Automated Test of ECUs in a Hardware-in-the-Loop Simulation Environment

Dr.-Ing. Rolf Boot, AUDI AG, I/EE-7, 85045 Ingolstadt, rolf.boot@audi.de,
Dr.-Ing. Jobst Richert, Dr.-Ing. Herbert Schütte dSPACE GmbH,
Technologiepark 25, 33100 Paderborn, {jrichert, hschuette}@dspace.de

1 Abstract

In this paper, several examples describe how real-time simulation tools can be used during the development process of modern automotive electronic control units (ECUs) in order to effectively meet the demands placed by the increasing complexity of the software and the resulting intricate ECU tests.

During the past few years, the software for engine ECUs in particular has shown an almost exponential growth in range and complexity, since they must fulfill the customers' demands for comfort as well as the government's requirements on emissions reduction and system self-diagnosis. As a result, the European Market is about to see the introduction of a new generation of engine ECUs with expanded on-board diagnosis (OBD) functionality. Until recently, such ECUs were produced only for the US market in small numbers and with little variety. In addition, the number of control loops within engine control systems is increasing, which means that offline operation of the ECU (for example, without the motor as a closed-loop controlled system) is no longer possible.

Because of these latest developments, hardware-in-the-loop (HIL) simulation is being used during the development of production type engine ECUs. During this development process, special requirements are primarily caused by the necessity to check the closed-loop and open-loop functions, self-diagnoses, failure memory management and triggering of failure lamps. Since malfunctions in these areas cause a high recall potential for vehicle manufacturers, especially when they are discovered by the customer first, they must be avoided by all means.

Test Automation (TA) now provides the functionality and power to meet the complexity of the increasing demands. This paper describes how the test automation method can be used to replace time-consuming manual tests with programmable, automated sequence control in an interactively operated test environment.

2 The Hardware-in-the-Loop Simulator

2.1 Hardware Components of the HIL System

To successfully run modern production type ECUs for engine management without using a real engine or vehicle it is no longer sufficient to just stimulate the inputs of the ECU with electrical signals. Today, the ECU also requires consistent, physically plausible data from the engine's sensors. Without such data, the ECU will enter emergency operation, which does not
permit any further functional tests. To provide the sensor values necessary for faultless operation of engine ECUs, even in transient engine operation, mean value models of combustion engines (spark ignition and diesel) are simulated in real time. Typical outputs of such a model are the engine speed to simulate the signals of the crankshaft- and camshaft sensors, the air mass flow of the air mass sensor, and the oxygen concentration in the exhaust system of the \( \lambda \)-sensors. On the input side of the simulation model, the throttle position, fuel amount and ignition angle must then be provided by the software engine exactly as they are generated from the ECU. Once this data is provided, the control loops between the software engine and real ECU are closed.

![Figure 1: The components of the HIL simulator](image)

Figure 1 shows the main hardware components of such an HIL simulator. In addition to the ECU under test, the other fundamental system components are the simulator with the real-time hardware, the input and output (I/O) modules (A/D, D/A, digital I/O, CAN) and the signal conditioning (SC).

It should be emphasized that the complete I/O of the simulator is built modularly using standard commercial off-the-shelf boards, and can thus be expanded flexibly if model expansions or changes on the ECU are necessary. The processor capacity can also be increased without any problems. For example, even extremely high demands for computing power, such as for diesel engines with common rail injection and/or turbochargers, can be met easily by integrating a DEC Alpha Processor.

The signal conditioning contains the bi-directional electrical connection between the ECU and the off-the-shelf I/O boards, which in general have TTL- and \( \pm 10V \) inputs and outputs. In addition to level adaptation and protection circuits, potential isolations for certain sensors and drivers are usually included here. To simulate short-circuits and wire faults, switchable relays
are controlled by the simulator. However, in some cases the signal preparation is considerably more complicated, such as for injection signals in common rail systems or the generation of the voltage levels on the primary side of ignition coils. Quite often there are additional components in the signal conditioning: for example, power supply units driven by the simulator with which the performance of the battery during start can be simulated. The signal conditioning used in this case has one signal conditioning board for each individual I/O board. These boards can either be integrated with the processor board and I/O hardware in a single box, or assembled as a separate unit.

Because the highside/lowside drivers of modern ECUs are often able to perform diagnoses, error-free operation is made possible only by integrating real components on a load panel. Another point in favor of using a load panel is that the real components themselves can be tested with this setup. Figure 1 shows the electronic engine power control (E-Gas) actuator, whose function would be very difficult to reproduce with simulation models, as well as the accelerator pedal position sensor, injection valves, and ignition coils. In addition, the camshaft actuator and valves such as the exhaust gas recirculation (EGR) valve and supplementary air valve are present as actual components. The dashboard primarily serves as a support for the end user because it provides optical feedback of the operating conditions. Quite often, though, there are also special functions included in the dashboard, such as the immobilizer without which the ECU cannot be run.

The PC itself is the actual user interface for operating the simulator. The code is automatically generated from the offline simulation of the engine model within MATLAB/Simulink (described in the following subsection) and downloaded on the real-time hardware. In addition, easily configurable standard tools are available to interactively operate the simulator and to perform continuous data recording. Furthermore, an instrument panel can effortlessly be set up graphically in order to manually alter parameters (for example, via sliders) or to visualize simulator-internal values via gauges or numeric displays. This graphical interface enables the user to gain intuitive access to the system, which is handled like a real vehicle rather than a simulator.

A diagnosis tool such as the VAG tester or a calibration system makes the HIL simulator complete. These tools are used to display the ECU-internal values and to reset error flags. Furthermore, diagnosis tools are indispensable during the multiple-step start-up procedure of the systems where the control loops must be closed successively. The central role of these tools during test automation will be explained in greater detail in chapter 3.

### 2.2 Software Components of an HIL System

Together with the I/O modules, the real-time-capable mean value models of the combustion engines described in the previous subsection constitutes the software core of the HIL simulator. The model used in this case was provided by the company Tesis (Munich) and is implemented in MATLAB/Simulink. To describe engine models, Tesis generally offers two Simulink block libraries that can be used to build models of different complexity. The base library,
for example, contains blocks for the control volumes in the air intake/exhaust areas with which the air mass flow through the engine can be simulated. The mean value of the torque is calculated individually for each cylinder. With the use of form functions, a position-dependent ripple is superimposed over the torque output in order to check the OBD-II functionality misfire detection of the ECU. The second block library offers the same functions, whereas the energy flow is also taken into consideration, not just the mass flow. These two libraries permit the user to flexibly model spark ignition engines and diesel engines with different numbers of cylinders, air intake benches and exhaust benches. There are also sub-modules for turbochargers, exhaust re-circulation, and simple drive train models with clutch, brake, automatic transmission, and manual transmission. Finally, dynamometer operation is also possible in addition to manual operation (driver operation) with the accelerator pedal, brake, clutch, etc. Both modes of operation can also be performed by remote control via automation scripts; for example, in order to set defined operating points (full load, partial load, etc.).

With a single press of a button, automatic C code generation implements the model on dSPACE real-time hardware. The standard I/O hardware implemented in the HIL simulator (such as A/D, D/A, digital I/O, PWM generation, etc.) can be integrated in the block diagram by simple drag-and-drop, even without any detailed knowledge about HIL.

During the first modeling step, the model outputs must be able to stimulate the ECU inputs in all operating conditions without creating any implausibilities that would be detected by the monitoring functions of the ECU. In other words, no emergency operation should occur. The next step should involve all OBD-II functions, which means functions such as supplementary air injection, monitoring the catalytic converter and $\lambda$-aging, the canister purge valve, etc. In the last step of the model refinement, the model should simulate the engine functions so well that at least a rough calibration can be performed on the HIL simulator. During this step, the calibration of safety functions is especially considered in addition to the rough setting of different look-up tables.

Figure 2 shows the structure of the loops between the model and the ECU. The main interface values of the model’s output are the engine speed, air mass flow and $\lambda$-values. For each cylinder, the injection times, ignition angles and throttle position are measured. In addition to regulating the ignition angle setting, the ECU is responsible for controlling the idle speed, knock and $\lambda$ of the software engine.
The signal generation and signal measurement blocks shown in Figure 2 are especially important. Signal generation includes, for example, the generation of the crankshaft/camshaft sensor signals and the knock signals. A standard dSPACE board (DDS Board, direct digital synthesis) is used for the signal generation. Each of the six C31 processors on this board are equipped with a D/A converter in order to be able to output complex sensor signals. The processors of these boards are programmed in C and communicate with the real-time simulator via dual port memory. Typical resolutions for the generated signals lie between 5 µs and 10 µs.

For test automation, the high flexibility of this board is especially useful whenever distorted sensor data is needed to test ECU diagnosis functions online. This means that a missing tooth or a shifted tooth gap in the 60-2 crankshaft sensor are as easy to simulate as continuous camshaft adjustments. For example, to test the knock control, the knock parameterization can be changed from the user interface and a cylinder-selective knock can be induced here as well.

Signal measurement involves the measurement of the injection time (also for multiple injections and permanent injection) and the cylinder-selective ignition angle. The angle-related measurement of the ignition pulses generated by the ECU is performed by the DDS Board in combination with a standard I/O board (DWC Board, digital waveform capture). A corresponding Simulink S-function can be modified for 4-, 6- and 8-cylinder engines.

By integrating a CAN board, the user can accommodate the constantly increasing interconnection among different automotive ECUs, such as engine management, ABS, TCS, etc. With this board, the dSPACE simulator can transmit and receive several vehicle-manufacturer-specific CAN messages.
3 Concepts and Tools of Test Automation

The HIL simulator contains all the tools necessary for interactive experiments for manual tests of particular ECU functions as well. The automation of these manually performed tasks, known as test automation, now offers the possibility to ensure the necessary test depth and test width despite the increasing requirements. With automation, safety-critical functions and parameters can be tested without any risk by means of remote control. This means that the user can run through situations in which the ECU under test operates at extreme driving conditions. The necessity to run these tests automatically results directly from the complexity of the tests, and especially the large number of functions to be tested.

In order to test as many ECU functions as desired, direct access to the ECU-internal variables is essential. The reaction to a failure stimulation can be measured only at the position where it would be detected in actual operation and possibly treated. The hardware and software extensions of the components necessary for this procedure will be covered in the following subsections.

The automatic test sequences performed by the test automation environment described in this paper cover many various application areas, for example, on-board diagnosis and functional tests. To date, most projects have basically emphasized the first category: on-board diagnosis. To be more specific, test functions are created to check the diagnoses of electrical connections (driver stage diagnoses) and then failures are stimulated on a relay box by corresponding software-driven switches for short-circuits and wire faults. The plausibility of the results of the on-board-diagnosis is then checked by other test sequences. For example, the permissible air mass flow ratio is calibrated in a two-dimensional engine characteristics map in the ECU in accordance to the engine speed and the throttle valve angle. It is thus necessary to read the permissible threshold value applicable to the current operating point (engine temperature and load). To stimulate a failure, the real-time simulation value that simulates the engine's air mass flow sensor is then distorted with a value that exceeds the permissible threshold. The reaction of the relevant failure bits must be observed and compared with the corresponding "good values" that, in part, are also parameterized via calibrated values. A third category of test sequences in the field of on-board diagnosis contains the test of diagnosis protocols and/or failure lamp triggers.

3.1 The Concept

To reach the highest possible degree of reusability for previously written test sequences, automobile manufacturers and suppliers of real-time simulation technology worked together to standardize the test sequences. In close cooperation they established a library of scripts that includes the settings of the operating conditions as well as test sequences for single function blocks. These library sequences can be varied easily with the use of an overall, flexible parameterization concept that employs all the available information from the ECU’s description files and data files. This means that the library sequences can also be reused in new
ECU projects and when the program versions change. In order to describe the test automation approach, a few fundamental terms must be explained first.

A **project** is defined as the sum of all accumulated files in relation to a particular engine ECU: namely, ECU description files, hex files and **project component lists** of different variations, the necessary model files, and the real-time programs and configuration files of the HIL simulator. All of the projects are managed in the **project data base**. A **project component list** is defined as a list of the components and failure bits designated in the corresponding variations. This list is sorted according to function groups or other user-defined categories. For example, an ECU can include the functions for a 4-cylinder in-line engine as well as for a V-6 engine. The user must specify that the failure bits that monitor the $\lambda$-control and that are assigned to the individual exhaust lines in the V-6 engine must not be used in the 4-cylinder in-line engine, or should only be used in part.

A **clearance table** is described as the management mechanism of the test automation script library. In this table, all the necessary information is assigned to the corresponding failure bits or test functions. This includes information such as the applicability for each specific failure type, the validity range of load and temperature operating points, the validity range of the referred diagnosis script of the program versions, etc. The range and sequence of the single tests can be determined by selecting the pre-defined standard operating points - **cold or warm engine**, **idle speed**, **partial load**, **full load**, **overrun mode** - or by selecting user-definable operating points.

Figure 3 shows the interaction of the components described above. It illustrates the separation of the project-specific information from the actual test automation library where only parameterized and thus re-usable scripts are filed. The operating concept shown here has been designed so that it can be operated by different users. Test engineers who are only interested in running and documenting the tests are just offered the information they need. In other words, they encounter only those files that they see during their
daily work (ECU description files and data files). Test engineers normally do not create the individual test scripts, but rather configure the individual test sequence from the main operating panel by using the test sequence editor. The information necessary to parameterize the desired test sequences is generated by the project data base and the clearance table (this process will be described in greater detail in the following section). In comparison, the development engineer creates the sequence controls for the stimulations, measurement tasks, test evaluations and test documentation by using dSPACE’s test automation script language. These parameterized scripts are then placed in the library and given the corresponding management information via the clearance table editor (Which failure bits are being treated? Which failure types are being tested? Which load and temperature operating points can be applied? Which script should be used?). Normally, the development engineer is also responsible for managing the projects with the project editor.

Figure 4 briefly illustrates the multiple-step configuration process for creating a test sequence. During preparation, the project engineer specifies the type information in the project data base while setting the selected variations. In the first step, an initial "filter process" then uses this type information in the project component list to check whether there is already a parameterization for the current ECU software version. In principle, this allows the treatment of the model-relevant failure bits. In the second step, a filter checks whether there are already diagnoses scripts that can be used. The information from the clearance table is used here. In addition, just the permissible load and temperature operating points are offered for subsequent user interactions. Users can also set their own operating points in addition to

<table>
<thead>
<tr>
<th>section</th>
<th>error bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>throttle valve</td>
<td>E_db, E_db1p, E_db2p</td>
</tr>
<tr>
<td>fuel injectors</td>
<td>E_twe, E_twe2</td>
</tr>
<tr>
<td>lambda probes</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: TA configuration process
selecting the above-mentioned standard operating points. The result is a completely parameterized, executable standard test specification that can be started immediately.

3.2 Scripting to Describe the Sequence Control

At dSPACE, a high-performance script language was integrated within the framework of a new generation of dSPACE experiment software [1]. This script language can be used for internal macro programming tasks (loading the desired real-time model and instrumentation layout, controlling the model parameters, etc.) as well as for the test automation tasks described in this paper.

The basis of this sequence control is the well-coordinated combination of non real-time-capable parts on the PC and real-time-capable service routines on the real-time simulator. The same description form is used to define the real-time signal forms described below, to determine measurement tasks for ECU or real-time simulation variables, or to formulate OLE automation sequences for communication with documentation tools. Furthermore, for certain standard programming tasks there are graphical editors that free test engineers from direct programming tasks.

3.3 Real-Time-Capable Signal Generation

For the type of test activities described in this paper, it is extremely important that the individual test sequences of the simulation model can be repeated with extreme precision in full real-time resolution. This requirement was fulfilled by using real-time-capable stimuli data generation for the model inputs as well as model parameter variations. With such a stimuli data generation, the above-mentioned driver stage diagnoses can be performed by using a break-out box with relay switches controlled via digital output channels.

This feature of dSPACE's test automation tools allows the multiple-channel definition of real-time signals. Some of the features are listed below:

- piecewise definition of signal from segments with description in relative times,
- conditional signal generation ("IFs") in connection with observed values (e.g., to abort the test sequence during exceptional cases),
- LOOP constructs to define cyclical signal segments,
- import of measured data and synchronous replay into the real-time simulator,
- (re-)parameterization at runtime without recompilation,
- configurability via a graphical editor.

3.4 Diagnosis and Measuring Interfaces to the ECU

Modern vehicle ECUs have failure memory mechanisms that enable the storage of specific failure situations that occur during operation, even in more than one drive cycle. If a failure
occurs, its type, source and specific operating conditions are stored. This information can then be retrieved later on, for example, during a diagnosis at the garage. This data is retrieved by diagnosis hardware that is usually connected to the serial diagnosis line of the ECU.

In an automated ECU test environment, the fundamental tasks are to check the correct entries in the failure memory after the stimulation of specific failure situations and reset them, if necessary. In the project described in this paper, the VAG 1551/1552 diagnosis hardware used by the VW Group was integrated in the test automation environment via an RS232 remote-control interface. In addition to failure memory management, the variant coding, release of optional functions, and reading of pre-defined measurement data blocks can also be used. During future diagnosis projects it will be possible to use the increasingly popular standardized diagnosis protocol KWP2000 [2], and thus to have solutions common for all relevant companies.

Diagnosis hardware normally does not allow time-stamped access to arbitrary RAM cells in the ECU. However, this is always necessary whenever reaction times need to be measured. It is therefore necessary to integrate a calibration interface into the test automation software. Since no dual-PC solution could be used in connection with an external calibration system, an ASAP-3 interface was ruled out [3]. With this calibration interface, static parameter labels (such as threshold values in the form of scales, curves, or engine characteristics maps) can be read. On the other hand, it is possible to measure dynamically changeable values with time stamps.

3.5 Documentation and Test Report Generation

The documentation and report generation software must be intelligent enough to influence the documentation's degree of detail. When an overnight test run is successful, there is just a list of relevant "PASSED" remarks waiting for the development engineer in the morning. An archive of recorded measurements is necessary only when there is a failure, since the engineers need this archive in order to perform troubleshooting. To fulfill such a requirement, a configurable logging mechanism was implemented and was specially tailored to the data formats of the diagnosis hardware and calibration interfaces, as well as to the access routines of the real-time simulation.

Test reports and diagnosis descriptions can be automatically generated in MS-WORD or MS-EXCEL documents via a programming interface that uses OLE automation, Window's standard communications technology. As a further enhancement, a hierarchically organized, hyperlink-equipped version is being planned for direct placement in company-internal intranets.

4 Application Example: Diagnosis of the λ-Probe

A typical application of on-board diagnosis is the monitoring of the λ-probe(s). In this case, the λ-probe is observed to determine numerous possible failure causes. Among other things, damages in the wiring of the λ-probe must be recognized indirectly via the electrical output
signal, and the current functional range detects different failure states of the \(\lambda\)-probe. The following section of this chapter provides an example of a \(\lambda\)-probe that was contaminated by leaded fuel. A two-point \(\lambda\)-probe contaminated in such a manner is usually not able to transmit the air-fuel ratio jump to the engine control unit within the normal voltage level. As a result, the performance of the \(\lambda\)-control is severely disturbed.

The diagnosis of the voltage limit is described by a voltage range (2 voltage thresholds) as well as by a few debouncing times that must first be extracted from the ECU’s memory during the pre-processing step.

- The engine simulator is started and the correct models and instrumentation panels are loaded. In this case, a specific operating point should not be started up because the monitoring function generally runs permanently.
- The fault is stimulated via the failure script. In this case, the fault is a restriction of the signal from the \(\lambda\)-probe to a defined limit. In parallel, a measurement of the diagnosis-relevant program values (\(\lambda\)-probe voltage, time, state value, etc.) is started. The resulting measurement data is stored temporarily during post-processing to evaluate the success of the diagnosis, and it can also be retrieved for further sequence control if needed.
- After all debouncing times are run through, an entry is placed in the failure memory. Afterwards, this is read and checked for plausibility.
- Next, the failure stimulation is removed (failure cure), and the success of the failure cure is monitored in the same manner as described above.

This test method of separating the failure stimulation and failure cure also allows the examination of several interwoven system faults so that any occurring transversal influences can be analyzed. During the final post-processing, a detailed analysis of the diagnosis run is carried out.

**References**

