FPGA accelerated model predictive control for autonomous driving

Yunfei Li
Tsinghua University, Beijing, China and Chongqing University, Chongqing, China
Shengbo Eben Li
Department of Automotive Engineering, Tsinghua University, Beijing, China, and
Xingheng Jia, Shulin Zeng and Yu Wang
Tsinghua University, Beijing, China

Abstract

Purpose – The purpose of this paper is to reduce the difficulty of model predictive control (MPC) deployment on FPGA so that researchers can make better use of FPGA technology for academic research.

Design/methodology/approach – In this paper, the MPC algorithm is written into FPGA by combining hardware with software. Experiments have verified this method.

Findings – This paper implements a ZYNQ-based design method, which could significantly reduce the difficulty of development. The comparison with the CPU solution results proves that FPGA has a significant acceleration effect on the solution of MPC through the method.

Research limitations implications – Due to the limitation of practical conditions, this paper cannot carry out a hardware-in-the-loop experiment for the time being, instead of an open-loop experiment.

Originality value – This paper proposes a new design method to deploy the MPC algorithm to the FPGA, reducing the development difficulty of the algorithm implementation on FPGA. It greatly facilitates researchers in the field of autonomous driving to carry out FPGA algorithm hardware acceleration research.

Keywords FPGA, Model predictive control, Autonomous driving, ZYNQ

Paper type Research paper

1. Introduction

Compared with other control methods, model predictive control (MPC) has many advantages: low requirements on model accuracy, good robustness and effective handling of multivariate constraint problems (Fernandez-Camacho and Bordons-Alba, 1995). Besides, MPC has succeeded in the field of industrial process control (Yu-Geng *et al.*, 2013). Therefore, in recent years, it has been widely used in the field of autonomous driving (Goli and Eskandarian, 2019; Quan and Chung, 2019; Li *et al.*, 2010). However, MPC often shows low efficiency in solving real-time tasks due to a large amount of calculation (Yu-Geng *et al.*, 2013). Researchers hope that the high-performance computing platform's computing capacity can make up for this defect of MPC. The core of the hardware computing platform is the processor chip. Currently, mainstream chips include CPU, graphics processing unit (GPU), FPGA and application specific integrated circuit (ASIC). The parallel computing capabilities of GPU, FPGA and ASIC are far superior to CPU. They are often used as hardware accelerators. Among these three chips, FPGA has the absolute advantage in power consumption over GPU and has reversible development characteristics compared with ASIC (Falsafi *et al.*, 2017; Nurvitadhi *et al.*,2016; Kestur *et al.*, 2010; Qasaimeh *et al.*, 2019; Kuon and Rose, 2007; Jones *et al.*, 2010; Russo *et al.*, 2012). With these characteristics, FPGA is more adaptable to algorithms update, making it widely welcomed by researchers.

How to use FPGA to accelerate MPC is a problematic point. Summarizing the existing studies, the primary way to realize the hardware-accelerated solution of MPC by FPGA is through using hardware description languages. For example, He and Ling (2005) used Handle-C hardware description language to implement the accelerated solution of MPC on FPGA for the first time.

The current issue and full text archive of this journal is available on Emerald Insight at: **https://www.emerald.com/insight/2399-9802.htm**

Journal of Intelligent and Connected Vehicles
5/2 (2022) 63–71
Emerald Publishing Limited [ISSN 2399-9802]
[DOI 10.1108/JICV-03-2021-0002]

Received 10 March 2021
Revised 22 September 2021
4 February 2022
Accepted 9 February 2022

Following this, Jerez *et al.* (2012) described a parameterizable FPGA application architecture, which mainly used a deep pipeline structure; as the solution scale increases, the MPC acceleration effect is gradually significant. Jerez *et al.* (2014) proposed a sharable hardware architecture based on the fast gradient descent method and the alternating multiplier method to solve the MPC problem, which can save a lot of hardware resources.

However, in the field of autonomous driving, researchers are better at high-level languages. Hardware language is too difficult for them. Benefitted from the development of electronic design automation technology, researchers can directly use high-level languages through high-level synthesis tools (Martin and Smith, 2009) to realize the mapping of algorithms to hardware. In this way, several achievements have been made in research. For example, Xu *et al.* (2015) successfully converted the C++ form of MPC into hardware language through Altera's Quartus II and Mentor's Catapult Synthesis, and deployed it on Altera Stratix III FPGA. Lucia *et al.* (2017) used the advanced synthesis tools provided by Xilinx to deploy MPC on XC7A200, which further proved the feasibility of bypassing the direct use of hardware languages and indirect deployment of high-level languages on FPGA.

ZYNQ, as a new generation of Xilinx FPGA products, integrates the processing system based on a dual-core Advanced RISC Machine (ARM) Cortex-A9 and the programmable logic composed of an XC7Z020 FPGA. Compared with the independent FPGA, it owns the high-performance computing power of FPGA and the unparalleled resource allocation ability of CPU. The combination of the two allows researchers to process the algorithm more flexibly. At the same time, ZYNQ also has the advantages of low power consumption and low price.

Although the above methods realized the deployment of MPC to FPGA and proved its feasibility, they were not for ZYNQ. We need a convenient and fast algorithm deployment method for the new generation of FPGA hardware.

The main contribution of this paper is to propose a method to deploy the MPC algorithm to FPGA (ZYNQ), which greatly reduces the difficulty of algorithm implementation on the latter. Our research results lay the foundation for the application of ZYNQ in actual vehicle experiments.

The paper is organized as follows. In Section 2, we design a lateral control algorithm for autonomous vehicles. In Section 3, a software and hardware combination method based on ZYNQ is proposed and realized. The control algorithm's feasibility, the solution performance of the quadratic programming solver and the acceleration effect of FPGA are verified in Section 4. Section 5 concludes this paper.

2. Lateral control algorithm of autonomous vehicles

Generally, vehicle control consists of lateral control and longitudinal control. For convenience, the trajectory tracking scenario in lateral control is discussed in this section, which will serve as the basis for subsequent study in this paper.

2.1 Dynamic model

As shown in Figure 1, we choose the single-track bicycle model assuming constant forward speed (Bevly *et al.*, 2006). The vehicle dynamics are described as:

$$\begin{bmatrix} \dot{y} \\ \dot{r} \\ \dot{\beta} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & v_x & v_x & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \dfrac{C_f + C_r}{mv_x} & \dfrac{C_f a - C_r b}{mv_x^2} - 1 \\ 0 & 0 & \dfrac{C_f a - C_r b}{I_z} & \dfrac{C_f a^2 + C_r b^2}{I_z v_x} \end{bmatrix} \begin{bmatrix} y \\ \psi \\ \beta \\ r \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\dfrac{C_f}{mv_x} \\ -\dfrac{C_f a}{I_z} \end{bmatrix} \delta \tag{1}$$

where y is the lateral displacement; v_x is the longitudinal speed; C_f is the front wheel cornering stiffness and C_r is the rear wheel cornering stiffness; m is the vehicle mass; a and b are the distances of front and rear axle from the center of gravity; I_z is the moment of inertia; ψ is the yaw angle; β is the vehicle slip angle; r is the yaw rate; δ is the front wheel steering angle.

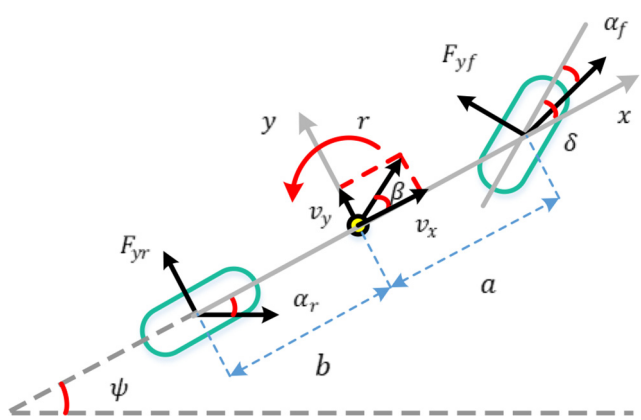
The state-space equations are obtained as:

$$\begin{cases} \dot{x}(k) = A_c x(k) + B_c u(k) \\ y(k) = C_c x(k) \end{cases} \tag{2}$$

where $x(k)$ is the state variable, $x(k) = [y\ \psi\ \beta\ r]^T$, $y(k)$ is the output variable, $y(k) = [y\ \psi]^T$, $u(k)$ is the control variable, $u(k) = \delta$,

$$A_c = \begin{bmatrix} 0 & v_x & v_x & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \dfrac{C_f + C_r}{mv_x} & \dfrac{C_f a - C_r b}{mv_x^2} - 1 \\ 0 & 0 & \dfrac{C_f a - C_r b}{I_z} & \dfrac{C_f a^2 + C_r b^2}{I_z v_x} \end{bmatrix},$$

Figure 1 Single-track bicycle model



$$B_c = \begin{bmatrix} 0 \\ 0 \\ -\dfrac{C_f}{mv_x} \\ -\dfrac{C_f a}{I_z} \end{bmatrix},\ C_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The discretization form of (2) is:

$$\begin{cases} x(k+1) = A \cdot x(k) + B \cdot u(k) \\ y(k) = C \cdot x(k) \end{cases} \tag{3}$$

where $A = I + T \cdot A_c$, $B = T \cdot B_c$, $C = C_c$, T is the sampling time, I is the identity matrix.

For $y(k) = C \cdot x(k)$, we set both the prediction horizon and control horizon to P, then

$$Y = C_P \cdot x(k) + D_P \cdot U \tag{4}$$

where

$$Y = \begin{bmatrix} y(k+1) \\ y(k+2) \\ \vdots \\ y(k+P) \end{bmatrix}, C_P = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^P \end{bmatrix}, U = \begin{bmatrix} u(k|k) \\ u(k+1|k) \\ \vdots \\ u(k+P-1|k) \end{bmatrix},$$

$$D_P = \begin{bmatrix} CA^0B & 0 & \cdots & 0 \\ CA^1B & CA^0B & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ CA^{P-1}B & CA^{P-2}B & \cdots & CA^0B \end{bmatrix}$$

2.2 Cost function and optimization problem

Vehicle lateral control needs to ensure that the autonomous vehicle can track the reference trajectory as close as possible, so in the cost function, we need to consider the deviation between the predicted value of the lateral displacement and the reference value, and the deviation between the predicted value of the yaw angle and the reference value. In summary, the cost function is designed as:

$$\begin{aligned} L = & \sum_{i=1}^{P} \Big\{ q_1 \left[\varphi(k+i|k) - \varphi_{ref}(k+i|k) \right]^2 \\ & + q_2 \left[\tilde{Y}(k+i|k) - \tilde{Y}_{ref}(k+i|k) \right]^2 \Big\} \\ & + \sum_{i=0}^{P-1} r[u(k+i|k)]^2 \\ = & \sum_{i=1}^{P} \| Y(k+i|k) - Y_{ref}(k+i|k) \|^2 Q \\ & + \sum_{i=0}^{P-1} \| u(k+i|k) \|^2 R \end{aligned} \tag{5}$$

where $\varphi(k+i|k)$ and $\varphi_{ref}(k+i|k)$ are the predicted yaw angle and the reference yaw angle, respectively. $\tilde{Y}(k+i|k)$ and $\tilde{Y}_{ref}(k+i|k)$ are the predicted lateral displacement and the reference lateral displacement, respectively. $u(k+i|k)$ is the control input, i.e. front-wheel steering angle. q_1 denotes the weight coefficient of the yaw angle, while q_2 denotes the weight coefficient of the lateral displacement. r is the weight coefficient of the control variable. $Y(k+i|k)$ and $Y_{ref}(k+i|k)$ are the predicted values and the reference values. Q is the weight matrix of output variables, $Q = \begin{bmatrix} q & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & q \end{bmatrix}, q = \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix}$, and R is the weight matrix of control variables, $R = \begin{bmatrix} r & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r \end{bmatrix}$.

Substituting (4) into (5):

$$\mathcal{J} = \frac{1}{2} U^T \cdot H \cdot U + G^T \cdot U \tag{6}$$

where

$$\mathrm{H} = 2\left(D_P{}^T Q D_P + R\right), \quad G^T = 2(QD_P)^T \left(C_P \cdot x(k) - Y_{\mathrm{ref}}\right),$$

$$Y_{\mathrm{ref}} = \begin{bmatrix} Y_{ref}(k+1|k) \\ Y_{ref}(k+2|k) \\ \vdots \\ Y_{ref}(k+P|k) \end{bmatrix}$$

Our goal is to minimize (6):

$$\min \mathcal{J} = \min \left(\frac{1}{2} U^T \cdot H \cdot U + G^T \cdot U \right) \tag{7}$$

The number of constraints directly determines the dimension of the solution to MPC. To save FPGA hardware resources in the following text, we only restrict the control variable (when hardware resources are sufficient, the performance of FPGA can be extended to MPC with state constraints):

$$U_{min} \le U \le U_{max} \tag{8}$$

where $U = \begin{bmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+P-1} \end{bmatrix}, U_{min} = \begin{bmatrix} u_{min} \\ u_{min} \\ \vdots \\ u_{min} \end{bmatrix}, U_{max} = \begin{bmatrix} u_{max} \\ u_{max} \\ \vdots \\ u_{max} \end{bmatrix}$,

u_{min} and u_{max} are the lower and upper limits of the control variable, respectively.

We can rewrite (8) as:

$$A^* U \le B^* \tag{9}$$

where $A^* = \begin{bmatrix} -T \\ T \end{bmatrix}, B^* = \begin{bmatrix} -U_{min} \\ U_{max} \end{bmatrix}, T = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$

By combining (7) and (9)

$$\begin{aligned} &min \ \frac{1}{2} U^T \cdot H \cdot U + G^T \cdot U \\ &s.t. \quad A^* U \le B^* \end{aligned} \tag{10}$$

Definition 1: Formula (10) is the standard form of the quadratic programming (QP) problem with constraints. The essence of solving the MPC problem is to solve the QP problem. Each time a set of optimal solution sequence U^* is obtained, the first element u^* is taken as the control variable.

2.3 Quadratic programming solver

The QP solver used in this paper is Quadprog++ (Di Gaspero, 2007). It is an open-source solver. Compared with other QP solvers such as quadprog, CVXGEN. Quadprog + + has the advantages of simplicity, easy modification and fewer hardware resources occupation (Mattingley and Boyd, 2012; Brandao *et al.*, 2019).

Quadprog++ is written in C++ by Luca Di Gaspero according to the Goldfarb–Idnani method (Goldfarb and Idnani, 1983). The Goldfarb–Idnani method combines the active set algorithm (Nocedal and Wright, 2006) and the dual algorithm to have a fast iteration speed.

Definition 2: The idea of the active set shows that (10) can be transformed into a form of equality constraints:

$$\begin{aligned} &min \ \frac{1}{2} U^T \cdot H \cdot U + G^T \cdot U \\ &s.t. \quad N^T U = B_S^* \end{aligned} \tag{11}$$

where S denotes the indices of the active set, N is the active set matrix determined by S, B_S^* are the elements of B^* indexed by S (Horowitz and Afonso, 2002).

According to the KKT conditions (under the transformation $x = H^{1/2} U$) (Goldfarb and Idnani, 1983; Horowitz and Afonso, 2002):

$$\begin{cases} x^* = -MG + N^{*T} B_S^* \\ \gamma^* = N^* G + W B_S^* \end{cases} \tag{12}$$

where γ^* is the Lagrangian multiplier, x^* is the optimal solution, and

$$N^* = (N^T H^{-1} N)^{-1} N^T H^{-1},$$

$$M = H^{-1} - H^{-1} N (N^T H^{-1} N)^{-1} N^T H^{-1},$$

$$W = (N^T H^{-1} N)^{-1}$$

Cholesky decomposition of H:

$$H = K^T K \tag{13}$$

Remark 1: The Goldfarb–Idnani method only supports solving positive definite problems (Goldfarb and Idnani, 1983), so H must be a positive definite matrix.

QR decomposition of $(K^{-T} N)$ (Horowitz and Afonso, 2002):

$$\begin{aligned} H^{-1} N &= K^{-1} K^{-T} N = K^{-1} Q \begin{bmatrix} R \\ 0 \end{bmatrix} \\ &= L \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} E \vdots F \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = ER \end{aligned} \tag{14}$$

where L is an orthogonal matrix, R is an upper triangular matrix and E contains as many columns as R.

Then N^*, M and W can be shown as (Goldfarb and Idnani, 1983):

$$N^* = R^{-1} E^T, \ M = FF^T, \ W = R^{-1} R^{-T} \tag{15}$$

We set S_k to be the currently active set, N_k, L_k and R_k are the matrixes corresponding to S_k. When S_k is not empty, through Givens rotations, we can get (Horowitz and Afonso, 2002):

$$H^{-1} = L_k L_k^T, \ R_k = E_k^T N_k \tag{16}$$

According to (12) and (15) (Horowitz and Afonso, 2002):

$$\begin{cases} x_k = -F_k F_k^T G + E_k R_k^{-T} B_k^* \\ \gamma_k = R_k^{-1} E_k^T G + R_k^{-1} R_k^{-T} B_k^* \end{cases} \tag{17}$$

where x_k is the solution and γ_k is the Lagrangian multiplier corresponding to S_k.

Also, because of the KKT conditions:

$$G = -K^T K x_{k+1} + N_k \gamma_{k+1} + n^+ t \tag{18}$$

where x_{k+1} is the solution corresponding to $S_k \cup m$. n^+ is the normal vector of the mth constraint, t is the corresponding Lagrangian multiplier (Horowitz and Afonso, 2002).

According to (15)–(18), the search directions of Goldfarb–Idnani method are defined as (Schmid and Biegler, 1994):

$$\begin{cases} x_{k+1} = x_k + F_k F_k^T n^+ t \\ \gamma_{k+1} = \begin{Bmatrix} \gamma_k \\ 0 \end{Bmatrix} + \begin{Bmatrix} -R_k^{-1} E_k^T n^+ \\ 1 \end{Bmatrix} t \end{cases} \tag{19}$$

Remark 2: The relevant proof processes of the Goldfarb–Idnani method are shown in literature (Goldfarb and Idnani, 1983).

The pseudo-code of the Goldfarb–Idnani method is shown in the algorithm.

Algorithm Goldfarb-Idnani method (Goldfarb and Idnani, 1983; Horowitz and Afonso, 2002)

Initializing $x = -H^{-1} G = -K^{-1} K^{-T} G, L = K^{-1}$,
$S = \{\emptyset\}, v = 0, v$ as the cardinality of S
1. if all constrains are satisfied **then** x is optimal, STOP.
else $n^+ = n_p, \gamma^+ = [\gamma \ 0]^T$,
if $v = 0$ **then** $\gamma^+ = 0$,
end if;
end if;
$S^+ = S \cup \{p\}, p$ as the index of constraint to be added to S

2. (a) Search direction in primal space: $z = FF^T n^+$
if $v > 0$ **then**
search direction in dual space: $r = R^{-1} E^T n^+$
end if;
(b) Step length:
maximum step in dual space: t_1
if $v = 0$ or $r \leq 0$ **then** $t_1 = \infty$
else $t_1 = \min_{j=1,\ldots,v} \left\{ \frac{\gamma_j^+}{r_j} \mid r_j > 0 \right\}$
end if;
minimum step in primal space: t_2
if $\|z\| = 0$ **then** $t_2 = \infty$
else $t_2 = \frac{B_p^* - (n^+)^T x}{z^T n^+}$
end if;
$t = \min(t_1, t_2)$
(c) **if** $t = \infty$ **then** problem infeasible
end if;
if $t_2 = \infty$, t_1 is finite **then** $\gamma^+ = \gamma^+ + t \begin{Bmatrix} -r \\ 1 \end{Bmatrix}$,
$S = S \backslash \{m\}$, $v = v - 1$, update L, R and γ^+, go to 2(a)
end if;
set $x = x + tz, \quad \gamma^+ = \gamma^+ + t \begin{Bmatrix} -r \\ 1 \end{Bmatrix}$
if, $t = t_2$ **then** $\gamma = \gamma^+$ $S = S \cup \{p\}$, $v = v + 1$,
update L, R. Go to 1.
else $t = t_1$ **then** $S = S \backslash\backslash \{m\}$, $v = v - 1$, update L, R, and γ^+. Go to 2(a).
end if;

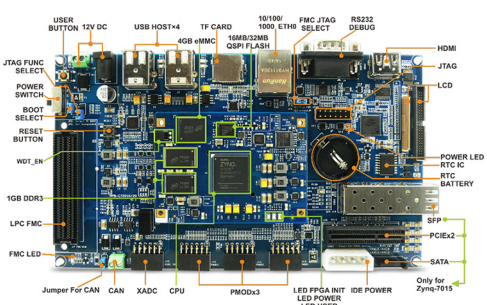
The update operations of the Cholesky, L and R in the Goldfarb–Idnani method account for a large proportion, and they are also the parts that consume the most hardware resources.

3. Implementation quadratic programming solver on FPGA

3.1 Hardware platform selection

According to our framework, the computation part of the QP solver is fully deployed on the FPGA as it is computing-intensive and time-consuming. The ARM processor is only responsible for data transmission and high-level system control. This kind of scheme can fully use the computing power of FPGA and the flexibility of the ARM processor. In this paper, the MYD-C7Z020 development board (Figure 2) is used as the hardware platform. Table 1 lists the parameters of MYD-C7Z020 and shows its strong ability to adapt to the environment. MYD-C7Z020 is composed of the core board and the bottom board. The core board is embedded with a core function chip such as ZYNQ SoC, while the bottom board is equipped with various functional interfaces, switches and indicators.

Figure 2 MYD-C7Z020 development board



3.2 Design flow

The overall design flow is shown in Figure 3; we follow a software-hardware codesign method to deploy the proposed algorithm. The hardware part is to deploy the QP algorithm to the programmable logic for fast computation and data movement optimization. The software is mainly aimed at the processing system. The purpose is to realize the deployment of the drivers, the data interaction between the on-chip memory and the off-chip interfaces and the self-starting of the hardware development platform.

3.1.1 Hardware design

In hardware design, we first use Xilinx Vivado HLS (Winterstein *et al.*, 2013) to convert the C++ form of the algorithm to register transfer level. We also need to select the functional interface type and the optimization method to implement effective algorithm deployment.

Considering the controller's control effect and the maximum utilization rate of hardware, we set both the prediction horizon and the control horizon to be five, so the maximum dimension of the matrix calculated on the FPGA is ten. Table 2 shows the hardware resource utilization information of FPGA. FPGA mainly contains four kinds of hardware resources: block

Table 1 Parameters of MYD-C7Z0210

Operating temperature	−40 to +85°C
Ambient temperature	−50°C to +100°C
Environment humidity	20%–90%
Mechanical dimensions	BP: 190 mm × 110 mm, CP: 75 mm × 55 mm
Power supply	BP: 12 V/0.5 A, CP: 5 V/0.5 A
Power consumption	BP: 6 W, CP: 2.5 W

Figure 3 Combined hardware-software design

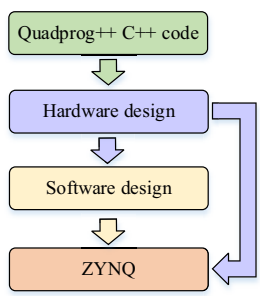


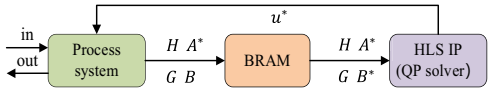
Table 2 Hardware resources utilization information of FPGA

	BRAM	DSP48E	FF	LUT
Total	22	78	30,059	47,923
Available	280	220	106,400	53,200
Utilization	7%	35%	28%	90%

Table 3 Comparison of optimization results

Optimization	Default state	Pipeline	Unroll	Pipeline and unroll
BRAM	22	22	22	22
DSP48E	78	78	78	78
FF	30,059	30,059	30,060	30,132
LUT	47,923	47,954	48,348	48,356
Simulation time	35,525 ns	35,445 ns	35,285 ns	35,365 ns
Delay	3,534	3,526	3,510	3,518

Figure 4 The main project of Vivado IDE



random access memory (BRAM), DSP48E, flip flop (FF) and look-up table (LUT). LUT resources are occupied so much because the Goldfarb–Idnani algorithm involves a large number of matrix multiplication and addition operations (the update operations of Cholesky, L and R).

The biggest advantage of FPGA is that it uses hardware to perform parallel operations. This type of process is very intuitive, such as $a{\cdot}b + c{\cdot}d$, which can perform $a{\cdot}b$ and $c{\cdot}d$ simultaneously. Vivado HLS can perform automatic parallel processing while generating the hardware language. We can also choose to select different optimization methods to process the C++ code manually. According to the specific situation, we select pipeline, unroll and pipeline&unroll. The results are shown in Table 3; neither the resource utilization rate nor the simulation time has been significantly improved (in the follow-up Vivado IDE-related process, these three optimization methods did not pass the verification due to excessive wiring resources). The above results are related to the algorithm structure; if it is composed of a relatively neat neural network structure, these optimization methods will produce significant results.

When the above work is completed, we need an environment to achieve corresponding hardware functions, so the algorithm module is imported into the environment generated by Vivado IDE (Crockett *et al.*, 2014). The main contribution of our design is shown in Figure 4. We mainly choose three modules to achieve the corresponding functions (the combination of modules needs to be designed according to the specific functions to be implemented). The advantage of this design is to take up as little additional hardware resources as possible. The entire project's workflow is that the processing system first writes the matrices H and A^*, the vectors G and B^* to BRAM and then the IP generated by Vivado HLS reads the data in BRAM and accelerates the solution. When the solution is completed, the processing system reads the result u^* from HLS IP (reading and writing data are done in a polling manner). The communication between different modules is realized through the AXI interface.

3.1.2 Software design

The design of the software part is mainly focused on the writing of driver code. The driver makes ARM the core of the entire architecture, and FPGA acts as a hardware accelerator to assist its work. Besides algorithm acceleration, multitasking functions also need to be supported in the development board's actual application. This situation requires complicated code programming to achieve, which is troublesome for us, so it is necessary to select an embedded system with mature architecture to complete these works. Linux system is the right choice. Popular Linux distributions mainly include Debian, Fedora and Ubuntu. As a newer distribution, Ubuntu inherits all the advantages of the Linux system and has highlights such as easy installation and various auxiliary functions (Al Housani *et al.*, 2009). We choose Ubuntu16.04 as the operating system deployed on the ARM processor.

4. Results

In this section, we mainly verify the effectiveness of the control algorithm, the reliability of the QP solver and the acceleration effect of FPGA through simulation and experiments.

4.1 Verification of lateral control algorithm

We use MATLAB/Simulink and CarSim for cosimulation in the PC to verify the effect of the control algorithm designed in Section 2. Tables 4 and 5 list the main parameters of the vehicle and the parameters of the lateral control algorithm, respectively.

Figure 5 shows the simulation results. We choose the double lane-change as the reference trajectory. The maximum error

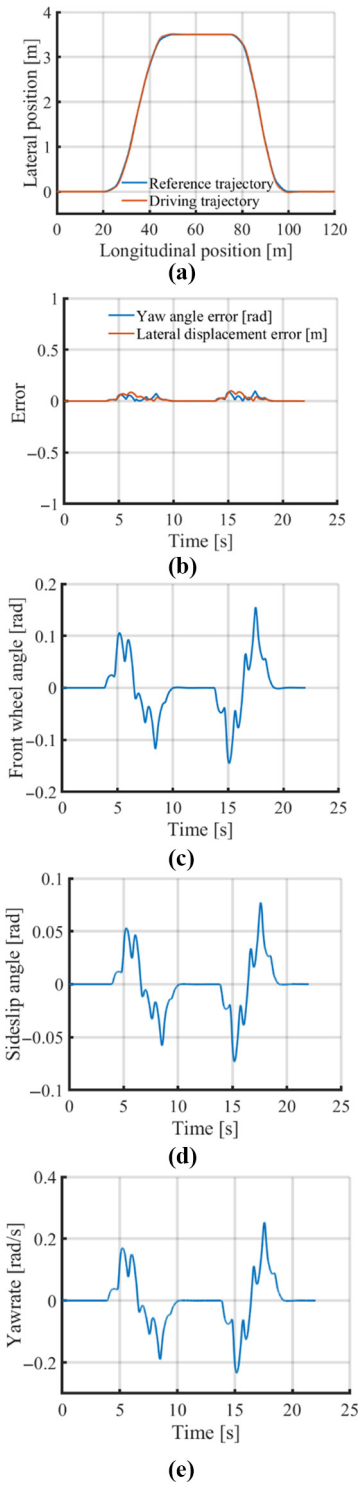
Table 4 Main parameters of the vehicle

Vehicle mass	m	1,420 [kg]
Distance of front axle from the center of gravity	a	1.015 [m]
Distance of rear axle from the center of gravity	b	1.895 [m]
Front wheel cornering stiffness	C_f	72,197 [N/rad]
Rear wheel cornering stiffness	C_r	39,930 [N/rad]
Moment of inertia	I_z	1,536.7 [kg/m^2]
Longitudinal speed	v_x	20 [km/h]

Table 5 Parameters of the lateral control algorithm

Prediction horizon/Control horizon	P	5
Sampling time	T_s	0.02 [s]
Output weighting matrix	q	[50 0;0 10]
Control weighting coefficient	r	4
Control variable upper limit	u_{max}	0.52 [rad]
Control variable lower limit	u_{min}	−0.52 [rad]

Figure 5 The simulation results. (a) Vehicle trajectory. (b) The error. (c) Front-wheel angle. (d) Sideslip angle. (e) Yaw rate



of the lateral displacement between the driving trajectory and the reference trajectory is less than 0.075 m, and the maximum error of the yaw angle is less than 0.098 rad. The above results prove that the lateral control algorithm in Section 2 is effective.

4.2 Verification of quadprog++

As shown in Figure 6, the performance verification method of Quadprog++ is to ensure the input that is precisely the same as the quadprog solver used in the simulation of part A (Section 4) and then compare the solution accuracy and the solution time of these two solvers. Both solvers run on the PC with the Intel i5 processor at 2.3 GHz. The software platform of quadprog is MATLAB, while Quadprog++'s is Visual Studio.

The comparison results of the solution accuracy are shown in Figure 7. The maximum percentage error of the two solvers is less than 0.008%.

Figure 8 and Table 6 present the solution time information of quadrog and Quadprog++. The solution performance of the two solvers is very close.

Figure 6 Verification scheme

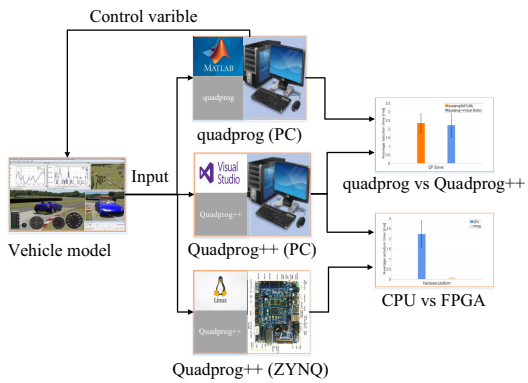


Figure 7 Comparison of the solution accuracy of quadprog and Quadprog++

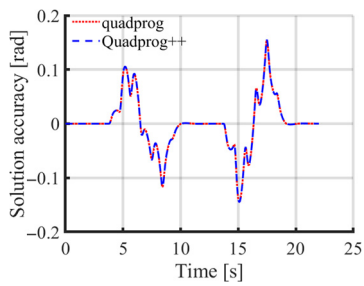


Figure 8 Comparison of the solution time of quadprog and Quadprog++

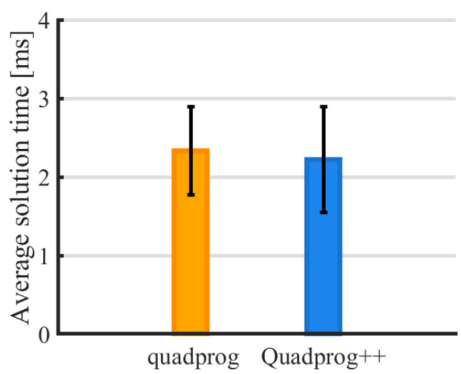


Table 6 Solution time information of QP solvers

QP Solver	Average solution time	Standard deviation of solution time
quadprog	2.332 ms	0.5645 ms
Quadprog++	2.216 ms	0.6723 ms

Table 7 Solution time information of hardware platforms

Hardware platform	Average solution time	Standard deviation of solution time
CPU	2.216 ms	0.6723 ms
FPGA	0.0803 ms	4.35×10^{-4} ms

The above results prove that Quadprog++ can well meet the solution requirements of the lateral control algorithm in this paper.

4.3 Verification of FPGA

The performance verification method of FPGA (Figure 6) is to use the input that is entirely consistent with the CPU platform and then compare the solution accuracy and the solution time of the two (both use the Quadprog++ solver). The configuration of the CPU is the same as part B (Section 4), and the frequency of ZYNQ is set to 50 MHz.

The comparison results of the solution accuracy are shown in Figure 9. The maximum percentage error of CPU and FPGA is less than 0.04%.

As shown in Figure 10 and Table 7, the average solution speed of FPGA is 27.162 times faster than that of CPU and the solution time fluctuation of FPGA is much less than that of the latter.

Figure 9 Comparison of the solution accuracy of CPU and FPGA

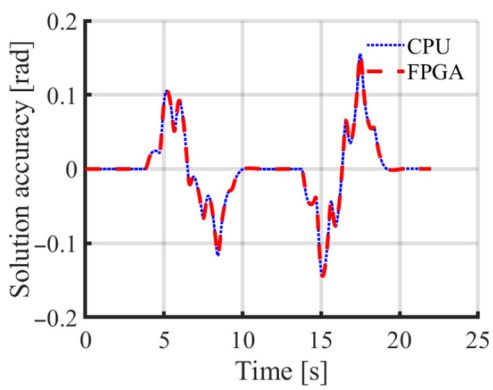
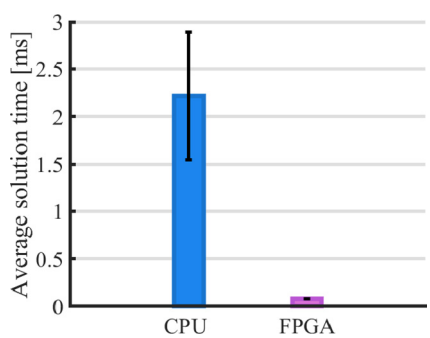


Figure 10 Comparison of the solution time of CPU and FPGA



The above experimental results indicate that compared with CPU, FPGA dramatically improves the speed of solving QP and improves the calculation efficiency of MPC.

5. Conclusion

This paper proposed an FPGA accelerated method of MPC for autonomous driving. Given the difficulty of combining MPC and FPGA. We implement a ZYNQ-based design method, which could significantly reduce the difficulty of development. The comparison with the CPU solution results shows that FPGA has a significant acceleration effect on the solution of MPC (the latter is 27.162 times faster than the former). Our method is effective.

In the future study, we will convert all the floating-point data to fixed-point data to save the hardware resources. We will also carry out relevant actual vehicle experiments to verify the control effect of the selected ZYNQ hardware. At the same time, we will also improve existing algorithms to adapt to more complex scenarios (Keskin *et al.*, 2020).

References

Al Housani, B., Mutrib, B. and Jaradi, H. (2009), "The linux review-Ubuntu desktop edition-version 8.10", *2009 International Conference on the Current Trends in Information Technology (CTIT)*, December, IEEE, pp. 1-6.

Bevly, D.M., Ryu, J. and Gerdes, J.C. (2006), "Integrating INS sensors with GPS measurements for continuous estimation of vehicle sideslip, roll, and tire cornering stiffness", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 7 No. 4, pp. 483-493.

Brandao, A.S.M., Lima, D.M., DA Costa Filho, M.V.A. and Rico, J.E.N. (2019), "A comparative study on embedded MPC for industrial processes", *Congresso Brasileiro de Automática-CBA*, Vol. 1 No. 1.

Crockett, L.H., Elliot, R., Enderwitz, M. and Stewart, R. (2014), *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*, Strathclyde Academic Media.

Di Gaspero, L. (2007), "Quadprog++: a c++ library implementing the algorithm of Goldfarb and Idnani for the solution of a (convex) quadratic programming problem by means of an active-set dual method".

Falsafi, B., Dally, B., Singh, D., Chiou, D., Joshua, J.Y. and Sendag, R. (2017), "FPGAs versus GPUs in data centers", *IEEE Micro*, Vol. 37 No. 1, pp. 60-72.

Fernandez-Camacho, E. and Bordons-Alba, C. (1995), "Introduction to model based predictive control", *Model Predictive Control in the Process Industry*, Springer, London, pp. 1-8.

Goldfarb, D. and Idnani, A. (1983), "A numerically stable dual method for solving strictly convex quadratic programs", *Mathematical Programming*, Vol. 27 No. 1, pp. 1-33.

Goli, M. and Eskandarian, A. (2019), "MPC-based lateral controller with look-ahead design for autonomous multi-vehicle merging into platoon", *2019 American Control Conference (ACC)*, July, IEEE, pp. 5284-5291.

He, M. and Ling, K.V. (2005), "Model predictive control on a chip", *2005 International Conference on Control and Automation*, June, IEEE, Vol. 1, pp. 528-532.

Horowitz, B. and Afonso, S.M. (2002), "Quadratic programming solver for structural optimisation using SQP algorithm", *Advances in Engineering Software*, Vol. 33 Nos 7/10, pp. 669-674.

Jerez, J.L., Ling, K.V., Constantinides, G.A. and Kerrigan, E.C. (2012), "Model predictive control for deeply pipelined field-programmable gate array implementation: algorithms and circuitry", *IET Control Theory & Applications*, Vol. 6 No. 8, pp. 1029-1041.

Jerez, J.L., Goulart, P.J., Richter, S., Constantinides, G.A., Kerrigan, E.C. and Morari, M. (2014), "Embedded online optimization for model predictive control at megahertz rates", *IEEE Transactions on Automatic Control*, Vol. 59 No. 12, pp. 3238-3251.

Jones, D.H., Powell, A., Bouganis, C.S. and Cheung, P.Y. (2010), "GPU versus FPGA for high productivity computing", *2010 International Conference on Field Programmable Logic and Applications*, August, IEEE, pp. 119-124.

Keskin, M.F., Peng, B., Kulcsar, B. and Wymeersch, H. (2020), "Altruistic control of connected automated vehicles in mixed-autonomy multi-lane highway traffic", *IFAC-PapersOnLine*, Vol. 53 No. 2, pp. 14966-14971.

Kestur, S., Davis, J.D. and Williams, O. (2010), "Blas comparison on FPGA, CPU and GPU", *2010 IEEE Computer Society Annual Symposium on VLSI*, July, IEEE, pp. 288-293.

Kuon, I. and Rose, J. (2007), "Measuring the gap between FPGAs and ASICs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26 No. 2, pp. 203-215.

Li, S., Li, K., Rajamani, R. and Wang, J. (2010), "Model predictive multi-objective vehicular adaptive cruise control", *IEEE Transactions on Control Systems Technology*, Vol. 19 No. 3, pp. 556-566.

Lucia, S., Navarro, D., Lucia, O., Zometa, P. and Findeisen, R. (2017), "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool", *IEEE Transactions on Industrial Informatics*, Vol. 14 No. 1, pp. 137-145.

Martin, G. and Smith, G. (2009), "High-level synthesis: past, present, and future", *IEEE Design & Test of Computers*, Vol. 26 No. 4, pp. 18-25.

Mattingley, J. and Boyd, S. (2012), "CVXGEN: a code generator for embedded convex optimization", *Optimization and Engineering*, Vol. 13 No. 1, pp. 1-27.

Nocedal, J. and Wright, S. (2006), *Numerical Optimization. Springer Science & Business Media*.

Nurvitadhi, E., Sim, J., Sheffield, D., Mishra, A., Krishnan, S. and Marr, D. (2016), "Accelerating recurrent neural networks in analytics servers: comparison of FPGA, CPU, GPU, and ASIC", *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, August, IEEE, pp. 1-4.

Qasaimeh, M., Denolf, K., Lo, J., Vissers, K., Zambreno, J. and Jones, P.H. (2019), "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels", *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, June, IEEE, pp. 1-8.

Quan, Y.S. and Chung, C.C. (2019), "Approximate model predictive control with recurrent neural network for autonomous driving vehicles", *2019 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, September, IEEE, pp. 1076-1081.

Russo, L.M., Pedrino, E.C., Kato, E. and Roda, V.O. (2012), "Image convolution processing: a GPU versus FPGA comparison", *2012 VIII Southern Conference on Programmable Logic*, March, IEEE, pp. 1-6.

Schmid, C. and Biegler, L.T. (1994), "Quadratic programming methods for reduced hessian SQP", *Computers & Chemical Engineering*, Vol. 18 No. 9, pp. 817-832.

Winterstein, F., Bayliss, S. and Constantinides, G.A. (2013), "High-level synthesis of dynamic data structures: a case study using vivado HLS", *2013 International Conference on Field-Programmable Technology (FPT)*, December, IEEE, pp. 362-365.

Xu, F., Chen, H., Gong, X. and Mei, Q. (2015), "Fast nonlinear model predictive control on FPGA using particle swarm optimization", *IEEE Transactions on Industrial Electronics*, Vol. 63 No. 1, pp. 310-321.

Yu-Geng, X.I., De-Wei, L. and Shu, L. (2013), "Model predictive control – status and challenges", *Acta Automatica Sinica*, Vol. 39 No. 3, pp. 222-236.

Corresponding author

Shengbo Eben Li can be contacted at: lishbo@tsinghua.edu.cn