FPGA Implementation of an Interior Point Solver for Linear Model Predictive Control

Juan L. Jerez [#1], George A. Constantinides [#2], Eric C. Kerrigan [#*3]

[#] *Department of Electrical and Electronic Engineering, Imperial College London*
London SW7 2AZ, United Kingdom
[1] juan.jerez-fullana@imperial.ac.uk
[2] george.constantinides@ieee.org
[3] e.kerrigan@imperial.ac.uk

[*] *Department of Aeronautics, Imperial College London*
London SW7 2AZ, United Kingdom

***Abstract*—Automatic control, the process of measuring, computing, and applying an input to control the behaviour of a physical system, is ubiquitous in engineering and industry. Model predictive control (MPC) is an advanced control technology that has been very successful in the chemical process industries due to its ability to handle large multiple input multiple output (MIMO) systems with physical constraints. It has recently been proposed to be applied to higher bandwidth systems, which add the requirement of greater sampling frequencies. The main hurdle is the need to solve a computationally intensive quadratic programming (QP) problem in real-time. In this paper we address the need for acceleration by proposing a highly efficient floating-point field-programmable gate array (FPGA) implementation that exploits the parallelism opportunities offered by interior-point optimization methods. The approach yields a 5x improvement in latency and a 40x improvement in throughput for large problems over a software implementation. This work builds on a previous FPGA implementation of an iterative linear solver, an operation at the heart of the interior-point method.**

I. Introduction

A control system tries to maintain the stable operation of a physical system, known as the plant, over time in the presence of uncertainties by means of feedback. In MPC, the controlling inputs at every sample instant are determined by solving a constrained optimization problem, which tries to minimize the deviation between the future outputs and the reference while using minimum control energy and satisfying the constraints of the physical system. As an example, a controller on an aeroplane could have the objective of steering the aircraft safely while minimizing fuel consumption.

Faster sampling frequencies reduce the maximum time between a disturbance acting on the system and the corresponding controlling action being applied, resulting in improved expected performance. However, in conventional MPC, the sampling frequency is determined by the time taken to solve the optimization problem. This task is very computationally intensive, which is why, until recently, MPC has only been a feasible option for systems with very slow dynamics. Examples of such systems arise in the chemical process industries [1], where the required sampling intervals can be of the order of seconds or minutes.

This paper presents a parametric FPGA design to accelerate the solution of the optimization problems arising in MPC so that the scheme can be applied to systems with faster dynamics that require shorter sampling intervals.

Existing work on hardware implementation of optimization solvers can be grouped into those that use interior-point methods [2]–[4] and those that use active-set methods [5], [6]. The major difference is that our approach can efficiently handle optimization problems that are several orders of magnitude larger than what has been presented in previous works.

II. Problem Description

Without loss of generality, the optimization problem can be described by the following equations:

$$\min_{u_k, x_k} x_T' \widetilde{Q} x_T + \sum_{k=0}^{T-1} (x_k' Q x_k + u_k' R u_k + 2 x_k' M u_k) \qquad (1)$$

subject to:

$$x_0 = \widehat{x} \qquad (2a)$$
$$x_{k+1} = A x_k + B u_k \qquad \text{for } k = 0, 1, 2, ..., T-1 \qquad (2b)$$
$$J x_k + E u_k \leq d \qquad \text{for } k = 0, 1, 2, ..., T-1 \qquad (2c)$$
$$J_T x_T \leq d_T \qquad (2d)$$

where $x_k \in \mathbf{R}^n$ is the state of the plant at sample instant k, $u_k \in \mathbf{R}^m$ is the input vector, $\widehat{x}$ is the current estimate of the state of the plant, T is the control horizon length, $Q \in \mathbf{R}^{n \times n}$, $R \in \mathbf{R}^{m \times m}$, $M \in \mathbf{R}^{n \times m}$, $\widetilde{Q} \in \mathbf{R}^{n \times n}$, x' denotes transposition, and $\leq$ denotes componentwise inequality. The equality constraints represent the model of the plant dynamics with $A \in \mathbf{R}^{n \times n}$ and $B \in \mathbf{R}^{n \times m}$. $J \in \mathbf{R}^{l \times n}$, $E \in \mathbf{R}^{l \times m}$ and $d \in \mathbf{R}^l$ describe the physical constraints of the system.

The optimal control problem (1)–(2) can be formulated as a sparse QP of the following form:

$$\min_{\theta} \frac{1}{2} \theta^T H \theta$$
$$\text{subject to} \quad F\theta = f(\widehat{x})$$
$$G\theta \leq g$$

where

$$\theta := [x_0', u_0', x_1', u_1', \ldots, x_{T-1}', u_{T-1}', x_T']'.$$

The primal-dual interior-point algorithm can be used to solve this problem. It uses Newton's method [7] for solving 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 slack variables, W_k is a diagonal matrix, and I_{NW} is the number of iterations (refer to [8] for more details).

Algorithm 1 QP pseudo-code

Choose initial point $(\theta_0, \nu_0, \lambda_0, s_0)$ with $[\lambda_0', s_0']' > 0$
for $k = 0$ to $I_{NW} - 1$ **do**

1. $\mathcal{A}_k := \begin{bmatrix} H + G'W_kG & F' \\ F & 0 \end{bmatrix}$
2. $b_k := \begin{bmatrix} -(H + G'W_kG)\theta_k - F'\nu_k - \\ G'(\lambda_k - W_kg + \sigma\mu_k s_k^{-1}) \\ -F\theta_k + f \end{bmatrix}$
3. Solve $\mathcal{A}_k z_k = b_k$ for $z_k =: \begin{bmatrix} \Delta\theta_k \\ \Delta\nu_k \end{bmatrix}$
4. Compute $\Delta\lambda_k := W_k(G(\theta_k + \Delta\theta_k) - g) + \sigma\mu_k s_k^{-1}$
5. Compute $\Delta s_k := -s_k - (G(\theta_k + \Delta\theta_k) - g)$
6. Find $\alpha_k := \max_{(0,1]} \alpha : \begin{bmatrix} \lambda_k + \alpha\Delta\lambda_k \\ s_k + \alpha\Delta s_k \end{bmatrix} > 0.$
7. $(\theta_{k+1}, \nu_{k+1}, \lambda_{k+1}, s_{k+1}) :=$
   $(\theta_k, \nu_k, \lambda_k, s_k) + \alpha_k(\Delta\theta_k, \Delta\nu_k, \Delta\lambda_k, \Delta s_k)$

end for

III. Implementation

A. Linear Solver

Most of the computational complexity in each iteration is associated with solving the system of linear equations $\mathcal{A}_k z_k = b_k$. After appropriate row re-ordering, matrix $\mathcal{A}_k$ becomes banded and symmetric but indefinite, hence has both positive and negative eigenvalues. The order and half-bandwidth of $\mathcal{A}_k$ in terms of the control problem parameters are given respectively by

$$N := T(2n + m) + 2n, \quad (3a)$$

$$M := 2n + m. \quad (3b)$$

Notice that the number of outputs p and the number of constraints l does not affect the size of A_k, which we will show to determine the total runtime.

In [9] the authors propose an FPGA implementation for solving this type of linear systems using the minimum residual (MINRES) method, reporting speed-ups of around one order of magnitude over microprocessor implementations. At each iteration of the MINRES method, a matrix-vector multiplication accounts for the majority of the computations. Most of the acceleration is achieved through a deeply pipelined dedicated hardware block that parallelizes dot-product operations ($2M - 1$ parallel multiplications followed by an adder reduction tree) for computing this 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 T (refer to (3b)).

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.

B. Pipelining

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.

Notice that if both blocks are to be doing useful work at all times, while we are solving the linear system for a specific problem we have to be updating the solution and linearizing for another independent problem. In addition, the architecture described in [9] has to process P independent problems simultaneously to match latency with throughput and make sure the dot product hardware is active at all times to achieve maximum efficiency. Hence, our design can process $2P$ independent QP problems simultaneously. We believe this to be an appropriate design approach, as it allows us to expose all the computational power of the device and open up new possibilities for new algorithm development.

The expression for P in terms of the matrix dimensions is

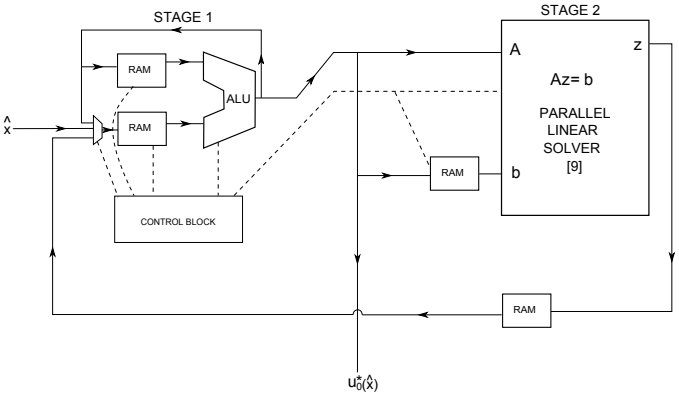

Fig. 1: Proposed two-stage hardware architecture. Solid lines represent data flow and dashed lines represent control signals. Stage 1 performs all computations apart from solving the linear system. The input is the current state measurement $\widehat{x}$ and the output is the next optimal control move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

given by

$$P := \left\lceil \frac{3N + 24\lceil \log_2(2M-1) \rceil + 154}{N} \right\rceil. \quad (4)$$

The linear term results from the row by row processing for the matrix-vector multiplication (N dot-products) and serial-to-parallel conversions, whereas the log term comes from the depth of the adder reduction tree in the dot-product block. It is important to note that P converges to a small number ($P = 4$) as the size of A_k increases, thus for large problems only four independent threads are required.

C. Architecture for Sequential Block

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, 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 of interest to us (large problems with large horizon lengths), 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 $\Theta(T^2)$ [9], whereas the complexity of Stage 1 is only $\Theta(T)$.

The computational block performs any of the main arithmetic operations: addition/subtraction, multiplication and division. Xilinx Core Generator [10] was used to generate highly optimized single-precision floating point units with maximum latency to achieve a high clock frequency and maximise throughput. Extra registers were added after the multiplier to match the latency of the adder for synchronization, as these are the most common operations. The latency of the divider was much larger (27 cycles) than the adder (12 cycles), therefore, it was decided not to mach the delay of the divider path as it would reduce our flexibility for ordering computations. The total number of floating point units in the circuit is $8n + 4m + 27$, where only three units belong to Stage 1.

Since the same computational units are being reused to perform many different operations, the necessary control is rather complex. The control block needs to provide the correct sequence of read and write addresses for the data RAMs, as well as other control signals, such as computation selection. An option would be to store the values for all control signals at every cycle in a program memory and have a counter iterating through them. However, this would take a large amount of memory. For this reason, it was decided to trade a small increase in computational resources for a much larger decrease in memory requirements. Frequently occurring memory access patterns have been identified and a dedicated address generator hardware block has been built to generate them from minimum storage. Each pattern is associated with a control instruction. Examples of these patterns are: simple increments $a, a+1, ..., a+b$ and the more complicated read patterns needed for matrix vector multiplication (standard and transposed). Control instructions to perform line search and linearization for one problem were stored. When the last instruction is reached, the counter goes back to instruction 0 and iterates again for the next problem with the necessary offsets being added to the control signals.

D. Latency and Throughput

Since the FPGA has deterministic timing, we can calculate the latency and throughput of our system. The overall latency of the circuit will be given by

$$\text{Latency} = \frac{2I_{NW}(PN + P)I_{MR}}{\text{FPGA}_{freq}} \text{ seconds}, \quad (5)$$

where I_{MR} is the number of iterations the MINRES method takes to solve the linear system to the required accuracy, I_{NW} is the number of outer iterations in the interior-point method (Algorithm 1), FPGA_{freq} is the FPGA's clock frequency, $(PN + P)$ is the latency of one MINRES iteration, and P is given by (4). In that time the controller will be able to output the result to $2P$ problems.

E. Input/Output

Stage 1 is responsible for handling the chip I/O. It reads the current state measurement $\widehat{x}$ as n 32-bit floating point values sequentially through a 32-bit parallel input data port. Outputting the m 32-bit values for the optimal control move $u_0^*(\widehat{x})$ is handled in a similar fashion. When processing $2P$ problems, the average I/O requirements are given by

$$\frac{2P(32(n+m))}{\text{Latency given by (5)}} \text{ bits/second.}$$

For the example problems that we have considered in Section IV, the I/O requirements range from 0.2 to 0.00015 Mbits/second, which is well within any standard FPGA platform interface, such as PCI Express. The combination of a very computationally intensive task with very low I/O requirements, highlight the affinity of the FPGA for MPC computation.

IV. Results

A. Resource Usage

The design was synthesized using Xilinx XST inside Xilinx ISE 12 targeting a Virtex 6 VSX 475T [11]. Figure 2 shows how the different resources scale with the problem size. For the fixed relationship, $n = 2m$ and $T = 4m$, the number of floating point units is $\Theta(m)$, hence the linear growth in registers, look-up tables and embedded DSP blocks. The memory requirements are $\Theta(m^2)$, which explain the quadratic asymptotic growth observed in Figure 2. The jumps occur when the number of elements to be stored in the RAMs for variable columns exceeds the size of Xilinx BlockRAMs. If the number of QP problems being processed simultaneously is chosen smaller than the value given by (4), the BlockRAM usage will decrease, whereas the other resources will remain unaffected.

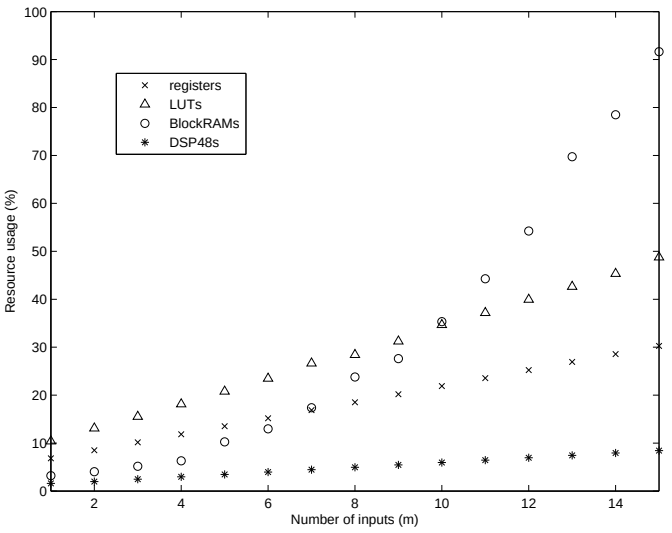

Fig. 2: Resource utilization on a Virtex 6 VSX 475T ($p = m, n = 2m, T = 2n, I_{ls} = 20, P$ given by (4)).

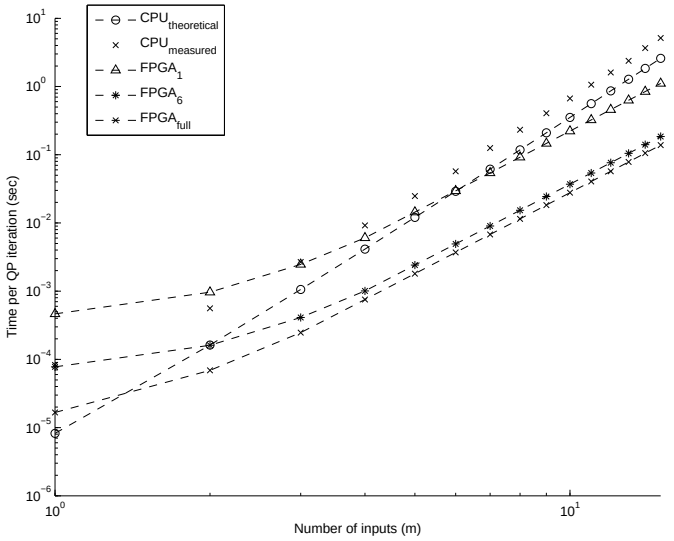

Fig. 3: Performance comparison showing theoretical and measured performance of the CPU, and FPGA performance for $2P = 1$, $2P = 6$ and P given by (4). All problem parameters scaling together ($p = m, n = 2m, T = 2n, I_{MR} = N$, FPGA$_{freq} = 150$MHz).

B. Performance

Post place-and-route results showed that a clock frequency above 150MHz is achievable with very small variations for different problem sizes, since the critical path is inside the control block in Stage 1. Figure 3 shows the latency and throughput performance of the FPGA and latency results for a microprocessor implementation. For the software benchmark, we have used a direct C sequential implementation, compiled using GCC -O4 optimizations running on a Intel Core2 Q8300 with 3.46GB of RAM, 4MB L2 cache, and a clock frequency of 2.5GHz. Note that for matrix operations of this size, this approach produces better performance software than using libraries such as Intel MKL.

The microprocessor performance was expected to be far from its peak for smaller problems, since the algorithm involves lots of movements of small blocks of data to exploit structure, especially for the operations outside the linear solver. Figure 3 shows how the performance gap between the CPU and FPGA widens as the size of the problem increases as a result of increased parallelism in the linear solver. Approximately a 5x improvement in latency and a 40x improvement in throughput are possible for large problems.

V. Conclusion

This paper has described a parameterizable FPGA architecture for solving QP optimization problems in linear MPC. The main source of acceleration is a parallel linear solver block, which reduces the latency of the main computational bottleneck in the optimization method. Results show that a significant reduction in latency is possible compared to a sequential software implementation, which translates to high sampling frequencies and better quality control. The potential for industrial take-up of this technology is currently being explored with our partners.

References

[1] J. Maciejowski, *Predictive Control with Constraints*. Harlow, UK: Pearson Education, 2001.

[2] 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.

[3] B. Leung, C.-H. Wu, S. O. Memik, and S. Mehrotra, "An interior point optimization solver for real time inter-frame collision detection: Exploring resource-accuracy-platform tradeoffs," in *Proc. Int. Conf. on Fiel Programmable Logic and Applications*, Milano, Italy, Sep 2010, pp. 113–118.

[4] 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 Conference*, Minneapolis, USA, Jun 2006, pp. 1912–1917.

[5] G. Knagge, A. Wills, A. Mills, and B. Ninnes, "ASIC and FPGA implementation strategies for model predictive control," in *Proc. European Control Conference*, Budapest, Hungary, Aug 2009.

[6] S. Bayliss, C. S. Bouganis, and G. A. Constantinides, "An FPGA implementation of the Simplex algorithm," in *Proc. Int. IEEE Conf. on Field Programmable Technology*, Bangkok, Thailand, Dec 2006, pp. 49–55.

[7] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.

[8] S. J. Wright, "Applying new optimization algorithms to model predictive control," in *Proc. Int. Conf. Chemical Process Control*. CACHE Publications, 1997, pp. 147–155.

[9] D. Boland and G. A. Constantinides, "An FPGA-based implementation of the MINRES algorithm," in *Proc. Int. Conf. on Field Programmable Logic and Applications*, Heidelberg, Germany, Sep 2008, pp. 379–384.

[10] (2010) Core Generator guide. Xilinx. [Online]. Available: http://www.xilinx.com/itp/xilinx6/books/docs/cgn/cgn.pdf

[11] (2010) Virtex-6 family overview. Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf