Control Engineering Practice 20 (2012) 1229–1235

Contents lists available at SciVerse ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

Model predictive controller design and implementation on FPGA with application to motor servo system

Nan Yang [a,b], Dewei Li [a,b,*], Jun Zhang [a,b,c], Yugeng Xi [a,b]

[a] Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China
[b] Key Laboratory of System Control and Information Processing, Ministry of Education, Shanghai 200240, China
[c] Joint Institute of UM-SJTU, Shanghai Jiao Tong University, Shanghai 200240, China

A R T I C L E   I N F O

Article history:
Received 29 April 2011
Accepted 16 June 2012
Available online 18 July 2012

Keywords:
Model predictive control
FPGA
Field controller
Motor servo
Quadratic programming

A B S T R A C T

In this paper, an integrated hardware and software design method is developed to implement an MPC algorithm on an FPGA chip. This makes it possible to achieve the long-desired goal of extending MPC algorithms to field control so as to deal with constraints effectively. To expedite the numerical procedure of solving quadratic programming (QP) in the MPC algorithm, a QP solver based on embedded chips is designed to exploit the flexibility and efficiency of FPGA chips. With a carefully devised software architecture, a universal platform is proposed to be facilely deployed to field control applications. To demonstrate the efficacy, a prototype system is built based on a Xilinx FPGA chip. It is then applied to a motor servo tracking control system and achieves satisfactory control performance.

1. Introduction

Thanks to its intrinsic capability of handling constraints, model predictive control (MPC) has been widely applied as an advanced control strategy to many complex industrial processes such as petro-refining plants (Bemporad, 2006; Ordys et al., 2001; Qin & Badgwell, 2003). Nevertheless, because the MPC controller needs to solve a constrained optimization problem in real time, the resulting computational burden restricts its implementations to those systems possessing high computing powers. It is usually difficult to directly apply MPC algorithms to field controllers and many other application fields equipped with only embedded hardware devices.

The current work is focused on fully exploiting the potential of MPC algorithms to field control. To this end, two key design factors in the MPC implementation need to be considered, namely, the embedded hardware device and the accordingly tailored design scheme. Currently, there are several commercially available embedded hardware platforms for field controllers, including advanced RISC machine (ARM), digital signal processing (DSP), application-specific integrated circuit (ASIC), and field programmable gate array (FPGA). Among them, FPGA distinguishes itself with salient features in both flexibility and computational efficiency.

FPGA devices have a number of programmable logic resources that can be directly configured to perform complex computations in hardware. Moreover, recent technical progresses make it possible to integrate FPGA with microprocessors and related peripherals to form a complete embedded system. Not only it guarantees the performance of entire system, but also it makes the system design more flexible and thus can significantly reduce design cycles (Kuon & Rose, 2007). In addition, with the adroitly designed architecture including features such as pipelining and parallel computing, FPGA can achieve a much higher processing speed than software implementations (Tessier & Burleson, 2001). This makes FPGA particularly attractive for computation intensive tasks.

Implementing MPC algorithm on FPGA chips has attracted wide interests in recent years (Bleris, Vouzis, Arnold, & Kothare, 2006; He & Ling, 2005; Ling, Yue, & Maciejowski, 2006; Monmasson & Cirstea, 2007). Monmasson and Cirstea (2007) provided an overview and several general guidelines on developing industrial control systems with FPGA. Bleris et al. (2006) used a co-processor to accelerate the major part of the computations in sequential order. In Ling et al. (2006) and He and Ling (2005), a MATLAB/Handel-C co-design procedure for fast prototyping is developed to implement MPC on an FPGA chip, where an interior point method is used to solve the online optimization. Lau, Yue, Ling, and Maciejowski (2009) compared the active set method with the interior point method by implementing it on FPGA chip with Handel-C procedures, and reflected that the active set method is more efficient for problems with small scale.

This paper develops an integrated hardware and software design method to implement an MPC algorithm on FPGA platforms for field controls. As the scale of field control is normally small, the active set method is adopted. To make the design more flexible for different applications and balance the required resources,

* Corresponding author at: Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China. Fax: +86 21 34205030.
E-mail address: dwli@sjtu.edu.cn (D. Li).

0967-0661/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.conengprac.2012.06.012

with the scrutinizing of the execution sequence in solving the quadratic programming (QP) problem by active set method, a highly modularized QP solver is designed to undertake most of the computational tasks. This QP solver can be customized to deal with different problem sizes so as to make full use of the logic resources as well as parallel processing capability. Furthermore, it has the flexibility to be readily ported to other FPGA devices.

To demonstrate the feasibility and efficacy of the proposed design, a prototype MPC system using the Xilinx Virtex-4 FPGA device is applied to a motor servo tracking control system and achieves good control performance. It shows that the proposed design approach can thus be applied to a wide range of field control applications with various constraints.

This paper is organized as follows. Section 2 introduces the technical backgrounds of the MPC algorithm and active set method in the optimization theory. In Section 3, the details of the MPC algorithm implementation on FPGA is discussed. Section 4 showcases an application of the proposed design to a motor servo tracking control system. The conclusion and future works are provided in Section 5.

2. Background

This section introduces the basic ideas of MPC algorithm and active set method in the optimization theory.

2.1. MPC algorithm

MPC is an advanced control strategy that is widely applied in many process control applications. The plant model can be described by the following discrete linear system:

$$x(k+1)=Ax(k)+Bu(k), \tag{1}$$

where $x\in\mathbb{R}^n$ is the state variable, $u\in\mathbb{R}^l$ the control input, $A\in\mathbb{R}^{n\times n}$ the system matrix, and $B\in\mathbb{R}^{n\times l}$ the input matrix. At each sampling instant, MPC needs to solve the following optimization problem (Morari & Lee, 1999):

$$\min_{U(k)} J=\frac{1}{2}x^T(k+p)P_f x(k+p)+\sum_{i=1}^{p-1}x^T(k+i)Q_i x(k+i)+\sum_{i=0}^{m-1}u^T(k+i)R_i u(k+i)$$
$$\text{subject to}\quad EX(k)+FU(k)\le b, \tag{2}$$

where p is the length of the prediction horizon, m the length of the control horizon, $Q_i\in\mathbb{R}^{n\times n}$ the weighting matrix on state variables, $R_i\in\mathbb{R}^{l\times l}$ the weighting matrix on control inputs, $P_f\in\mathbb{R}^{n\times n}$ the weighting matrix on the terminal state, and

$$X(k)=[x^T(k+1),\dots,x^T(k+p)]^T\in\mathbb{R}^{pn}, \tag{3}$$

$$U(k)=[u^T(k),u^T(k+1),\dots,u^T(k+m-1)]^T\in\mathbb{R}^{lm}. \tag{4}$$

To reduce the computational complexity of the optimization solver, it is a common practice to adopt linear constraints on state variables and control inputs. Incorporating the model in Eq. (1), one can cast the optimization problem (2) into the following standard Quadratic Programming (QP) problem:

$$\min_{U(k)} J=\frac{1}{2}U^T(k)HU(k)+c^T(k)U(k)$$
$$\text{subject to}\quad GU(k)\le b, \tag{5}$$

where $H\in\mathbb{R}^{lm\times lm}$ is a constant matrix derived from weighting matrices P_f, Q_i and R_i, $c(k)\in\mathbb{R}^{lm}$ is composed of the state vector $x(k)$, and G is a constant matrix determined by E and F. For brevity, the detailed expressions for H, c, and G are omitted. Interested readers may refer to Rawlings (2000). Note that the optimization variable $U(k)$ for (5) is defined in Eq. (4).

MPC algorithm is implemented in a receding manner, that is, at each sampling time instant, it solves the optimization problem (5) and then applies only the first control signal $u(k)$ to the plant. At the next instant, the entire procedure is repeated in the same fashion. Depending on the size and complexity, solving the online optimization problem may require powerful and expensive computing devices, which prohibits MPC algorithms from being applied to field control.

2.2. Active set method

There are several methods available for the optimization algorithm design, including interior point, conjugate gradient, and active set. Because the MPC online optimization scale for field control is usually small, the active set method is a preferred option since it is less computationally demanding in comparison to other methods (Bartlett, Wächter, & Biegler, 2000).

The basic idea of active set method is to convert the inequality constrained QP problem into a series of equality constrained QP (ECQP) problems, and then use Lagrange method to solve these subproblems in sequence.

To conform to the convention in the optimization community, by an abuse of notation, x is used to denote the optimization variable in this subsection, which corresponds to the optimization variable $U(k)$ in Eq. (5). Let $x^{(l)}$ be an optimal solution after the l-th iteration. Define the active constraint index set as

$$I^{(l)}=\{i|g^i x=b^i\},$$

where g^i is the i-th row of the matrix G in (5), and b^i is the i-th element in b. Define the following ECQP subproblem:

$$\min\quad f(x)=\tfrac{1}{2}x^T Hx+c^T x$$
$$\text{subject to}\quad G_l x=b_l, \tag{6}$$

where G_l and b_l are composed of all the active constraints in $Gx=b$ at the l-th iteration, and they may vary from one iteration to the next. Chen (2005) proposes an approach to solve (6) by calculating

$$x^{(l+1)}=-Qc+R^T b_l, \tag{7}$$

$$\lambda^{(l)}=Rc-Sb_l, \tag{8}$$

where $\lambda^{(l)}$ is the Lagrange multiplier, and

$$Q=H^{-1}-H^{-1}G_l^T(G_lH^{-1}G_l^T)^{-1}G_lH^{-1}, \tag{9}$$

$$R=(G_lH^{-1}G_l^T)^{-1}G_lH^{-1}, \tag{10}$$

$$S=-(G_lH^{-1}G_l^T)^{-1}. \tag{11}$$

Now the active set method algorithm can be described as:

Step 1: Set $l=0$, pick an initial feasible solution $x^{(0)}$, and determine the initial active constraint index set $I^{(0)}$.

Step 2: Calculate $x^{(l+1)}$ and $\lambda^{(l)}$ from Eqs. (7) and (8).

Step 3: Let $d^{(l)}=x^{(l+1)}-x^{(l)}$. If $d^{(l)}$ is equal to 0, go to Step 4; otherwise, go to Step 5.

Step 4: If $\lambda^{(l)}\ge 0$, $x^{(l+1)}$ is the optimal solution, terminate the iteration; otherwise, find the least entry in $\lambda^{(l)}$ and then remove the corresponding constraint in $I^{(l)}$ to form a new active constraint index set $I^{(l+1)}$; let $x^{(l+1)}=x^{(l)}$, then go to Step 2.

Step 5: Search the optimal step size α_l along search direction $d^{(l)}$, and let $x^{(l+1)}=x^{(l)}+\alpha_l d^{(l)}$; find the current active constraint index set $I^{(l+1)}$, then go to Step 2.

Note that the initial feasible solution $x^{(0)}$ can be obtained by solving a Linear Programming (LP) problem. See Fletcher (1970) for details.

3. MPC design

In this section, an integrated hardware and software design method is developed to implement the MPC algorithm on a FPGA chip. The basic structure of MPC system is reviewed, then the technical details of hardware and software implementation on FPGA to achieve the design objective of high efficiency and flexibility are discussed.

3.1. Structure of MPC control system

A typical MPC system has the following components as shown in Fig. 1:

Data acquisition: Collect the output data from the plant.

Pre-processing: Prepare the data to feed into the optimization solver.

Optimization: Solve the online optimization problem (5).

Post-processing: Retrieve the control signal from the optimal solution.

Actuation: Apply the control to the plant.

Among these functional blocks, the hardware realization of data acquisition, pre-processing, post-processing, and actuation may vary depending on the specific application. In some applications, additional blocks may be required, *e.g.*, state observer in output feedback systems.

Moreover, a flexible MPC design is needed to enable easy customization to different applications. To satisfy these demands, an integrated structure is developed to take full advantage of FPGA. This structure consists of a QP solver that can significantly shorten the computational time, and a microprocessor with embedded software that controls the work flow and provides an interface to adjust parameters of the MPC controller if necessary. With this integrated structure, only the software part needs to be modified to meet the requirements of different control tasks and environments, thus the workload for redesigning and deploying the controller to various fields is greatly reduced.

The scheme of MPC implementation on FPGA is shown in Fig. 2. For the hardware part, an ECQP solver that calculates the optimal solution and Lagrange multiplier of (7)–(11) at each sampling instant is carefully designed to cooperate with embedded software to construct the QP solver. The FPGA microprocessor runs the embedded software, whereas the memory controllers connect external memory chips. All the components are connected via FPGA internal bus. Furthermore, the QP solver should be able to handle different problem sizes so as to make the design flexible.

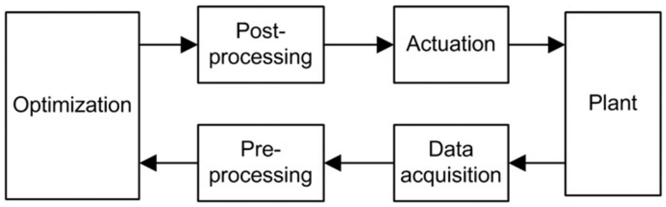

Fig. 1. Block diagram of MPC system.

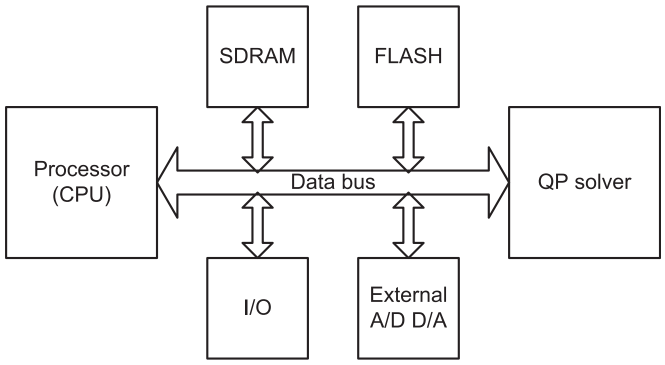

Fig. 2. Scheme of MPC implementation on FPGA.

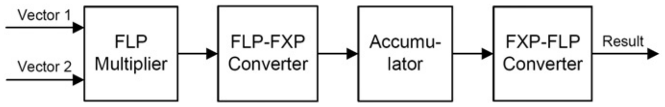

Fig. 3. Pipeline structure of vector multiplier.

The software part is used to determine the stop criterion, update the active constraint index set, and control the optimization flow. The control parameters are stored in a dedicated storage, so they can be changed on-the-fly.

3.2. QP solver design

This subsection presents the QP solver that is implemented by the logic resources of FPGA. This is one of the major contributions of the current paper.

The main calculations in Eqs. (7)–(11) are matrix operations at a prescribed dimension. Due to the physical characteristics of FPGA, matrix inversion, which consists of divisions and branch operations, are resource-expensive, so it is carried out by software to balance performance and resource utilization (Xilinx, 2008). Since H^{-1} is a constant matrix and can be computed off-line, only the inverse of the matrix $G_l H^{-1} G_l^T$ in Eqs. (9)–(11) needs to be calculated in real time.

For other matrix operations, hardware matrix multiplier and adder are designed to improve efficiency. Following the guideline of matrix multiplier design in Dou, Vassiliadis, Kuzmanov, and Gaydadjiev (2005), FPGA built-in floating point (FLP) components are used to calculate each entry in the product matrix by using a vector multiplier to calculate inner product of corresponding row and column, as shown in Fig. 3. Here, in order to simplify the matrix division in Dou et al. (2005) and then reduce the resource utilization, which is important to make the design easy to be applied to various applications, the vector inner product is chosen as the basic operation in the design.

As shown in Fig. 3, this vector multiplier reads two vectors in an element-by-element fashion, then process the data with four stages: FLP multiply, FLP to fixed point (FXP) conversion, FXP accumulation, and FXP to FLP conversion. FXP numbers are used here to increase the throughput of vector multiplier, since FLP accumulation is hard to be accomplished within one clock cycle. All input FLP data are pre-scaled to an appropriate range to avoid data overflow and underflow. All of these stages have a throughput of one floating point operation per cycle, which means they can accept new data at every clock cycle, even if the result of previous data is not ready yet. This structure thus can fully exploit the computing power of each component. By this pipelined structure, the vector multiplier takes only one clock cycle per floating point multiplication and accumulation in average.

A special Direct Memory Access (DMA) block is designed to guarantee correct data are delivered and calculated in every cycle by visiting each row or column of a matrix in a predefined order. It can fetch matrix data in either normal or transposed form to avoid unnecessary transpositions. The result is stored to local memory at the end of each vector multiplication. This design is suitable for matrix multiplication with arbitrary dimensions. The computational complexity is approximately n^3 cycles, where n is matrix dimension.

In a similar structure, a matrix adder is designed to use a FLP adder to perform addition and subtraction with computational complexity at n^2. In these modules, IEEE-754 compliant single floating point precision is adopted to avoid numerical unstable issues and improve compatibility.

Besides matrix operations, the data transfer between microprocessor and QP solver via data bus is also time-consuming. An experimental investigation of a single matrix multiplication with data transfer is shown in Table 1. The results from pure software approach are listed for comparison. It is clear that a considerable amount of time is wasted on the data transfer, which is something that should certainly be avoided.

The basic structure of ECQP solver is shown in Fig. 4. From Eqs. (7)–(11), it can be observed that other than input variables $\{H^{-1}, G, c, b\}$ and output variables $\{x, \lambda\}$, all the other variables are temporary and can be stored locally instead of being transferred back and forth. In the proposed design, the ECQP solver contains one matrix multiplier and one adder, and they can be executed in parallel. Then, the computing procedure of Eqs. (7)–(11) is divided into 12 steps as listed in Table 2. Each step has at most one multiplication and one addition/subtraction, so they can be calculated at the same time.

In the computation process, there are 16 temporary variables to be stored in the ECQP solver, among which eight of them are matrices and the others are vectors. Each variable needs to have a unique internal address to avoid access conflict. Here, on-chip block RAMs are combined into a large memory region as the local memory to store these variables. Each memory region has two data ports that can be accessed independently. The ECQP solver uses four memory regions to store all these 16 variables, and two of them are connected to FPGA internal data bus for data transfer of input/output variables. In addition, a multiplexer array is used to make other data ports accessible by the adder and multiplier. It is controlled by the DMA block for internal data delivery.

Note that additional multipliers and adders can be used to combine several steps in Table 2 to reduce the amount of data transfer and further improve the computational efficiency.

Table 3 shows the time needed by the ECQP solver to compute Eqs. (7)–(11). The time for calculation and data transfer are listed separately. A single multiplier and adder approach without local memories is also shown on the right side of Table 3 for comparison. The result shows clearly that the ECQP solver improves the efficiency significantly by reducing data transfers.

Table 1
Calculation and data transfer time breakdown. Unit: clock cycle.

Matrix dim	Calc.	Data trans.	HW total	SW total
5×5	141	413	554	2512
10×10	1021	1652	2673	20 543
15×15	3401	3715	7116	68 033

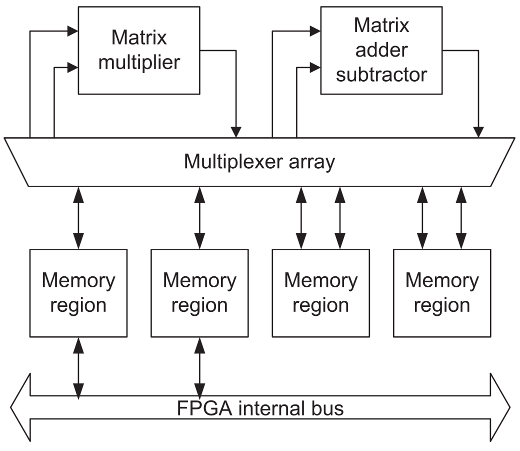

Fig. 4. Structure of ECQP solver.

Table 2
Execution sequence of ECQP solver.

#	Data transfer	Multi.	Add/sub	Complexity
1	In: H^{-1}, G			$O(n^2)$
2	In: c	$T=G\times H^{-1}$		$O(n^3)$
3	In: b	$P^T=G\times T^T$		$O(n^3)$
4		P^{-1} (by software)		$O(n^3)$
5		$R=P^{-1}\times T$		$O(n^3)$
6		$K=T^T\times R$		$O(n^3)$
7		$m^T=c^T\times R^T$	$Q=H^{-1}-K$	$O(n^2)$
8		$n=P^{-1}\times b$		$O(n^2)$
9		$o^T=b^T\times R$	$\lambda=m+n$	$O(n^2)$
10	Out: λ	$p^T=-c^T\times Q^T$		$O(n^2)$
11			$x=o+p$	$O(n)$
12	Out: x			$O(n)$

Table 3
Computing time comparison (unit: clock cycle).

Scale	ECQP solver			Single multiplier & adder		
	Calc.	Trans.	Total	Calc.	Trans.	Total
5×5	738	597	1335	722	3081	3803
10×10	4638	2305	6943	4596	11 338	15 934
15×15	14 772	5258	20 030	14 674	14 777	39 451

3.3. Hardware implementation on Xilinx Virtex-4 FPGA

In this subsection, a prototype system is built to implement the MPC design on a Xilinx ML403 development board. A picture of this experimental system is shown in Fig. 5. As an embedded platform, it matches well with the tight budget requirements set by the field controllers.

This platform has a Virtex-4 series FPGA chip with a hard core PowerPC 405 microprocessor. The PowerPC processor runs at a frequency of 200 MHz and other components at 100 MHz. The hardware ECQP solver is coded in Verilog HDL, and one ECQP solver module is instantiated in the implementation. Combined with other hardware modules provided by Xilinx, the whole hardware circuit design is downloaded into the FPGA.

Logic resource utilization is an important factor in FPGA implementation. Generally, FPGA chip provides four kinds of logic resources: Flip-flop (FF), Look-up table (LUT), DSP48, and Block RAM. FF is a distributed storage element that can store 1-bit information, and several FFs can be combined as registers or buffers to store digital data. LUT is a 4-input and 1-output logic block, whose input–output relation is defined by its truth table. Single LUT can be configured as basic logic gates, while large numbers of LUTs can be used to form complex logic. DSP48 is a

Fig. 5. MPC implementation on ML403 development board.

Table 4
Resource utilization ratio.

Flip-flop	Look-up table	DSP48	RAM block
7223/10944 (66%)	9084/10944 (83%)	13/32 (40%)	33/36 (91%)

dedicated digital signal processing unit, which contains an 18-bit integer multiplier and a 48-bit accumulator. Several DSP48 can be combined to perform floating point arithmetic. Block RAM is an on-chip memory with 18 kb capacity, which is often used to store large block of data such as floating point matrices. These resources build up the ECQP solver that solves the optimization problem in the MPC algorithm.

Table 4 shows the logic resource utilization of the proposed design. Flip-flop, Look-up table, and DSP 48 are moderately used, but RAM block is almost exhausted in storing intermediate data. This limitation restricts the dimension of H and other variables in Eqs. (7)–(11) be smaller than 32, which is dependent on the provided resource by the FPGA chip.

The proposed integrated design is particularly flexible because the hardware part does not need any additional design or configuration when deploying to new applications. What an application engineer has to do is to change some parameters in the embedded software. This integrated structure makes this design especially practical and it can be readily deployed to field control applications.

4. Application to an angle servo system

This integrated software and hardware design of the MPC algorithm can be applied to many industrial processes. In this section, the proposed FPGA based MPC design is applied to an angle servo control system to demonstrate its high performance and efficacy. The motion control system is chosen because it has high requirements on online computing efficiency.

The angle servo system under study is shown in Fig. 6. There are two DC motors: the right one is master and the left one is slave, and two needles indicating their rotating angles. The control objective is to drive the slave motor to track the master motor's rotation, whereas the angular speed of the master motor is unknown and unmeasurable to the controller. Fig. 7 shows the block diagram of the angle servo control system. Driving voltages of the two motors drive them to a certain rotation speed. Angle difference is the angle between the pointing directions of master and slave motors' needles. For this servo system, the control input is the driving voltage of the slave motor, and the output is the angle difference between the two needles, which is the only feedback signal. This system can be used to emulate many practical systems such as a radar tracking system.

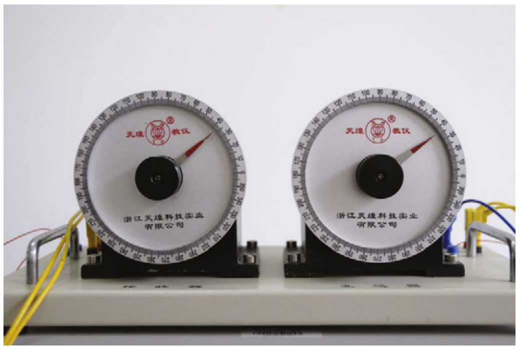

Fig. 6. Experimental platform for motor rotating angle tracking system.

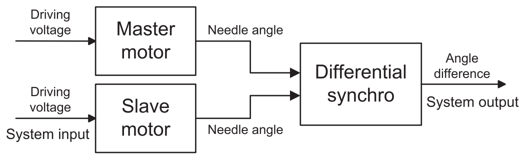

Fig. 7. Block diagram of the angle servo control system.

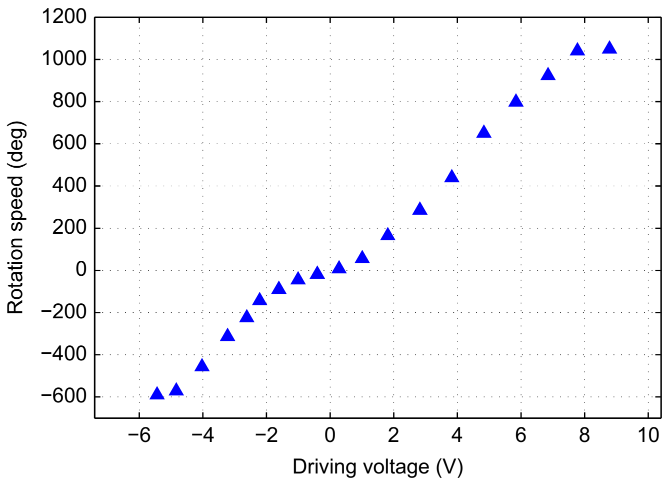

Fig. 8. Steady rotating speed and driving voltage characteristics for slave motor.

4.1. System modeling and controller design

To simplify the design, the slave motor can be modeled as an inertial system:

$$\begin{pmatrix} \dot{\theta} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & -1/\tau_s \end{pmatrix} \begin{pmatrix} \theta \\ \omega \end{pmatrix} + \begin{pmatrix} 0 \\ \alpha_s/\tau_s \end{pmatrix} u, \tag{12}$$

where θ is the angle of the rotor, ω is the angular speed, u is the driving voltage, τ_s is the time constant of the motor, and α_s is the steady speed factor.

From the motor specifications, the motor parameters can be obtained as $\alpha_s = 0.59$ r/s V and $\tau_s = 9$ ms. The relation between the steady rotating speed and driving voltage of the slave motor is depicted in Fig. 8. The physical characteristics of the motor restricts the driving voltage to the range $[\underline{u}, \overline{u}]$, where $\overline{u} = 8$ V and $\underline{u} = -5$ V. The master motor's characteristics are slightly different from the slave motor, and this information is unknown to the controller and cannot be used in the design process, so it is assumed that $\tau_m = \tau_s$ and $\alpha_m = \alpha_s$ in the modeling process and rely on MPC's robustness against model mismatch.

The servo tracking system has a differential synchro to measure the angle difference between two needles. The differential synchro introduces a high frequency sinusoidal stimulus signal on one motor's coil and measures the envelope curve of stimulated signal on the other motor's coil to determine the angle difference. This physical nature makes that some noise exists on the output signal of the differential synchro due to the stimulated signal. Fig. 9 plots the induced noise when $\Delta\theta = 30^\circ$. A low-pass filter is used to attenuate this noise.

As the tracking problem is considered, the desired operation point of the slave motor is obviously the current angle θ_m and angular speed ω_m of the master motor. Denote the operating voltage of the slave motor corresponding to the desired operation point as $\hat{u}_m$ and let the system state be $\Delta\theta = \theta_m - \theta_s$ and $\Delta\omega = \omega_m - \omega_s$. The predictive model of MPC controller is shown as follows:

$$\begin{pmatrix} \Delta\dot{\theta} \\ \Delta\dot{\omega} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & -1/\tau_s \end{pmatrix} \begin{pmatrix} \Delta\theta \\ \Delta\omega \end{pmatrix} + \begin{pmatrix} 0 \\ \alpha_s/\tau_s \end{pmatrix} (\hat{u}_m - u_s). \tag{13}$$

In Eq. (13), $\hat{u}_m$ and $\Delta\omega$ cannot be measured directly. The estimation of them will be discussed in the later part.

Setting the sampling period $T = 4$ ms and discretizing the plant (13), one can obtain that

$$x(k+1) = \begin{pmatrix} 1 & 0.004 \\ 0 & 0.58 \end{pmatrix} x(k) + \begin{pmatrix} 0 \\ 0.255 \end{pmatrix} \Delta u(k), \tag{14}$$

where $\Delta u(k) = \hat{u}_m(k) - u_s(k)$ is the control input, $x(k) = [\Delta\theta(k), \Delta\omega(k)]^T$ is the state vector. Furthermore, $\Delta\omega(k)$ can be approximated by the Backward Euler method:

$$\Delta\omega(k) = (\Delta\theta(k) - \Delta\theta(k-1))/T. \tag{15}$$

Now the MPC optimization problem is given by

$$\min_{U(k)} J = \frac{1}{2} U(k)^T H U(k) + x(k)^T F^T U(k)$$

$$\text{subject to} \quad \hat{u}_m(k+i) - \overline{u} \le u(k+i) \le \hat{u}_m(k+i) - \underline{u},$$
$$i = 0, \ldots, m-1, \tag{16}$$

where H and F can be calculated from the model (14) and weighting matrices P_f, Q_i, and R_i.

Since the tracking problem is considered here, the control horizon and predictive horizon are normally chosen to be small. In this experiment, the control horizon m is chosen to be equal to 3 to simplify the online computation, and the optimization horizon p is equal to 10 for better performance. Because $\Delta\theta$ is the only measurable output, a heavier weight is put on $\Delta\theta$ than on $\Delta\omega$. Furthermore, a terminal cost function $\frac{1}{2} x^T P_f x$ is adopted in (2) to improve the performance and ensure closed-loop stability rather than using terminal set (cf. Mayne, Rawlings, Rao, & Scokaert, 2000). The matrix P_f is determined by solving a discrete Riccati equation of system (13). In this experiment, the weighting matrices are chosen as

$$Q_i = \begin{pmatrix} 200 & 0 \\ 0 & 1 \end{pmatrix}, \quad R_i = 0.08, \quad P_f = \begin{pmatrix} 4200 & 22 \\ 22 & 0.33 \end{pmatrix}. \tag{17}$$

Note that $U(k) = 0$ is an immediate feasible solution for (16), since there is no constraint on the state variables.

Finally, the driving voltage of the imagined motor $\hat{u}_m$ needs to be estimated. Eq. (14) yields

$$\hat{u}_m(k-1) = (\Delta\omega(k) - 0.58\Delta\omega(k-1))/0.255 + u_s(k-1). \tag{18}$$

Since the sampling rate is much faster than the change of u_m, it is safe to assume that $\hat{u}_m(k) \approx \hat{u}_m(k-1)$. To attenuate the noise effects, a moving average filter is applied to remove the high frequency noise in $\hat{u}_m$:

$$\hat{u}_m(k) = \frac{1}{L} \sum_{i=1}^{L} \hat{u}_m(k-i). \tag{19}$$

Notice that a compromise has to be made for the value of L. A large L leads to smooth signal but the time response will be slowed down. In this experimental setting, L is chosen as 5.

4.2. Evaluation of control algorithm

Before proceeding to the final experiment, it is necessary to show that MPC algorithm is indeed more suitable in this application. A comparison between MPC and PID is carried out by a simulation with the motor model discussed earlier. Here, the PID controller is to calculate the increment of driving voltage of the slave motor since the considered control problem is a tracking problem. At time $t = 0$, the master motor is driven by a 5 V voltage source. The goal is to drive slave motor's needle to catch up with master motor's needle. As shown in Fig. 10, it is obvious that the MPC algorithm can drive the slave needle more quickly tracking the master needle with less transitional angle difference, which matches the control goal.

4.3. Experimental results

As mentioned earlier, ML403 FPGA development board is used in this experiment. It is connected to an A/D converter to get the sampled angle difference $\Delta\theta$, and to a D/A converter with driving circuit to actuate the slave motor. The driving voltage of the master motor is set manually.

In this experiment, it takes the hardware ECQP solver 0.02 ms to calculate Eqs. (7)–(11) one time. Since the ECQP problem of MPC controller is a 3 × 3 problem, this result verifies Table 3 by considering the additional initial time and the bus frequency

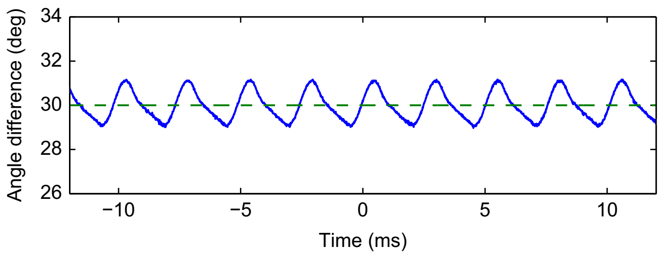

Fig. 9. Induced noise in differential synchro. Dashed line: desired output; solid line: actual output.

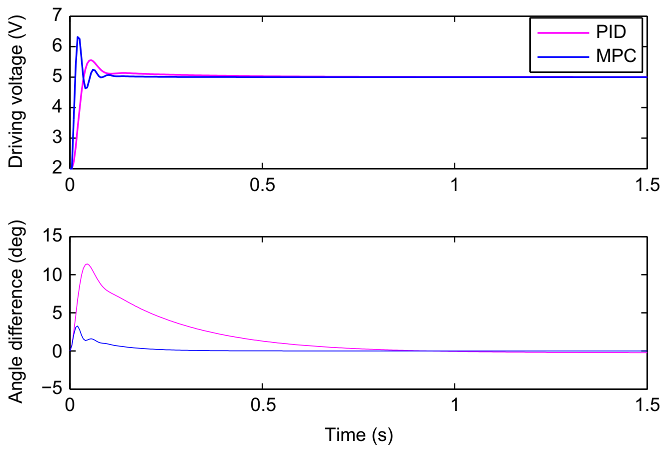

Fig. 10. Simulation result of MPC and PID algorithm.

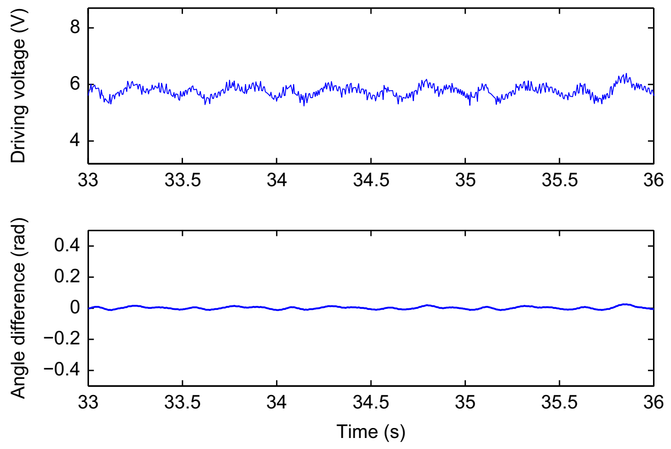

Fig. 11. Experimental results when master motor has a constant driving voltage.

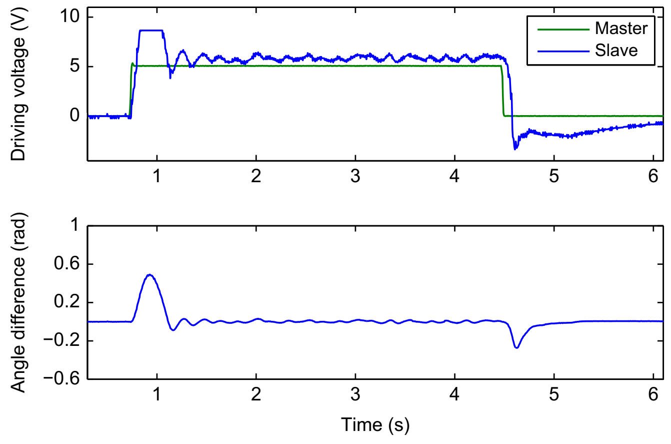

Fig. 12. Experimental results when master motor has a step input.

100 MHz. As a comparison, the software solver needs 0.4 ms to accomplish the same task. It is clear that the ECQP solver can enormously shorten the computing time of MPC.

The control performance is satisfying. In Fig. 11, it is shown that when the driving voltage of master motor is set at 5 V (amounting to about 120 rounds/min), MPC controller manages to maintain the angle difference within ± 0.03 rad (or equivalently, $\pm 1.72^\circ$). Fig. 12 shows that when the driving voltage of master motor has a steep change from 0 to 5 V, the control signal of MPC controller first hits the upper bound and then promptly get out of the constraint boundary. The total regulation time is less than 0.5 s. Note that the driving voltages for two motors are different due to the different mechanical parameters.

5. Conclusions

This paper focused on extending widely used MPC algorithm to field controllers so as to deal with constraints effectively. An integrated design of both hardware and software was developed to implement MPC on FPGA platforms, which achieves the goal of providing embedded yet computational efficient control devices as required by most field control applications. A QP solver based on embedded chips has been developed to facilitate the optimization procedure in the MPC algorithm. This QP solver is platform independent, and it can find a satisfactory solution much faster than traditional approaches. Moreover, the fundamental design guideline is universal so that it can be easily customized to other applications. This design can be ported to ASIC due to the similarity between their design flows, which leads to further single-chip cost reduction and performance improvements.

A prototype experimental system was built to demonstrate the feasibility and effectiveness of the proposed design. The prototype system has been further applied to a motor servo tracking system and achieved good control performance. The proposed design thus can provide a portal to extend MPC algorithms to many industrial applications of field control. It also strikes a good balance between cost and performance, which makes it particularly appealing to field control where a large number of embedded yet capable controllers are required.

It is worth pointing out that depending on specific application, sophisticated chips with more resources can be used if budget permits. With more multipliers and adders under disposal, some steps in Table 2 can be combined to form a more efficient execution sequence. This will further reduce the computational time of the MPC algorithm and makes it applicable to those field controls mandating fast responses.

Acknowledgments

This work is supported in part by the National Science Foundation of China (Grant nos. 60934007, 61074060), China Postdoctoral Science Foundation Special Support (Grand no. 201003272), and Innovation Program of Shanghai Municipal Education Commission (Grant nos. 11ZZ20, 11CXY08).

References

Bartlett, R. A., Wächter, A., & Biegler, L. T. (2000). Active set vs. interior point strategies for model predictive control. In *American control conference* (pp. 4229–4233). IEEE.
Bemporad, A. (2006). Model predictive control design: New trends and tools. In *45th IEEE conference on decision and control* (pp. 6678–6683). IEEE.
Bleris, L., Vouzis, P., Arnold, M., & Kothare, M. (2006). A co-processor fpga platform for the implementation of real-time model predictive control. In *American control conference* (pp. 1912–1917). IEEE.
Chen, B. L. (2005). *Optimization theory and algorithms*. China: Tsinghua University Press.
Dou, Y., Vassiliadis, S., Kuzmanov, G., & Gaydadjiev, G. (2005). 64-bit floating-point fpga matrix multiplication. In *Proceedings of the ACM/SIGDA13th international symposium on field-programmable gate arrays* (pp. 86–95). ACM.
Fletcher, R. (1970). *Calculation of feasible points for linearly constrained optimization problems. OSTI Identifier*, OSTI ID: 4485671. Report Number(s), AERE-R 6354.
He, M., & Ling, K. (2005). Model predictive control on a chip. In *International conference on control and automation* (Vol. 1, pp. 528–532). IEEE.
Kuon, I., & Rose, J. (2007). Measuring the gap between fpgas and asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *26*(2), 203–215.
Lau, M., Yue, S., Ling, K., & Maciejowski, J. (2009). A comparison of interior point and active set methods for fpga implementation of model predictive control. In *Proceedings of the European control conference* (pp. 156–160).
Ling, K. V., Yue, S. P., & Maciejowski, J. M. (2006). A fpga implementation of model predictive control. In *American control conference* (pp. 1930–1935). IEEE.
Mayne, D., Rawlings, J., Rao, C., & Scokaert, P. (2000). Constrained model predictive control: Stability and optimality. *Automatica*, *36*(6), 789–814.
Monmasson, E., & Cirstea, M. (2007). Fpga design methodology for industrial control systems—A review. *IEEE Transactions on Industrial Electronics*, *54*(4), 1824–1842.
Morari, M., & Lee, J. (1999). Model predictive control: Past, present and future. *Computers & Chemical Engineering*, *23*(4–5), 667–682.
Ordys, A., Grimble, M. J., & Ordys, A. W. (2001). Predictive control for industrial applications. *Annual Reviews in Control*, *25*, 13–24.
Qin, S., & Badgwell, T. (2003). A survey of industrial model predictive control technology. *Control Engineering Practice*, *11*(7), 733–764.
Rawlings, J. (2000). Tutorial overview of model predictive control. *IEEE Control Systems*, *20*(3), 38–52.
Tessier, R., & Burleson, W. (2001). Reconfigurable computing for digital signal processing: A survey. *Journal of VLSI Signal Processing*, *28*(1), 7–27.
Xilinx, I. (2008). Virtex-4 FPGA user guide. Xilinx Inc.