Study on Engine Control Software Testing Based on Hardware-in-the-Loop Simulation Platform

Wenwen Zeng, Ying Huang, Xuelong Zheng and Wenqiang Zhao

1 Introduction

Traditional ECU development methodologies have been gradually eliminated, and V-cycle development mode, a model-based development mode mainly including control function design, rapid control prototyping, target code generation, HIL, and calibration as shown in Fig. 1, is recently most favored by vehicle developers [1–3]. Typically, off-line simulation and HIL are used as development methodologies according to different test requirements in different test stages. Normally, the control logic and control algorithm as well as the switching and coordination between various engine operation conditions can be verified through off-line simulation [4]. Different from logical connection in off-line simulation, corresponding interface programs are used in HIL to connect the real ECU and the digital engine model in the processor [5]. The virtual engine, which provides actuator signals for ECU, precisely simulates various sensor signals and to the greatest extent makes the ECU run in a real working environment of engine.

Notice that the dominant merit of HIL is that it can be used for software testing, and it is an efficient methodology for testing software functions and achieving parameter pre-calibration. Besides, software testing is extremely necessary, while the engine control system becomes increasingly complex [6–8].

Referring to the mature software testing methodology classification, engine control software testing technique is classified into many types as shown in Fig. 2 [9]. Typically, it is divided into static test technique and dynamic test technique

W. Zeng · Y. Huang (✉) · X. Zheng · W. Zhao
College of Mechanical and Vehicle Engineering, Beijing Institute of Technology, 100081 Beijing, China
e-mail: hy111@bit.edu.cn

W. Zeng
e-mail: 936754111@qq.com

Society of Automotive Engineers of China (SAE-China) (ed.), *Proceedings of the 19th Asia Pacific Automotive Engineering Conference & SAE-China Congress 2017: Selected Papers*, Lecture Notes in Electrical Engineering 486, https://doi.org/10.1007/978-981-10-8506-2_67

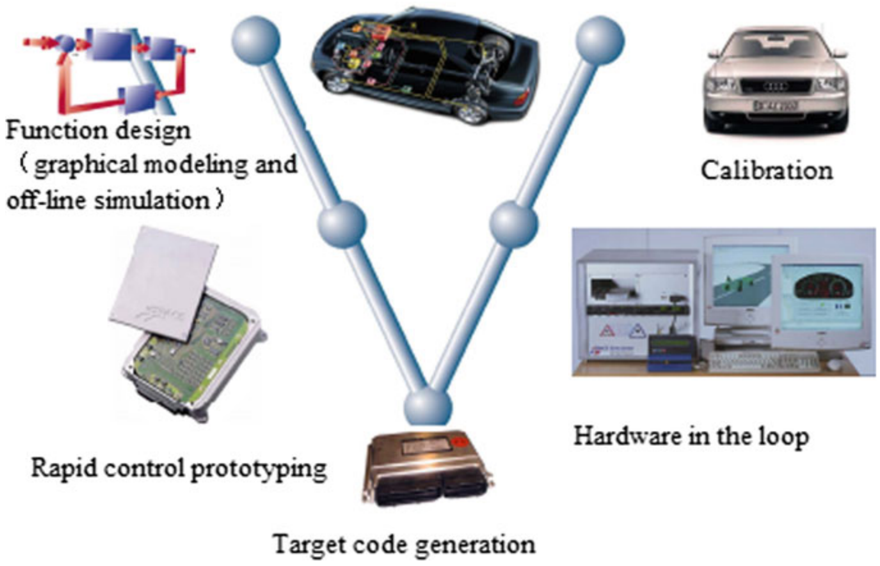

Function design
(graphical modeling and off-line simulation)
Calibration
Rapid control prototyping
Hardware in the loop
Target code generation

Fig. 1 V-cycle mode in vehicle development

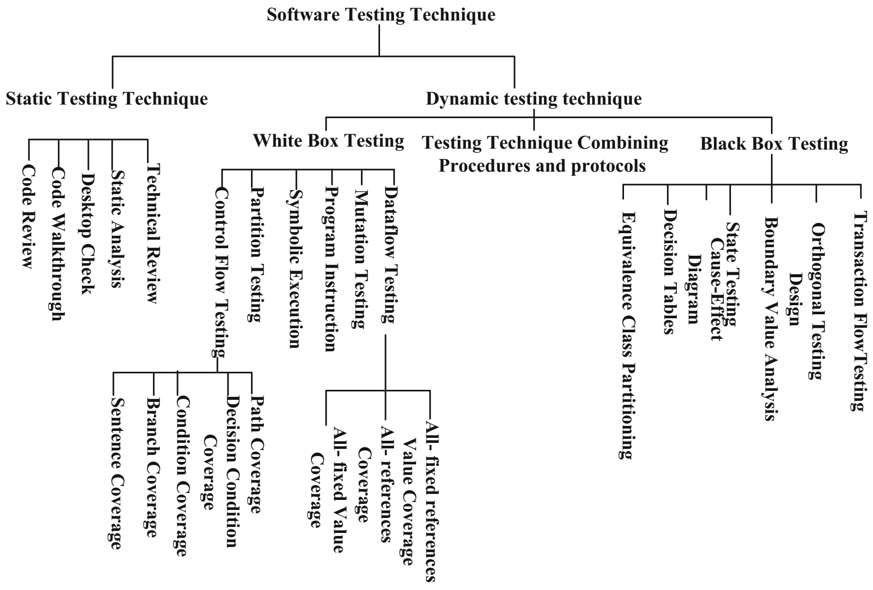


Fig. 2 Classification of software testing technique

depending on whether the software runs dynamically, and dynamic test technique is the most commonly used test methodology [10, 11]. As one of the branches of dynamic testing technique, black-box testing is divided into methodology of equivalence class partitioning and methodology of boundary value, etc. [12–14]. When the black-box testing is performed, the whole tested program is regarded as a sealed black box. To be more specific, the only information available for testers is the software function specification, and the testing can be accomplished by comparing software function specification and the testing results [15, 16].

In this paper, the HIL-based test system was developed by Beijing Institute of Technology. The engine control software function requirements and test requirements were analyzed, and the test items were extracted through considering the coverage ratio and difficulties such as the limiting and dangerous conditions which are hard to be achieved in real engine test bench. In the test case distribution strategy, the reasonable test case distribution of the engine control software involves two procedures: At the first stage, the first-time test case distribution was conducted based on the matrix distribution methodology, and then the test results were analyzed to guide the redistribution of the test cases. For the design of test cases, appropriate black-box test methods such as methodology of equivalence class partitioning, methodology of boundary value, and methodology based on scenario were used. In addition, for the automatic testing of software, the test scripts were designed according to different test cases. Finally, several test results were analyzed in detail.

2 HIL-Based Test System

In order to implement HIL-based software testing, AutoBox, which was developed by dSPACE Company, was selected as the real-time platform to run the engine model. When the software testing is performed, the real-time engine model and electronic control unit (ECU) exchange data through I/O interfaces, which replace the logical interfaces used in off-line simulation. The HIL system supports the implementation of various scheduled ECU software testing in a virtual environment, and the testing results are close to or even equivalent to what is achieved in engine's real working environment.

Figure 3 shows the schematic diagram of HIL-based software testing system. The hardware environment includes PC1, PC2, emulator AutoBox, sensor signal processing board, actuator signal processing board, and the engine controller. The software environment includes engine mean-value model, engine control software, ControlDesk, AutomationDesk, and CANape.

CANape, a software which is used to download the engine control software into the controller, is installed on PC1. Both software ControlDesk and automatic testing software AutomationDesk are installed on PC2. ControlDesk is used to download the engine mean-value model into the real-time emulator, and AutomationDesk is

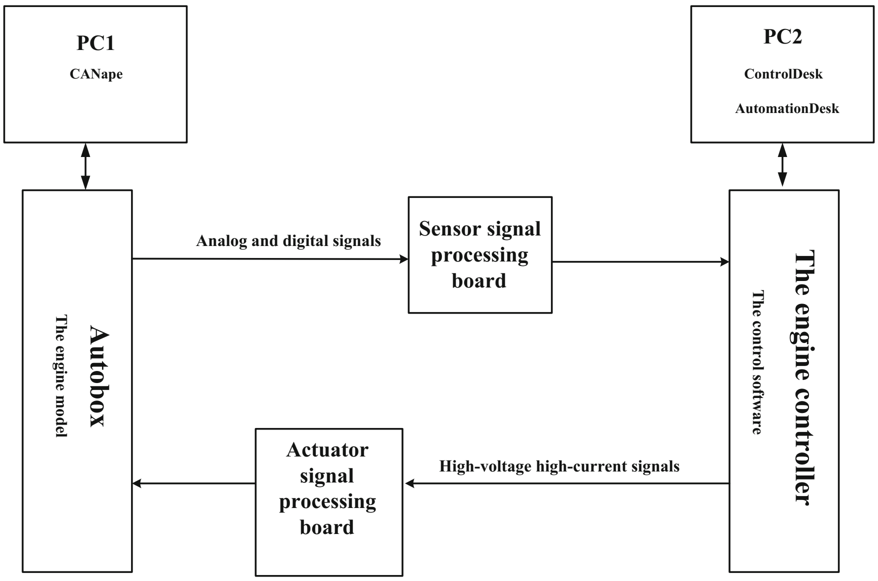

PC1
CANape
PC2
ControlDesk
AutomationDesk
Autobox
The engine model
Analog and digital signals
Sensor signal processing board
The engine controller
The control software
Actuator signal processing board
High-voltage high-current signals

Fig. 3 Schematic diagram of HIL system

used to design and implement test script files. Emulator AutoBox and the engine controller are used to run engine model and the control software in real time, respectively. Analog and digital signals generated by AutoBox are transformed into signals that can be identified by the controller through sensor signal processing board; meanwhile, the high-voltage and high-current driving signals generated by the engine controller are transformed into signals that can be identified by AutoBox via actuator signal processing board. The engine mean-value model, which has a reasonable fidelity and excellent real-time performance, was built by MATLAB/Simulink. So did the engine control software. They interact via I/O interfaces and CAN bus to constitute a real-time closed-loop test system.

3 Test Item Extraction

Test items play an important part in software testing; they correspond to the software function points and can be extracted through the test requirement analysis. Besides, test items ensure the integrity of the software function testing as well as the test coverage ratio. Typically, the test coverage ratio is divided into test code coverage ratio and test requirement coverage ratio, and the test coverage ratio is equivalent to test requirement coverage ratio in terms of software function testing.

3.1 Extraction Strategy

In general, test requirement coverage ratio is determined by congruent relationship between software function requirements and test requirements, if all the software function requirements have been established congruent test requirements, the test requirement coverage ratio is regarded as 100%. Because the test items are extracted through test requirement analysis, then the test item coverage ratio will be 100% too.

3.2 Extraction

Figure 4 illustrates the control logic, depending on which the control software can realize the control functions. Basic control function points include stop condition control function, start condition control function, idling condition control function,

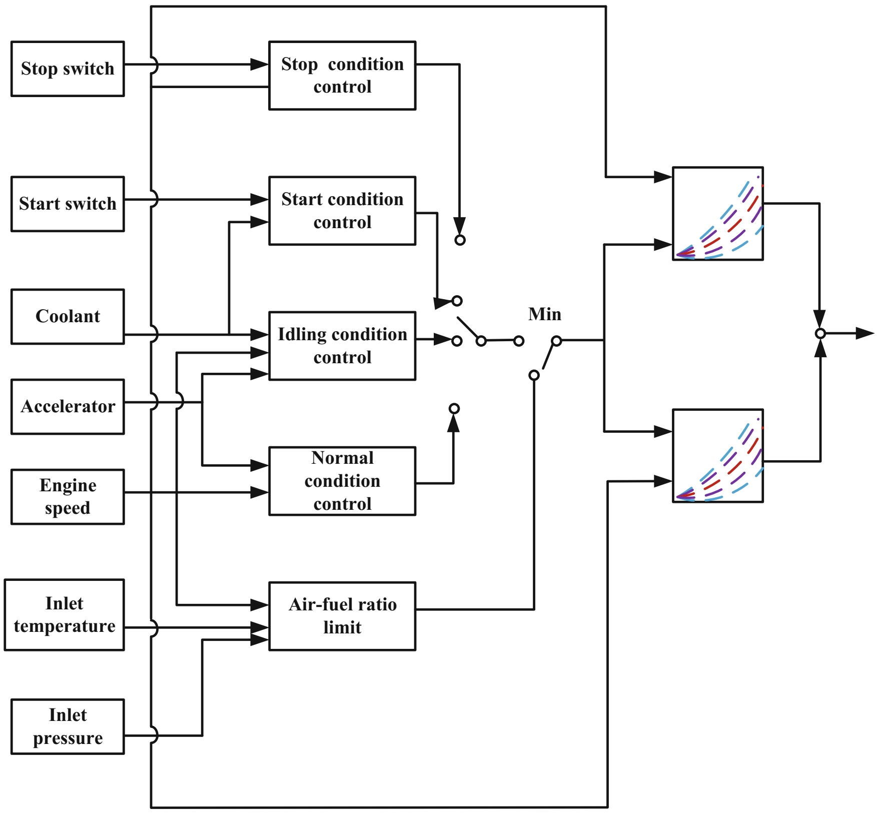


Fig. 4 Diagram of control software function

normal condition control function, and air–fuel ratio limit function. Besides, analog signal acquisition function, digital signal acquisition function, fuel injection pulse function, and data transmission function are also taken into consideration in this study.

Through full analysis for the control software function requirements, the corresponding relationship between software function requirements and test requirements is established to ensure the full coverage of the all function points. Nevertheless, test cases extracted directly on test items could hardly satisfy the detailed test requirements of large-scale or complex function points; most possibly, there are testing omissions that can affect the test adequacy.

Here, only the signal acquisition function and data transmission function are relatively simple and no complex control logic is included, so one test point is enough. However, some large-scale software function points such as start control function and idling control function need to be subdivided into more test points; refer to Table 1.

4 Test Case Distribution and Design

After the extraction of test items, specific test cases for each test point need to be designed, according to which software testing is to be implemented afterward. Notice that the number of test cases should be reasonable; too many test cases could result in a waste of labor and financial resources of programmers, but not enough test cases could lead to testing omissions; thereby, reasonable distribution and design of test cases is the key to improve testing efficiency. Usually, the optimal number is determined by the scale of the software as well as the demands of testers and test time.

Table 1 Test item extraction

Test Items	Test points
Analog signal acquisition	Data acquisition
Digital signal acquisition	Data acquisition
Start condition	Data acquisition, look-up table function, logical judgement
Idling condition	Data acquisition, look-up table function, logical judgement, complex calculation
Normal condition	Data acquisition, look-up table function, logical judgement, complex calculation
Stop condition	Data acquisition, logical testing function
Air–fuel ratio limit	Data acquisition, look-up table function, logical judgement, complex calculation
Fuel injection pulse	Data acquisition, look-up table function
Data transmission	Data acquisition

4.1 Test Case Distribution Strategy

In this study, assuming that the number of test cases grows linearly with the software scale, the complexity and other factors of the engine control software are neglected, only the scale of the software is taken into account. The distribution of the test case number is based on matrix distribution methodology. Specifically, when test cases are distributed preliminarily, test items such as data transmission, fuel injection pulse, air–fuel ratio limit, idling condition control, and signal acquisition are taken as rows of the matrix, and the basic test points such as look-up table function, logic judgement, complex calculation, and data acquisition are taken as columns of the matrix. The total number of test cases required is calculated using Eq. (1):

$$Q_A = \sum S_i * R_i \tag{1}$$

where Q_A is the total number of test cases, i (i = 1,…,n) is the number of function modules after software partition, S_i is the scale of the ith function module, and R_i is the test case density of the ith function module. In general, test case density varies with number of testing participants, permitted time as well as testing scales. The number of test case predicted by the cells in row i and column j of the matrix is calculated using Eq. (2):

$$QC_{ij} = S_i * R_i * W_{ij} \tag{2}$$

where QC_{ij} is the number of test cases in row i and column j of the matrix, W_{ij} is the weight of basic test point in column j for row i of the matrix. Notice that the sum of the weight of all basic test points for each row is 100%, and each weight of the basic test points in each row can be different. In addition, values of S_i and R_i are equivalent to the values defined in the model. Through matrix distribution methodology, the number of test cases can be distributed reasonably.

4.2 Test Case Distribution

The first-time distribution of test cases was shown in Table 2, the total number is calculated by matrix distribution methodology according to software scale and test case density of each test item.

In this study, after the implementation of test cases designed according to test points, the failed test cases were filled into the failed cases matrix, which has the same structure with the first-time distribution matrix. Figures 5 and 6 show the proportion represented by failed test cases based on the rows and columns of the matrix, respectively.

Table 2 First-time distribution of test cases

Test items	Scale	Density	Total number	Data acquisition	Logical judgement	Look-up table function	Complex calculation
Accelerator pedal	10	0.1	1	1			
Coolant temperature	10	0.1	1	1			
Inlet temperature	10	0.1	1	1			
Inlet pressure	10	0.1	1	1			
Crankshaft signal	150	0.01	2	2			
Camshaft signal	100	0.01	1	1			
Start condition	50	0.1	5	2	2	1	
Idling condition	100	0.09	9	3	4	1	1
Normal condition	100	0.07	7	2	2	1	2
Stop condition	30	0.2	6	3	3		
Air–fuel ratio limit	60	0.1	6	2	2	1	1
Fuel injection pulse	80	0.05	4	2		2	
Data transmission	120	0.01	2	2			

The first-time distribution of test cases is based on software scale and test case density; however, this simple distribution is not enough for a full-scale software testing. The redistribution of test cases is based on the error distribution characteristics which determine whether to continue testing after the first-time distribution. If the test case increase is necessary, the redistribution strategy based on the analysis results can be utilized to determine the distribution of increased test cases.

Taking the engine speed acquisition function testing as an example, the engine speed acquisition is based on a wide engine speed range, from low speed to high speed. In essence, in the process of engine speed acquisition, the speed is calculated by only one CPU frequency division in the bottom program of the engine controller, resulting in obvious accuracy variant in different speed segments. Thus, for the detailed testing of speed accuracy calculation function in each speed segment, extra test cases are introduced to speed acquisition function to make the test cases cover all speed segments.

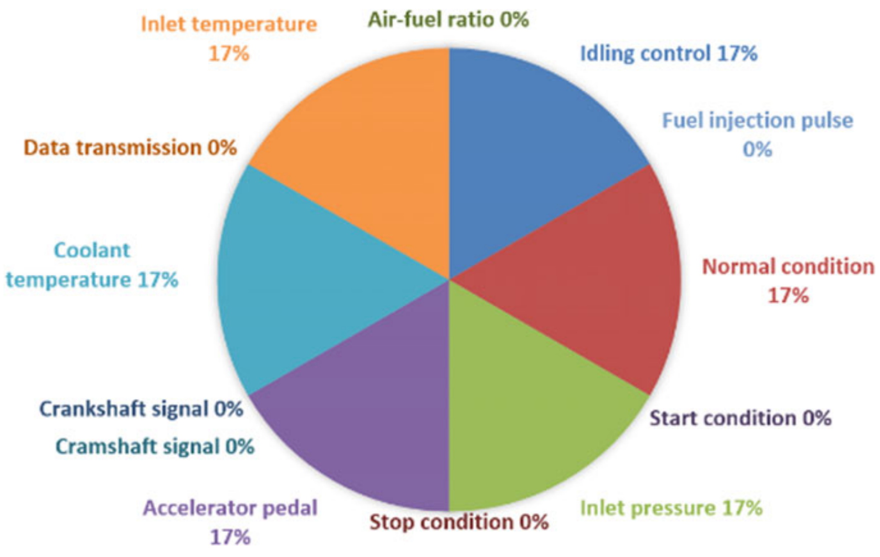


Fig. 5 Failed test case distribution based on test items

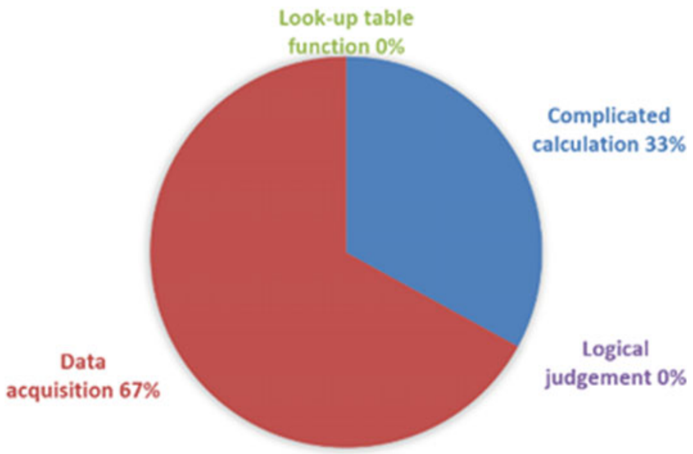


Fig. 6 Failed test case distribution based on test points

4.3 Test Case Design Strategy

Since the engine control software testing is a kind of function testing, the detailed realization process of software function could be completely ignored, so the black-box testing technique is adopted in this paper. Three typical testing methods of black-box testing technique used in this paper are illustrated as follows.

4.3.1 Methodology of Equivalence Class Partitioning

As one of the typical black-box testing methods, methodology of equivalence class partitioning ignores the internal structure of the program and partitions the inputs according to the software specification. To be specific, it divides all possible input data into a number of disjoint subsets, and then selects a small number of representative data from each equivalence class subset as test cases. In this manner, the test points can be remained in a small scale, avoiding heavy workload for software testers.

4.3.2 Methodology of Boundary Value

Different from the methodology of equivalence class partitioning, the methodology of boundary value, which aims at boundary values instead of any internal element within the input and output range of equivalence classes, selects the boundary value of the equivalent class to design the test cases. Besides, it is a supplement of equivalence class partitioning since errors often occur near boundary values. Figure 7 shows the schematic of methodology of equivalence class partitioning and methodology of boundary value.

4.3.3 Methodology Based on Scenario

The scenario is actually the flow of affairs, including elementary streams and alternative streams as shown in Fig. 8. The transaction-driven approach is used to drive the software, and the scenario is developed once the scene is triggered by an event.

From the view of software users, testers simulate ideas of designers by analyzing the design, imagine the operation order to use the software in practice, and list such possibilities as test points one by one; in this way, defects of the software are easier to be found out.

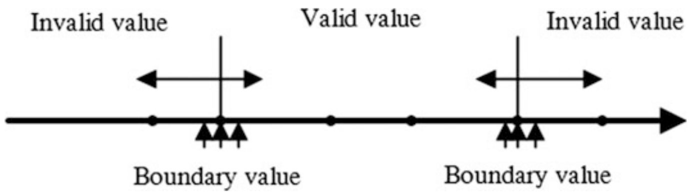


Fig. 7 Schematic of equivalence class partitioning and boundary value

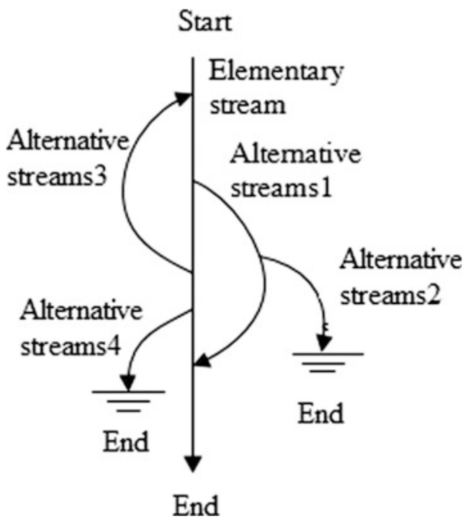


Fig. 8 Schematic of methodology based on scenario

4.4 Test Case Design of Idling Control Function

A complete test case contains the test item name, test methodology, test content, test input, expected output, and judgement criteria. Combined with the test requirement analysis, the test case design of idling control is taken as an example.

The control strategy for idling condition control function is relatively complicated as shown in Fig. 9. The controller judges whether the engine is running at the idling condition according to the accelerator pedal position signal collected from accelerator pedal position sensor and the engine speed signal collected from engine speed sensor. The basic values of the target idling speed, which decrease as the coolant temperatures increase, is obtained by looking up MAP using coolant temperature. Furthermore, the application layer obtains the difference between engine speed and target idling speed via speed contrast, and then inputs it into the

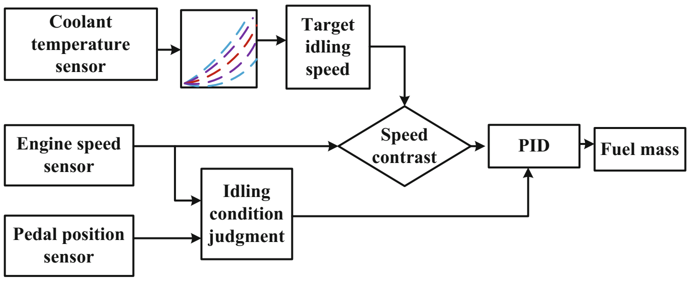


Fig. 9 Control strategy for idling conditions

PID module, through which can calculate out the fuel mass and adjust the engine speed stabilized near the target idling speed.

In this case, the methodology based on scenario is suitable, and test case scenario of idling condition control function is determined by the combination of elementary streams and alternative streams. Fig. 10 shows the flowchart of the test case. First, start the engine, check if the engine status is 2 (i.e., the engine is running at idling condition) when the speed is higher than 500 r/min; if so, check if the target idling speed value is consistent with the expected idling value; and if so, check whether the fuel mass calculated by the PID algorithm is correct; and if so, check whether the error between engine speed and the target idling speed is reasonable when the engine speed is stabilized. Notice that the allowable error between engine speed and the target idling speed is less than 10 rpm, and the test failed if any of the steps above not passed.

As for the detailed testing of look-up table function, both methodology of equivalence class partitioning and methodology of boundary value are indispensable. Notice that the range of input coolant temperature is 0–100 °C in the MAP, and nine values are selected, namely −50, −1, 0, 1, 50, 99, 100, 101 and 150 °C, which include the invalid values, valid values, and near boundary values, covering all the possibilities. In the process of testing, the inputs are set as the values defined above, respectively, and then the output target idling speed values are measured whether they are consistent with expected idling values.

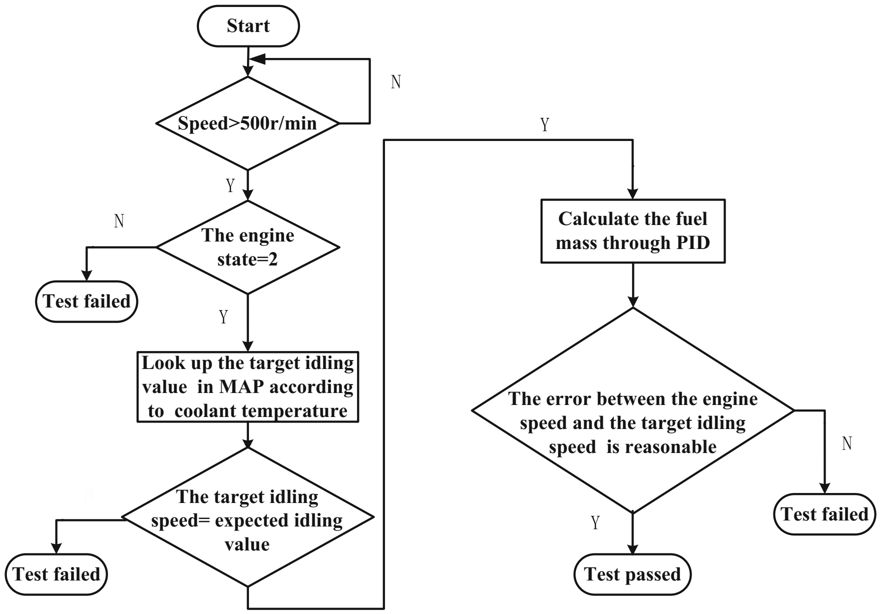

Start
Speed>500r/min
N
Y
The engine state=2
N
Test failed
Y
Look up the target idling value in MAP according to coolant temperature
The target idling speed= expected idling value
Test failed
Y
Calculate the fuel mass through PID
The error between the engine speed and the target idling speed is reasonable
N
Test failed
Y
Test passed

Fig. 10 Flowchart of idling control based on scenario methodology

5 Automatic Software Testing

HIL testing is a critical step in the development and validation of controller,and the aim of function testing is to find out defects in software system. Usually, manual and automatic software testing methods are both used; however, because the constantly improving complexity and frequently updating versions of the software bring heavy and repetitive workload to manual testing, the automatic software testing is used more.

5.1 Development of Automatic Test Scripts

AutomationDesk, a software tool based on dSPACE, is used to design the test scripts. This software tool can not only ensure the consistency of test cases in the regression test but also avoid the repetitive workload in manual testing.

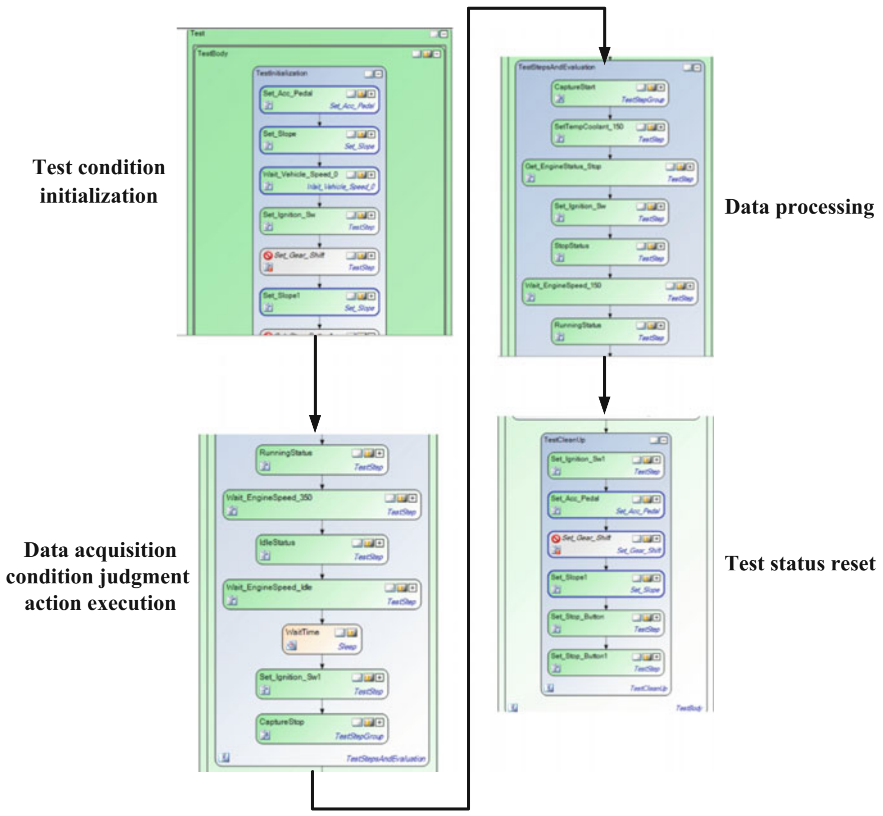


Fig. 11 Test script of automatic testing

Figure 11 shows the test script of idling condition control function, and each test script basically includes test condition initialization, data acquisition, condition judgement, action execution, data processing, and test status reset.

5.2 *Automatic Testing and Result Analysis*

On the basis of the automatic test scripts developed, the automatic testing is implemented and testing reports are generated. The detailed testing and analysis of digital signal acquisition function, analog sensor signal acquisition function, idling condition control function, and normal condition control function are introduced as follows. And the testing analysis contains all the four categories of test points mentioned in Chap. 3.

5.2.1 Testing and Result Analysis of Digital Signal Acquisition Function

The engine speed, which is a typical kind of digital signal, is taken as an example for digital signal acquisition function. Engine speed is obviously a significant parameter for describing engine conditions, and it can be seen from Chap. 4 that most of the control parameters in the control strategy are related to engine speed. Besides, the engine speed accuracy directly determines whether the relevant engine control function can be achieved correctly. Figure 12 shows the testing results of engine speed acquisition function; notice that the allowable acquisition error of speed in this testing is less than ±3%.

It is shown in Fig. 12 that the speed acquisition errors are all within the allowable error range; that is to say, the test passed.

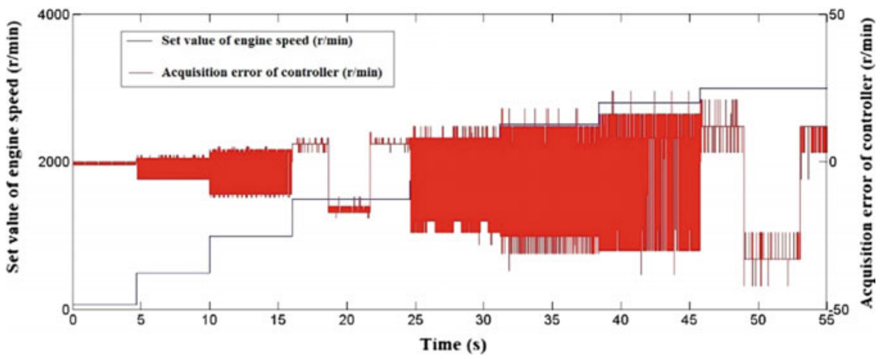


Fig. 12 Function testing of engine speed acquisition

5.2.2 Testing and Result Analysis of Analog Signal Acquisition Function

Likewise, the analog signal acquisition is important in the realization of engine control function. In this paper, the accelerator pedal position signal acquisition is taken as an example; also, the allowable acquisition error is less than ±3%. The judgement criterion is filled in the automatic testing script; the test passes if the collected signal meets the criterion. However, the test failed when the test script was executed for the first time. The results show that the error is a constant. Given this, the reason for the failure emerges: There is a zero drift in the acquisition circuit of controller, which was out of consideration in the development process of software.

In order to solve the problem, the zero drift of the hardware is corrected at the software layer. The correction algorithm is shown in Fig. 13. The voltage signal (0–5 V) collected by voltage signal input module is converted to digital signal (0–5000). The value of Constant1 is set to 50 to eliminate linearity deviation of zero drift. The function of the remaining part is to turn digital signal (0–5000) into the corresponding throttle opening (0–100%).

Figure 14 shows the contradistinction between set value and collected value of the corrected accelerator pedal position. Apparently, the two lines coincide perfectly, particularly when the throttle opening is small; meanwhile, the maximal error is less than 3%, and the test passed.

5.2.3 Testing and Result Analysis of Idling Condition Control Function

(1) Target idling speed control function

The target idling speed control function is taken as an example for look-up table function. As was analyzed in Chap. 4, in order to test the target idling speed control function, the input coolant temperature was divided to nine values, namely −50, −1, 0, 1, 50, 99, 100, 101 and 150 °C; thereafter, corresponding test cases and test scripts were designed.

When the testing was implemented, the input coolant temperature was set as the eight values in turns, and the output target idling speed values were all equal to the expected values; the test passed.

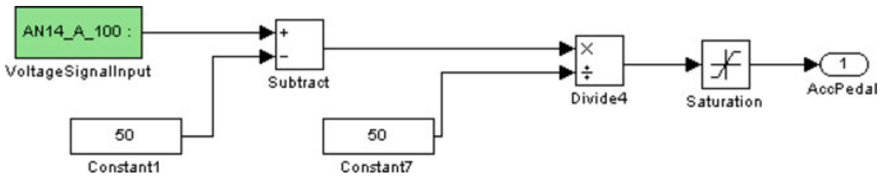


Fig. 13 Correction algorithm of accelerator pedal position sensor

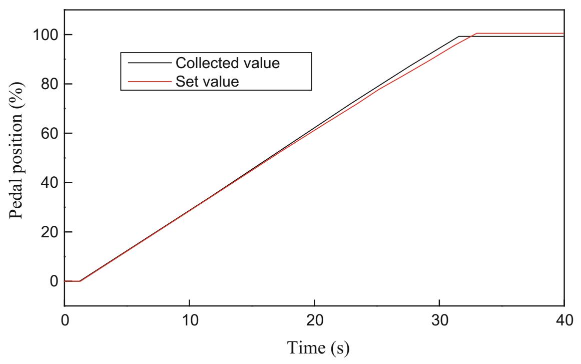


Fig. 14 Contrast diagram after correction of zero drift

(2) Idling speed control function

Likewise, idling condition is important for engine control, and it is rather complex. In Chap. 4, the test item of idling condition is divided into four test points. The PID control is used in the idling speed control function and is taken as an example for complex calculation. Several representative target idling speed values, namely 600, 800, 1000, 1500, and 2000 r/min, were selected for PID control function testing. The test passes only when the error between speed calculated by PID and the set speed is less than 10 r/min. Despite the fact that the PID control function passed the testing in the off-line simulation, it did not work correctly in the HIL. Figure 15 shows the testing result of speed at 800 r/min; the test failed due to the high fluctuation of approximately 150 r/min when the engine speed was stabilized 40 s later.

The reason is that the real-time performances of the model vary in different platforms, and the parameters calibrated before may not be suitable for the HIL platform. As a consequence, CANape, the online calibration software produced by Vector, is used to re-calibrate parameters in PID module. Figure 16 shows the online calibration interface.

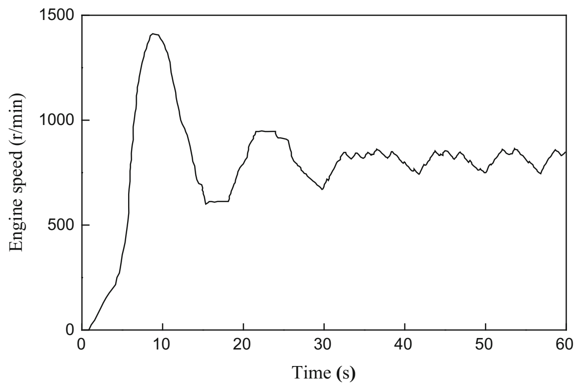
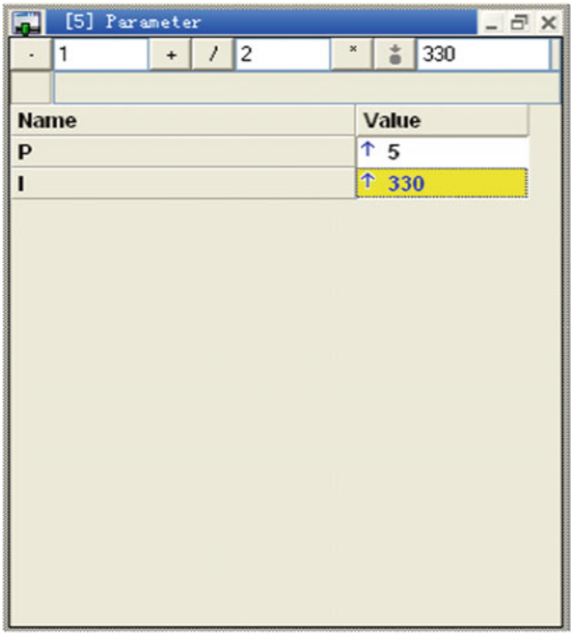


Fig. 15 Testing result of PID control function



Name	Value
P	↑ 5
I	↑ 330

Fig. 16 Online calibration interface

Moreover, regression testing is executed after re-calibration; Fig. 17 shows the testing result; apparently, after re-calibration, the idling speed stabilizes near the target speed (800 r/min) in about 2 s with a speed error of less than 2 r/min, and the test case passed.

5.2.4 Testing and Result Analysis of Normal Condition Control Function

The correct judgement of engine status determines whether the controller could calculate out the right control signals. The judgement of engine status in normal condition function is taken as an example for logical judgement function.

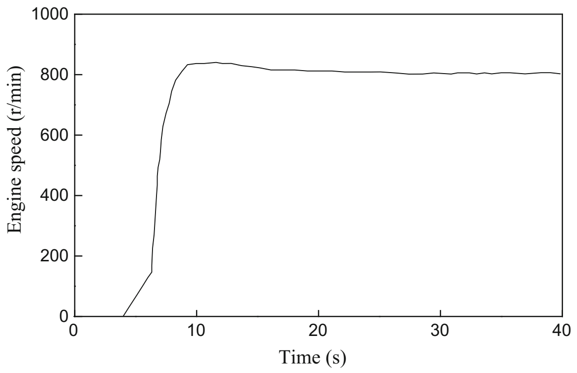


Fig. 17 Testing result of PID control function after re-calibration

The controller determines whether the engine is running at normal conditions according to the accelerator pedal signal. When the position of the accelerator pedal is greater than 4%, the status of the engine will be 3; meanwhile, the engine will turn into the normal control condition. In order to test this logic judgement test point in normal condition, the engine was started to enter idling condition, when the speed was stable and the accelerator pedal opening was increased to over 4%, check whether the engine status was 3. When the engine was running at normal condition stably, the controller will take the corresponding judgements and control based on the collected accelerator pedal position. When the accelerator pedal position is less than 3%, the engine will switch from normal condition to idling condition; at that time, the engine status will be 2. Figure 18 shows the test result.

As shown in Fig. 18, at the 13th second, the accelerator pedal position was greater than or approximately equal to 4%, the engine status switched from 2 to 3, and the engine entered into normal condition. At probably the 35th second, the accelerator pedal position was less than 3%, the engine status switched from 3 to 2, and simultaneously, the engine switched from normal condition to idling condition; the test case passed.

When the engine speed drops suddenly, the controller will make corresponding change based on speed value, when the speed is decreased to less than 100 rpm, the engine condition will be switched from normal condition to stop condition, meanwhile the engine status will switch from 3 to 0, and the fuel supply will be cut off. The test result was shown in Fig. 19.

When the engine was running in normal condition, the load torque suddenly increased at the 43rd second, leading to a sudden drop of engine speed. At the 43rd second, the engine speed decreased to 100 rpm, the engine status switched from 3 to 0, and the engine turned into stop condition. As shown in Fig. 19, this control function was realized correctly, and the test case passed.

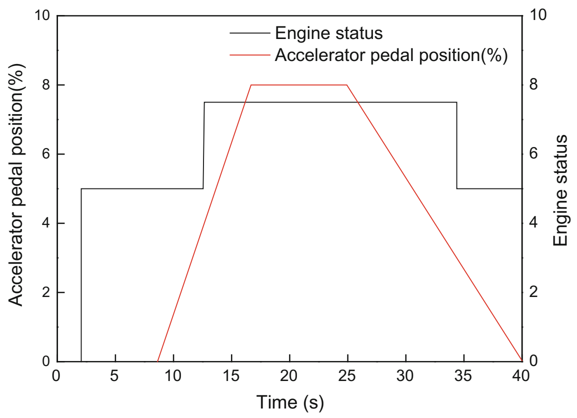


Fig. 18 Logical judgement testing result

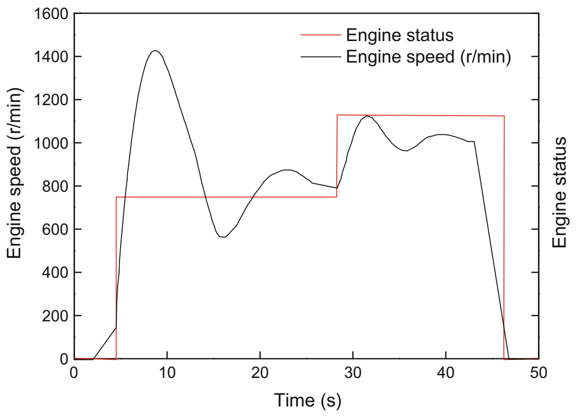


Fig. 19 Engine status testing result

6 Conclusions

In this study, based on the HIL simulation technology, the integrated test platform of hardware and software was developed for the engine ECU; meanwhile, the automatic testing of various functions of the engine control software was accomplished in the semi-physical simulation environment. Moreover, the test case distribution was optimized based on the matrix distribution methodology, and the test case redistribution strategy was applied based on the error distribution characteristics which contributed to a tremendous enhancement of testing efficiency. Besides, the engine control software test cases were designed and the automatic testing scripts were developed using the methodology of equivalence class partitioning, methodology of boundary value, and the methodology based on scenario. The developed automatic testing script library lays the foundation for reuse in subsequent software testing and improvement of testing efficiency. Application on engine control software testing shows that the HIL-based platform can not only test the software function and real-time performance, but also find the further software problems related to the hardware. It was noticed that problems found in the engine control software testing mainly exist in the software relating to hardware circuit such as the zero drift in the controller acquisition circuit. This will be valuable experience for future software testing using HIL-based platform.

References

1. Roche M, Mammetti M (2015) An innovative vehicle behaviour modeling methodology for model-based development. SAE Technical Papers
2. Xu H, Niimi Y, Ono T (2013) Virtual development of engine ECU by modeling technology. Lect Notes Electr Eng 67:66–71

3. Haukap C, Röpke K, Barzantny B (2006) Hardware-in-the-Ioop simulation (HIL) for production engine development. Elsevier Inc.
4. Piscaglia F, Ferrari G (2007) Development of an offline simulation tool to test the on-board diagnostic software for diesel after-treatment systems. SAE Technical Papers
5. Li JW, Dong HG, Wang Y et al (2014) Research on the dynamic model and ECU HIL simulation system of electronic-controlled engine. Adv Mater Res 889–890:962–969
6. Allen J, Dhaliwal A, Warra J (2011) A novel approach to implementing HIL systems for ECU validation and verification for commercial vehicle applications. Chem Nat Compd 7(6): 840–841
7. Mazhari SA, Nampoothiri S (2013) PSO tuned vehicle climate system model for HIL based ECU testing. SAE Technical Papers, 2
8. Ni J, Li X, Shi X et al (2014) Design of host program for engine ECU HIL system based on NI PXI platform. Automobile Technology
9. Vegas S, Basili V (2005) A characterisation schema for software testing techniques. Empirical Softw Eng 10(4):437–466
10. Gupta A, Jalote P (2008) An approach for experimentally evaluating effectiveness and efficiency of coverage criteria for software testing. Int J Softw Tools Technol Transfer 10(2): 145–160
11. Chen TY, Kuo FC, Liu H et al (2013) Code coverage of adaptive random testing. IEEE Trans Reliab 62(1):226–237
12. Katyal RK, Srinath S (2011) Virtualization for ECU platform software testing in automotive embedded. SAE Technical Papers
13. Conrad M, Sadeghipour S, Wiesbrock HW (2005) Automatic evaluation of ECU software tests. SAE Technical Papers, No. 4, pp 595–599
14. Barrett S, Bouchez M (2015) Addressing engine ECU testing challenges with FPGA-based engine simulation. SAE Technical Papers
15. Bryce R, Kuhn R (2014) Software testing. Computer 47(2):21–22
16. Godefroid P, De Halleux P, Nori AV et al (2008) Automating software testing using program analysis. IEEE Softw 25(5):30–37