Real-Time Hardware-In-Loop simulation for automated validation of diagnostic services

Charu Garg, Amit Kumar, Ajay Kumar Vashisth
Maruti Suzuki India Limited

ABSTRACT

Due to stringent emission regulations and the customer demand for more power and less fuel consumption, the functionality in today's engine management systems continues to grow. The electronic engine control units (ECUs) have to perform more control tasks using new sensors and actuators, along with the corresponding self-diagnostics (OBD, on-board diagnostics). Hence, the need is to adopt automated techniques to efficiently validate the ECUs.
This paper illustrates the detailed methodology of the automated testing on Hardware-In-Loop (HIL) simulator i.e. the Test Automation framework which uses a high-level programming language to control test sequences and coordinate with a variety of interfaces necessary for a particular test, or a set of tests and explains its effectiveness by comparing with the traditional methodology of testing on a real vehicle.

INTRODUCTION

Legislations regarding environmental protection such as reduction in fuel consumption and exhaust emissions are getting more and more stringent E.g., CAFC and BS-VI with RDE monitoring. These changes require more complex ECU control functions. So, conventional testing methods need to be replaced by MBD approach to develop and test control strategies for frontloading of the development activities, increasing the testing quality and reducing the development time and cost.

MBD in automotive follows a typical V-cycle (MiL/ SiL/ HiL) for the controller testing and validation as illustrated by Figure1. First, on the left side of the V cycle, MIL is applied, followed by SIL. Second, on the right side of the V cycle, HIL is applied.

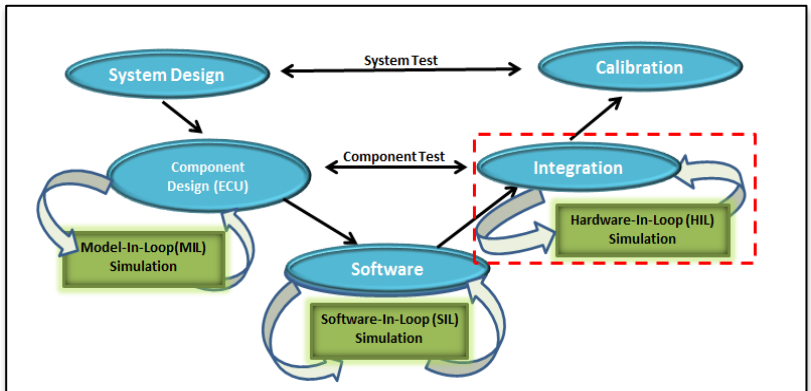

Figure 1. V-Cycle (MIL, SIL, HIL)

HIL Simulation is characterized by the operation of real components (actual prototype of controller) in connection with real time simulated components (Engine Plant Model) in close-loop. At the end of the design process, the control system is finally calibrated for the specific application using the HILs and the in-vehicle test procedures. The above mentioned process collectively forms a V-Model development cycle.

Now, automatic close-loop tests have become the standard test procedure for all advanced engine control functions. Onboard diagnostics testing is one of the typical examples of advanced engine control and therefore a typical application area for HIL simulation.
As an initial step, automated validation of On-Board-Diagnostic (OBD) communication services and electrical discontinuity checks are done on Electronic Control Module (ECM) by using a HIL simulator integrated with several off-the-shelf diagnostics and calibration tools in a closed-loop environment and this paper will describe the methodology for automating diagnostic communication testing with failure simulation in detail.

TRADITIONAL APPROACH

The picture below roughly describes the traditional approach for EMS OBD software validation. The test engineer performs the following tasks in the traditional approach:

1. Drive the vehicle as per the test procedure in which test steps are mentioned.

2. For any failure simulation, test operator simulates the fault by connecting Break-Out Box in between ECM and Engine/Vehicle Body harness.

3. For diagnostic communication, diagnostic tools are connected at the OBD port of the vehicle and the diagnostic software is operated by the test operator.

4. Also, to access the ECU data in running conditions, measurement or calibration tools are connected with the ECM and the measurement & calibration software are operated by the operator.

5. Test measurements are then analyzed for expected results and final test report is generated.

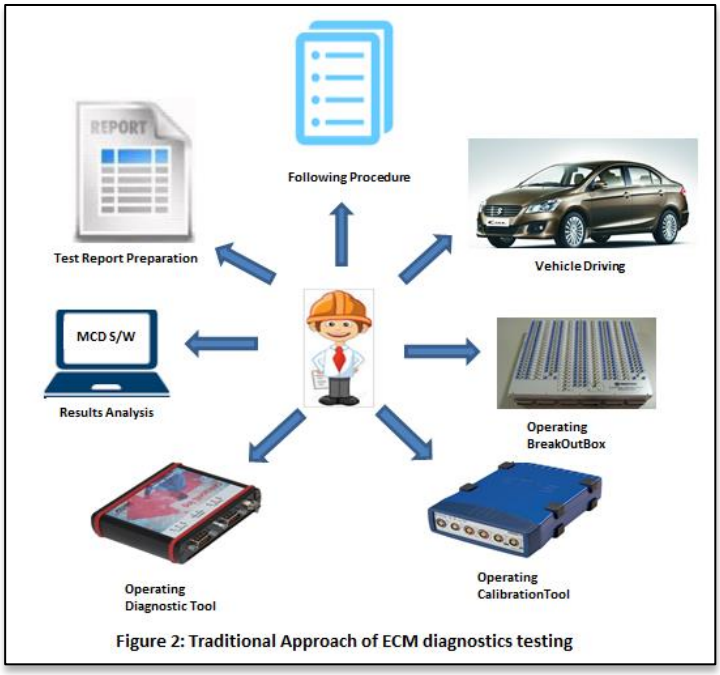

Figure 2: Traditional Approach of ECM diagnostics testing

AUTOMATED APPROACH

To automate the traditional test sequence, vehicle/engine will be replaced by HIL Simulator with engine plant model on HIL PC, MCD tools will be accessed using their APIs through HILs control software and faults will be simulated using FIUs.

In order to achieve above mentioned purpose, the following integration is required:

1. Control of test sequence (test steps for any test case)

2. Access to the plant model parameters

3. Access & Control of measurement and calibration tools

4. Access & Control of diagnostic tools

5. Test data analysis during the test (Actual vs Expected values comparison)

6. Ability to save test data and generate test report automatically.

As shown in Figure.3, Test operator only defines the test cases.

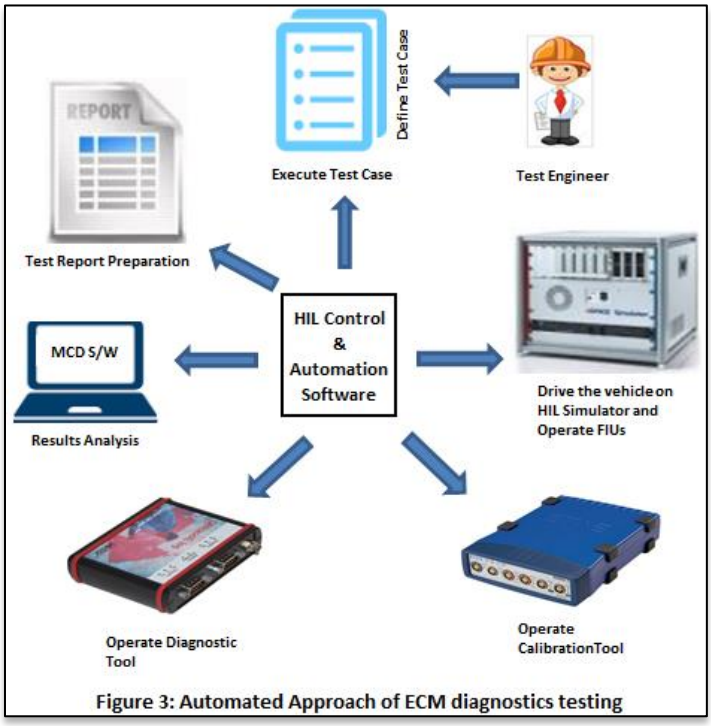

Figure 3: Automated Approach of ECM diagnostics testing

For the above automation, following tools are used:

1. HIL Control Software is used for hardware/model management, data acquisition from HIL in real-time in graphical user-interface.

2. HIL Automation Software extension of HIL Control Software is based on an advanced test scripting language, to control test sequences and coordinate the variety of interfaces necessary to automate third-party MCD tools for a particular test, or a set of tests. It is also responsible for parsing and validating the Test Set-Up, Test Execution and Test Case sheets, controlling HIL automation software and collects the results from HIL automation software and write them back into the Test Case.

METHODOLOGY FOR AUTOMATED TESTING

1. SETTING UP OF HIL

The architecture of the HIL system is shown in Figure 4. The simulator is connected to the DUT by wire-harness either directly or through switchable breakout boxes. In this case DUT is the ECM ECU.

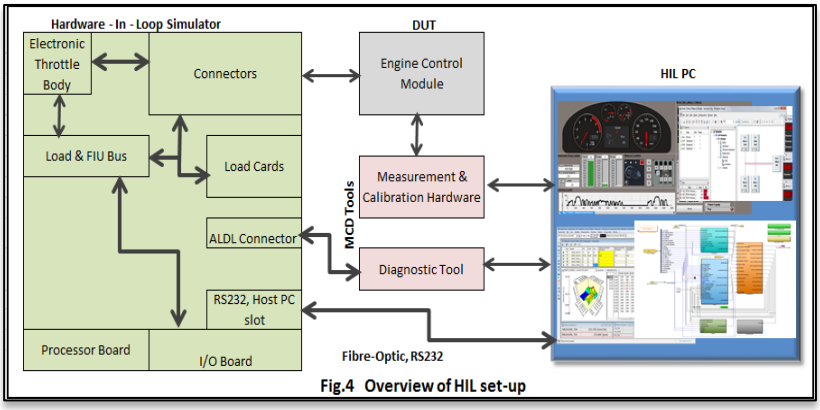

Fig.4 Overview of HIL set-up

The HIL simulator rack contains a PC processor board for real time computation, a specialized automotive I/O board, signal conditioning boards, load cards with fault simulation capability as the main components. Further, it is possible to connect real loads (e.g. Electronic Throttle Body) to the HIL simulator. The power supply can be controlled in real time during simulation, which provides the capability of vehicle battery simulation. The system also accommodates integration of off-the-shelf calibration and diagnostic tools. The integration of calibration and diagnostic tools with the HIL system provides the capability of automated testing of very complex Engine control system software.

HILs system is controlled from the HIL PC, which is connected via Ethernet over an interface card to the simulator. FIU is also controlled by HIL-PC over a serial port. On the HIL-PC, HIL Automation Software and HIL Control software are installed for controlling the simulator and test automation. Third-party software are also installed on HIL-PC for accessing the ECU. For the complete closed loop system simulation another significant requirement is the representative plant model. The plant model should represent the physical behavior of the plant; in our implementation the plant consists of the engine. The real time executable plant model is based on MATLAB® and Simulink®.

2. DEVELOPMENT OF TEST AUTOMATION FRAMEWORK

In Test Automation Framework, there are basically three different files for test data and test management:

1. Test-Item-List
2. Test-Setup-Sheet
3. Test-Parameter-Sheet

Those three different files result in several logical objects and relationships. Following picture shows the relationship between those logical test objects:

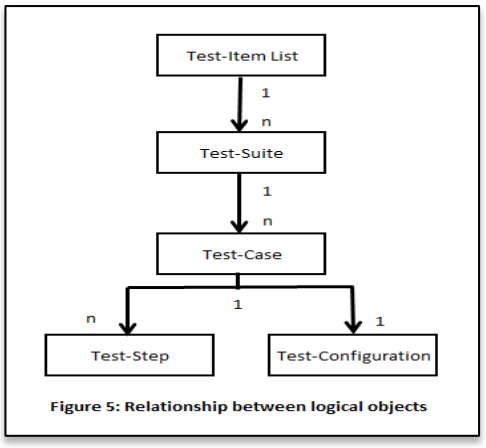

Figure 5: Relationship between logical objects

The main entry point is the Test-Item-List. The Test-Item-List can have several Test-Suites related to it. This can be seen as 1-to-n relationship. Each Test-Suite contains one or more Test-Cases. A Test-Case consists of several Test-Steps, which are held in a Test-Parameter-Sheet and uses one specific Test-Configuration which is held in the Test-Setup-Sheet.

Test-Item list is used to send command for parsing, building and executing the Test Automation project. Reference of Test-Case and Test-Configuration is defined in Test-Suite of Test-Item List.

Test-Configuration sheet has reference to all the third-party tool configuration files, HIL sdf file etc.

Test-Case sheet is used to define the test-steps.Values for Diagnostic Messages Read & Write, Control of FIU, HIL control and data read, ECM data read and write are defined and results are also compiled in this sheet.

A brief header layout of the Test-Case Sheet is shown in Figure 6.

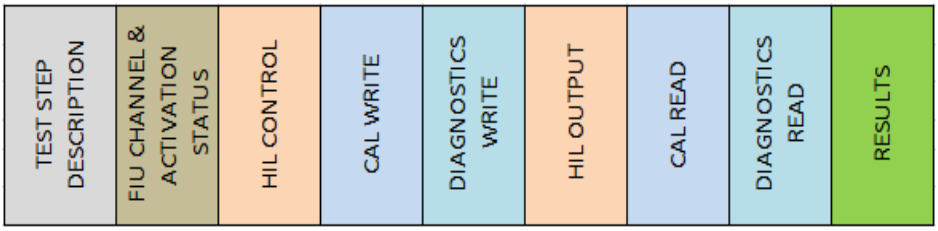

Figure 6: Test- Case Layout

Each row of Test-Case sheet indicates the test step. Values for the headers are filled in each row as per the test conditions. HIL control header can be used to control any switches or sensors. CAL write header can be used to set the constants/tables/maps. Diagnostics write is used to send diagnostic messages to ECU. Diagnostics Read can be used either to read data in raw format(bytes) or the final values by using odx file. For each header of output(HIL/ CAL/ Diagnostics), two columns are used for each parameter. One is filled by user for expected result and another is filled through automation with actual results. Expected value can be defined by using logical operands and also reference to the output of different head can be given. After the comparison between actual and expected value, pass or fail judgment is done for each test step and filled as verdict in results column.

For each test execution, an AutomationSoftware project is built up. Source for building up a project is a project template. The project template is extended by test data. The data objects from the project template remains present in an execution project.
Figure 7 shows the structure of the AutomationSoftware project template with test data.

The AutomationSoftware project template contains data containers for accessing the real time simulator, storing information about the used CAL tool project, accessing the FIU interface over HIL Control Software and handling Diagnostics Communication Tool. GlobalConfiguration container is used to indicate if MCD tools are used or not and stores the IP address and port for connection with MCD tools service.
For a test execution, this project template is loaded to AutomationSoftware and saved as a temporary project for test execution and it is further extended by test data objects.
An overview of added test data objects and their purpose is mentioned below:

1. HILData : This data container contains test case relevant simulator data such as used variables, flag for reloading the real time application and all available variables on the real time platform.
2. DiagnosticsData: This data container contains test case specific configuration data. It contains a configuration file and a logfile path.
3. CALData: This data container contains all test case specific calibration data. It stores information about the A2L file, HEX file, used hardware device name, etc.
4. FIUData: This data container contains test case specific information for accessing the failure simulation and used CSV file.
5. TestCaseInitialization: The TestCaseInitialization is responsible for loading the MCD tools configuration, loading / creating the MCD tools project and setting up the FIU access.
6. Steps : This folder contains all single steps from a test case. Each single row from a test case sheet can be found as sequence within this steps folder.
7. Cleanup: This folder contains a test case specific cleanup sequence.
8. TestCaseCleanup : This sequence cleans up the test case. It stops recording in MCD Tools and closes the experiment, it releases the MAPort for HIL access and removes the failure system from HIL Control Software.
9. GlobalCleanup: This sequence is used to clean up a whole test suite and is basically the counterpart of the GlobalInitialization sequence. It closes MCD Tools as well.

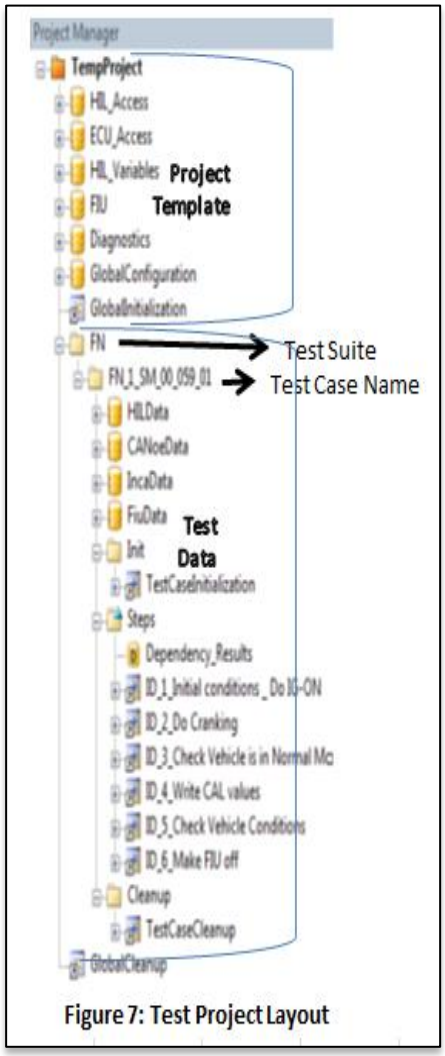

Figure 7: Test Project Layout

In AutomationSoftware, in addition to built-in libraries, implementation of custom libraries was required to access third party tools. For example: One library for accessing CAL tool, another library for accessing DiagnosticTool and the main library for building test case. Figure 8 illustrates the DiagnosticTool Service Library.
The library contains blocks for basic access to the normal DiagnosticTool instance. There are blocks for starting/stopping the DiagnosticTool service, for executing diagnostic requests, starting/stopping measurements and reading/writing to environment variables. Each block uses an exec block that performs a function call on the DiagnosticToolAutomation module. The exec block imports the DiagnosticToolAutomation module and performs a function call to executeDiagService. The return value is evaluated and a success flag is set. In a last step the response is stored. CALToolServiceLib is also having the same mechanism.

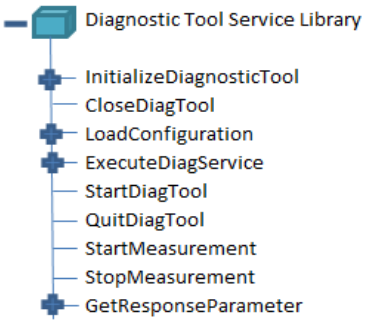

Figure 8: DiagnosticTool Service Library

3. DEFINITION OF TEST CASES

a) Diagnostic Communication Testing

Testing diagnostic functions requires accessing the diagnostic memory of the ECU. There are different ways of accessing the diagnostic memory of ECUs from a HIL environment. One is to implement the diagnostic protocol as a software component within the HIL system itself. The test can then use the protocol layer, which provides an appropriate API. The other method, which has been implemented here, is to make use of commercial diagnostic tools. Such tools can be controlled from within a test and serve as an interface to the ECU diagnostic functionality.
For testing of diagnostic services communication of one vehicle model, execution of more than one hundred test cases is required. Test cases are based on accessing real time data from diagnostic memory, freeze frame data, controlling I/Os, checking appropriateness of data under normal operation of vehicle and in faulty conditions, OBD monitoring, to check timing of DTC information storage and erasing, accessing ECU and vehicle information etc. Such test cases need to be checked regressively on various models. So, here lies the huge scope of taking benefits of test automation. Hence, following figure describes the diagnostic ram header and sub-headers of a Test Case Sheet in detail that how all the described test cases are covered by this framework:

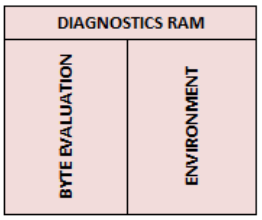

Figure 9: Diagnostics Header

1. **Byte Evaluation** This section is used to perform a byte evaluation on a hexadecimal response from a previous diagnostic message request. This header can have an arbitrary number of sub-headers. Each sub header represents a byte position. Each byte has two cells, one for expected results and another one is kept empty, to write back the actual results.

2. **Environment** This header contains the names of environment variables (Qualifier names of the odx file). During the validation process the occurrence of these variable names is checked as qualifier names. The last response from the ECU is checked for the occurrence of the given environment variable and the read value is compared against the expected value. Valid values for an environment variable are single values like “800” or syntax conforming logical expression like “> 800 and < 1000”. As mentioned above, each parameter has an empty column next to each value. This empty column is used to write back read values. Also, to confirm the appropriateness of diagnostic ram values with CAL ram values, a mechanism is implemented to give reference to the previously read values of different headers.

b) Electrical Fault Simulation Testing

Fault simulation test consists of different steps to be performed sequentially, as follows:
1. Drive into specified operating point
2. Activate electrical fault by remote-control relay
3. Read out ECU diagnostic memory
4. Evaluate test by comparing the detected fault with the expected fault
5. Generate report automatically

Since this kind of test has a very uniform structure and can easily be reused for different kinds of electrical faults, it has very great potential for savings in time and therefore cost.
The ECU diagnostic software is required to detect electrical faults on the ECU input and output pins in the defined DC and to correctly handle the fault, for example, by limp home functions, and to store the information about the fault in the

ECU internal diagnostic memory. And also the healing of the fault is confirmed on the ECU input and output pins in stated number of DCs. Vehicle is driven in a specific pattern for the diagnosis of fault.
Typical electrical faults to be simulated using relays are short circuits to battery voltage and to ground or broken wires. In this case, these faults are simulated by using an automation interface for sending switching commands via the serial interface (RS232).

After the execution of test cases, test data loggings are saved and detailed reports of the test cases are generated automatically.

CONCLUSION

As the number and functionalities of the controllers have increased significantly, automatic testing has become substantially important. The designed Test Automation Framework helped us in achievement of the following:

- Execute repetitive and time intensive tests efficiently
- Eliminate errors due to manual testing
- Improve Quality and Efficiency
- Generate automatic Report at the end of test
- Re-usability of the Test Scripts

Figure 10 shows the results of using automated approach over traditional approach for the diagnostics communication testing.

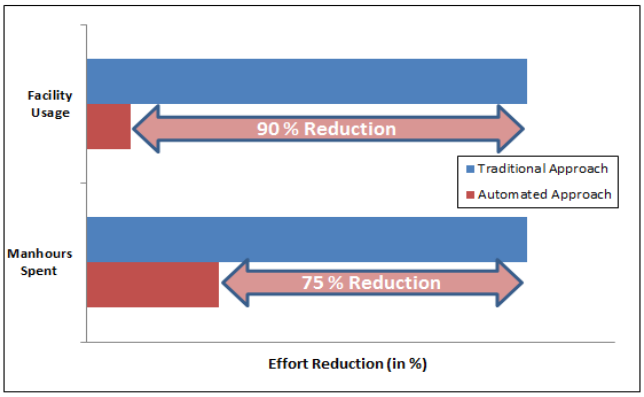

Figure 10: Effectiveness of Automated approach for Diagnostics Communication Testing

ACKNOWLEDGEMENTS

The authors wish to thank the team members from Maruti Suzuki for their extensive support throughout the project.

REFERENCES

1. Herbert Schuette and Markus Ploeger, “Hardware-in-the-Loop Testing of Engine Control Units - A Technical Survey”, SAE Paper 2007-01-0500.
2. Development and Calibration of On-Board-Diagnostic Strategies Using a Micro-HiL Approach, SAE Paper 2011-01-0703.
3. Klaus Lamberg, Jobst Richert and Rainer Rasche, “A New Environment for Integrated Development and Management of ECU Tests”, SAE Paper 2003-01-1024.
4. Syed Nabi and Mahesh Balike, Jace Allen and Kevin Rzemien, “An Overview of Hardware-In-the-Loop Testing Systems at Visteon”, SAE Paper 2004-01-1240.

CONTACTS

Charu Garg
Deputy Manager – Vehicle Testing & Calibration
Maruti Suzuki India Limited
E-mail: charu.garg@maruti.co.in

Amit Kumar
Deputy Manager – Vehicle Testing & Calibration
Maruti Suzuki India Limited
E-mail: amit.kumar@maruti.co.in

Ajay Kumar
Deputy General Manager – Vehicle Testing & Calibration
Maruti Suzuki India Limited
E-mail: kumar.ajay@maruti.co.in

DEFINITIONS, ACRONYMS, ABBREVIATIONS

API	Application Programming Interface
CAN	Controller Area Network
DC	Driving Cycle
DTC	Diagnostic Trouble Code
DUT	Device Under Test
ECU	Engine Control Unit
EMS	Engine Management System
FIU	Fault Insertion Unit
HIL	Hardware-In-the-Loop
HW	Hardware
IUPR	In-Use Monitoring Performance Ratio
I/O	Input / Output
MIL	Model-In-Loop
MBD	Model Based Development
MCD	Measurement, Calibration & Diagnostics
OBD	On Board Diagnostics
OEM	Original Equipment Manufacturer
SIL	Software-In-Loop