Wuhan University
Journal of Natural Sciences

2016, Vol.21 No.2, 171-177

Article ID 1007-1202(2016)02-0171-07
DOI 10.1007/s11859-016-1155-8

A Model-Based Calibration Method of Automotive Electronic Control Unit

☐ **CHENG Anyu, LI Haining, XIONG Liangbo**

Automotive Electronics Engineering Research Center, College of Automation, Chongqing University of Posts and Telecommunications, Chongqing 400065, China

Abstract: This paper presents a systematic method of designing the calibration toolbox of automotive electronic control unit (ECU) based on real-time workshop (RTW). To break the strong coupling of each functional layer, the hierarchical architecture of the calibration system is divided into the bottom driver layer, the intermediate interface layer and the top application layer. The driver functions meeting the specification of the automotive open system are sent and received in the intermediate interface layer. To reduce the development costs, the portable user codes are generated by RTW which provides a development environment from system simulation to hardware implementation. Specifically, the calibration codes yielded from the controller area network (CAN) calibration protocol (CCP) module are integrated into the control codes, called by a compiler in the daemons to build a corresponding project, and then downloaded into the object board to provide the A2L file. The experiments illustrate that the different drive modules are only needed to be replaced for the implementation of the calibration system applied in different hardware platforms.

Key words: calibration system; electronic control unit; hierarchical architecture; real-time workshop
CLC number: TP 217

Received date: 2015-09-30
Foundation item: Supported by the Youth Science and Technology Innovation Talents Project of Chongqing Science & Technology Commission(cstc2013 kjrc-qnrc40005), and the Research Project of Chongqing Education Committee (KJ120511)
Biography: CHENG Anyu, male, Associate professor, research direction: vehicular network and information system. E-mail: chengay@cqupt.edu.cn

0 Introduction

The system calibration is one of the key technologies to effectively design automotive electronic products. The calibrated system with high efficiency and adaptability can greatly improve the design of electronic control unit (ECU). Today, the model-based calibration methods have progressed significantly in terms of theory and applications. There have been increased applications of model-based design and testing in the automotive industry to reduce design errors and perform rapid prototyping [1, 2]. Bifulco *et al* [3] presented a system in which an on-board ECU is required by the driver to calibrate its own parameters after a few minutes' manual drive. The calibration system in Ref. [4] used the calibration board and the serial communication of battery management system (BMS) to perform the calibration of battery management system via the keyboard. And a multi-node calibration system based on controller area network (CAN) calibration protocol (CCP) is developed for a multi-ECU being used by vehicle [5]. In this system, the message buffers, message filter registers and bit timing of identifiers were reasonably configured. The calibration system proposed a method of calibrating multi ECUs simultaneously, which can also make the parameters of different ECUs match better.

The ASAP standards is a set of widely-agreed calibration criterion in industry. As an important component of ASAP standards, the CCP is most commonly used to achieve the communication of host and ECU for the calibration systems based on CAN bus technology. Many well-known companies including Vector and ETAS have their own ECU products supported by the calibration and testing tools adopting CCP. In the entire development

process of calibration system, it is prerequisite that the measurement calibration system (MCS) and applications programme of ECU should be supported by CCP. The implication of the CCP driver and the communication between the calibration tool and ECU are also needed to be further understood during the CCP driver is being incorporated into the ECU [6].

In addition, most of the calibration softwares, such as the CANape developed by Vector Company and the Measurement & Calibration Toolkit developed by NI Company, have more complicated upfront configurations. For example, each ECU needing calibration is required to be configured in the CANape. Although the calibration software offers a tight junction of the CCP driver and the related hardware driver, the combination of the CCP driver and the bottom program is operated manually. In this sense, the workload increase for supplying extra program code of calibration interface and driver due to the strong coupling and poor flexibility of calibration program's architecture [7, 8].

In this work, to solve the inefficient problem mentioned above and reduce the development time and cost of calibration system, we present a systematic method of designing automotive ECU calibration toolbox based on real-time workshop (RTW). The hierarchical architecture of bottom calibration system is constructed by the functionalities of the individual submodules in the calibration program. An intermediate interface layer is designed to improve the versatility and reusability of the bottom calibration program. The driver functions meeting the interface specification of the automotive open system architecture (AutoSAR) are sent and received in this intermediate interface layer. The different drive modules are only needed to be replaced for the applications of the calibration system applied to different hardware platforms. The personalized and portable user codes are generated by RTW which provides a development environment from system simulation to hardware implementation.

1 Design of the Calibration Toolbox

Graphic language is adopted for the design method instead of high-level abstract language for the development of calibration toolbox. Specifically, in the visualization environment of MATLAB, each module of the bottom calibration system is encapsulated into separate module by the C-MEX S-functions in a manner of the customized signal flow graph. By means of calling the template target file, the control codes are automatically generated by the RTW tool, then downloaded to the hardware platform for achieving the design and debugging of calibration program.

The block diagram of designing the calibration toolbox is shown in Fig. 1. From Fig. 1, it is clearly drawn that the designed work generally includes three aspects: the analysis of the hierarchical architecture in ECU calibration system, the model design based on S-function and the customization of module object file.

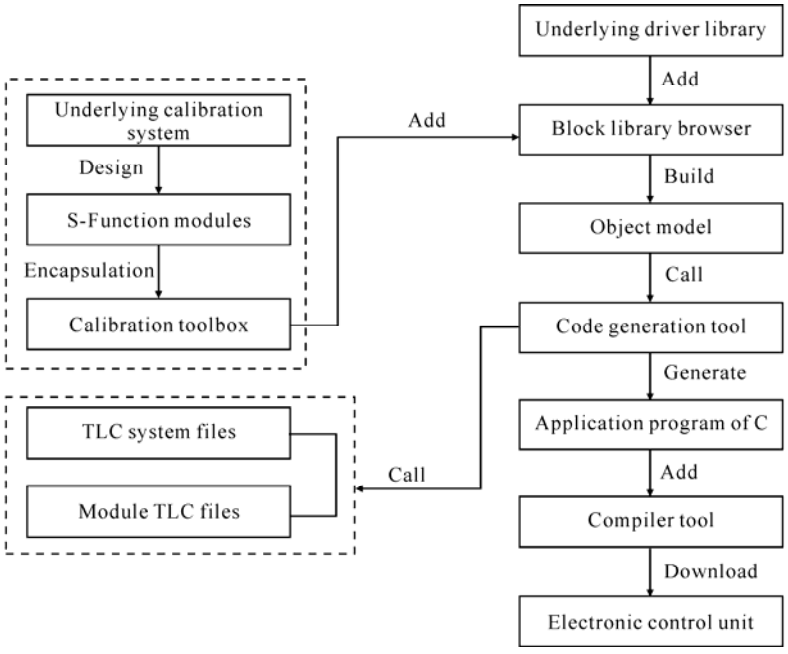

Fig. 1 Block diagram of the proposed design method

1.1 Hierarchical Architecture of ECU Calibration System

According to the functionalities of the individual submodules in ECU, the hierarchical architecture of bottom calibration system, as shown in Fig. 2, is divided into three levels: the bottom driver layer, the intermediate interface layer and the top application layer.

In the bottom driver layer, the bottom programs of a certain micro-controller, such as the module initialization program, the signal acquisition and output program, are written here in accordance with the corresponding situation of the interface and the relationship between input and output signal. The intermediate interface layer mainly plays a role of providing the interfaces for the CAN application. The CAN messages passed down from the upper calibration system are accepted and parsed out the CCP messages by the "CAN Receive Function". The CCP messages are submitted to the "Command Processor", then re-packaged as CCP packets being fed to the "CAN Transmit Function" for further encapsulation of CCP. At last, the encapsulated CCP is sent to the upper calibration system.

The purpose of the top application layer is to achieve the calibration of the ECU control parameter and upload these monitoring parameters. The "CCP Driver", as one of the "CAN Driver" users, is responsible for communication of the calibration software and PC using the CAN function [9].

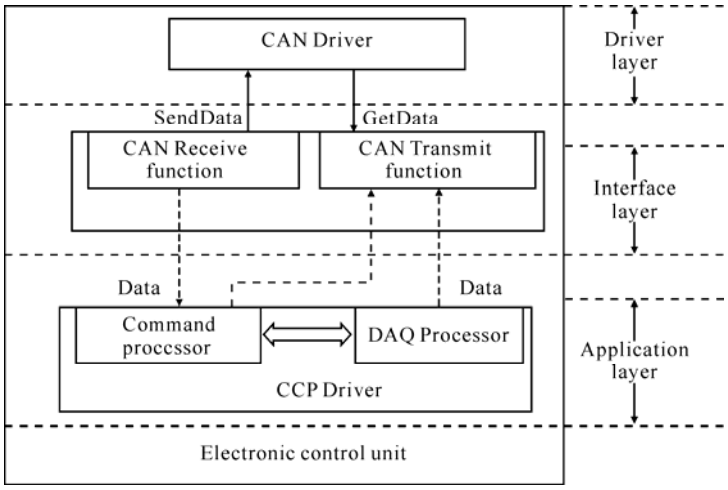

Fig. 2 Hierarchy of the ECU calibration system

1.2 Model Design Based on S-Function

The S-function, an autonomous file interface in MATLAB, is always employed to build the simulink module with a special function calls. Both of M-files and MEX-files are the mainly implemented ways of the S-function. Particularly, the C-MEX S-functions have many advantages, such as a common language specification, high execution speed, being called by any open source code. The common language specification (CLS), which is a set of basic language features needed by many applications, has been defined. The CLS rules define a subset of the common type system; that is, all the rules that are applied to the common type system which is applied to the CLS. These advantages make the C-MEX S-functions much more suitable for the hardware development.

The S-function, an autonomous file interface in MATLAB, is always employed to build the simulink module with a special function calls. Both of M-files and MEX-files are the mainly implemented ways of the S-function. Particularly, the C-MEX S-functions have many advantages, such as a common language specification, high execution speed, being called by any open source code. The common language specification (CLS), which is a set of basic language features needed by many applications, has been defined. The CLS rules define a subset of the common type system; that is, all the rules that are applied to the common type system which is applied to the CLS. These advantages make the C-MEX S-functions much more suitable for the hardware development.

If the models in MATLAB do not meet user requirements, the S-function can be programmed according to the module parameters and the input or output ports. Then an encapsulation of S-function module is achieved by the configuration of the Mast Editor parameter dialog. Finally, a customized and personalized interface module is displayed in the simulink block library browser. Figure 3 gives the encapsulated process of the S-function modules.

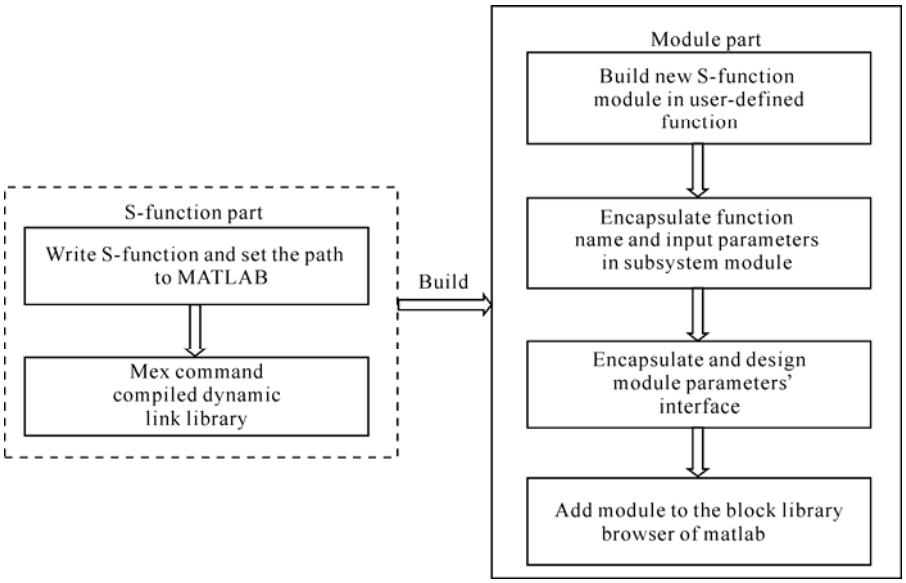

Fig. 3 Encapsulated process of the S-function modules

1.3 Customization of Module Object File

After the encapsulated process of the S-function modules is completed, the customized module programs are embedded into the source codes being generated automatically. The module file, so called target language complier (TLC), is programmed to customize the code generation template of each sub-model in the model library. The TLC file contains four functions: "BlockTypeSetup", "Start", "Outputs" and "Terminate".

This section takes "CCP Block" in calibration toolbox as an example to illustrate the method of creating the TLC file. The primary purpose of "CCP Block" module is getting the channel number of CAN, the baud rate, the message ID of the command receive object (CRO) and the data transfer object (DTO). Such obtained knowledge is delivered to the CAN transmitting and receiving modules for initialization of the CCP driver in the main function. The "BlockType-Setup" function, before the starting of the code generation, is executed only once in each module. The general operations of each module of a given type, such as the macro declaration, the function prototype declaration, the header file contains (# include), the data type alias (typedef), etc., are performed by this "BlockTypeSetup" function. The exact procedure about the "BlockType-Setup" function is shown as follows:

```
Line 1:  %function BlockTypeSetup ( block,
         system ) void
Line 2:  %%Used to determine how many CCP
         modules exist
Line 3:  %assign::CCP BlockTotal = 0
Line 4:  %assign hFile = LibGetModelDotHFile()
Line 5:  %openfile tmpBuf
Line 6:  #include "Platform Types.h" ⇒ Templates
         generation
Line 7:  %closefile tmpBuf %LibSetSource
         FileSection (hFile, "Includes", tmpBuf)
Line 8:  %assign hFile = LibGetModelDotHFile ()
Line 9:  %openfile tmpBuf
Line 10: void ccpInit(uint8 id )
Line 11: %closefile tmpBuf
Line 12: %LibSetSourceFileSection(hFile,
         "Functions", tmpBuf)
Line 13: %endfunction
```

The "Line 2"，started with the specific character "%%" ，is the annotation of the "BlockType-Setup" function. The "Line 3", in the aforementioned code segment, defines a variable named CCP BlockTotal being assigned to 0. The "Line 4" employs the function "LibGetModelDotHFile ()" to create a header file with the same name of the model ".mdl". The "Line 5" denotes creating a text buffer to temporarily store the code generation templates of a module designed by its functionality. The text buffer is turned off in "Line 11". The "Line 12" denotes that the contents of the specified templates in the buffer are written into the "Function" section of the corresponding header file. For the section of the "Start" function, the programme is given as follows:

```
Line 1:  %function Start ( block, system ) Output
Line 2:  %assign:: CCP_BlockTotal
          = CCP_BlockTotal+1
Line 3:  %if ( CCP_BlockTotal == 1 )
```

```
Line 4:  %%Initialize the CCP module only once
Line 5:  ccpInit ( %SFcnParamSettings.Can DAQ
         ID ) ⇒Initialization function calls
Line 6:  %endif
Line 7:  %endfunction
```

In the code segment mentioned above, the "Line 2" represents a value growing of the variable CCP Block-Total. This value will be increased by 1 if each CCP module appears once in the model. The judge sentence in "Line 4" ensures that the CCP is initialized only once by the generated code. So far, the TLC files of the CCP initialization module have been created. The interface diagram of the CCP module is plotted in Fig. 4.

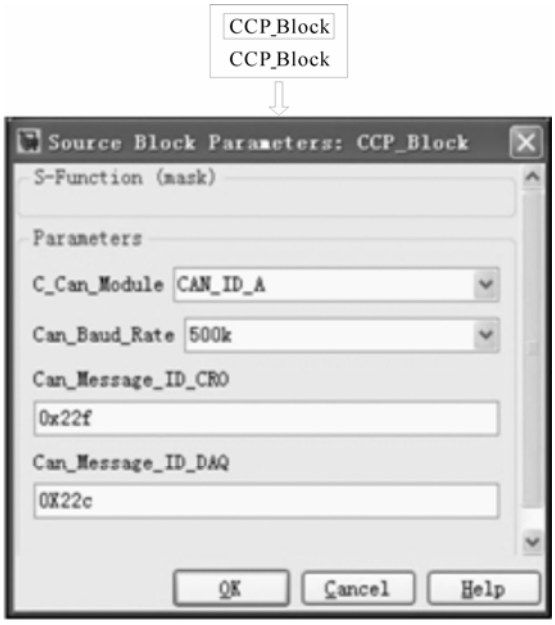

Fig. 4 Interface diagram of CCP module

It is noteworthy that we dealt with only both of the "BlockTypeSetup" and "Start" functions in this example. According to the needs of the users, "Outputs" and "Terminate" functions are also used in the TLC module. The corresponding TLC files of the modules in calibration toolbox can be customized in the aforementioned manners.

2 Experiments

In this section, the Freescale MPC5634 board is selected as our object board. The CCP module, in our calibration toolbox, is combined with other modules to a simple calibrated model for verifying the effectiveness of the proposed calibration method.

The system model built as "demone.mdl" has the following desired features. The conversion value of the 21th channel in the ADC module is amplified by $k1$. This amplified conversion value, namely input1, is used to gain the frequency of the PWM waveform exporting from the 8th channel. The duty cycle of the PWM waveform is obtained by searching the values of input1 amplified via $k2$. Here, $k1$ and $k2$ are the calibrated variables in a presetting table. In addition, the frequency and the duty cycle of the PWM waveform exporting from the 14th channel are determined by the conversion value of the 22th channel in the ADC module and the constant module in the Simulink block library, respectively.

The hardware of the calibration system consists of the engine ECU (the object board), the USBCAN adapter and the PC host computer. In the process of generating the codes automatically, the calibration codes yielding from the CCP module are integrated into the control codes, called by the "Codewarrior" compiler in the daemons to build the corresponding project, and then downloaded into the object board. Meanwhile, the A2L file meeting the specifications ASAP are generated automatically by the RTW. The related parameters in the A2L file are given by Fig. 5.

By imputing the A2L file into the calibration system, the calibration and monitoring parameters can be observed in the client interface of the host computer. The client interface of the ECU calibration system is shown in Fig. 6. The aforementioned client interface takes the calibration of the pulse width of fuel injection as an example. Because the pulse width of fuel injection, as the most important part of the engine ECU calibrated parameters, affects the air-fuel ratio directly, injection quantity in different conditions can be determined [10]. According to the engine performance parameters provided by the manufacturer, the testing range of the rotational speed are selected between 1 600 r/min and 4 600 r/min. The related testing interval is selected as 200 r/min. For each rotational speed, the absolute pressures of the intake manifold are increased from 30 kPa to 100 kPa. The corresponding additional interval is 7 kPa. Therefore, the engine operating conditions are divided into 176 kinds of situations.

As shown in Fig. 6, the mapping graph of the pulse width for fuel injection is magnified to illustrate the comprehensive details. In the magnified mapping graph, the blue-axis and the red-axis denote the engine rotational speed and the intake manifold pressure, respectively. The detailed values of the pulse width in at different rotational speeds and absolute pressures P are given in Table 1.

Name	gEnSpdMi
Long identifier	"Engine speed"
Characteristic type	VALUE
Memory address	40001640
Record layout	Scalar FLOAT64 IEEE
Maximum difference	0
Conversion method	COMPU METHOD 1
Lower limit	1 000.0
Upper limit	5 000.0

(a) Characteristics of gEnSpdMi

Name	gWaterTemp
Long identifier	"Engine water temp."
Data type	FLOAT64 IEEE
Conversion method	COMPU METHOD 1
Resolution (Not used)	0
Accuracy (Not used)	0
Lower limit	−20.0
Upper limit	20.0
ECU ADDRESS	400016b0

(b) Measurement of gWaterTemp

Fig. 5 Related parameters in A2L file

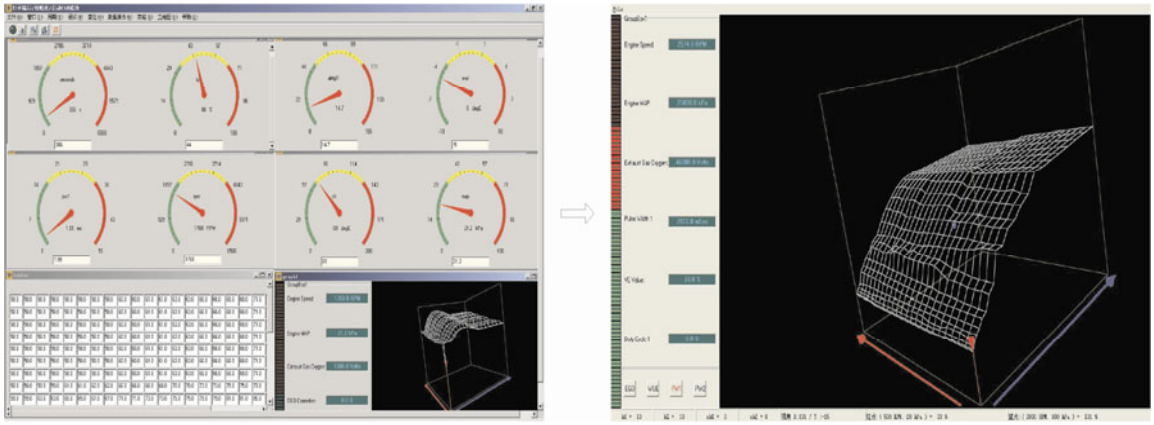

Fig. 6 Hierarchy of the ECU calibration system

Table 1 Detailed values of the pulse width for fuel injection

ms

P/kPa	Rotational speed / 10^3 r • min^{-1}															
	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0	3.2	3.4	3.6	3.8	4.0	4.2	4.4	4.6
30	5.6	5.5	5.4	5.3	5.2	5.0	5.4	5.0	5.5	5.9	6.1	6.1	6.1	6.2	6.2	6.2
37	8.9	8.4	7.9	7.4	7.6	7.1	7.7	7.1	8.0	9.0	8.8	8.8	8.8	8.7	8.7	8.7
44	10.3	12.1	11.6	11.4	11.2	10.4	10.8	10.5	11.1	12.0	11.7	11.7	11.8	11.9	11.9	11.9
51	11.8	15.0	14.3	13.8	13.2	12.0	13.3	12.1	13.2	14.8	14.4	14.4	14.4	14.5	14.6	14.6
58	13.9	16.0	16.2	16.2	16.2	16.2	15.9	15.8	16.4	17.1	16.8	16.8	16.8	16.9	17.0	17.0
65	14.2	16.3	16.5	16.0	16.8	16.0	16.2	16.5	17.0	17.5	18.0	18.2	18.5	18.6	18.7	18.7
72	14.4	17.0	17.4	17.3	17.5	17.9	17.5	18.5	19.0	19.5	19.0	19.0	19.0	19.0	19.1	19.1
79	14.6	16.8	17.2	17.1	17.3	17.0	17.5	19.1	20.1	20.4	20.0	20.0	20.0	20.1	20.1	20.2
86	14.3	16.7	17.3	17.4	17.1	17.0	17.9	19.3	20.0	21.0	20.5	20.3	20.1	20.5	20.3	20.4
93	14.4	16.8	17.3	17.3	17.2	17.3	18.0	19.4	20.0	20.4	21.0	21.0	20.2	20.1	20.5	20.5
100	14.4	16.7	17.3	17.3	17.2	17.2	18.0	19.5	20.0	20.5	20.5	20.3	20.2	20.3	20.6	21.2

3 Conclusion

This paper has focused on the scenario to build a calibration toolbox systematically for the automotive ECU. The idea behind the proposed calibration method is the modular designing and RTW-based programming.

This new developed method, compared with the traditional calibration methods, can improve the reusability and portability of the bottom calibration program, and reduce the coupling nature and the developed cycle

of the calibrated software. In addition, the designed calibration toolbox is also applicable for the ECUs with different hardware interfaces. Especially when the bottom hardware changes, the corresponding driver modules are only required to be replaced for the calibration. In this sense, the efficiency and the adaptability of the calibration system are enhanced significantly.

References

[1] Luo J, Krishna R P, Liu Q, *et al*. An integrated diagnostic development process for automotive engine control systems [J]. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, *Part C*: *Applications and Reviews*, 2007, **37**(6): 1163-1173.

[2] Klaus D M, Gerd F, Eric S, *et al*. Multiparadigm modeling in embedded systems design [J]. *IEEE Transactions on Control Systems Technology*, 2004, **12**(2): 279-292.

[3] Bifulco G N, Pariota L, Simonelli F, *et al*. Development and testing of a fully adaptive cruise control system [J]. *Transportation Research Part C*: *Emerging Technologies*, 2013, **29**(4): 156-170.

[4] Chatzakis J, Kalaitzakis K, Voulgaris N C, *et al*. Designing a new generalized battery management system [J]. *IEEE Transactions on Industrial Electronics*, 2003, **50**(5): 990-999.

[5] Yang S, Yang L, Zhuo B. Developing a multi-node calibration system for can bus based vehicle [C] // *Proc of IEEE International Conference on Vehicular Electronics and Safety*, Piscataway N J: IEEE Press, 2006: 199-203.

[6] Cen M, Yan Y, Dai H. General calibration system architecture of automotive electronic control unit [J]. *Journal of Computers*, 2010, **5**(12): 1894-1898.

[7] Wong P K, Tam L M, Ke L. Automotive engine power performance tuning under numerical and nominal data [J]. *Control Engineering Practice*, 2012, **20**(3): 300-314.

[8] Beham M, Etzel M, Yu D L. Development of a new automatic calibration method for control of variable valve timing [J]. *Proceedings of the Institution of Mechanical Engineers*, *Part D*: *Journal of Automobile Engineering*, 2004, **218**(7): 707-718.

[9] Wang X, Waschl H, Alberer D, *et al*. A design framework for predictive engine control [J]. *Oil & Gas Science and Technology-Revue d'IFP Energies Nouvelles*, 2011, **66**(4): 599-612.

[10] Deng J, Winward E, Stobart R, *et al*. Modeling techniques to support fuel path control in medium duty diesel engines [J]. *SAE Technical Paper*, 2010, (1): 332-336.

□