Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Parallel MPC for Real-Time FPGA-based Implementation

Juan L. Jerez * **George A. Constantinides** **
Eric C. Kerrigan *** **Keck-Voon Ling** ****

* *Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, United Kingdom (e-mail: juan.jerez-fullana@imperial.ac.uk)*
** *Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, United Kingdom (e-mail: george.constantinides@ieee.org)*
*** *Department of Electrical and Electronic Engineering and Department of Aeronatics, Imperial College London, SW7 2AZ, United Kingdom (e-mail: e.kerrigan@imperial.ac.uk)*
**** *School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: EKVLING@ntu.edu.sg)*

Abstract: The succesful application of model predictive control (MPC) in fast embedded systems relies on faster and more energy efficient ways of solving complex optimization problems. A custom quadratic programming (QP) solver implementation on a field-programmable gate array (FPGA) can provide substantial acceleration by exploiting the parallelism inherent in some optimization algorithms, apart from providing novel computational opportunities arising from deep pipelining. This paper presents a new MPC algorithm based on multiplexed MPC that can take advantage of the full potential of an existing FPGA design by utilizing the provided 'free' parallel computational channels arising from such pipelining. The result is greater acceleration over a conventional MPC implementation and reduced silicon usage. The FPGA implementation is shown to be approximately 200x more energy efficient than a high performance general purpose processor (GPP) for large control problems.

Keywords: Parallel Algorithms, Parallel Computation, Pipelined Architectures VLSI, Optimization Problems, Optimal Control, Computer Architecture

1. INTRODUCTION

MPC is an advanced optimal control technology that has proven to be very succesful due to its capability of returning an optimal strategy without violating the physical limitations of the system. The need to solve a computationally intensive QP problem at every sampling instant has restricted its applicability to slow plants, such as those encountered in the chemical process industries (Maciejowski, 2001), where sampling times can be on the order of seconds or minutes. As the computational power of new devices continues to rise, MPC is now being proposed for higher bandwidth applications, such as aerospace, robotics, electrical power and automotive. There is a growing demand for ways of accelerating the solution of QP problems so that the success of MPC can be extended to areas where the computational burden has so far been considered too great.

One approach is through hardware acceleration. Recent advances in reconfigurable hardware technology have made the FPGA a suitable platform for accelerating scientific computations. FPGAs are a good alternative to application specific integrated circuits (ASICs) for high speed embedded MPC applications since they offer much reduced low-volume cost, greater flexibility, and a shorter design cycle, reducing the risk while still maintaining deterministic execution time and a high power efficiency.

A parametric FPGA implementation of an interior-point solver for linear MPC has been presented in (Jerez et al., 2011). The design is able to efficiently process problems that are several orders of magnitude larger than previous implementations (Knagge et al., 2009; Ling et al., 2008; Vouzis et al., 2009) and achieves substantial acceleration over a general purpose microprocessor. In this paper we present a new MPC algorithm based on multiplexed MPC (Ling et al., 2010b) that makes use of the slack in the FPGA implementation, in the form of unused parallel computational channels, to achieve lower closed-loop cost. We also demonstrate the large advantage in terms of power efficiency that the FPGA implementation brings over a GPP implementation.

⋆ The authors would like to acknowledge the support of the EPSRC (Grant EP/G031576/1), discussions with Prof. Jan Maciejowski, and industrial support from Xilinx, the Mathworks and the European Space Agency.

A review of MPC and the primal-dual interior point algorithm (Wright, 1997) is presented in Sections 2 and 3. The main features of the FPGA implementation are highlighted in Section 4. Section 5 presents the new MPC algorithm and Section 6 shows the results of the parameterized FPGA design. The performance improvement is demonstrated with an example in Section 7.

2. REVIEW OF LINEAR MPC

Unlike conventional control techniques, MPC takes into account the physical limitations of the system by incorporating constraints into the problem formulation, delivering extra performance gains (Maciejowski, 2001). However, due to the presence of constraints it is not possible to obtain an analytic expression for the optimum solution and we have to solve an optimization problem at every sample instant, resulting in very high computational demands. In linear MPC, we have a linear model of the plant, linear constraints, and a positive definite quadratic cost function, hence the resulting optimization problem is a convex quadratic program (Boyd and Vandenberghe, 2004). Without loss of generality, the time-invariant problem can be described by the following equations:

$$\min_{\boldsymbol{u},\boldsymbol{x}} \frac{1}{2}x_N^T\widetilde{Q}x_N+\sum_{k=0}^{N-1}(\frac{1}{2}x_k^TQ_dx_k+\frac{1}{2}u_k^TR_du_k+x_k^TS_du_k) \quad (1)$$

subject to:

$$x_0 = \widehat{x} \quad (2a)$$

$$x_{k+1} = Ax_k + Bu_k \quad \text{for } k = 0,1,2,...,N-1 \quad (2b)$$

$$Jx_k + Eu_k \leq d \quad \text{for } k = 0,1,2,...,N-1 \quad (2c)$$

$$J_Nx_N \leq d_N \quad (2d)$$

where $u_k \in \mathbf{R}^m$, $x_k \in \mathbf{R}^n$, $\boldsymbol{u} \in \mathbf{R}^{Nm}$ and $\boldsymbol{x} \in \mathbf{R}^{(N+1)n}$ contain all u_k and x_k respectively, $\widehat{x}$ is the current estimate of the state of the plant, N is the horizon length, $Q_d \in \mathbf{R}^{n\times n}$ is symmetric positive semi-definite (SPSD), $R_d \in \mathbf{R}^{m\times m}$ is symmetric positive definite (SPD) to guarantee uniqueness of the solution, $S_d \in \mathbf{R}^{n\times m}$ is such that (1) is convex, $\widetilde{Q} \in \mathbf{R}^{n\times n}$ is an approximation of the cost from $k = N$ to infinity and is SPSD and $\leq$ denotes componentwise inequality. $J \in \mathbf{R}^{l\times n}$, $E \in \mathbf{R}^{l\times m}$ and $d \in \mathbf{R}^l$ describe the physical constraints of the system and l is the number of constraints.

At every sample instance a measurement of the system's output is taken, from which the current state of the plant is inferred (Maciejowski, 2001). The optimization problem (1)–(2) is then solved but only the first part of the solution $u_0^*(\widehat{x})$ is implemented. Due to disturbances, model uncertainties and measurement noise, there will be a mismatch between the next output measurement and what the controller had predicted, hence the whole process has to be repeated again at every sample instant to provide closed-loop stability and robustness.

3. PRIMAL-DUAL INTERIOR-POINT ALGORITHM

Following the non-condensed method, where the states are left as decision variables (Rao et al., 1998), the optimal control problem (1)–(2) can be formulated as a sparse QP of the following form:

$$\min_\theta \frac{1}{2}\theta^TH\theta \quad (3)$$

subject to

$$F\theta = f(\widehat{x}) \quad (4a)$$

$$G\theta \leq g \quad (4b)$$

where

$$\theta := [x_0^T\ u_0^T\ x_1^T\ u_1^T\ \dots\ x_{N-1}^T\ u_{N-1}^T\ x_N^T]^T,$$

$H \in \mathbf{R}^{N(n+m)+n\times N(n+m)+n}$, $F \in \mathbf{R}^{(N+1)n\times N(n+m)+n}$, $f \in \mathbf{R}^{(N+1)n}$, $G \in \mathbf{R}^{Nl+2p\times N(n+m)+n}$, $g \in \mathbf{R}^{Nl+2p}$, and p is the number of outputs to the plant.

The primal-dual interior-point algorithm is a suitable algorithm, which uses Newton's method (Boyd and Vandenberghe, 2004) for solving a perturbed version of a nonlinear system of equations, known as the Karush-Kuhn-Tucker (KKT) optimality conditions. At each iteration, three tasks need to be performed: linearization around the current point, solving the resulting linear system to obtain a search direction, and performing a line search to update the solution to a new point.

The algorithm can be described by the following pseudo-code, where ν and λ are known as Lagrange multipliers, s are known as the slack variables, Λ_k and S_k are diagonal matrices containing the elements of λ_k and s_k respectively, and I_{IP} is the number of iterations (refer to (Wright, 1997) for more details).

Algorithm 1. QP pseudo-code

1. **Initialization** $(\theta_0, \nu_0, \lambda_0, s_0) : [\lambda_0^T\ s_0^T]^T > 0$
   All variables are arbitrarily set to 1.

for $k = 0$ to $I_{IP} - 1$

2. **Linearization** $\mathcal{A}_k := \begin{bmatrix} H + G'W_kG & F' \\ F & 0 \end{bmatrix}$ $b_k := \begin{bmatrix} r_k^\theta \\ r_k^\nu \end{bmatrix}$
   where

$$W_k := \Lambda_kS_k^{-1},$$
$$r_k^\theta := -(H + G^TW_kG)\theta_k - F^T\nu_k - G^T(\lambda_k - W_kg + \sigma\mu_ks_k^{-1}),$$
$$r_k^\nu := -F\theta_k + f,$$
$$\mu_k := \frac{\lambda_k^Ts_k}{Nl + 2p},$$
$$\sigma \in (0, 1].$$

3. **Solve** $\mathcal{A}_kz_k = b_k$ for $z_k =: \begin{bmatrix} \Delta\theta_k \\ \Delta\nu_k \end{bmatrix}$

4. Compute $\Delta\lambda_k := W_k(G(\theta_k + \Delta\theta_k) - g) + \sigma\mu_ks_k^{-1}$
   and $\Delta s_k := -s_k - (G(\theta_k + \Delta\theta_k) - g)$

5. **Line Search** $\alpha_k := \max_{(0,1]} \alpha : \begin{bmatrix} \lambda_k + \alpha\Delta\lambda_k \\ s_k + \alpha\Delta s_k \end{bmatrix} > 0.$

6. **Update** $(\theta_{k+1}, \nu_{k+1}, \lambda_{k+1}, s_{k+1}) :=$
   $(\theta_k, \nu_k, \lambda_k, s_k) + \alpha_k(\Delta\theta_k, \Delta\nu_k, \Delta\lambda_k, \Delta s_k)$

end

4. FPGA IMPLEMENTATION

4.1 Linear Solver

Most of the computational complexity in each iteration of the interior-point method is associated with solving the system of linear equations $\mathcal{A}_k z_k = b_k$. After appropriate row re-ordering (interleaving elements of θ and ν), the indefinite symmetric matrix $\mathcal{A}_k$ becomes banded. The size and half-bandwidth of $\mathcal{A}_k$ in terms of the control problem parameters are given, respectively, by

$$Z := N(2n + m) + 2n, \tag{5a}$$

$$M := 2n + m. \tag{5b}$$

Notice that the number of outputs p and the number of constraints l does not affect the size of A_k, which determines the computation time.

The minimum residual (MINRES) method is a suitable iterative algorithm for solving linear systems with indefinite symmetric matrices (Fisher, 1996). At each MINRES iteration, a matrix-vector multiplication accounts for the majority of the computations. This kind of operation is easy to parallelize and consists of multiply-accumulate instructions, which are known to map efficiently into hardware in terms of resources, in contrast with division operations abundant in factorization based methods. In addition, iterative methods allow one to trade off accuracy for computation time.

In (Boland and Constantinides, 2008) the authors propose an FPGA implementation for solving this type of linear system using the MINRES method, reporting speed-ups of approximately one order of magnitude over software implementations. Most of the acceleration is achieved through a deeply pipelined dedicated hardware block that parallelizes dot-product operations for computing the matrix-vector multiplication in a row-by-row fashion. We use this architecture in our design with a few modifications to customize it to the special characteristics of the matrices that arise in MPC. Notice that the size of the dot-products that are computed in parallel is independent of the control horizon length N (refer to (5b)), thus computational resource usage does not scale with N.

4.2 Sequential Block

The remaining operations in the interior-point iteration are undertaken by a separate hardware block, which we call Stage 1. The resulting two-stage architecture is shown in Figure 1. Since the linear solver will provide most of the acceleration by consuming most resources it is vital that it remains busy at all times. Hence, the parallelism in Stage 1 is chosen to be the smallest possible such that the linear solver is always active. The whole circuit can be seen as a two-stage high-level pipeline where the computational times have to be matched to achieve the highest hardware efficiency.

When computing the coefficient matrix $\mathcal{A}_k$, only the diagonal matrix W_k, changes from one iteration to the next, thus the complexity of this calculation is relatively small. If the structure of the problem is taken into account,

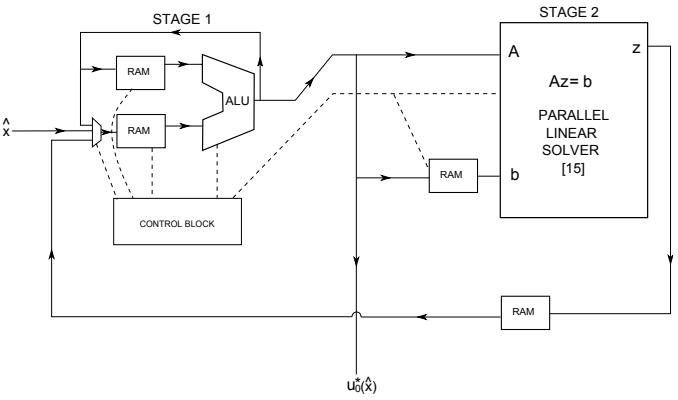

Fig. 1. Proposed two-stage hardware architecture. Solid lines represent data flow and dashed lines represent hardware control signals. Stage 1 performs all computations apart from solving the linear system. The input is the current state measurement $\widehat{x}$ and the output is the next optimal control move $u_0^*(\widehat{x})$.

we find that the remaining calculations in an interior-point iteration are all sparse and very simple compared to solving the linear system. Comparing the computational count of all the operations to be carried out in Stage 1 with the latency of the linear solver implementation, we come to the conclusion that for most control problems, the optimum implementation of Stage 1 is sequential, as this will be enough to keep the linear solver busy at all times. This is a consequence of the latency of the linear solver being $\Theta(N^2)$ (Boland and Constantinides, 2008), whereas the number of operations in Stage 1 is only $\Theta(N)$. For cases where computing Stage 1 takes longer than solving the linear system, another instance of Stage 1 can be synthesized to operate in parallel and share the same common hardware control block.

4.3 Parallel Channels

Notice that if both blocks are to be doing useful work at all times, while the linear system for a specific problem is being solved, Stage 1 has to be updating the solution and linearizing for the next iteration for another independent problem. In addition, the dot-product architecture described in (Boland and Constantinides, 2008) is deeply pipelined, thus it can complete a matrix-vector multiplication in Z cycles (Z rows in $\mathcal{A}_k$), but the latency of one MINRES iteration is longer than Z cycles due to other operations. In order to keep the dot-product hardware busy at all times, P independent problems can be multiplexed into the linear solver for simultaneous processing. Hence, our design can process $2P$ independent QP problems simultaneously at no extra cost in terms of computational resources.

The expression for P for the current implementation is given by

$$P := \left\lceil \frac{2Z + M + 12\lceil \log_2(2M-1) \rceil + 230}{Z} \right\rceil. \tag{6}$$

The linear terms result from the row by row processing for the matrix-vector multiplication (Z dot-products) and serial-to-parallel conversions, whereas the log term comes

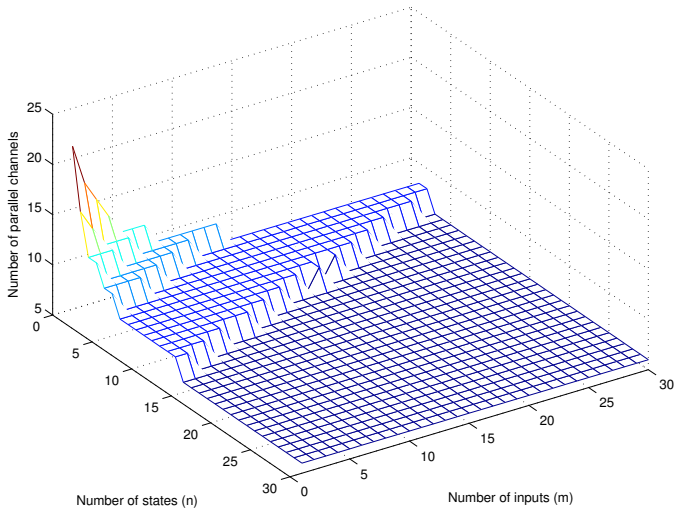

Fig. 2. Number of parallel computational channels available for problems with $N = 10$ and different number of inputs and states.

from the depth of the adder reduction tree in the dot-product block. The constant term comes from the other operations in the MINRES iteration. Figure 2 combines (6) and (5) to show the number of parallel channels available in terms of the parameters of the control problem. For large problems, six parallel channels are available.

4.4 Latency and Throughput

Unlike a software implementation, the FPGA's deterministic execution time can provide the precise timing guarantees required for interfacing a controller to the physical system. The overall latency of the circuit will be given by

$$\text{Latency} = \frac{2I_{IP}PN(I_{MINRES}+1)}{\text{FPGA}_{freq}} \text{ seconds}, \qquad (7)$$

where I_{MINRES} is the number of iterations the MINRES method takes to solve the linear system to the required accuracy, I_{IP} is the number of outer iterations in the interior-point method (Algorithm 1), FPGA_{freq} is the FPGA's clock frequency, PN is the latency of one MINRES iteration (Boland and Constantinides, 2008), and P is given by (6). In that time the controller will be able to output the result to $2P$ problems.

5. PARALLEL MULTIPLEXED MPC

Multiplexed MPC (MMPC) has been proposed elsewhere (Ling et al., 2010b). This section extends the MMPC algorithm in a way such that the architecture proposed in Section 4, which is capable of solving $2P$ QP problems in parallel, can be exploited. We called this version of MMPC, Parallel MMPC.

The original formulation of MMPC was for implementation on a single core processor, solving one QP problem per sampling interval. The key idea in MMPC is that, for an m-input plant, instead of optimizing over all the m input channels in one large QP, the inputs are optimized one channel at a time, in a pre-planned periodic sequence, and the control moves updated as soon as the solution becomes available. This results in a smaller QP at each sampling instant, hence reduced online computational load which in turn enables faster sampling, leading to faster response to disturbances, despite finding a sub-optimal solution to the original optimization problem (Ling et al., 2010a).

MMPC is closer to industrial practice in cases where there is a complex plant with network constraints, meaning that all control inputs cannot be updated simultaneously due to limitations in the communication channels between the actuators and the controller. Parallel MMPC helps to choose which inputs are best to update at any given sampling interval.

A detailed derivation of the Parallel MMPC is beyond the scope of this paper. Instead, we set out the following algorithm which outlines the key steps in Parallel MMPC:

Algorithm 2. Parallel MMPC

1. Initialize by optimizing over all the control moves.
2. Stored planned moves (N moves for each input).

while(1)

3. Apply the first control move for all inputs and shift the plan.
4. Obtain new measurement $\widehat{x}$.
5. Solve m different copies of MMPC in parallel. For each copy, optimize with respect to different subsets of control moves.
6. Evaluate and select from these m copies of MMPC, the set of control moves that gives the smallest cost.
7. Update the plan for the set of control moves that gives the smallest cost.

end

As can be seen from Algorithm 2, Parallel MMPC uses MMPC as an elementary building block. In Parallel MMPC, for a plant with m inputs, at a given time, there can be up to m copies of MMPC. Each of these operates independently and in parallel, and when given the current plant state $\widehat{x}$, optimized with respect to different subsets of control moves. The set of control moves which produces the smallest cost is selected and applied to the plant. The process is repeated at the next updating instant. The resulting updating sequence does not follow a pre-planned sequence and is not necessarily periodic.

Note that Step 1 of Algorithm 2 involves solving for inputs across all input channels. This type of initialization requirement is common in distributed MPC. Subsequent optimizations use this initial solution, but optimize with respect to a subset of control moves. The stability property of MMPC does not depend on the optimality of this initial solution, only its feasibility (Ling et al., 2010b).

Proposition 1. Parallel MMPC, obtained by implementing Algorithm 2, gives closed-loop stability.

Proof. A detailed proof is beyond the scope of this paper and only an outline is provided. The proof follows standard arguments used by most MPC stability proofs. It depends on the constrained optimization being feasible at each step. In the proposed Parallel MMPC algorithm, the default MMPC is always evaluated at every iteration, among the

m parallel copies of MMPC. It then follows that closed-loop stability can be achieved by applying the default MMPC, which is stabilising. This gives the worst case since the Parallel MMPC algorithm ensures that switching to a different MMPC will further reduce the cost. □

6. RESULTS

The design was synthesized using Xilinx XST and placed and routed using Xilinx ISE 12 targeting a Virtex 6 VSX 475T (Vir, 2010). Post place-and-route results showed that a clock frequency above 250MHz is achievable with very small variations for different problem sizes, since the critical path is inside the control block in Stage 1. For the software benchmark, we have used a direct C sequential implementation, compiled using GCC -O4 optimizations running on an Intel Core2 Q8300 with 3GB of RAM, 4MB L2 cache, and a clock frequency of 2.5GHz running Linux. Note that for matrix operations of this size, this approach produces better performance software than using libraries such as Intel MKL.

In order to estimate the device power for the FPGA implementation, we used XPower analyzer (Xpo, 2010) from Xilinx. The tool gives a conservative power consumption estimate based on the design's operating frequency and post-place-and-route resource utilization data. For the range of problems considered in Figure 3, the device power varied almost linearly from 6.7Watts to 20.7Watts. This is a consequence of the computational resources growing linearly with the number of states. It is important to note that no power optimization flags were turned on during synthesis since the main goal is high performance. In order to include the energy consumed by the FPGA's peripherals, we measured the idle power consumption of an RC300 board from Celoxica to be 7.9Watts and added it to the FPGA's power requirements.

For the high performance GPP implementation running Linux, the average power drained from the mains supply was approximately 76Watts. Figure 3 combines measurements of execution time with the power consumption estimates to calculate the energy efficiency of each implementation. For most problems, the FPGA is both faster than the GPP and consumes less power, hence the overall energy performance is much better. For the largest problem reported, the FPGA provides a 38x improvement in energy efficiency when processing one problem or 227x improvement when processing $2P = 6$ problems simultaneously.

7. CASE STUDY

To illustrate the performance improvement that can be achieved with the sampling frequency upgrade enabled by our FPGA implementation, we apply our MPC controller to an example system under different sampling intervals. The example system consists of 18 equal masses (0.15kg) connected by equal springs ($1\text{N}m^{-1}$) (shown in Figure 4). There are two states per mass, its position and velocity, hence the system has 36 states. Each mass can be actuated by a horizontal force ($m = 18$) and the reference for the outputs to track is the zero position for all masses ($p = 18$). The continuous time regulator matrices are

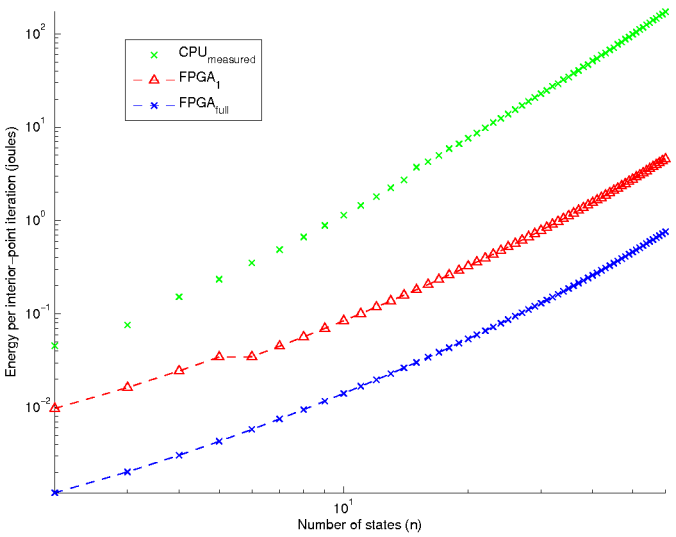

Fig. 3. Energy per interior-point iteration for the GPP, and FPGA implementations when solving one problem and $2P$ problems, where P is given by (6). Problem parameters are $m = 3$, $N = 20$, $p = 3$, and $\text{FPGA}_{freq} = 250\text{MHz}$.

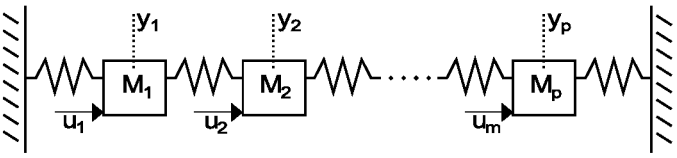

Fig. 4. Spring-mass system.

chosen as $Q_c = C'C$, $R_c = I$ and $S_c = 0$. Notice that the matrices in (1) are the discrete-time version of these matrices assuming a zero order hold (ZOH) in between sampling instants.

The horizon length (T_h) for a specific system is specified in seconds, therefore sampling faster leads to more steps in the horizon (N) and larger optimization problems to solve at each sampling instant. For the example system we found that a horizon of $T_h = 3.1$ seconds was sufficient. Table 1 shows the sampling interval and computational delays for the GPP and FPGA implementations for different number of steps in the horizon. For each implementation, the operating sampling interval is chosen to be the smallest possible such that the computational delay allows solving the optimization problem before the next sample needs to be taken. For this example system, employing parallel MMPC allows sampling 22% faster than with conventional MPC.

Even though the sampling frequency upgrade is modest, there is a reduction in control cost, as shown by the simulation results presented in Figure 5. In addition, employing parallel MMPC not only leads to lower sampling intervals but also lower resource usage, since the optimization problems are smaller (as shown in Table 2). The extra resources could be used to increase the level of parallelism and achieve greater speed-ups. This is a current subject of investigation.

Table 1. Computational delay for each implementation when $I_{IP} = 14$ and $I_{MINRES} = Z$. The grey region represents cases where the computational delay is larger than the sampling interval, hence the implementation is not possible. The smallest sampling interval that the FPGA can handle is 0.281 seconds (3.56Hz) when computing parallel MMPC and 0.344 seconds (2.91Hz) when computing conventional MPC, whereas the GPP samples every 0.775 seconds (1.29Hz). The relationship $Ts = \frac{T_h}{N}$ holds.

N	FPGA_1	FPGA_{MMPC}	GPP	Sampling interval, T_s
4	0.063	0.062	0.580	**0.775**
5	0.092	0.067	0.850	0.620
6	0.126	0.092	1.180	0.517
7	0.166	0.120	1.570	0.442
8	0.211	0.152	2.030	0.388
9	0.262	0.188	2.500	**0.344**
10	0.318	0.227	3.070	0.310
11	0.379	0.270	3.650	**0.281**
12	0.446	0.318	4.350	0.258

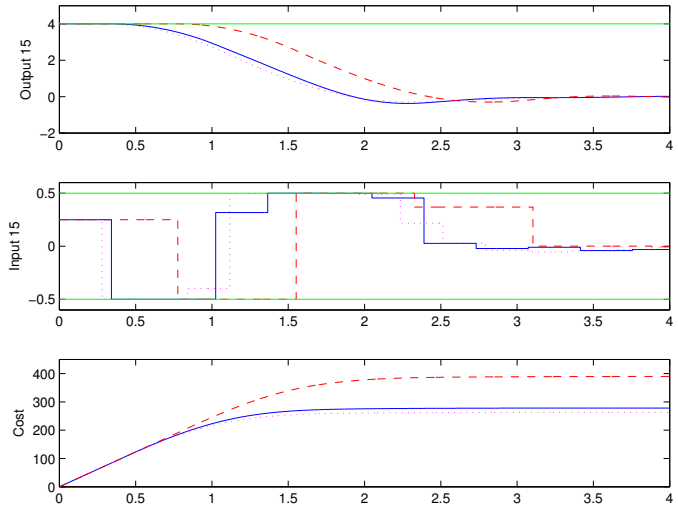

Fig. 5. Comparison of the closed-loop performance of the FPGA using conventional MPC (solid) and parallel MMPC (dotted), and the GPP (dashed). The horizontal lines represent the physical constraints of the system. The closed-loop continuous-time cost represents $\int_0^t x(t)'Q_c x(t) + u(t)'R_c u(t)\,\mathrm{d}t$.

Table 2. Size of QP problems solved by each implementation. Parallel MMPC solves six of these problems simultaneously.

	Decision Variables	Constraints
GPP - MPC	252	324
FPGA - MPC	522	684
FPGA - Parallel MMPC	465	498

8. CONCLUSION

This paper has described the characteristics of an existing FPGA implementation of a QP solver for the optimization problems arising in MPC. A new MPC algorithm that makes use of the slack parallel computational channels offered by the FPGA architecture has been presented. Employing this new strategy allows to save resources and achieve greater acceleration, leading to better quality control. The large power efficiency advantage of the FPGA over a high performance GPP implementation has been established.

We are currently working on new MPC algorithms that can take advantage of the parallel computational channels offered by our design without requiring many inputs to the system, hence applicable to a more general class of systems.

REFERENCES

(2010). Virtex-6 family overview. URL `http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf`.

(2010). Xpower tutorial. URL `ftp://ftp.xilinx.com/pub/documentation/tutorials/xpowerfpgatutorial.pdf`.

Boland, D. and Constantinides, G.A. (2008). An FPGA-based implementation of the MINRES algorithm. In *Proc. Int. Conf. on Field Programmable Logic and Applications*, 379–384. Heidelberg, Germany.

Boyd, S.P. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, Cambridge, UK.

Fisher, B. (1996). *Polynomial based iteration methods for symmetric linear systems*. Wiley, Baltimore, MD, USA.

Jerez, J.L., Constantinides, G.A., and Kerrigan, E.C. (2011). An FPGA implementation of a sparse quadratic programming solver for constrained predictive control. In *Proc. ACM Symposium on Field Programmable Gate Arrays*, 209–218. Monterey, CA, USA.

Knagge, G., Wills, A., Mills, A., and Ninnes, B. (2009). ASIC and FPGA implementation strategies for model predictive control. In *Proc. European Control Conference*. Budapest, Hungary.

Ling, K.V., Ho, W.K., Wu, B.F., Lo, A., and Yan, H. (2010a). Multiplexed MPC for multi-zone thermal processing in semiconductor manufacturing. *IEEE Transactions on Control Systems Technology*, (Accepted for publication).

Ling, K.V., Maciejowski, J.M., Richards, A.G., and Wu, B.F. (2010b). Multiplexed model predictive control. Technical Report CUED/F-INFENG/TR.657, Cambridge University Engineering Department.

Ling, K.V., Wu, B.F., and Maciejowski, J.M. (2008). Embedded model predictive control (MPC) using a FPGA. In *Proc. 17th IFAC World Congress*, 15250–15255. Seoul, Korea.

Maciejowski, J. (2001). *Predictive Control with Constraints*. Pearson Education, Harlow, UK.

Rao, C.V., Wright, S.J., and Rawlings, J.B. (1998). Application of interior-point methods to model predictive control. *Journal of Optimization Theory and Applications*, 99(3), 723–757.

Vouzis, P.D., Bleris, L.G., Arnold, M.G., and Kothare, M.V. (2009). A system-on-a-chip implementation for embedded real-time model predictive control. *IEEE Transactions on Control Systems Technology*, 17(5), 1006–1017.

Wright, S.J. (1997). Applying new optimization algorithms to model predictive control. In *Proc. Int. Conf. Chemical Process Control*, 147–155. CACHE Publications.