presented at 9th International CTI Symposium Innovative Automotive Transmissions, Berlin, 30.11. - 01.12.2010, Berlin, Germany

Automated test of the AMG Speedshift DCT control software

M. Tatar
QTronic GmbH, Berlin

R. Schaich, T. Breitinger
Mercedes-AMG GmbH, Affalterbach

Abstract

AMG has recently completed the development of the AMG SPEEDSHIFT DCT. For development and test of the control software AMG used a novel test method based on a test case generator, and a software-in-the-loop (SiL) setup used to run the DCT control software in closed loop on a Windows PC. In this paper, we motivate and describe the corresponding test process and tool chain in detail, and discuss costs and benefit of this approach.

1 Introduction

For the validation and test of transmission controllers, established methods based on hand-written test scripts do not scale well. Testing the controller in real life by trying to expose the transmission under test to all relevant driving situations repeatedly during the development cycle is very time consuming or even not feasible without excessive effort. New methods and tools supporting a much higher degree of automation are required here, to meet shorter time-to-market and high quality demands.

In this paper we present one such method based on fully automated generation, execution and evaluation of useful test cases. In particular, we report how the corresponding tool, TestWeaver, has been used to validate and iteratively improve the control software of the DCT for a super sports car, the Mercedes-Benz SLS AMG [1].

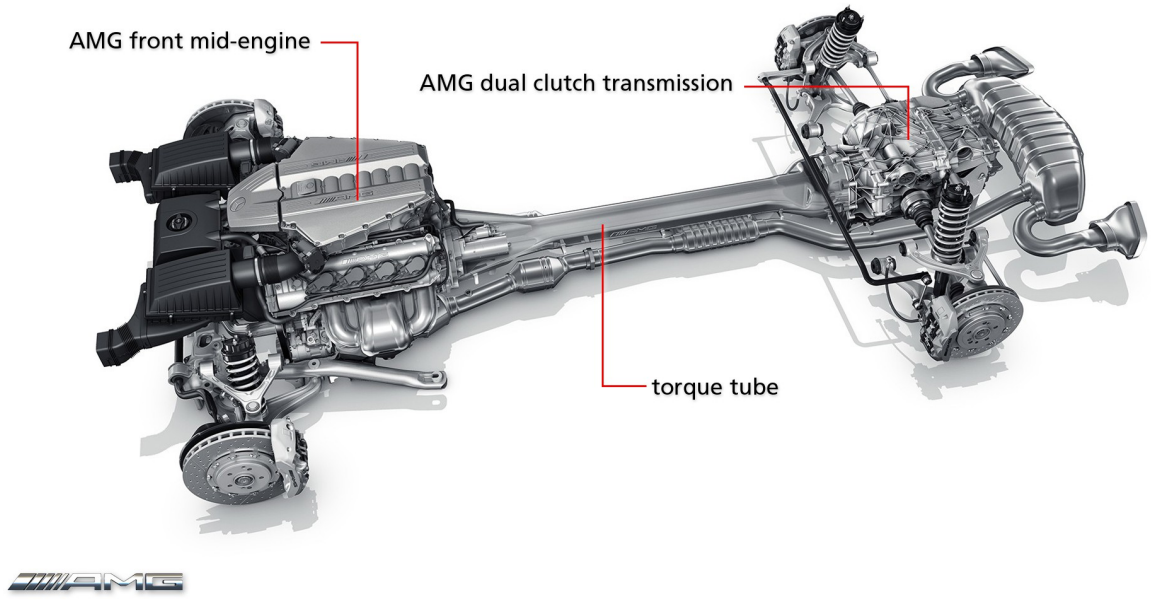

Fig. 1: The drivetrain of the Mercedes-Benz SLS AMG with the AMG front mid-engine M159, torque tube and the AMG SPEEDSHIFT DCT

The development environment for the DCT control software integrates the following components

- MATLAB/Simulink is used for model-based development of the DCT control software.
- TargetLink turns the Simulink model (about 150 modules) into high quality C code for two targets: the real TCU and the SiL / Silver platform described below.
- A well calibrated simulation model of the SLS AMG and the DCT hardware. The simulation code has been generated using Dymola from Modelica source code of the model.
- Silver as a tool for virtual integration using software-in-the-loop simulation. Silver imports both the vehicle model generated by Dymola and the TCU software generated by TargetLink as DLLs and runs them in a co-simulation on standard Windows PC. In addition, Silver provides interfaces to automated system test, the A2L database to integrate calibration data into the simulation loop, and XCP, to support virtual calibration and measurement, using the same protocols as in a real car.
- The test case generator TestWeaver [2, 3] to automatically generate, run and assess thousands of different driving scenarios for comprehensive system test during development of the DCT control software.

We see the main benefit achieved by using this tool chain in the:

- *speed-up of the development process*: Development and application of the DCT control software has been performed with only 6 software development cycles in a relatively short time. The observed shortening of development time is partly credited to the simulation-based tool chain sketch above. It enabled the developers to perform more engineering and test tasks on their PCs, avoiding blocking of rare resources like physical prototypes and HiL test rigs, and thus contribute to speedup the entire workflow.
- *increased safety of the development process*: The automated generation of high-quality test cases in conjunction with automated test execution and assessment via SiL enabled us to perform a much higher number of test cases than possible with same effort using conventional test methods. This increased the overall safety of the development process.

The paper is structured as follows: in the next section, we describe the environment used for the virtual integration of the essential modules of the DCT control software. Section 3 gives a brief description of the DCT simulation model. Section 4 describes the automated test and validation process, and Section 5 provides a critical assessment of the presented approach, with focus on costs and benefits.

2 Virtual integration of the DCT software modules

The development environment for the DCT software contains an incremental build system used to integrate and build the control software for two target execution platforms:

(a) the binary for the target TriCore TCU and

(b) the binary (DLL file) for the PC target used by the software-in-the-loop (SiL) setup.

This way a developer can compile the module that he is currently developing, link it with the object files of all other modules and immediately run the resulting integrated control software on his PC, to test the effects of his last changes in the context of the whole system (Fig 2). The entire compile and build loop takes less than 10 minutes. The SiL setup provides access (plot and control) to thousands of variables of the control software that are listed in the A2L file, and to over 2800 variables of the simulation model. Simulation can be stopped at any time to inspect these variables, to attach a debugger, and to inject faults by changing variable values. The simulation can be driven by a measurement MDF file or by a Python script, for instance, in order to trigger the adaptation algorithms that adapt certain parameters of the DCT control software to properties of the simulated car. The resulting adaptation values can be saved to files and reused later during the automated test. Moreover, the application data (hex or par files as used for the target) can be 'flashed' into the simulation. This way, it is possible to test the control software including the

application data and the data resulting from adaptation procedures on the developer's PC. Measurement and calibration tools (such as CANape or INCA) can be attached to the SiL setup via XCP. This way, the plotter and measurement masks used in the real vehicle or on test rigs can be used on the PC as well.

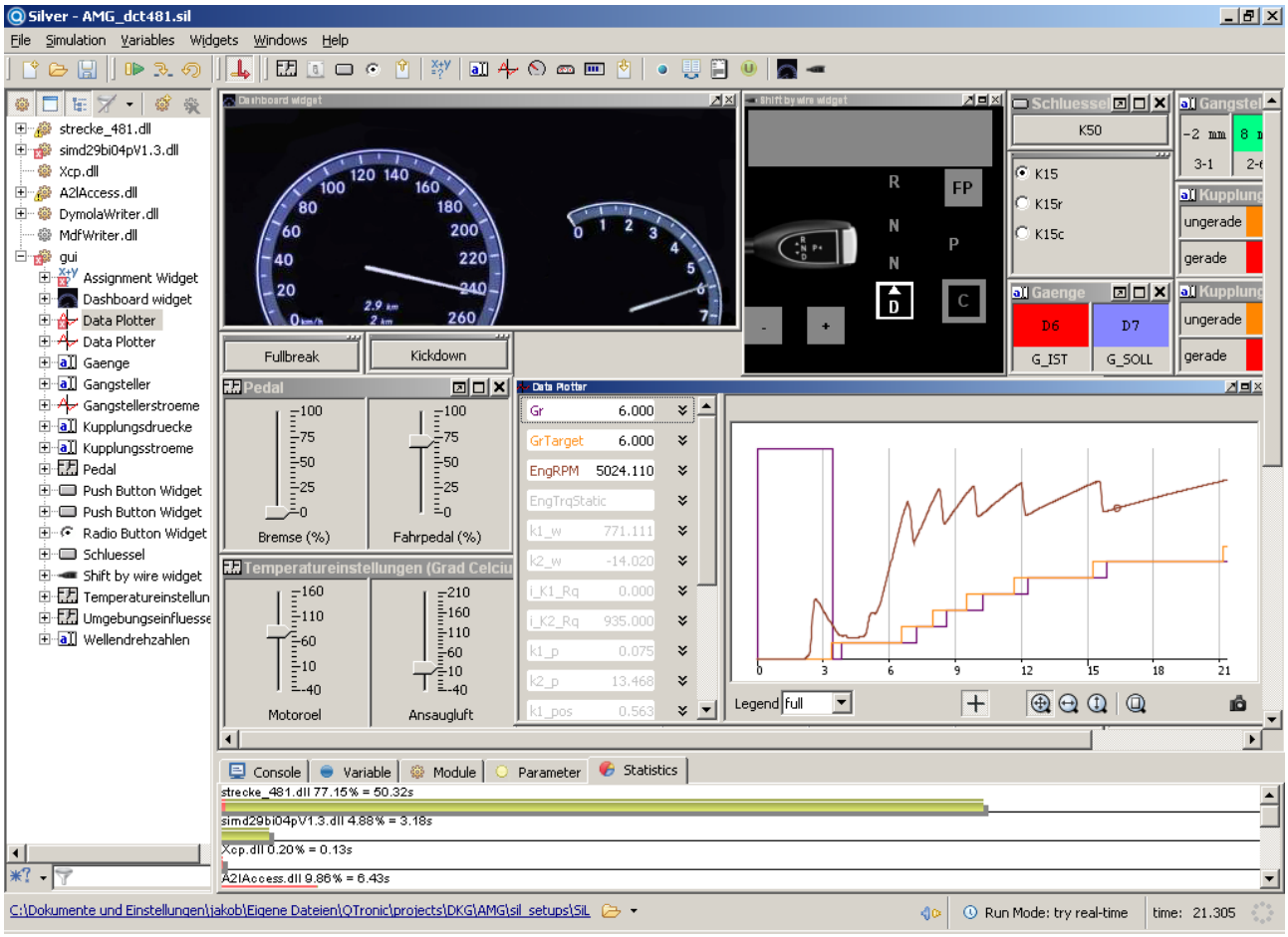

Fig. 2: DCT control software running in Silver

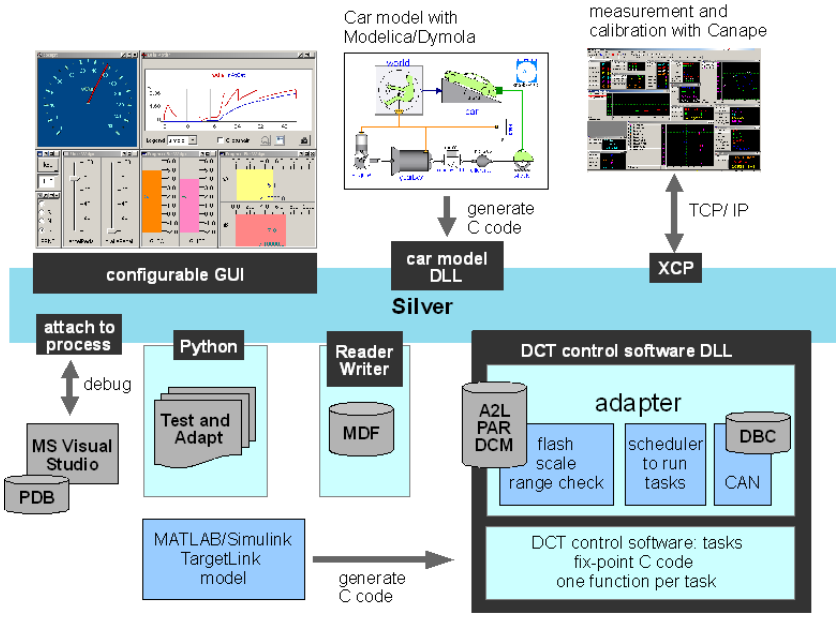

Fig. 3: Features of the SiL setup for the DCT control software

A key issue when setting up a SiL is how to actually integrate the control software. In the case reported here, we used adapter code (bottom right in Fig. 3) to integrate the controller code. The

adapter contains interfaces to support flashing of hex or par files, automatic scaling of integer variables based on given A2L files, scheduling of all tasks with given sample rates, CAN emulation based on given DBC files and error-handling emulation.

3 Simulation model of SLS AMG with DCT

Model-based test of control software requires a simulation model of the controlled system. Quality assessment of the control software typically requires a well calibrated simulation model, while coding errors can often be found using quite simple models. In the case reported here, the required vehicle model of the SLS AMG sports car was developed by QTronic using Modelica/Dymola and QTronics's Auto component library. The model contains 2844 variables, including 34 continuous state variables. It models the longitudinal dynamics of the vehicle and includes (cf. Fig. 4): a road model with different surface properties (dry / wet asphalt, ice), wheels with given slip characteristics, a car body with given air and driving resistance and dynamically varying axle load, a model of the combustion engine M159 including the protocol to serve torque requests of the TCU, and a detailed model of the DCT hydraulic unit.

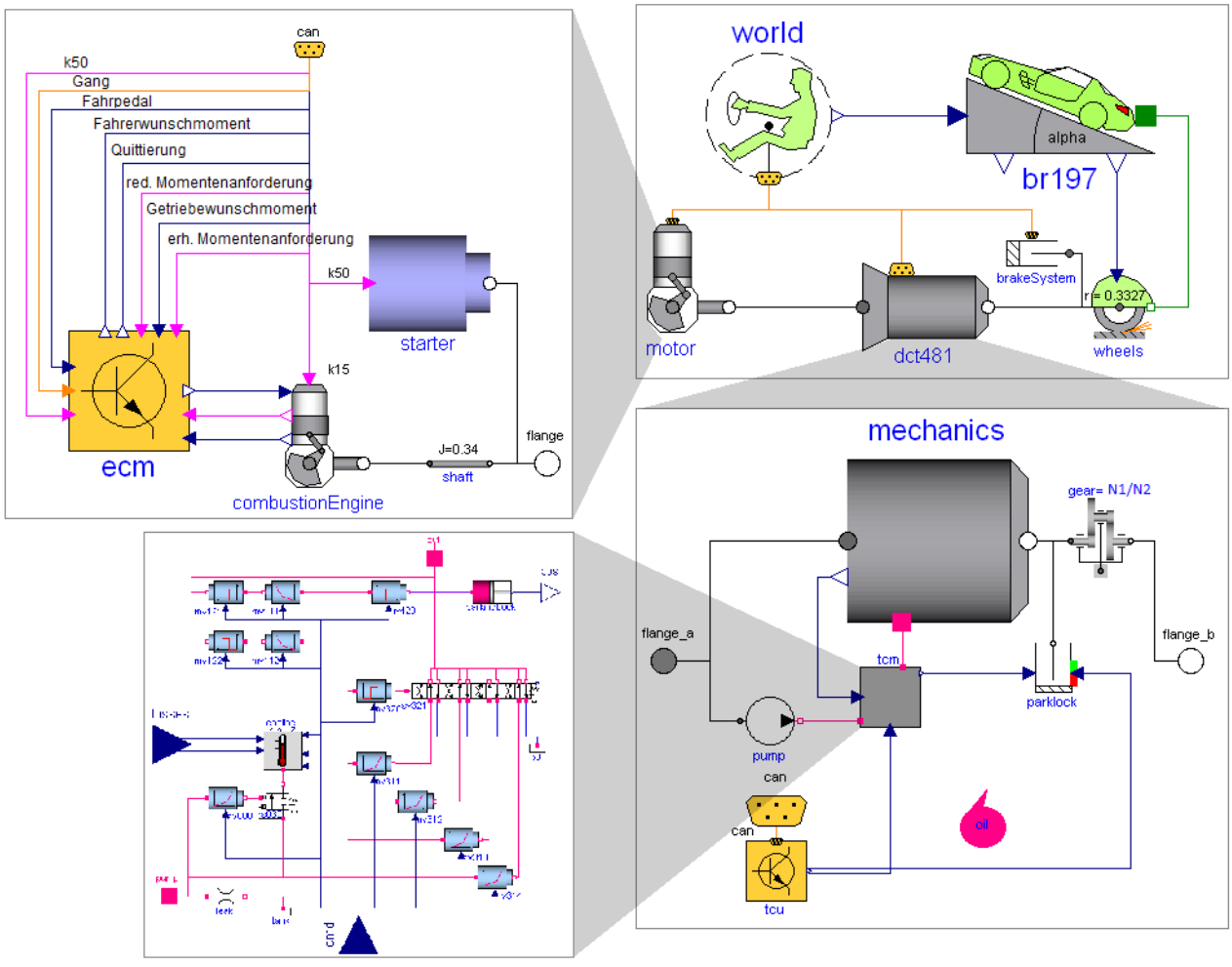

Fig. 4: Modelica model of the Mercedes-Benz SLS AMG

The DCT hardware has been modeled as shown in Fig 5. Each clutch of the double clutch is modeled by a hydraulic piston moved against friction forces and two springs with given non-linear characteristics, generating a force to control the torque transmitted by the clutch (bottom left of Fig 5). The mechanical part of the DCT is decomposed into four gear actuators. Each actuator allows to hydraulically activate one of two gears. Special care was taken to calibrate the dynamic behavior of the double clutch and of the gear actuators. The simulation code generated by Dymola from the Modelica model executes approximately in real time on a Windows PC with 1.69 GHz Pentium and 2GB Ram.

It took about two person weeks to develop an initial version of the simulation model as sketched above. This surprisingly low initial effort is credited to the use of Modelica as a high-level modeling language and also to the use of the Auto library which provides a mature starting point for modeling transmission systems. The initial model was, however, only good enough to run in closed-loop with the DCT control software without forcing the control software into emergency mode. At this stage, the SiL setup could only be used to find hard coding or application errors and to check whether the modules roughly work together as expected. In order to also support more subtle quality assessments, e.g. of micro slip control of the double clutch, further work was needed for calibrating the parameters of the model. This took a few months of continuous work, based on repeated measurements performed on test rigs and vehicle prototypes.

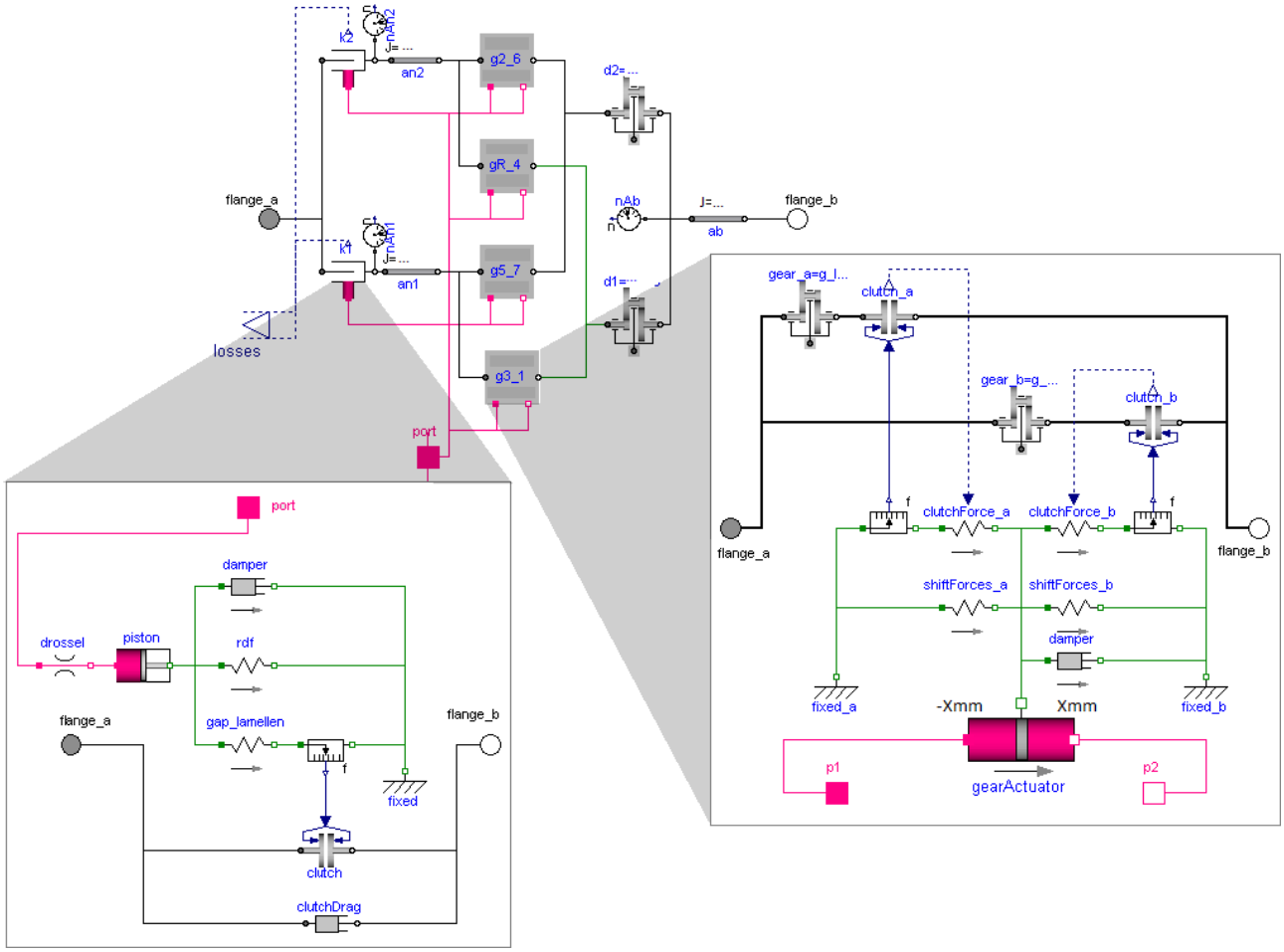

Fig. 5: Modelica model of DCT mechanics, clutch k1 and actuation of gear 1 and 3 detailed

4 Automated testing the DCT software

For the automated test of the DCT control software, the SiL setup is driven by a sequence of inputs dynamically generated by the test case generator TestWeaver. The inputs control the road properties, acceleration and brake pedals, PRND lever position, and may also be used to activate dynamically (simulated) component faults, e. g. of sensors or hydraulic valves. Selected outputs of the simulation (such as car speed, engine torque and key variables of the controller) are observed by TestWeaver and stored together with the inputs in a data base, labeled 'state DB' in Fig. 6. The test case generation, execution and evaluation does not require any user interaction and is interleaved: a new test case depends on the outcome of all previously generated tests. TestWeaver generates tests not randomly (this does not help much), but in a reactive, informed way, trying to worsen actively scenarios that are already sub-optimal until system behavior is really bad, i. e. a bug or flaw has been found. Here, a 'bad' scenario is by definition a scenario where an output variable reaches a value classified as 'bad' in the test specification, see below. TestWeaver

also attempts to maximize the coverage of the system state space, i. e. to reach every reachable state in at least one of the generated scenarios. As indicated in Fig. 6, the state space is here the space spanned by all inputs and outputs that connect the system under test to TestWeaver. Maximizing state coverage is non-trivial, because TestWeaver can only control the inputs directly, not the outputs. For example, TestWeaver cannot set the speed of the car (an output of the model), but it can learn that pushing the acceleration pedal (an input of the model) for a while leads to high vehicle speed. To guide scenario generation, TestWeaver stores each state reached during simulation into a state data base, together with the sequence of inputs that leads into this state. Thereby TestWeaver successively learns how to control the system under test. TestWeaver uses this knowledge to drive the system into states not reached before (to maximize state coverage) and to worsen scenarios locally by automated variation of those already generated scenarios that got worst scores.

For testing a system with TestWeaver, no test scripts need to be specified. Instead, a test or development engineer provides a compact test specification with the following information:

- names of input variables, allowed set of discrete values, and classification of these input values on a good-bad scale, to support fault injection
- names of output variables and classification of output values on a good-bad scale, to support automated evaluation of generated scenarios during execution
- templates for reporting the reached coverage in the state space, reached code coverage and other test results
- general specification data, such as maximal duration of generated scenarios, upper-bounds for injected faults per scenario, command used to start the simulation, etc.

TestWeaver reports the test results using HTML (Fig. 7). Report templates use SQL - a standard for data bases - to define the content of the tables. All scenarios generated by TestWeaver can be replayed by the test engineer on demand, e. g. all TCU and all model signals can be plotted for detailed investigation and debugging.

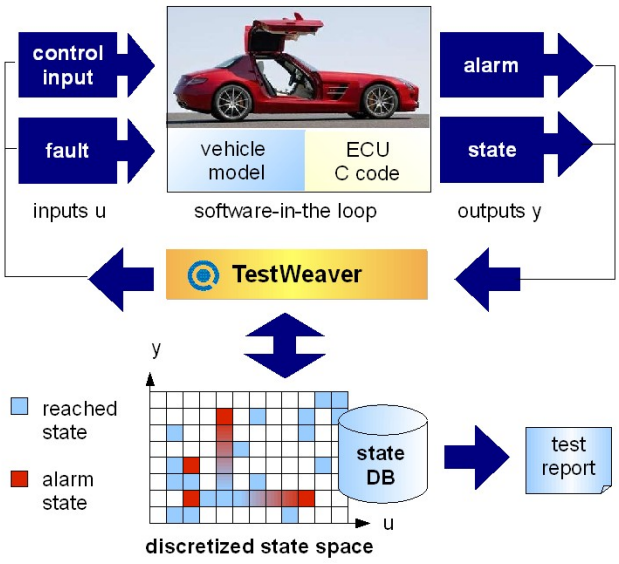

Fig. 6: Setup of the automated DCT software test

For testing the DCT control software, TestWeaver was configured as follows. First, alarm variables where defined in the SiL setup (or activated if predefined) to detect and report the following situations:

- *Runtime exceptions*: report division by zero, access violation, stack overflow, timeouts e. g. due to slow simulation, and other exceptions. This kind of alarm is a built-in feature of Silver and does not require special setup.

- *A2L range violation*: monitor and report out-of-range values of integer variables. These kind of alarm is configured by passing the corresponding A2L file to Silver, which activates range monitoring for all TCU signals (thousands of variables). The range of each variable is checked at the end of each control cycle, i.e. every 10 ms.
- *Shift duration*: all shifts are monitored, average and maximal shift durations are reported for each shift class.
- *DCT condition monitoring*: alarm variables have been defined to detect and report about 200 symptoms that indicate potential problems. Examples for such alarms: diagnostic trouble code generated by DCT software without a reason (i. e. without a fault injected by TestWeaver into the simulation), unwanted oscillations of certain signals generated by the DCT control software, e. g. of the target gear, unwanted oscillations of controlled variables, e.g. current for magnetic valves, high temperature of a (simulated) clutch, engine over-speed or stalled.
- *Code coverage*: the code coverage achieved by running the scenarios generated by TestWeaver is measured and reported. This feature is achieved by compiling the DCT control software with a special flag that activates measurement code. For measuring the code coverage, CTC++ by Testwell has been used, which is integrated with the TestWeaver report generator.

TestWeaver was further configured to automatically vary the following inputs to create scenarios:

- acceleration pedal: 0%, 10% 25%, 50%, 85%, 100%
- brake pedal: 0%, 25%, 50%, 75%, 100%
- transmission selector lever: one of P, R, N, D
- drive program selector: comfort, sport, super-sport, manual
- manual shift: up, down, or neutral, used if manual drive program is activated
- road steepness: -20%, 0%, 20%
- fault injection: several input variable used to inject component faults into the simulation.

Test results are reported by TestWeaver using tables (HTML), whose content is specified by report templates when setting up a test project. Fig 7. is an example for such a generated report. The table lists scenarios that contain range violations. Other problems detected by TestWeaver are reported in a similar way.

Monitoring the ranges of signals after time 1s

14 out of range signals -- names not containing "_L2":

Name	A2L Min	A2L Max	Sim Min	Sim Max	Scen Min	Time Min	Scen Max	Time Max	Coverage
accVehX_VSC	-10.24	10.16	-12	14	s638	27.515	s209	17.885	127.451
trqClPrev_CctcVSWA	-500	1000	-500	1145.44	s0	1.005	s1186	28.755	109.696
EngRPM_Max_Rq_DCT_OcptVUW	0	8190	8192	8192	s0	1.005	s0	1.005	0
EngRPM_Max_Rq_DCTCp_OcptVUW	122880	131070	122878	122878	s0	1.005	s0	1.005	0
EngRPM_Rq_TCM_OcptVUW	0	8190	8192	8192	s0	1.005	s0	1.005	0
EngRPM_Rq_TCMCp_OcptVUW	122880	131070	122878	122878	s0	1.005	s0	1.005	0
intrvtnMd_TCM_OcptVUC	0	1	1	3	s0	1.005	s0	6.555	200
intrvtnMd_TCMCp_OcptVUC	254	255	252	254	s0	6.555	s0	1.005	200
TxDnShiftMd_OcptVUC	0	1	3	3	s0	1.005	s0	1.005	0
TxDnShiftMdCp_OcptVUC	254	255	252	252	s0	1.005	s0	1.005	0
TxShtRcmnd_DispRqTCMCp_OcptVUC	0	252	254	255	s6	6.875	s0	1.005	0.396825
rpmTgtRefNom_SccpVSW	0	9999	0	10129	s0	1.005	s292	7.715	101.3
prctAccpUpsh_SdgpVUW	0	100	0	107.313	s0	1.005	s905	35.49	107.313
trqBrkEsp_RcesVUW	0	12285	0	59151	s0	3.305	s0	1.005	481.49

3 out of range signals -- names containing "_L2":

Name	A2L Min	A2L Max	Sim Min	Sim Max	Scen Min	Time Min	Scen Max	Time Max	Coverage
intrvtnMd_TCM_L2siVUC	0	1	1	3	s0	1.005	s0	6.555	200
TxDnShiftMd_L2siVUC	0	1	3	3	s0	1.005	s0	1.005	0
TxShtRcmnd_DispRqTCMCp_L2siVUC	0	252	254	255	s6	6.875	s0	1.005	0.396825

Fig. 7: Test results reported by TestWeaver

A scenario (e. g. **s638**) listed in the table can be inspected and replayed as a SiL simulation for detailed debugging by clicking on the corresponding link. The ability to exactly reproduce found problems as often as needed is an essential feature for fast debugging and tracking of found problems. Consider as an example the scenario shown on top of Fig. 8. In this case, the target

gear computed by the DCT control software toggled between gear 5 and 6. Fig 8 (bottom) shows the same scenario after fixing the corresponding problem.

Fig 8: Oscillating target gear (green) before and after a bug fix

The development of the DCT control software took only 6 software development cycles. Each release of the software was tested also with TestWeaver, performing over 3000 qualitatively different driving scenarios for each test. This means that, for each software release, thousands of different shift actions have been tested against hundreds of correctness and quality indicators. Many usual but also many unusual driving situations were generated and tested this way, for instance: changing the pedal positions and pressing all kind of buttons during the shifts or during the engine interventions, for all kind of shifts, with all drive programs, all slope classes, etc. This systematic analysis of the possible system conditions is only possible in simulation and is an important complement to the other test and quality assurance measures, such as test benches and prototype test.

5 Conclusion

We presented an approach for an automated test of automotive control software based on SiL simulation on standard PCs and intelligent generation of test cases. We demonstrated the successful application of the method during the development of the control software of the AMG SPEEDSHIFT DCT. The presented approach has an excellent cost/benefit ratio, resulting in high test coverage with moderate work load for development engineers. Moreover, the SiL setup of the DCT software, initially meant to just support test automation, proved to be a valuable tool for assessing, analyzing and debugging the system behavior in itself.

Acknowledgments

The authors wish to thank Victor Torres Toledo from Daimler AG for analysis and help during the calibration of the vehicle model and for valuable improvements of the software test setup.

References

[1] M. Hart , R. Schaich, T. Breitinger, F. Eichler: The function development and application of the DCT in the Mercedes-Benz SLS AMG. VDI-Berichte 2081: Getriebe in Fahrzeugen 2010, pp. 599-615.

[2] A. Junghanns, J. Mauss, M. Tatar: Testautomation based on Computer Chess Principles. 7th International CTI Symposium Innovative Automotive Transmissions, Berlin, 2 - 3.12.2008.

[3] K.-D. Hilf, I. Matheis, J. Mauss, J. Rauh: Automated simulation of scenarios to guide the development of a crosswind stabilization function. 6th IFAC Symposium Advances in Automotive Control, July 12-14, 2010, Munich, Germany.