To be presented at ACC'06, June.

A FPGA Implementation of Model Predictive Control*

K.V. Ling, S.P. Yue and J.M. Maciejowski

***Abstract*— With its natural ability in handling constraints, Model Predictive Control (MPC) has become an established control technology in the petrochemical industry, and its use is currently being pioneered in an increasingly wide range of process industries. It is also being proposed for a range of higher bandwidth applications, such as ships, aerospace and road vehicles. To extend its applications to miniaturized devices and/or embedded systems, this paper explores the implementation of the MPC technology into reconfigurable hardware such as a FPGA chip. A rapid prototyping environment suitable for exploring the various implementation issues to bring MPC onto a chip is described. Tests were conducted to verify the applicability of the "MPC on a Chip" idea. It is shown that a modest FPGA chip could be used to implement a reasonably sized constrained MPC controller.**

Keywords: predictive control, constrained optimization, reconfigurable hardware, FPGA.

I. INTRODUCTION

Model Predictive Control (MPC) has become an established control technology in the petrochemical industry. Its use is currently being pioneered in an increasingly wide range of high bandwidth applications, such as ships [12], aerospace [11] [14], and road vehicles [10]and microscale devices[3].

Fundamentally, MPC can be formulated as a quadratic programming (QP) problem. It thus has the natural ability to handle physical constraints arising in industrial applications. Alternatively called receding horizon control, MPC computes optimal current and future control inputs by minimizing the difference between set-points and future outputs predicted from a given plant model. Then only the optimal current input is applied to the plant and this procedure is repeated at the next sampling instance.

Two important factors determines a successful MPC applications. First, is the availability of a suitable plant model. The second, is the ability to solve the quadratic programming problem within the prescribed sampling period. The ability to solve the QP problem online become critical when applying MPC to complex systems with fast response time and/or embedded applications where computational resource may be limited. In addition, there would be a need for a scalable and low-cost embedded control solution for "lab-on-chip" devices on which the number of actuators and sensors could be large. A time-multiplexed version of MPC was recently proposed to address the scalability issue in applying MPC to such situation[8].

In the last decade, reconfigurable hardware is becoming a promising alternative to both ASIC and general purpose off-the-shelf processors for embedded applications[2], [4]. As a reconfigurable hardware, Field Programmable Gate Array, or FPGA, is gaining popularity. FPGA-based systems have been applied in applications ranging from signal processing, image processing, to network processors and robotics, just to name a few (see e.g., Andraka [1] and Tessier [15]). A low precision, Logarithmic Number System (LNS) based microprocessor architecture for embedded MPC has also been explored[6]. FPGA has better flexibility and shorter design cycle than ASIC.

In this paper, the encapsulation of the constrained MPC algorithms as suitable modules for embedded control is investigated. A Handel-C model of the MPC algorithm was created which could be synthesized and implemented as FPGA module. This allows us to investigate time-area trade-off in implementing embedded MPC on FPGA. In Section 2, we review the constrained MPC problem formulation and its solution. A rapid prototyping environment for developing the MPC algorithm for FPGA implementation is described in Section 3. The resulting MPC on a chip is demonstrated in Section 4 through a simulated aircraft control example. Finally, Section 5 concludes this paper.

II. REVIEW OF CONSTRAINED MPC AND ITS SOLUTION

Constrained MPC can be formulated as a QP problem. Given a discrete linear time-invariant plant in the state space form,

$$\Sigma : \left\{ \begin{array}{lcl} x(k+1) & = & Ax(k) + B\Delta u(k) \\ y(k) & = & Cx(k), \end{array} \right. \tag{1}$$

where $y(k) \in \mathrm{R}^p$, $u(k) \in \mathrm{R}^m$ and $x(k) \in \mathrm{R}^n$ represent its system outputs, inputs and internal states respectively, the constrained MPC problem is to minimize the cost function,

$$f = \sum_{j=1}^{N_\mathrm{p}} \|y(k+j) - \omega(k+j)\|^2 + \sum_{j=0}^{N_\mathrm{u}-1} \|\Delta u(k+j)\|^2$$

subject to linear inequality constraints on the system outputs, inputs and states. Here, ω is the set-point; N_p and N_u are the prediction and control horizons respectively.

For such a QP problem, Rao et. al. [13] formulated it in a sparse form containing both equality and inequality constraints. Block factorization was used to handle the

*This research was supported by A*STAR project "Model Predictive Control on a Chip" (Ref: 022-106-0044). K.V. Ling and S.P. Yue are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, ekvling@ntu.edu.sg. J.M. Maciejowski is with the Cambridge University Engineering Department, United Kingdom, jmm@eng.cam.ac.uk

resulting banded sparse matrix within the interior point method. According to ([9], page 93), although there are computational advantages in factorising a banded matrix with a fixed bandwidth compare to a dense matrix of the same size, this comparison should be done more carefully for any particular applications. This is because the outcome of the comparison would be affected by many factors such as how many constraints that are likely to be active typically. Here, we adopted a compact formulation by eliminating the equality constraints of (1) through replacing predicted system output $y(k+j)$ as,

$$\vec{y} = \Psi_{\mathrm{x}} x(k) + \Psi_{\mathrm{u}} z, \tag{2}$$

with

$$\begin{aligned}
\vec{y} &= [\, y(k+1)^T \quad y(k+2)^T \quad \dots \quad y(k+N_p)^T \,]^T, \\
z &= [\, \Delta u(k)^T \quad \Delta u(k+1)^T \quad \dots \quad \Delta u(k+N_u-1)^T \,]^T, \\
\Psi_{\mathrm{x}} &= \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^{N_{\mathrm{p}}} \end{bmatrix}, \\
\Psi_{\mathrm{u}} &= \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N_{\mathrm{p}}-1}B & CA^{N_{\mathrm{p}}-2}B & \cdots & CA^{N_{\mathrm{p}}-N_{\mathrm{u}}}B \end{bmatrix}.
\end{aligned}$$

In this way, the constrained MPC problem is formulated as a compact QP problem

$$\Phi = \frac{1}{2} z^{\mathrm{T}} Q z + c^{\mathrm{T}} z \tag{3}$$

with inequality constraints

$$Jz \le g, \tag{4}$$

where Q is a $N_{\mathrm{u}}m \times N_{\mathrm{u}}m$ matrix and J is $m_{\mathrm{c}} \times N_{\mathrm{u}}m$ in size. Here m_{c} is the total number of inequality constraints.

Active set [5] and Interior Point method [16] are two popular methods used to solve QP problems. In this paper, we adopted the infeasible interior point method for our FPGA implementation. The basic idea of the infeasible interior point method for solving a QP problem follows from the well-known Karush-Kuhn-Tucker (or KKT) conditions,

$$Qz + J^{\mathrm{T}}\lambda = -c, \tag{5}$$

$$-Jz - t = -g, \tag{6}$$

$$\lambda \ge 0, \quad t \ge 0, \quad t^{\mathrm{T}}\lambda = 0. \tag{7}$$

Equations (5) and (6) are called feasibility conditions while the equation (7) is called complementary condition.

The infeasible interior point method perturbs the complementary condition in (7) with the following scalar

$$\mu^k = (t^k)^{\mathrm{T}} \lambda^k / m_{\mathrm{c}}, \tag{8}$$

where k is the iteration sequence and m_{c} is the number of inequality constraints in (4). When the iteration goes on, the infeasibility and μ^k are gradually reduced to zero.

According to the infeasible interior point framework introduced by Wright [16], an optimal control signal can be computed by solving the QP problem using the following algorithm:

Step 1:

Choose an initial condition (z^0, λ^0, t^0) with $(\lambda^0, t^0) > 0$.

Step 2:

At the k-th iteration step, solve for the increments $(\Delta z^k, \Delta\lambda^k, \Delta t^k)$ with

$$\begin{bmatrix} Q & J^{\mathrm{T}} \\ J & \Gamma \end{bmatrix} \begin{bmatrix} \Delta z^k \\ \Delta\lambda^k \end{bmatrix} = \begin{bmatrix} r_1^k \\ r_2^k \end{bmatrix}, \tag{9}$$

and

$$\Delta t^k = -t^k + (\Lambda^k)^{-1}(\sigma^k \mu^k e - T^k \Delta\lambda^k). \tag{10}$$

Here Q is a symmetric matrix, $\Gamma = -(\Lambda^k)^{-1}T^k$,

$$\Lambda = \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_{m_{\mathrm{c}}}^k \end{bmatrix}, \quad T = \begin{bmatrix} t_1^k & & \\ & \ddots & \\ & & t_{m_{\mathrm{c}}}^k \end{bmatrix}, \quad e = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{11}$$

In addition,

$$\begin{aligned}
r_1^k &= -Qz^k - J^{\mathrm{T}}\lambda^k - c, \\
r_2^k &= -Jz^k + g - \sigma^k \mu^k (\Lambda^k)^{-1} e.
\end{aligned} \tag{12}$$

Step 3:

Increment the variables by

$$(z^{k+1}, \lambda^{k+1}, t^{k+1}) = (z^k, \lambda^k, t^k) + \alpha^k (\Delta z^k, \Delta\lambda^k, \Delta t^k), \tag{13}$$

for some $\alpha^k \in (0, 1]$ subject to $(\lambda^{k+1}, t^{k+1}) > 0$.

Step 4:

Judge the convergence. If the iterations converges, stop the process and the optimal control z^{k+1} is obtained; otherwise, go back to Step 2 with $(z^{k+1}, \lambda^{k+1}, t^{k+1})$ and continue the iteration process.

Remark 2.1: It can be seen that solving such a QP problem is an iterative process and the main computational load is in solving equation (9) at each iteration.

Equation (9) can be solved either as

$$\begin{cases} \Delta\lambda^k &= (\Gamma - JQ^{-1}J^{\mathrm{T}})^{-1}(r_2^k - JQ^{-1}r_1^k) \\ \Delta z^k &= Q^{-1}r_1^k - Q^{-1}J^{\mathrm{T}}\Delta\lambda^k, \end{cases} \tag{14}$$

or

$$\begin{cases} \Delta z^k &= (Q - J^{\mathrm{T}}\Gamma^{-1}J)^{-1}(r_1^k - J^{\mathrm{T}}\Gamma^{-1}r_2^k) \\ \Delta\lambda^k &= \Gamma^{-1}r_2^k - \Gamma^{-1}J\Delta z^k. \end{cases} \tag{15}$$

Both schemes need to invert a matrix. For (14), the matrix to be inverted at each iteration is $\Gamma - JQ^{-1}J^{\mathrm{T}}$, which has dimensions m_{c} by m_{c}. In (15), the matrix is $Q - J^{\mathrm{T}}\Gamma^{-1}J$, which has dimensions $N_{\mathrm{u}}m$ by $N_{\mathrm{u}}m$. $\Gamma - JQ^{-1}J^{\mathrm{T}}$ has a special structure in that only the diagonal elements change at each iteration. In most practical MPC applications, m_{c} is generally much larger than $N_{\mathrm{u}}m$. In such a situation, (15) is more attractive for our current FPGA implementation and is adopted for the rest of the paper.

Remark 2.2: With (15), (10) can also be written as

$$\Delta t^k = -t^k + g - J(z^k + \Delta z^k) \tag{16}$$

III. A PROTOTYPING ENVIRONMENT FOR MPC ON A CHIP

The main factors to be considered when implementing MPC on reconfigurable hardware include computational speed, hardware resource usage, power consumption, etc. For a particular application, specific requirements on these factors need to be met and the final implementation is usually a compromise between all these factors. Hence, an effective and efficient rapid prototyping environment which allows for experimentation and verification of various algorithm configurations, architecture and implementation schemes would be useful. The tools we employed in achieving our "MPC on a chip" includes a RC10 FPGA prototyping board, the DK Design Suite from Celoxica and Matlab/Simulink software from Mathworks.

The core of RC10 is a Xilinx Spartan-3L (XC3S1500L-4-fg320) FPGA chip which has 1.5 million logic gates and some useful on-chip resources such as multiplier and on-chip memory. Celoxica DK design suite is an integrated environment for FPGA implementation using the Handel-C programming language. It provides a complete tool set which includes a compiler, a debugger, an optimizer and a simulator. MATLAB/Simulink provides an excellent platform for plant modelling, MPC algorithm design and simulation, Handel-C/MATLAB co-simulation and hardware-in-the-loop verification.

Handel-C is a high level FPGA implementation language with an ANSI C syntax and some hardware related language features such as parallel execution, channel communication, interface definition, etc. Compared with other hardware description languages such as VHDL or Verlog, Handel-C is more convenient for rapid prototyping of control-centric algorithms.

The main procedure of prototyping of our "MPC on a Chip" design is illustrated in Figure 1.

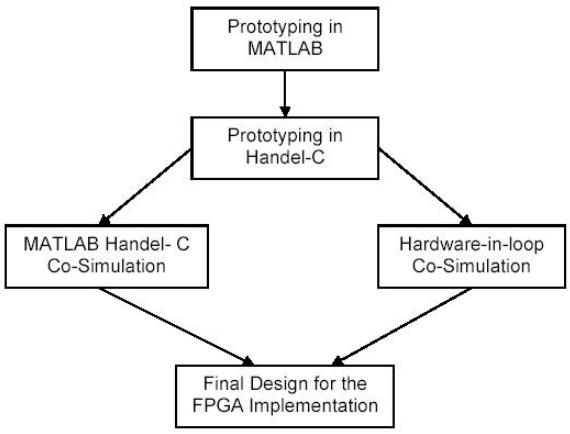

Fig. 1. Prototyping of MPC on a Chip

In the following, we briefly describe the main steps in prototyping MPC into a FPGA implementation.

Step 1: Prototyping in MATLAB Code
MATLAB provides an excellent computation and simulation environment for designing and implementing control algorithms. The MPC algorithm is first prototyped in MATLAB code and then simulated and verified in the MATAB/SIMULINK environment.

Step 2: Prototyping in Handel-C Code
The prototype MPC in the form of MATLAB code is translated into Handel-C code for FPGA realisation. The code is then compiled and optimized in the DK design suite. It is mapped, placed and routed by Xilinx ISE to a target FPGA. The Xilinx tool would report hardware resource usage and timing performance. If the results do not meet the specified requirements, design iterations would need to be carried out.

Step 3: Handel-C/MATLAB Co-Simulation
Two options are available for algorithm verification: software or hardware verification. For software verification, the Handel-C code will be packaged into a DLL file and then be called by Simulink as a S-function (see Fig. 2).

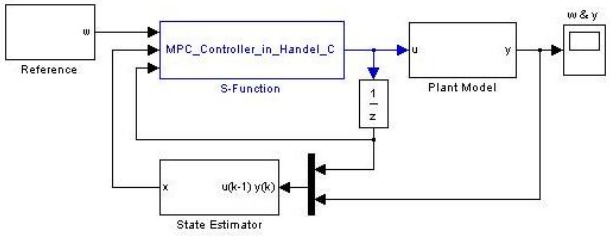

Fig. 2. Simulink/Handel C Co-Simulation

Step 4: Hardware-in-the-loop Verification
For hardware verification, the Handel-C code will be compiled into a bitstream file which will subsequently be downloaded onto the FPGA on the RC10 prototyping board to perform the MPC calculations. A test suite can then be written to verify the MPC implementation on the FPGA. Test data and test results can be transferred between MATLAB on the PC and FPGA on the RC10 board through the RS232 serial link (see Fig. 3). Table I shows a typical MATLAB test suite. If there is any error in the hardware-in-the-loop verification, it would be trapped and investigated.

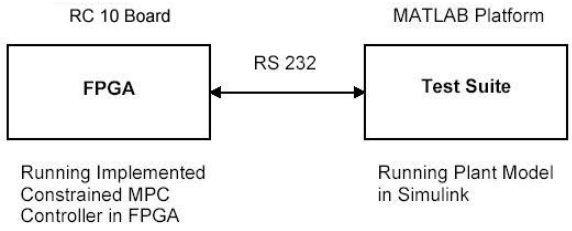

Fig. 3. Hardware-in-the-loop Verification

A. IMPLEMENTING MPC ON FPGA

Although FPGA implementation of MPC is highly application dependent, there exists some common core components. The first, is an efficient floating point library (available in the DK design suite). Next, is a matrix inversion core.

TABLE I
A SAMPLE TEST SUITE FOR FPGA IMPLEMENTATION OF CONSTRAINED MPC

```
%-------------------------------------------------
clear all

TOL = 1e-3; %acceptable accuracy
N = 50;     %how many test
n = 6;      %size of QP
mc = 32;
...

for i = 1:N
  % generate a QP program randomly
  H = rand(n,n); Q = H'*H;
  ...

  u_qp = quadprog(Q,c,J,g); %use MATLAB's QP solver

  disp('Download program and data to RC10 board...')
  for i=1:n
    for j=1:n, fwrite(s,Q(i,j),'single'); end
  end
  ...

  disp('Read back results from RC10 board...')
  for i=1:n, u_RC10(i) = fread(s,1,'single'); end
  ...

  err = max(abs(u_qp-u_ip));

  if (err > TOL)
    disp('-----Error!----'), break
  end
end
%-------------------------------------------------
```

Our experience showed that a 1.5 million gates Spartan-3L FPGA could easily handle a 128x128 matrix inversion problem with IEEE single precision floating point arithmetic (8-bit exponent and 23-bit mantissa)[7].

When using the interior point method to solve the constrained MPC problem, the solution depends on the precision of the floating point arithmetic and the criteria used in the convergence test.

IV. TESTING THE FPGA-MPC ON AN AIRCRAFT EXAMPLE

To verify our FPGA implementation, we tested it using the Cessna Citation 500 aircraft model from [9], p.64. It has the following continuous-time state space form

$$A = \begin{bmatrix} -1.2822 & 0 & 0.98 & 0 \\ 0 & 0 & 1 & 0 \\ -5.4293 & 0 & -1.8366 & 0 \\ -128.2 & 128.2 & 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} -0.3 \\ 0 \\ -17 \\ 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -128.2 & 128.2 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{17}$$

The model has the elevator angle (rad) as its input, and the pitch angle (rad), altitude (m) and altitude rate (m/s) as outputs. The elevator angle is limited to $\pm 15^{\mathrm{o}}$ (± 0.262 rad), and the elevator slew rate is limited to $\pm 30^{\mathrm{o}}$/s (± 0.524 rad/s). These are limits imposed by the equipment design and cannot be exceeded. For passenger comfort the pitch angle is limit to $\pm 20^{\mathrm{o}}$(± 0.349 rad).

A MPC controller was designed with a sampling interval of 0.5s, $N_{\mathrm{p}} = 10$, $N_{\mathrm{u}} = 3$. The following constraints,

$$|u| \leq 0.262, \quad |\Delta u| \leq 0.524, \quad |y_1| \leq 0.349.$$

were also included. This translates to a QP problem of 3 unknowns with 60 constraints that the FPGA has to compute on-line at every sampling time.

We implemented our MPC using IEEE single precision arithmetic (8-bit exponent and 23-bit mantissa) using a Xilinx FPGA. The aircraft model was simulated in MATLAB/ SIMULINK on a PC and controlled by the FPGA implementation of MPC on the RC10 board. The controller and plant interacted through the RS232 serial link.

Fig. 4 shows the response to a step change of 40m in the altitude set-point. For this magnitude of change none of the constraints are active.

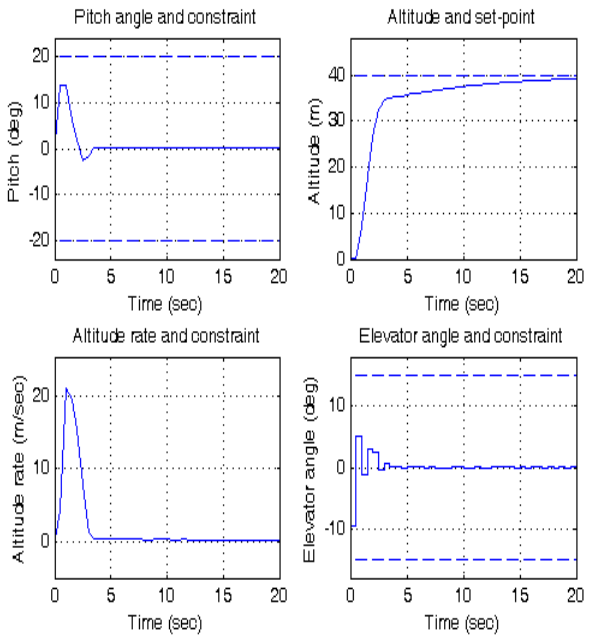

Fig. 4. Scenario 1: Response to 40m step change in altitude set-point.

Next, the set point for altitude was set to 400m, corresponds to Fig.2.7 in [9]. In our work, we found that solving the QP problem as in (3) and (4) sometimes gave incorrect results. To overcome this problem, we re-scaled the QP and wrote it as

$$f(z) = \frac{1}{2} z^{\mathrm{T}} \tilde{Q} z + \tilde{c}^{\mathrm{T}} z \tag{18}$$

and

$$\tilde{J} z \leq \tilde{g}, \tag{19}$$

with

$$\tilde{Q} = \alpha Q, \quad \tilde{c} = \alpha c, \quad \tilde{J} = \beta J, \quad \tilde{g} = \beta g. \tag{20}$$

where α and β are scalar constants. The introduction of α and β do not change the solution of the original QP problem. However, our experience showed that, by scaling the elements of Q, c, J and g matrices to a range of ± 1, it was useful in obtaining accurate solutions of QP using the interior point method (see Fig. 5).

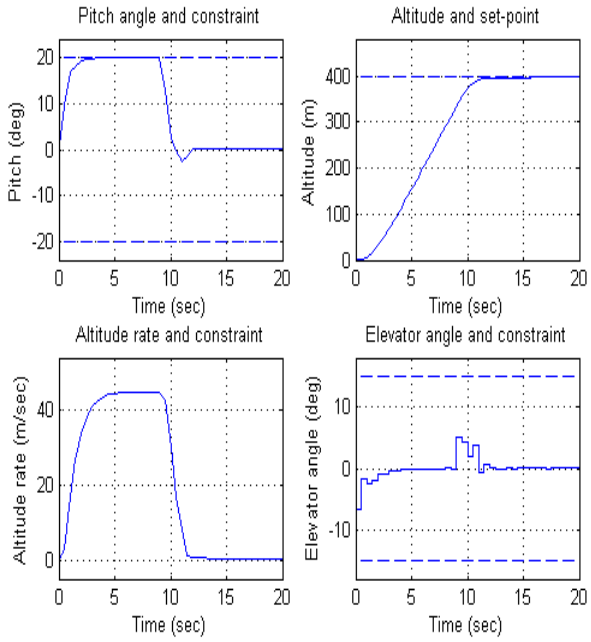

Fig. 5. Scenario 2: Response to 400m step change in altitude set-point.

Next, a constraint on the altitude rate with a limit of 30m/s was introduced, i.e. another 20 constraints were added to the original QP problem. Fig. 6 shows the results with the added constraint.

Fig. 7, a disturbance was introduced at time 5 seconds, on the altitude rate for a duration of 5 seconds and an amplitude of 5 m/sec. The figure showed that this implementation of MPC is able to handle such disturbance.

The hardware resources used to implement the constrained MPC algorithm on a FPGA chip is shown in Table II. About 30% of the Look-Up-Table (LUT) on the FPGA chip were used. Note that in this implementation, all vector-matrix computation were carried out sequentially. We have not exploited the possibility of pipelining and parallel processing which can be implemented on the FPGA.

Table IV list the performance index of our "MPC on a Chip" implementation. In the four scenarios tested, the MPC calculation can be completed in about 20 milliseconds.

TABLE II

FPGA RESOURCE USAGE WITH (8,23) FLOATING POINT ARITHMETIC

Slice	LUT	FF	Block Memory	System Clock
4565 (34%)	8451 (31%)	1860 (6%)	19 (59%)	20MHz

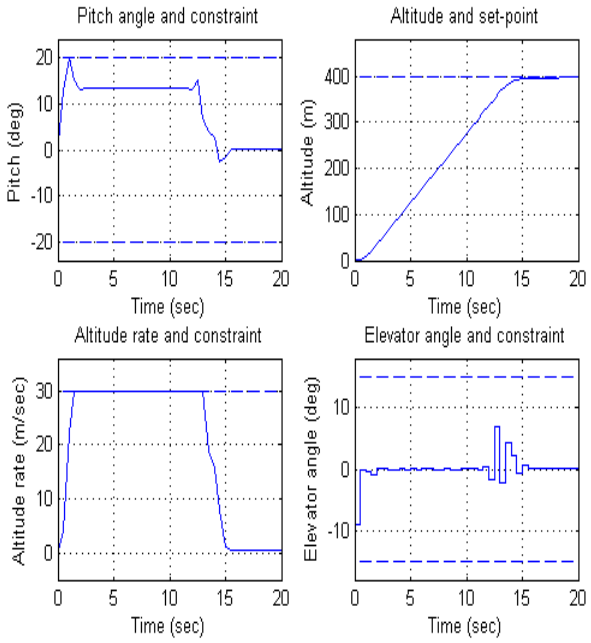

Fig. 6. Scenario 3: Response to 400m step change in altitude set-point, with altitude rate constraint.

TABLE III

"MPC ON A CHIP" PERFORMANCE (20MHZ CLOCK)

Control Scenario	Average number of Interior Point Iterations	Average number of clock cycles per sample	Average execution time (msec) per sample
1	6.6	398,920	19.9
2	8.6	475,133	23.7
3	8.3	368,527	18.4
4	8.4	390,229	19.5

V. CONCLUSIONS

In this paper, we explored the implementation of constrained MPC algorithm using a FPGA chip. Interior point method, with dense matrix formulation, was employed to solve the resulting QP problem. A rapid prototyping environment suitable for exploring the various implementation issues to bring MPC onto a FPGA chip was described. Simulation tests were conducted to verify the applicability of the "MPC on a Chip" idea. It was shown that a modest FPGA chip could be used to implement a reasonably sized constrained MPC controller. Further work is needed to investigate the possible parallelising of MPC computations to take advantage of the available on-chip resources on the FPGA chip. In this work, we have used Matlab/Simulink, Handel-C, Xilinx ISE, etc. to take a MPC solution from design to embedded implementation. Further effort should also be directed at achieving a higher level of automation in implementing embedded MPC technology. This would facilitate the embedded system community to explore the design space available in realizing a customized embedded MPC design.

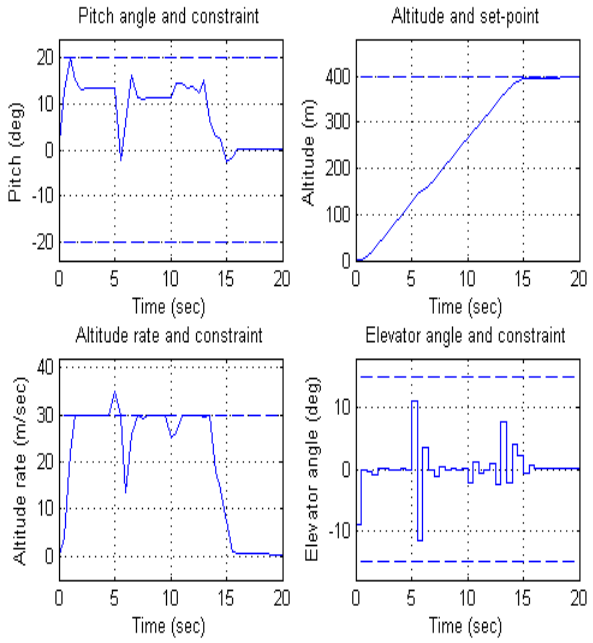

Fig. 7. Scenario 4: Response to disturbance with altitude rate constraint.

REFERENCES

[1] R. Andrak, "A survey of CORDIC algorithms for FPGA based computers", *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, March 1998, pp. 191-200

[2] M. Baleani, F. Gennari, Y. Jiang, Y. Patel, R. K. Brayton, A. Sangiovanni-Vincentelli, "System partitioning and timing analysis: HW/SW partitioning and code generation of embedded control applications on a reconfigurable architecture platform", *Proceedings of the tenth international symposium on Hardware/software codesign*, May 2002, pp.151-156.

[3] L. Bleris, J. Garcia and M. Kothare, Model Predictive Hydrodynamic Regulation of Microflows, American Control Conference, Portland, OR, June 8-10, 2005, pp.1752-1757.

[4] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software", *ACM Computing Surveys (CSUR)*, Vol. 34, Issue 2, June 2002, pp.171-210.

[5] R. R. Fletcher, *Practical Methods of Optimization*, Wiley, 2nd Edition, 1987.

[6] J. G. Garcia, M. G. Arnold, L. G. Bleris and M. V. Kothare. LNS architectures for embedded model predictive control processors. In 2004 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, pp.7984, Washington, D.C., September 2004.

[7] M. H. He and K. V. Ling, Model Predictive Control on a Chip, The 5th International Conference on Control & Automation, Budapest, Hungary, June 26-29, 2005.

[8] K. V. Ling, J. M. Maciejowski and B. F. Wu, Multiplexed Model Predictive Control, 16th IFAC World Congress, Prague, July 2005.

[9] J. M. Maciejowski, *Predictive Control with Constraints*, Prentice Hall, 2002.

[10] M. Morari, M. Baotić and F. Borrelli, "Hybrid systems modeling and control", *European Journal of Control*, Vol. 9, 2003, pp.177-27.

[11] R.M. Murray, J. Hauser, A. Jadbabie, M.B. Milam, N. Petit, W.B. Dunbar and R. Franz, "Online control customization via optimization-based control", In: *Software-Enabled Control* (T. Samad and G. Balas, Eds.). IEEE Press and Wiley, 2003.

[12] T. Perez, G.C. Goodwin and C.W. Tzeng, Model predictive rudder roll stabilization control for ships", In: *Proc. 5th IFAC Conf. on Manoeuvring and Control of Marine Craft*, Aalborg, Denmark, 2000.

[13] C. V. Rao, S. J. Wright and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications* , Vol. 99, 1998, pp. 723-757.

[14] A. Richards and J.P. How., "Model predictive control of vehicle maneuvers with guaranteed completion time and robust feasibility", In: *Proc. American Control Conference*, Denver, 2003.

[15] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: a survey", *Journal of VLSI Singnal Processing*, Vol. 28, 2001, pp. 7-27

[16] S. J. Wright, "Applying new optimizaiton algorithms to model predictive control," *Chemical Process Control-V*, CACHE, AIChE Symposium Series No. 316, Vol. 93, 1997, pp. 147-155.