES algorithm used to solve $\mathcal{A}_k z_k = b_k$, and the accuracy of the final estimate of z_k are influenced by the eigenvalue distribution of $\mathcal{A}_k$. When no scaling is performed on the prediction model and cost matrices for this application, and no preconditioning is applied on-line, inaccuracy in the estimates of z_k leads the PDIP algorithm to not converge to a satisfactory solution. Increasing the number of MINRES iterations fails to improve the solution, yet increases the computational burden.

Preconditioning applied online at each iteration of the PDIP algorithm can accelerate convergence and reduce the worst-case solution error of z_k. In [26], an off-line pre-scaling was used in lieu of an on-line preconditioner, with the control performance demonstrated competitive with the use of conventional factorisation-based algorithms on a general purpose platform. The rationale behind the pre-scaling is now restated and numerical results presented to demonstrate that combining systematic off-line pre-scaling with on-line preconditioning yields better performance compared to mutually exclusive use.

Matrix $\mathcal{A}_k$ is not constant, but W_k^{-1} is diagonal. Since there are only upper and lower bounds on inputs, the varying component of $\mathcal{A}_k$, Φ_k only has diagonal elements. Moreover, as $k \to I_{IP} - 1$, the elements of W_k^{-1} corresponding to inactive constraints approach zero. Therefore, despite the diagonal elements of W_k^{-1} corresponding to active constraints becoming large, as long as only a handful of these exist at any point, the perturbation to $\hat{\mathcal{A}}$ is of low rank, and will have a relatively minor effect on the convergence of MINRES. Hence, rescaling the control problem to improve the conditioning of $\hat{\mathcal{A}}$ should also improve the conditioning of $\mathcal{A}_k$ in some sense.

Prior to scaling, for $N = 12$, $\mathrm{cond}(\hat{\mathcal{A}}) = 1.77 \times 10^7$. The objective of the following procedure is to obtain diagonal matrices $T_Q > 0$ and $T_R > 0$ to scale the linear state space prediction model and quadratic cost weighting matrices as follows: $A \leftarrow T_Q A T_Q^{-1}$, $B \leftarrow T_Q B T_R^{-1}$, $B_d \leftarrow T_Q B_d$, $Q \leftarrow T_Q^{-1} Q T_Q^{-1}$, $R \leftarrow T_R^{-1} R T_R^{-1}$, $\delta u_{\min} \leftarrow T_R \delta u_{\min}$, $\delta u_{\max} \leftarrow T_R \delta u_{\max}$. This substitution is equivalent to

$$\hat{\mathcal{A}} \leftarrow \hat{M}\hat{\mathcal{A}}\hat{M}, \quad \text{where} \tag{10}$$

$$\hat{M} \triangleq \left(\left(I_N \otimes \left(T_Q^{-1} \oplus T_R^{-1}\right)\right) \oplus T_Q^{-1}\right) \oplus \left(I_{N+1} \otimes T_Q\right). \tag{11}$$

Since T_Q and T_R are diagonal, the diagonal structure of Φ_k is retained. The transformation (10) is a function of both T_Q and its inverse, and both of these appear quadratically.

In [37] some guidelines are provided for desirable scaling properties. In particular, it is desirable to scale the rows and columns of $\hat{\mathcal{A}}$ so that they are all of similar magnitude. Whilst not exactly the original purpose, it should be noted that if the preconditioner (8) is applied repeatedly (i.e. re-preconditioning the same matrix multiple times) to a general square matrix of full rank, the 1-norm of each of the rows converges asymptotically to unity. The method proposed here for normalising $\hat{\mathcal{A}}$ follows naturally but with the further caveat that the structure of $\hat{M}$ is imposed to be of the form (11). Consequently, it is not (in general) possible to scale $\hat{\mathcal{A}}$ such that all row norms are equal to an arbitrary value. Instead, the objective is to reduce the variation in row (and column) norms. Empirical testing suggests that normalising the 2-norm of the rows of $\hat{\mathcal{A}}$ (subject to (11)) gives the most accurate solutions from the PDIP method for the present application.

Noting the structure of $\hat{\mathcal{A}}$, define the following vectors:

$$s_x \triangleq \left\{ s_x \in \mathbb{R}^n \colon s_{x,\{i\}} = \left(\sum_{j=1}^n Q_{ij}^2 + \sum_{j=1}^n A_{ji}^2 + 1 \right)^{1/2} \right\}$$

$$s_u \triangleq \left\{ s_u \in \mathbb{R}^m \colon s_{u,\{i\}} = \left(\sum_{j=1}^m R_{ij}^2 + \sum_{j=1}^n B_{ji}^2 + 1 \right)^{1/2} \right\}$$

$$s_N \triangleq \left\{ s_N \in \mathbb{R}^n \colon s_{N,\{i\}} = \left(\sum_{j=1}^n P_{ij}^2 + 1 \right)^{1/2} \right\}$$

$$s_\lambda \triangleq \left\{ s_\lambda \in \mathbb{R}^n \colon s_{\lambda,\{i\}} = \left(\sum_{j=1}^n A_{ij}^2 + \sum_{j=1}^m B_{ij}^2 + 1 \right)^{1/2} \right\}.$$

Also, define elementwise, $l_1 \triangleq \sqrt{s_u/\mu}$ and

$$l_2 \triangleq \{ l_2 \in \mathbb{R}^n > 0 : \ (l_2)^4 = ((N s_x + s_N)/(1 + N s_\lambda)) \}$$

where $\mu \triangleq (N \sum s_x + \sum s_N + N \sum s_\lambda + n)/(2(N+1)n)$, and apply the Algorithm in Figure 3. Table VIII shows properties of $\hat{\mathcal{A}}$ that influence solution quality, before and after application of the prescaling with $\epsilon = 10^{-7}$: the condition

Data: A, B, Q, R, P, ϵ
1. Let $t_Q \leftarrow \mathbf{1}_n$, and $t_R \leftarrow \mathbf{1}_m$.
Repeat:
2. Calculate l_1, l_2 as functions of current data, and define $L_1 \triangleq \mathrm{diag}(l_1)$, $L_2 \triangleq \mathrm{diag}(l_2)$.
3. Update: $t_Q \leftarrow L_2 t_Q$, $t_R \leftarrow L_1 t_R$, $A \leftarrow L_2 A L_2^{-1}$, $B \leftarrow L_2 B L_1^{-1}$, $Q \leftarrow L_2^{-1} Q L_2^{-1}$, $P \leftarrow L_2^{-1} P L_2^{-1}$, $R \leftarrow L_1^{-1} R L_1^{-1}$.
Until: $(\|l_2 - \mathbf{1}\| < \epsilon) \cap (\|l_1 - \mathbf{1}\| < \epsilon)$
Output: $T_Q \triangleq \mathrm{diag}(t_Q)$, $T_R \triangleq \mathrm{diag}(t_R)$.

Fig. 3. Offline prescaling algorithm

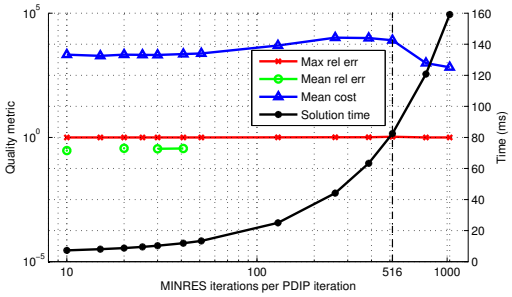
TABLE VIII
EFFECTS OF OFFLINE PRECONDITIONING

Scaling	$\mathrm{cond}(\hat{\mathcal{A}})$	std $\|\hat{\mathcal{A}}_{\{i,:\}}\|_1$	std $\|\hat{\mathcal{A}}_{\{i,:\}}\|_2$	std $\|\lambda_i(\hat{\mathcal{A}})\|$
Original	1.77×10^7	5.51×10^3	4.33×10^3	4.35×10^3
Scaled	2.99×10^4	0.6845	0.5984	0.6226

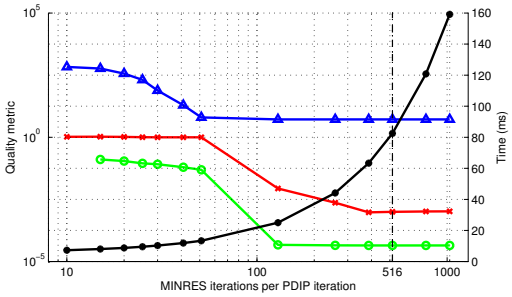
number of $\hat{\mathcal{A}}$; the standard deviation of the row 1– and 2–norms; and the standard deviation of the magnitude of the eigenvalues, which are substantially reduced by the scaling.

Figure 4 shows three metrics for the quality of the solution from the MINRES-based PDIP solver over the duration of a closed-loop simulation with a prediction horizon $N = 12$. The number of MINRES iterations per PDIP iteration is varied for four different approaches to preconditioning (none, offline, online, and combined online and offline). These experiments were performed in software, but a theoretical computation time using (9) for the FPGA implementation is also shown.

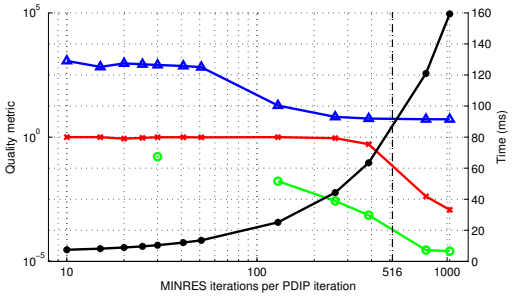
With neither preconditioning nor offline scaling, the control performance is unacceptable. Even when the number of MINRES iterations is equal to $2Z = 2 \times 516 = 1032$, the mean stage cost over the simulation is high (the controller failed to stabilise the aircraft), and the worst case control error in comparison to a conventional PDIP solver using double precision arithmetic and a factorisation-based approach is of the same order as the range of the control inputs. Using solely online preconditioning, control performance (in terms of the cost function) does not start to deteriorate significantly until the number of MINRES iterations is reduced to $\lfloor 0.25Z \rfloor = 129$, although at this stage the worst case relative accuracy is still poor (but mean relative accuracy is tolerable). With only offline preconditioning, worst case relative control error does not deteriorate until the number of MINRES iterations is reduced to $\lfloor 0.75Z \rfloor = 387$ and control performance does not deteriorate until this is reduced to $\lfloor 0.1Z \rfloor = 51$. When combined, control performance is maintained with $\lfloor 0.03Z \rfloor = 15$ iterations, and worst case control accuracy is maintained with $\lfloor 0.08Z \rfloor = 41$ iterations.



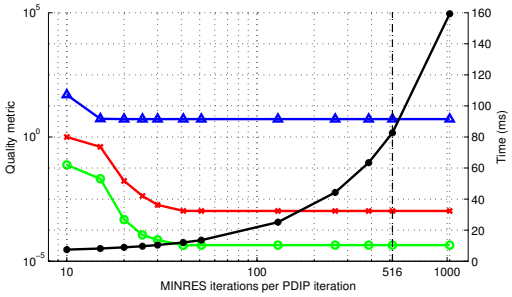
(a) No preconditioning



(b) Offline preconditioning only



(c) Online preconditioning only



(d) Online & Offline Preconditioning

Fig. 4. Numerical performance for a closed-loop simulation with $N = 12$, using PC-based MINRES-PDIP implementation with online and offline preconditioning (missing markers for the mean error indicate that at least one control evaluation failed due to numerical errors). Offline prescaling appears to be more effective than the online preconditioning alone, but combined use yields significantly better numerical performance than using either alone.

VI. FPGA-IN-THE-LOOP TESTBENCH

The hardware-in-the-loop experimental setup used to test the predictive controller design has two goals: providing a reliable real-time closed-loop simulation framework for controller design verification; and demonstrating that the controller could be plugged into a plant presenting an appropriate interface. Figure 5 shows a schematic of the experimental setup. The QP solver, running on a Xilinx FPGA ML605 evaluation board [38], controls the nonlinear model of the B747 aircraft running in SIMULINK on a PC. At every sampling instant k, the observer estimates the next state $\delta\hat{x}(k+1|k)$ and disturbance $\hat{w}(k+1|k)$. For the testbench, the roll, pitch and airspeed setpoints comprising the reference signal $r(k)$ in the target

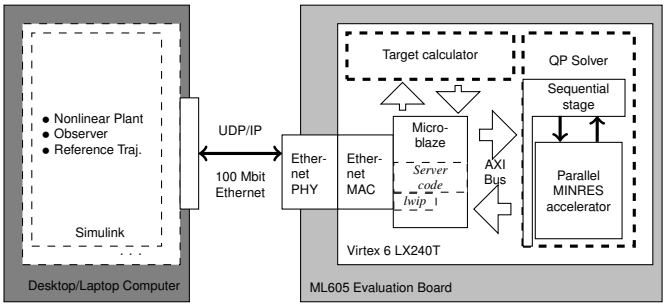


Fig. 5. Hardware-in-the-loop experimental setup.

calculator (5) that the predictive controller is designed to track, are provided by simple linear control loops, with the roll and pitch setpoints as a function of a reference yaw angle and reference altitude, respectively. The airspeed setpoint is passed through a low-pass filter. The vectors $\delta\hat{x}(k+1|k)$, $\hat{w}(k+1|k)$ and $r(k)$ are represented as a sequence of single-precision floating point numbers in the payload of a UDP packet via an *S*-function and this is transmitted over $100\ \mathrm{Mbit/s}$ Ethernet. The FPGA returns the control action in another UDP packet. This is applied to the plant model at the next sampling instant.

On the FPGA the two custom hardware circuits implementing the QP solvers for target calculation and MPC regulation are connected to a Xilinx MicroBlaze soft core processor, upon which a software application, bridges the communication between the Ethernet interface and the two QP solvers. As well as being simpler to implement, this architecture provides some system flexibility in comparison to a dedicated custom interface, with a small increase in FPGA resource usage (Table IX) and communication delay, and allows easy portability to other standard interfaces, e.g. SpaceWire, CAN bus, etc., as well as an option for direct monitoring of controller behaviour.

Table IX shows the FPGA resource usage of the different components in the system-on-a-chip testbench, as well as the proportion of the FPGA used for two mid-range devices with approximately the same silicon area from the last two technology generations (the newer FPGA offers more resources per unit area, meaning a smaller, cheaper, lower power model can be chosen). The linear solver uses the majority of the resources in the MPC QP solver, while the MPC QP solver consumes substantially more resources than the target calculator, since it is solving a larger optimization problem (refer to Section II). Table IX also highlights the cost of using preconditioning.

Using Xilinx Power Analyser (XPA) it is estimated that the Virtex 6 LX240T FPGA-based system draws $9.75\ \mathrm{W}$ of power, of which $4.277\ \mathrm{W}$ is quiescent (e.g. leakage) and $5.469\ \mathrm{W}$ is dynamic. This is less than the $35\ \mathrm{W}$ thermal design power (TDP) associated with the microprocessor in the laptop used for performance comparison in the following section. A smaller, low-voltage or more modern FPGA model can reduce power consumption (particularly leakage). Within the custom circuit, $3.00\ \mathrm{W}$ is used in the interior point QP solver and $0.12\ \mathrm{W}$ in the FGM target calculator.

VII. FPGA-in-the-loop closed loop performance

A closed-loop system with the FPGA in the loop, controlling the nonlinear model of the Boeing 747 from [32] is compared with running the complete control system on a conventional computer using factorisation-based methods. The two QP solvers are first evaluated separately, and then trajectory plots of the closed loop trajectories for the complete system are presented. The reference trajectory is continuous, piecewise continuous in its first derivative, and consists of a period of level flight, followed by a 90° change in heading, then by a $200\ \mathrm{m}$ descent, followed by a $10\ \mathrm{ms}^{-1}$ deceleration.

A. MPC Regulator

Solution times and control quality metrics for the regulator QP solver are presented for a $360\ \mathrm{s}$ simulation, with $N = 12$ and $N = 5$ in Table X. Based on the results of Section V, for $N = 12$, the number of MINRES iterations per PDIP iteration $I_{MR} = 51$. For $N = 5$, $I_{MR} = 30$. This is higher than was empirically determined to be necessary; however, the architecture of the QP solver requires that the MINRES stage must run for at least as long as the sequential stage. The control accuracy metrics presented are

$$e_{\max} \triangleq \max_{i,k} \left| u^F_{\{i\}}(k) - u^*_{\{i\}}(k) \right| / \left(\delta u_{\max,\{i\}} - \delta u_{\min,\{i\}} \right)$$
$$e_{\mu} \triangleq \mathrm{mean}_{i,k} \left| u^F_{\{i\}}(k) - u^*_{\{i\}}(k) \right| / \left(\delta u_{\max,\{i\}} - \delta u_{\min,\{i\}} \right)$$

where $u^F(k)$ is the calculated control input, and $u^*(k)$ is the hypothetical true solution and the subscript $\cdot_{\{i\}}$ indicates an elementwise index. Since the true solution is not possible to obtain analytically, the algorithm of [27], implemented using MATLAB Coder, is used as a baseline.

The metrics are presented alongside those for custom software QP solvers generated using current versions of CVXGEN [39] (for $N = 5$ only since for $N = 12$ the problem was too large to handle) and FORCES [40] (for $N = 12$ and $N = 5$). PC-based comparisons are made using double precision arithmetic on a laptop with a $2.4\ \mathrm{GHz}$ Intel Core 2 Duo processor. The code from CVXGEN and FORCES is modified to use single precision arithmetic and timed running directly on the $100\ \mathrm{MHz}$ MicroBlaze soft core on the FPGA for the number of iterations observed necessary on the PC. (Double precision floating point arithmetic would be emulated in software, and not provide a useful timing comparison.) Whilst obtaining results useful for control from the single precision modification to the CVXGEN solver proved to be too challenging, the timing result is presented assuming random data for the number of iterations needed on the PC. The MicroBlaze used for the software solvers is configured with throughput optimisations, single precision floating point unit (including square root), maximum cache sizes ($64\ \mathrm{kB}$ data and $64\ \mathrm{kB}$ instruction) and maximum cache line length (8 words).

For $N = 12$, the FPGA-based QP solver (at $250\ \mathrm{MHz}$) is slightly faster than the PC-based QP solver generated using FORCES (at $2.4\ \mathrm{GHz}$) based on wall-clock time but $\approx 10\times$ faster on a cycle-by-cycle basis. It is also $\approx 65\times$ faster than the FORCES solver on the MicroBlaze (at $100\ \mathrm{MHz}$), which would fail to meet the real-time deadline of $T_s = 0.2\ \mathrm{s}$ by a

TABLE IX
FPGA RESOURCE USAGE

	MicroBlaze	Target calculator	MINRES solver unpreconditioned	Sequential stage unpreconditioned	MINRES solver preconditioned	Sequential stage preconditioned
LUT	9081 (6%) [3%]	4469 (3%) [1%]	70183 (47%) [23%]	2613 (2%) [1%]	94308 (63%) [31%]	3274 (2%) [1%]
REG	7814 (3%) [1%]	9211 (3%) [2%]	89927 (30%) [15%]	3575 (1%) [1%]	123920 (41%) [20%]	4581 (2%) [1%]
BRAM	40 (10%) [4%]	5 (1%) [0%]	77 (19%) [7%]	14 (3%) [1%]	77 (19%) [7%]	20 (5%) [2%]
DSP48E	5 (1%) [0%]	66 (9%) [2%]	205 (27%) [7%]	2 (0%) [0%]	529 (69%) [19%]	2 (0%) [0%]

Synthesis estimate of absolute and precentage resource usage of the FPGA mounted on the Xilinx ML605 (round brackets) and Xilinx VC707 (square brackets) Evaluation Boards. An FPGA consists of look-up tables (LUT), registers (REG), embedded RAM blocks (BRAM) and multiplier blocks (DSP48E).

TABLE XI
COMPARISON OF FPGA-BASED TARGET-CALCULATOR PERFORMANCE

Implementation QP Solver		Numerical accuracy $e_{\max}$	Numerical accuracy e_{μ}	FGM Solution Time ms	FGM Solution Time Cycles
F /HDL-FGM	Fix	2.0×10^{-2}	3.4×10^{-4}	0.39	7.3×10^{4}
PC/EML-FGM	Flt64	–	–	1.04	2.5×10^{6}
PC/CVXGEN	Flt64	1.7×10^{-3}	1.2×10^{-4}	0.58	1.4×10^{6}
UB/EML-FGM	Flt32	1.4×10^{-5}	2.6×10^{-7}	148.32	1.5×10^{7}

FPGA (F) running at 250 MHz*, PC (PC) at* 2.4 GHz*, (UB) at* 100 MHz*. HDL-FGM is hardware FGM solver, EML-FGM is compiled M-code solver. (–) indicates a baseline.*

factor of approximately 10. By contrast, the clock frequency for the FPGA-based QP solver could be reduced by a factor of ≈ 15 (reducing power requirements, and making a higher FPGA resource utilisation factor possible), or the sampling rate increased by the same factor (improving disturbance rejection) whilst still meeting requirements. Worst-case and mean control error are competitive. A similar trend is visible for $N = 5$ with the FPGA-based solver only marginally slower than the CVXGEN solver on the PC in terms of wall-clock time.

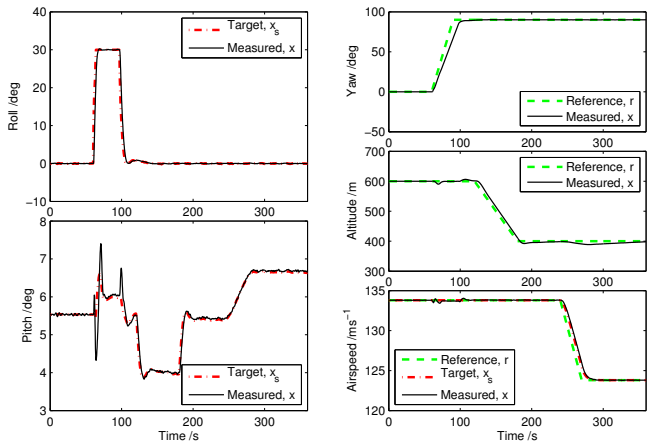
The maximum communication time over Ethernet, experimentally obtained by bypassing the interface with the QP solvers in the software component is 0.6990 ms. The values for FPGA-based implementation in Table X are normalised by subtracting this, since it is independent of the QP solver.

B. Target calculator

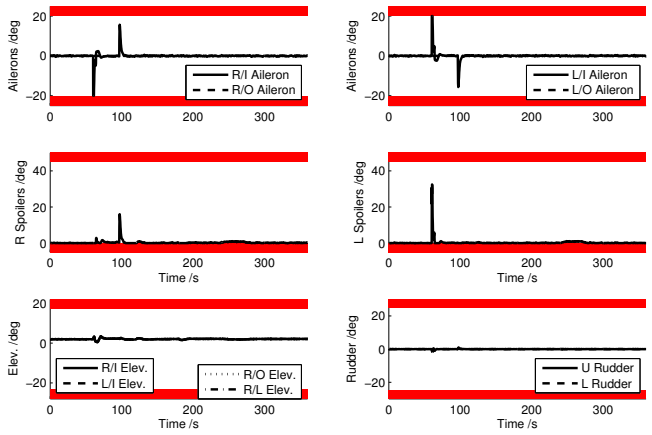
The accuracy and computation time of the fixed-point FPGA-based target calculator is compared with double and single precision variants of the same algorithm generated using MATLAB Coder and with a solver generated using CVXGEN, in Table XI. FORCES is not considered here since the target calculation problem does not have a multi-stage structure. Whilst the accuracy of the FPGA-based implementation is slightly inferior to the other options (but as demonstrated by closed-loop simulation still adequate), the solution wall-clock time is faster than the fastest PC-based algorithm compared, and more than $200\times$ faster on a cycle-per-cycle basis than running a single precision FGM directly on the MicroBlaze, and negligible in comparison to the MPC computation time.

C. Combined system

Trajectories from the closed-loop setup, with $N = 12$ for the PDIP-based MPC regulator, and the FGM-based target calculator running on the FPGA are shown in Figure 6. The



(a) Roll, pitch, yaw, altitude and airspeed trajectories



(b) Selected input trajectories (forbidden region shaded red)

Fig. 6. Closed loop state and input trajectory plots from FPGA-in-the-loop testbench

reference trajectory is tracked, inputs constraints are enforced during transients, and the zero-value lower bound on the spoiler panels is not violated in steady state.

VIII. CONCLUSIONS

This paper has demonstrated the implementation of a "system-on-a-chip" MPC control system, including predictive control regulator and steady-state target calculation on an FPGA. A Xilinx MicroBlaze soft-core processor is used to bridge communication between the two custom QP solvers, and the outside world over Ethernet. The controller is tested

TABLE X
COMPARISON OF FPGA-BASED MPC REGULATOR PERFORMANCE (WITH BASELINE FLOATING POINT TARGET CALCULATION IN SOFTWARE)

Implementation				Relative numerical accuracy		Mean	Max Solution time	
QP Solver	Bits	N	I_{MR}	$e_{\max}$	e_{μ}	cost	QP (ms)	Clock cycles
F /P-MINRES	32	12	51	9.67×10^{-4}	3.02×10^{-5}	5.2246	12	2.89×10^{6}
PC/RWR1998	64	12	–	–	–	5.2247	23	5.59×10^{7}
PC/FORCES	64	12	–	5.89×10^{-3}	1.69×10^{-4}	5.2250	13	3.09×10^{7}
UB/FORCES	32	12	–	3.83×10^{-3}	7.31×10^{-5}	5.2249	1911	1.91×10^{8}
F /P-MINRES	32	5	30	9.10×10^{-4}	2.95×10^{-5}	5.2203	4	1.09×10^{6}
PC/RWR1998	64	5	–	–	–	5.2204	11	2.64×10^{7}
PC/CVXGEN	64	5	–	1.04×10^{-3}	1.84×10^{-5}	5.2203	3	7.20×10^{6}
PC/FORCES	64	5	–	5.00×10^{-3}	1.24×10^{-4}	5.2207	6	1.44×10^{7}
UB/CVXGEN	32	5	–	??	??	??	(269)	(2.69×10^{7})
UB/FORCES	32	5	–	4.14×10^{-3}	8.01×10^{-5}	5.2205	823	8.23×10^{7}

(FPGA QP solver (F) running at 250 MHz*, PC (PC) at* 2.4 GHz *and MicroBlaze (UB) at* 100 MHz*. (–) indicates a baseline. (??) indicates that meaningful data for control could not be obtained). P-MINRES indicates preconditioned MINRES. RWR1998 indicates the algorithm of [27].*

in closed-loop with a non-linear simulation of a large airliner — a plant with substantially more states and inputs than any previous FPGA-based predictive controller.

The MPC regulator employs a PDIP algorithm using single precision floating point arithmetic, with a preconditioned iterative method used to solve systems of linear equations. A numerical investigation shows that with preconditioning and the correct plant model scaling, a relatively small number of MINRES iterations is required to achieve sufficient control accuracy for this application. The steady state target calculator uses the fast gradient method (FGM) implemented using fixed point arithmetic. This is economical in terms of FPGA resources, and is faster than an equivalent algorithm in floating point arithmetic on a laptop PC, whilst running at approximately 10% of the clock frequency. The whole system can fit onto a mid-range FPGA, and uses less than 1/3 of the power of the laptop microprocessor with which it is compared. Lower clock frequencies could be used to further reduce power consumption further, whilst still meeting real-time control deadlines.

REFERENCES

[1] J. M. Maciejowski, *Predictive Control with Constraints*. Pearson Education, 2002.

[2] E. F. Camacho and C. Bordons, *Model predictive control*. London: Springer-Verlag, 2004.

[3] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Nob Hill Publishing, 2009.

[4] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Eng. Pract.*, vol. 11, no. 7, pp. 733–764, Jul 2003.

[5] K. R. Muske and T. A. Badgwell, "Disturbance modeling for offset-free linear model predictive control," *J. Process Control*, vol. 12, no. 5, pp. 617–632, 2002.

[6] G. Pannocchia and J. B. Rawlings, "Disturbance models for offset-free model predictive control," *AIChE J.*, vol. 49, no. 2, pp. 426–437, Feb 2003.

[7] U. Maeder, F. Borrelli, and M. Morari, "Linear offset-free model predictive control," *Automatica*, vol. 45, no. 10, pp. 2214–2222, Oct 2009.

[8] 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, Jan 2002.

[9] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear Model Predictive Control*, ser. Lecture Notes in Control and Information Sciences. Springer Berlin / Heidelberg, 2009, vol. 384, pp. 345–369.

[10] J. M. Maciejowski and C. N. Jones, "MPC fault-tolerant flight control case study: Flight 1862," in *Proc. IFAC Safeprocess Conf.*, Washington, USA, Jun 2003, pp. 9–11.

[11] C. Edwards, T. Lombaerts, and H. Smaili, Eds., *Fault Tolerant Flight Control: A Benchmark Challenge*, ser. Lecture Notes in Control and Information Sciences. Springer, 2010.

[12] H.-G. Geisseler, M. Kopf, P. Varutti, T. Faulwasser, and R. Findeisen, "Model predictive control for gust load alleviation," in *Proc. IFAC Conf. Nonlinear Model Predictive Control*, Noordwijkerhout, Netherlands, 2012, pp. 27–32.

[13] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, Jun 2000.

[14] K. Basterretxea and K. Benkrid, "Embedded high-speed model predictive controller on a FPGA," in *Proc. NASA/ESA Conf. Adaptive Hardware and Systems*, San Diego, CA, Jun 2011, pp. 327–335.

[15] X. Chen and X. Wu, "Design and implementation of model predictive control algorithms for small satellite three-axis stabilization," in *Proc. Int. Conf. Information and Automation*, Shenzhen, China, Jun 2011, pp. 666–671.

[16] A. Wills, A. Mills, and B. Ninness, "FPGA implementation of an interior-point solution for linear model predictive," in *Proc. 18th IFAC World Congress*, Milan, Italy, Aug 2011, pp. 14 527–14 532.

[17] N. Yang, D. Li, J. Zhang, and Y. Xi, "Model predictive controller design and implementation on FPGA with application to motor servo system," *Control Eng. Pract.*, vol. 20, no. 11, pp. 1229–1235, Nov 2012.

[18] A. G. Wills, G. Knagge, and B. Ninness, "Fast linear model predictive control via custom integrated circuit architecture," *IEEE Trans. Control. Syst. Technol.*, vol. 20, no. 1, pp. 59–71, 2012.

[19] A. Mills, A. G. Wills, S. R. Weller, and B. Ninness, "Implementation of linear model predictive control using a field-programmable gate array," *IET Control Theory Appl.*, vol. 6, no. 8, pp. 1042–1054, Jul 2012.

[20] H. Chen, F. Xu, and Y. Xi, "Field programmable gate array/system on a programmable chip-based implementation of model predictive controller," *IET Control Theory Appl.*, vol. 6, no. 8, pp. 1055–1063, Jul 2012.

[21] K.-V. Ling, B. F. Wu, and J. M. Maciejowski, "Embedded model predictive control (MPC) using a FPGA," in *Proc. 17th IFAC World Congress*, Seoul, Korea, Jul 2008, pp. 15 250–15 255.

[22] L. G. Bleris, P. D. Vouzis, M. G. Arnold, and M. V. Kothare, "A co-processor FPGA platform for the implementation of real-time model predictive control," in *Proc. American Control Conf.*, Minneapolis, MN, Jun 2006, pp. 1912–1917.

[23] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare, "A system-on-a-chip implementation for embedded real-time model predictive control," *IEEE Trans. Control. Syst. Technol.*, vol. 17, no. 5, pp. 1006–1017, Sep 2009.

[24] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, "An FPGA implementation of a sparse quadratic programming solver for constrained predictive control," in *Proc. ACM Symp. Field Programmable Gate Arrays*, Monterey, CA, USA, Mar 2011.

[25] J. L. Jerez, K.-V. Ling, G. A. Constantinides, and E. C. Kerrigan, "Model predictive control for deeply pipelined field-programmable gate array implementation: Algorithms and circuitry," *IET Control Theory Appl.*, vol. 6, no. 8, pp. 1029–1041, 2012.

[26] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, "Predictive control of a Boeing 747 aircraft using an FPGA," in *Proc. IFAC Conf. Nonlinear Model Predictive Control*, Noordwijkerhout, Netherlands, Aug 2012, pp. 80–85.
[27] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *J. Optim. Theory Appl*, vol. 99, no. 3, pp. 723–757, Dec 1998.
[28] S. J. Wright, "Interior-point method for optimal control of discrete-time systems," *J. Optim. Theory Appl.*, vol. 77, pp. 161–187, 1993.
[29] S. Richter, C. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Trans. Autom. Control*, vol. 57, no. 6, pp. 1391–1403, 2012.
[30] *HDL Coder user's guide R2012a*, The MathWorks, Natick.
[31] *MicroBlaze Processor Reference Guide – Embedded Development Kit (UG081)*, Xilinx, 2012.
[32] C. van der Linden, H. Smaili, A. Marcos, G. Balas, D. Breeds, S. Runham, C. Edwards, H. Alwi, T. Lombaerts, J. Groeneweg, R. Verhoeven, and J. Breeman, "GARTEUR RECOVER benchmark," 2011. [Online]. Available: http://www.faulttolerantcontrol.nl
[33] E. C. Kerrigan, J. L. Jerez, S. Longo, and G. A. Constantinides, "Number representation in predictive control," in *Proc. IFAC Conf. Nonlinear Model Predictive Control*, Noordwijkerhout, Netherlands, Aug 2012, pp. 60–67.
[34] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM J. Optim*, vol. 2, no. 4, pp. 575–601, Nov 1992.
[35] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, "Towards a fixed point QP solver for predictive control," in *Proc. 51st IEEE Conf. Decision and Control*, Maui, HI, USA, Dec 2012.
[36] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded predictive control on an FPGA using the fast gradient method," in *(To appear in European Control Conf.)*, Zurich, 2013.
[37] J. T. Betts, *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*, 2nd ed. SIAM, 2010.
[38] *ML605 Hardware User Guide*, Xilinx, February 15 2011.
[39] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.
[40] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *Proc. 51st IEEE Conf. Decision and Control*, Maui, HI, USA, Dec 2012.

Edward N. Hartley received the M.Eng degree from University of Cambridge, UK in 2006. He received the Ph.D degree from the same university in 2010. He is currently a Research Associate in the Control group at the University of Cambridge. His research work focuses on predictive control designs for aerospace and space systems and embedded control implementations.

Juan Luis Jerez (S'11) received the M.Eng degree in electrical engineering from Imperial College London, UK in 2009. He is currently a Ph.D student at the Circuits and Systems research group at the same institution. The focus of his research work is on developing tailored linear algebra and optimization algorithms for efficient implementation on custom parallel computing platforms.

Andrea Suardi received his M.Sc. in Electronic Engineering from the Politecnico di Milano, Italy in 2006 and he gained his Ph.D. in Electronics and Communication Engineering at the same university in 2010. Currently, he is a research associate at Imperial College London, UK. His research interests are in digital architecture design, in particular high efficiency FPGA based systems for supercomputing and control applications.

Jan M. Maciejowski (M'81-SM'96-F'11) received a B.Sc degree in Automatic Control from Sussex University in 1971 and a Ph.D degree in Control Engineering from Cambridge University in 1978. From 1971 to 1974 he was a Systems Engineer with Marconi Space and Defence Systems Ltd, working mostly on attitude control of spacecraft and high-altitude balloon platforms. Since 1981 he has been at the University of Cambridge, where he is now Professor of Control Engineering and Head of the Information Engineering Division. He was President of the European Union Control Association from 2003 to 2005, and President of the Institute of Measurement and Control in 2002. He was awarded the Honeywell International Medal by the InstMC in 2008. His current research interests are in the theory and applications of predictive control, its application to fault-tolerant control, in system identification, and in the control of autonomous systems.

Eric C. Kerrigan (S'94-M'02) Dr Kerrigan's research is focused on the development of efficient numerical methods and computer architectures for solving optimal control, estimation, optimization and modeling problems that arise in a variety of aerospace and renewable energy applications. He is a member of the IEEE, IET, MOS and SIAM, is on the IEEE CSS Conference Editorial Board and is an associate editor of the IEEE Transactions on Control Systems Technology, Control Engineering Practice, and Optimal Control Applications and Methods.

George A. Constantinides S'96-M'01-SM'08) received the M.Eng. degree and the Ph.D. degree from Imperial College London in 1998 and 2001. Since 2002, he has been with the faculty at Imperial College, where he leads the Circuits and Systems group. He is a recipient of the Eryl Cadwaladar Davies Prize for the best doctoral thesis in Electrical Engineering at Imperial College (2001), an Imperial College Research Excellence Award (2006), and a fellowship from the EPSRC. He will be program (general) chair of the ACM FPGA Symposium in 2014 (2015) and has previously claired FPL (2003) and FPT (2006) amongst other conferences. Dr Constantinides is a Senior Member of the IEEE and a Fellow of the British Computer Society.