Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Embedded Model Predictive Control (MPC) using a FPGA[1]

K.V. Ling*, B.F. Wu* and J.M. Maciejowski**

**School of Electrical and Electronic Engineering,*
Nanyang Technological University,
Singapore 639798 (e-mail:ekvling@ntu.edu.sg)
***Cambridge University Engineering Department,*
United Kingdom (e-mail: jmm@eng.cam.ac.uk)

Abstract: Model Predictive Control (MPC) is increasingly being proposed for application to miniaturized devices, fast and/or embedded systems. A major obstacle to this is its computation time requirement. Continuing our previous studies of implementing constrained MPC on Field Programmable Gate Arrays (FPGA), this paper begins to exploit the possibilities of parallel computation, with the aim of speeding up the MPC implementation. Simulation studies on a realistic example show that it is possible to implement constrained MPC on an FPGA chip with a 25MHz clock and achieve MPC implementation rates comparable to those achievable on a Pentium 3.0 GHz PC.

1. INTRODUCTION

Model Predictive Control (MPC) has become an established control technology owing to its capability of solving problems with physical constraints. Its applications were originally in the petrochemical industry, but its use has now been proposed for a variety of industries, including high bandwidth applications, such as ships (Perez et al., 2000), aerospace (Richards et al., 2003), road vehicles (Morari et al., 2003) and also in micro scale devices (Bleris et al., 2005).

Basic MPC solves a quadratic programming (QP) or a linear programming (LP) problem. Its successful application depends on the ability to generate a feasible solution within one sampling period. One approach to achieving MPC for high speed embedded applications which are constrained by computational time is through hardware acceleration of the MPC implementation.

Reconfigurable hardware such as Field Programmable Gate Array (FPGA) is a promising platform for deploying embedded MPC solutions to applications which require flexibility and short design cycle. The encapsulation of the constrained MPC algorithms as suitable modules for embedded control has been investigated in (Ling et al., 2006), where a *Handel-C* implementation of an MPC algorithm was described and realized on a modest *Xilinx Spartan* 3L(XC3S1500L-4-fg320) FPGA. The computation carried out in that FPGA design was sequential without exploiting the parallel computing capability of an FPGA. This paper focuses on improving on our earlier work by exploiting the opportunities for parallel computation.

The paper is organized as follows: A review of the constrained MPC formulation and the Interior Point algorithm for solving the resulting Quadratic Programming (QP) problem is presented in Section 2. The prototyping environment for this study, as well as an illustration of how the *Handel-C* language could be used to explore opportunities for parallel computations of the MPC algorithm through better use of the available FPGA resources, is briefly described in Section 3. The speed up improvement of the new implementation is verified through an aircraft simulation example in Section 4.

2. REVIEW OF CONSTRAINED MPC AND THE INTERIOR POINT METHOD

2.1 Review of Constrained MPC

Constrained MPC can be formulated as a QP problem. For simplicity, given a single input single output discrete linear time-invariant plant in the state space form,

$$\Sigma : \begin{cases} x(k+1) = Ax(k) + B\Delta u(k) \\ \quad y(k) = Cx(k) \end{cases} \tag{1}$$

where $y(k), u(k),$ and $x(k)$ are the system output, input and internal states respectively. The constrained MPC problem is to minimize the cost function,

$$f(\Delta u) = \sum_{j=1}^{N_P} \left\| y(k+j) - \omega(k+j) \right\|^2 + \sum_{j=1}^{N_u - 1} \left\| \Delta u(k+j) \right\|^2$$

subject to linear inequality constraints on the system outputs, inputs and states. Here, ω is the set-point; $\mathrm{N_P}$ and $\mathrm{N_u}$ are the prediction and control horizons respectively.

We follow the standard approach (eg Rao et al. 1998, Maciejowski 2002) of eliminating the equality constraints

[1] This work was supported in part by the A*STAR grant reference: 052-118-0059.

of (1), replacing the predicted system output $y(k+j)$ as follows:

$$\vec{y} = \Psi_x x(k) + \Psi_u z \tag{2}$$

with

$$\vec{y} = \begin{bmatrix} y(k+1) & y(k+2) & \ldots & y(k+N_P) \end{bmatrix}^T,$$

$$z = \begin{bmatrix} \Delta u(k) & \Delta u(k+1) & \ldots & \Delta u(k+N_u-1) \end{bmatrix}^T,$$

$$\Psi_x = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^{N_P} \end{bmatrix},$$

$$\Psi_u = \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N_P-1}B & CA^{N_P-2}B & \cdots & CA^{N_P-N_u}B \end{bmatrix}$$

The constrained MPC problem reduces to a QP problem where the cost function

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

is minimised subject to inequality constraints

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

Here, Q is a symmetric $N_u \times N_u$ matrix and J is $m_c \times N_u$ in size, where m_c is the total number of inequality constraints. The infeasible Interior Point (IP) method (Rao et al, 1997) was adopted for solving the QP problem and implemented in a FPGA. The IP method can be derived from the well-known Karush-Kuhn-Tucker (or KKT) conditions, which are necessary and sufficient conditions for characterising the global optimum, providing that $Q \ge 0$:

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

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

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

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

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

$$\mu_k = (t_k)^T \lambda_k / m_c \tag{8}$$

where k is the iteration counter and m_c is the number of inequality constraints in (4). As the iteration proceeds, the infeasibility and μ_k are gradually reduced to zero.

2.2 The Interior Point Method

According to the infeasible interior point framework introduced by (Wright, 1997), the QP can be solved iteratively as follows:

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, solve for the increments

$(\Delta z_k, \Delta\lambda_k, \Delta t_k)$ with

$$\begin{bmatrix} Q & J^T \\ J & \Gamma_k \end{bmatrix}\begin{bmatrix} \Delta z_k \\ \Delta\lambda_k \end{bmatrix} = \begin{bmatrix} r_k \\ s_k \end{bmatrix}, \tag{9}$$

and

$$\Delta t_k = -t_k + (\Lambda_k)^{-1}(\sigma\mu_k e - T_k \Delta\lambda_k). \tag{10}$$

Here $\Gamma_k = -(\Lambda_k)^{-1} T_k$,

$$\Lambda_k = \begin{bmatrix} \lambda_k(1) & & \\ & \ddots & \\ & & \lambda_k(m_c) \end{bmatrix}, T_k = \begin{bmatrix} t_k(1) & & \\ & \ddots & \\ & & t_k(m_c) \end{bmatrix}, e = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{11}$$

In addition,

$$\begin{aligned} r_k &= -Qz_k - J^T \Lambda_k e - c, \\ s_k &= -Jz_k + g - \sigma\mu_k (\Lambda_k)^{-1} e \end{aligned} \tag{12}$$

Step 3: Increment the variables according to:

$$(z_{k+1}, \lambda_{k+1}, t_{k+1}) = (z_k, \lambda_k, t_k) + \alpha_k(\Delta z_k, \Delta\lambda_k, \Delta t_k) \tag{13}$$

for some $\alpha_k \in (0, 1]$, subject to $(\lambda_{k+1}, t_{k+1}) > 0$.

Step 4: Judge the convergence. If the iterations have converged, stop the iterations; the optimal control sequence is obtained as z_{k+1}. Otherwise, go back to Step 2 with $(z_{k+1}, \lambda_{k+1}, t_{k+1})$ and continue iterating. $\square$

It is clear that Step 2 of the interior point algorithm is where most of the computation occurs and hence optimizing the computational efficiency of step 2 should be the most fruitful. In most embedded MPC application, the number of constraints m_c is usually much larger than the number of variables N_u – note that the same constraint at different points in the prediction horizon appears as several constraints in (4). In this case, it can be computationally advantageous to solve (9) in two stages as follows:

$$\begin{cases} (Q - J^T \Gamma_k^{-1} J)\Delta z_k & = & (r_k - J^T \Gamma_k^{-1} s_k) \\ \Gamma_k \Delta\lambda_k & = & s_k - J\Delta z_k \end{cases} \tag{14}$$

Note that Γ_k is a diagonal matrix.

Using (14), (10) can also be written as

$$\Delta t_k = -t_k + g - J(z_k + \Delta z_k) \tag{15}$$

3. PROTOTYPING ENVIRONMENT FOR MPC-ON-A CHIP

The rapid prototyping environment for optimization of the "MPC on a Chip" design is shown in Fig. 1. The plant simulation is implemented in *MATLAB* while MPC is implemented either in *MATLAB* or on a FPGA prototyping board. This prototype environment provides flexibility and allows convenience in experimentation and verification of the various FPGA implementations of the MPC algorithm.

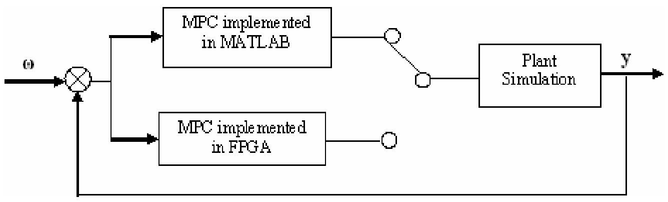

Fig. 1 Prototyping of MPC on a Chip

The tools used in this study include a RC203 FPGA board, the *DK Design Suite* from *Celoxica* and *Matlab/Simulink* software from *The Mathworks*.

The RC203 is a platform for evaluation and development of FPGA-based applications. The platform include a *Xilinx Virtex-II* FPGA, external memory, programmable clocks, Ethernet, Audio, Parallel port, RS-232 etc. The FPGA used in this study is XC2V3000, which has three million logic gates. The XC2V3000 has 96 multiplier blocks, 96 block of 18K on-chip RAM and a maximum 720 I/O pads.

For hardware implementation on an FPGA, the constrained MPC algorithm is coded in *Handel-C* which is a *C*-like language for digital logic design. Programming in *Handel-C* allows one to focus on algorithm design without worrying about how the underlying computation engine works. As an illustration, Fig.4(a) and Fig.4(b) show, respectively, fragments of the *Handel-C* code for the sequential and parallel implementation of the computation of $e = a \times b + c \times d$, where a, b, c, d are scalar single precision floating-point variables. Implementing the computation sequentially, as illustrated in Fig. 4(a), the computation would require 9 clock cycles, since in our implementation, one floating-point multiplication and addition would require 3 clock cycles each to complete.

The opportunity for parallelism depends very much on the data dependency of the computation tasks. Variables that do not have any data dependency could be computed simultaneously provided additional hardware resources are available. As illustrated in Fig. 4(b), if an additional copy of the floating point multiplier hardware is created on a FPGA, then the multiplications can be performed in parallel and the clock cycles needed are reduced to 6. During the time when the multiplications are carried out, the adder hardware is left idle and can be deployed for other computational tasks (not shown here). In our current implementation, we achieved a degree of parallelism by ad hoc programming. In particular, the execution of some FOR loops were combined and parallelised when solving (14).

The *Handel-C* implementation of the MPC algorithm was compiled into a bitstream file which was then downloaded to the RC203 board to configure the FPGA chip to perform the constrained MPC calculations. The data exchange between the PC which runs *MATLAB* and the FPGA board which carries out the MPC calculations was achieved via the serial link.

```
seq {
      myFloatMult(a,b,&tmp1);
      myFloatMult(c,d,&tmp2);
      myFloatAdd(tmp1,tmp2,&e);
      }
```

Fig 4(a) Sequential Implementation

```
par {
      myFloatMult1(a,b,&tmp1);
      myFloatMult2(c,d,&tmp2);
     }

  myFloatAdd(tmp1,tmp2,&e);
```

Fig 4(b) Parallel Implementation

4. RESULTS

4.1 Aircraft Model and Constraints

To verify the implementation of our FPGA implementation of the constrained MPC algorithm, the Cessna Citation 500 Aircraft example (Example 2.7) from Maciejowski (2002) is used. It has the following continuous-time state-space model

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

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -128.8 & 128.8 & 0 & 0 \end{bmatrix}; \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

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 ±15° (±0.262rad) and the elevator slew rate is limited to ±30°/s (±0.524rad/s). These are limits imposed by equipment design and cannot be exceeded. For passenger's comfort the pitch angle limit is limited to ±20° (±0.349rad)

A MPC controller was designed with a sampling interval of 0.5s, $N_p = 10$, and $N_u = 3$. The following constraints,

$$|u| \le 0.262, \qquad |\Delta u| \le 0.524, \qquad |y_1| \le 0.349$$

were also included.

Table 1 Comparison on Computational Time (*ms*) for Different Implementation Schemes

Scenario		Quadprog	IP	Sequential	Parallel
Case2 (32 inequality constraints)	m-function	3.3	3.9	N.A	N.A
	Stand alone executable	3.2	2.7	5.2	2.9
Case3 (52 inequality constraints)	m-function	4.2	17.6	N.A	N.A
	Stand alone executable	4	4.5	9.1	4.9

We implemented our MPC on the *Xilinx* FPGA using the IEEE single precision arithmetic (8-bit exponent and 23-bit mantissa). Two FPGA implementations of the constrained MPC algorithms were created: one implemented the computations sequentially while the other exploited the opportunities for parallel computation. The aircraft model was simulated in *MATLAB* (Version 7.1) on a *Windows XP* PC (CPU P4 3.0GHz with 1GB RAM) and controlled by the FPGA implementation of MPC on the RC203 board. The controller and plant interacted through the serial link.

4.2 Simulation Results

Simulations were conducted based on two scenarios which corresponded to Case 2 and Case 3 of (Maciejowski, 2002). These two scenarios were chosen because they resulted in QP problems with different numbers of inequality constraints: 32 for Case 2 and 52 for Case 3.

We tested several versions of the constrained MPC implementations using the prototype environment described in Section 3. In order to create a benchmark for our FPGA implementation of MPC, a version of the constrained MPC algorithm implemented in *MATLAB* using its standard `quadprog` function (which solves QP problems using the active set method) was created. Also, the interior point method described in Section 2 was coded as an *m*-function in *MATLAB*. The timing of these two implementations running in the interactive environment of MATLAB was recorded. We also used the "`mcc -m`" command available in *MATLAB Compiler* to obtain stand-alone executables of these two *MATLAB* implementations. We recorded their timings.

Table 1 compares the results. It can be seen that the parallel implementation of MPC in the FPGA was approximately twice as fast as the sequential implementation. Although the computational time of the Interior Point (IP) implementation increases with the number of inequality constraints, the computational speed achieved by exploiting parallelism using an FPGA is comparable to that of the stand alone executable running on a Pentium4 3.0GHz CPU.

Table 2 presents the resource requirements and the clock counts measured for the two FPGA implementations. The clock rate achieved was 25MHz. The clock counts needed to complete one iteration of the Interior Point (Step 2 only) was 7564 for the sequential implementation and 3088 for the parallel implementation; a saving of 4476 and 6920 in case 2 and 3 per iteration step. The reduction in the clock counts was achieved at the expense of a modest increase in the hardware resources needed for the parallel implementation: 2 additional Arithmetic Logic Units (ALU) -- which are 18x18 multipliers, 20% more Look-Up Tables (LUT), and 2% more Flip-Flops (FF). Nevertheless, it should be noted that even with these increased resource requirements; there are still unused resources on the FPGA chip that can be further exploited.

Table 2 Timing and Resources Analysis on RC203 Board with a Vertex II FPGA Chip

		Sequential	Parallel
Requested Clock Rate (MHz)		25	25
Achieved Clock Rate (MHz)		25.07	25.03
Clock Count[†]	**Case 2**	7,564	3,089
	Case 3	11,827	4,907
ALUs		3	5
LUTs		8,773 (30%)	16,338 (50%)
FFs		2,063 (7%)	2,750 (9%)
Memory Bits		497,776	379,120

The response of the aircraft for each of the simulation scenarios is shown in Figure 5(a) and 6(a), respectively. For Case 2, the 400m step change in altitude set-point resulted in the pitch angle constraint being active during the transient, while the rate of change of altitude was unconstrained. For Case 3, the rate of change of altitude was also constrained, and the simulation result showed that the altitude rate constraint is active during most of the transient while the pitch angle constraint is only active at the beginning of the transient.

Solving a constrained MPC problem is a repetitive process. At each MPC sampling instant, a new QP problem is created which requires several Interior Point iterations to obtain the solution. The number of IP iterations required varies and depends on factors such as the set of active constraints.

Figure 5(b) shows the plot of the number of IP iterations and clock counts recorded at every MPC sampling instants for

[†] For step 2 in the IP algorithm, one iteration only

Case 2. It can be seen that when the constraints become active from 4s to 9s, both the number of iterations and clock counts increase dramatically. After the transient they fall to a relatively low number. The most difficult QP problem occurred at about t=5s. Comparing the results of sequential and parallel implementations, it can be seen that although the trends are similar, the clock counts needed in the latter was reduced by nearly half. The corresponding plots of the number of iterations and clock counts for Case 3 are given in Fig. 6(b). The trends of iterations and clock counts are similar to those in Figure 5(b). For Case 3, the most difficult QP problem occurs at t=9s, which takes more than 30 IP iterations to solve. Table 3 lists the average number of Interior Point iterations (at each step) for Cases 2 and 3. Such information may be useful in determining an appropriate sample rate for an embedded MPC application.

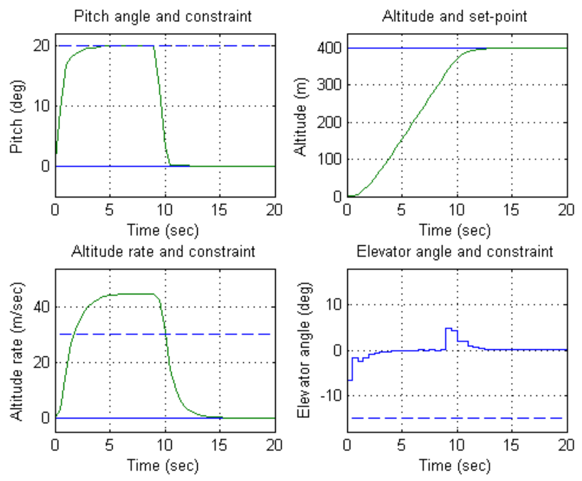

Fig. 5(a) Response of aircraft in Case 2

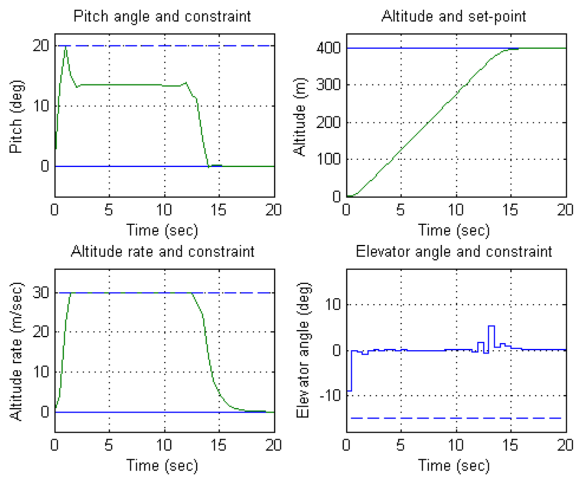

Fig. 6(a) Response of aircraft in Case 3

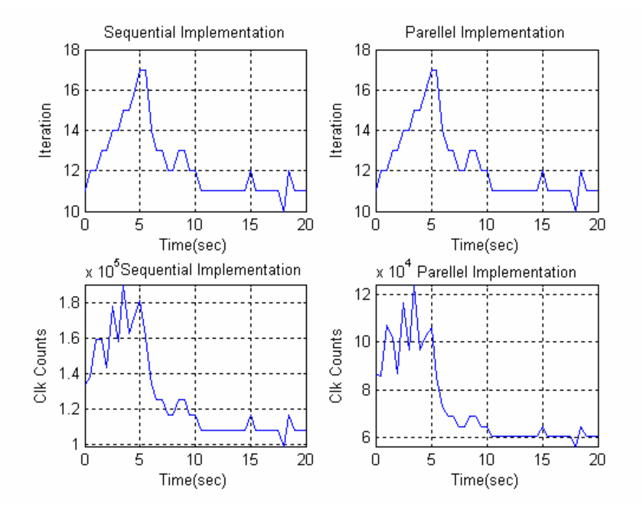

Fig. 5(b) Iterations and clock counts in step 2 of IP

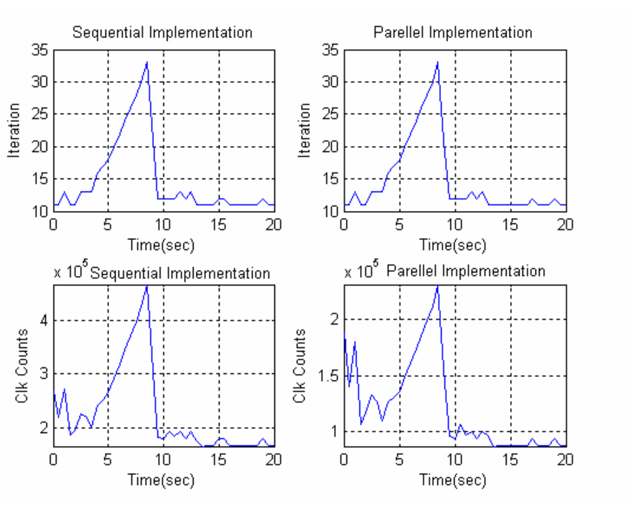

Fig. 6(b) Iterations and clock counts in step 2 of IP

Table 3 Average numbers of IP iterations per MPC step (RC203 Board with Clock Rate@25M)

Scenarios	Schemes	Average number of IP Iterations per MPC sample	Average Clock Counts per MPC sample	Average MPC Sampling Interval (msec)	Average Clock Count per IP Iteration	Average Time per IP iteration (msec)
Case 2	Sequential	12.3	130,058	5.2	10,580	0.42
	Parallel	12.3	73,512	2.9	5,980	0.24
Case 3	Sequential	14.8	226,555	9.1	15,353	0.61
	Parallel	14.7	121,299	4.9	8,234	0.33

4.3 Scalability of the current FPGA implementation

The scalability of the current FPGA implementation of the Interior Point solver is investigated by solving randomly generated QP problems parameterised by n_v and m_c, where n_v is the number of variables to be optimised and m_c is the number of inequality constraints.

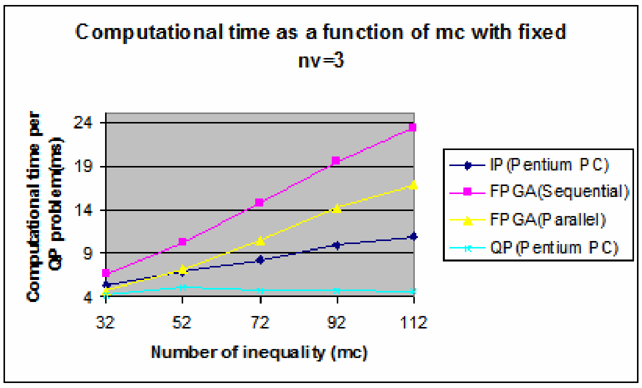

Fig. 7 Computational time as a function of m_c.

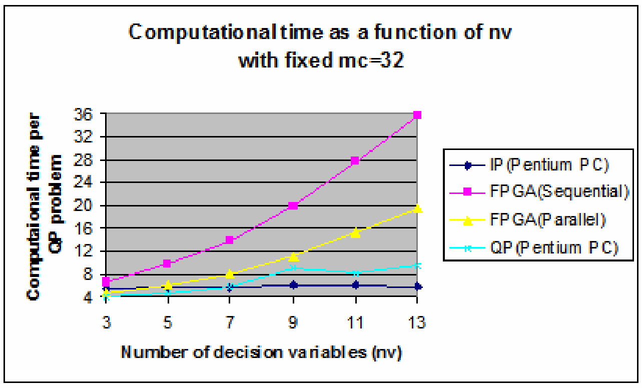

Fig. 8 Computational time as a function of n_v.

In Fig.7 the average computational time for one QP problem is plotted as a function of m_c with n_v fixed at 3. It can be seen that the computational speed appears to increase linearly with m_c, and the parallel FPGA implementation has a slope which is in between the sequential FPGA implementation and that of Pentium CPU. However, as shown in Fig. 8, when the number of optimisation variables increases, the increase in the computation time varies faster than linear and the Pentium CPU outperformed the FPGA implementation. The reason could be that the current FPGA implementation uses a straightforward Gaussian elimination method to solve the resulting Ax=b problem in Step 2 and hence less efficient in handling larger Ax=b problems. This issue is a current subject of investigation.

5. CONCLUSION

In this paper, two FPGA implementations of constrained MPC were described and evaluated. The parallel implementation outperformed the sequential implementation in terms of computation speed, at the expense of a modest increased in the hardware resource requirement.

This paper continues our investigation of the potential benefits of using FPGAs to implement MPC. There is still much detailed analysis to be done, which should indicate further possibilities for achieving faster implementations. More study of the hardware trade-offs is needed. It is noted that there are still unused hardware resource available which can be further exploited. A more systematic method to exploit these unused resources is a subject of current research. This should allow MPC to be used in application areas where the computational load has been considered too great until now, such as UAVs, automobile control systems and gas turbine control.

REFERENCES

Bleris, L., Garcia, J. and, Kothare, M. (2005). Model predictive hydrodynamic regulation of microflows. *American Control Conference,* Portland, OR, June, 1752-1757
Ling, K.V., Yue, S.P. and Maciejowski, J.M. (2006). A FPGA implementation of model predictive control. *American Control Conference,* Minneapolis, Minnesota, USA, June 14-16, 1930-1935
Maciejowski, J.M. (2002). *Predictive Control with Constraints*, Prentice Hall.
Morari, M., Baotić, M. and Borrelli, F. (2003). Hybrid systems modelling and control. *European Journal of Control*, **9**, 177-189
Perez, T., Goodwin, G.C and Tzeng, C.W. (2000). Model predictive rudder rolls stabilization control for ships, In: *Proc. 5th IFAC Conf. on Manoeuvring and Control of Marine Craft*, Aalborg, Denmark,
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**, 723-757
Richards, A. and How, J.P. (2003). Model predictive control of vehicle manoeuvres with guaranteed completion time and robust feasibility, In: *Proc. American Control Conference*, Denver.
Wright, S..J (1997). Applying new optimization algorithms to model predictive control, *Chemical Process Control-V, CACHE, AIChE Symposium,* **316(93),** 147-155.