INTELLIGENT TEST-CASE GENERATION FOR AUTOMATED VALIDATION OF TCUs.

Lionel Belmon, Technical Director, Global Crown Technology, China
Yijia Xu, Software Engineer, DCT Project engineering dept., SAGW/SAIC group, China

ABSTRACT

Automatic transmissions are difficult to test and validate, due to the complex interaction of mechanics, hydraulics, electronics, and transmission control software. This paper introduces how simulation-based automatic generation of test scenarios can maximize test coverage and reduce test workload at the same time. This tool, TestWeaver, has been applied by SAGW during the development of the application software for a new Dual-Clutch Transmission.

TestWeaver is based on a novel approach that aims at maximizing test coverage with minimal work load for the test engineer specifying test cases. TestWeaver can be applied to Simulink models (MiL), SiL setups or HiL setups. For testing a system, TestWeaver does not require any test scripts. Instead, TestWeaver generates, runs and evaluates thousands of tests cases (driving maneuvers) automatically - using a unique technology for intelligent test generation and evaluation of reactive systems. The intelligent generation strategy guarantees high test coverage with reduced work effort for engineers. For their DCT application, SAGW used TestWeaver with Simulink models to identify issues such as gearshift oscillations, overtime shifts or clutch overheat. 10,000's of scenarios are generated overnight and provides fast feedback about maturity of a software release, typically within 12 hours. Specific uses cases and applications at SAGW are introduced.

Besides MiL application, TestWeaver can also be connected to HiL systems in various manners. However, the main drawbacks of HiL testing are (1) much slower speed compared to MiL/SiL and (2) increased complexity due to simulation non-determinisms, management of fault codes and EEPROM reset. At SAGW, TestWeaver scenario generation is done on MiL to quickly reach a high coverage. Selected scenarios are then reproduced and verified on HiL.

INTRODUCTION

Testing and validating automatic transmissions is a complex, time-consuming and error prone process. The TCU software must interact dynamically with the vehicle and this leads to a very large numbers of states to be covered by testing. If some states are not covered, it is possible that issues and bugs slips out of the test phase, only to be found out later during the development cycle, or worse, after the SOP. Intensive testing with high test coverage during the TCU software development should be performed in order to reduce the risks that bugs might slip out.

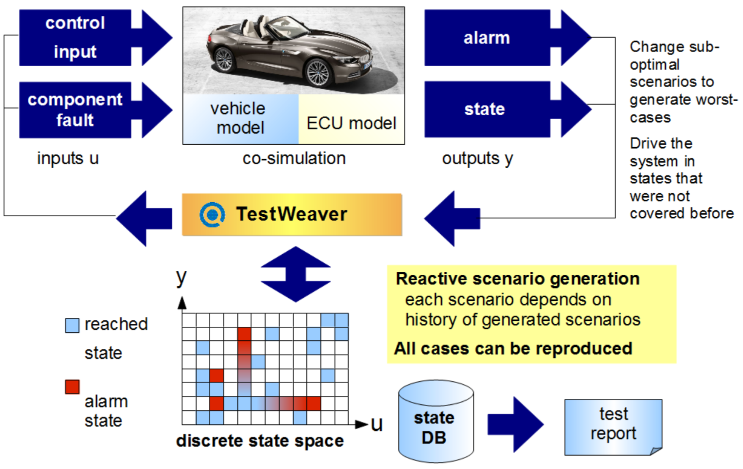
SAGW is currently developing a new DCT and is applying an automated test method for validating the transmission control unit. The System Under Test (SUT) is a Simulink model containing the following parts, running in closed loop simulation:

- The TCU application software model
- A plant model of the DCT gearbox and vehicle

This SUT is a relevant object for testing because SAGW uses the application software model to perform code generation. The resulting c-code is the one which is downloaded to the TCU hardware in the car. Moreover, the plant model used is realistic enough to reproduce the main dynamics and behavior of the real vehicle and gearbox.

PRINCIPLES AND IMPLEMENTATION OF TESTWEAVER

The driving concept in TestWeaver is to automate the generation and the analysis of test scenarios. The SUT provided to TestWeaver is a black-box which only exposes pre-defined input/ouputs, so called instruments. The nature of the SUT does not matter and could be a Simulink model[1], a Virtual-ECU/SiL based simulation [2],[3] or a HiL based simulation. Determinism of the SUT is however expected and un-deterministic behavior is checked and reported. TestWeaver will generate and execute scenarios for the SUT and apply complex algorithms in order to maximize the reached states coverage. A schematic is provided in the following figure. Further details on TestWeaver can be found in [4].


control input
component fault
inputs u
vehicle model
ECU model
co-simulation
alarm
state
outputs y
Change sub-optimal scenarios to generate worst-cases
Drive the system in states that were not covered before
TestWeaver
y
u
reached state
alarm state
discrete state space
Reactive scenario generation
each scenario depends on history of generated scenarios
All cases can be reproduced
state DB
test report

For Simulink based SUT, TestWeaver provides a blockset containing the TestWeaver instruments. Instrumenting a Simulink for use with TestWeaver is done by adding and configuring the required instruments to the Simulink model. This step should be as non-intrusive as possible and we took care during the implementation to avoid modification to the original model structure. At SAGW, the following additions where done to the model :

- Instruments for letting TestWeaver control system inputs such as the accelerator/brakes/gear lever/slope where added to the driver subsystem.
- Instruments for reporting all necessary outputs to TestWeaver where collected intto a new subsystem, using GO-FROM blocks from various parts of the system.

The instruments don’t modify the simulation behavior and are inactive/bypassed as long as TestWeaver is not connected to the SUT. The configuration of each individual instrument is based on system specifications and requirements. For instance, engine and shafts speed limits/overspeed are specified in the reporting instruments. We illustrate the implementation in the simulink models in the following figures.

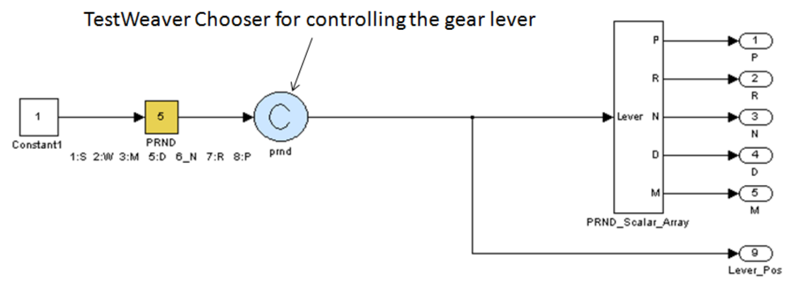

TestWeaver Chooser for controlling the gear lever
1
Constant1
5
PRND
1:S 2:W 3:M 5:D 6_N 7:R 8:P
prnd
Lever
P
R
N
D
M
PRND_Scalar_Array
1 P
2 R
3 N
4 D
5 M
9 Lever_Pos

Figure 1: Example of implementation of a Chooser for the gear lever control

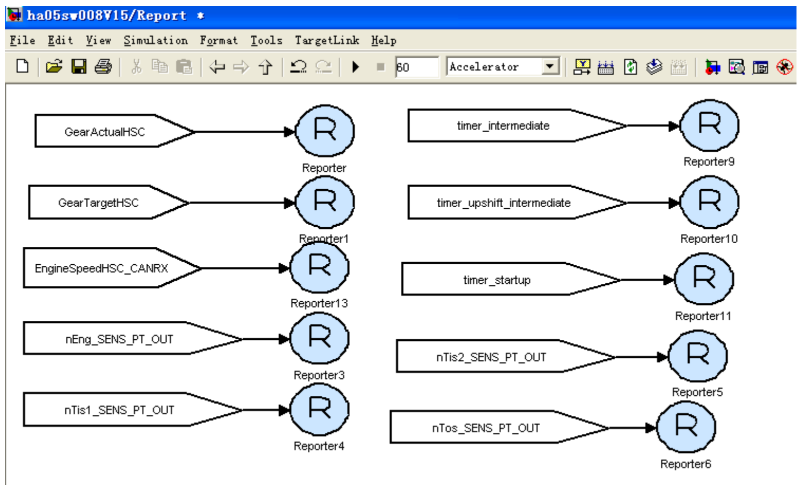


Figure 2 : Example of implementation of a Simulink subsystem for reporting outputs to TestWeaver

Once instrumented, the SUT is then compiled with Simulink RTW as an executable program that contains the interface functions to TestWeaver. This executable program (.exe) is then called by TestWeaver to perform simulations. This allows high speed simulation and scenario generation, several times faster than real-time.

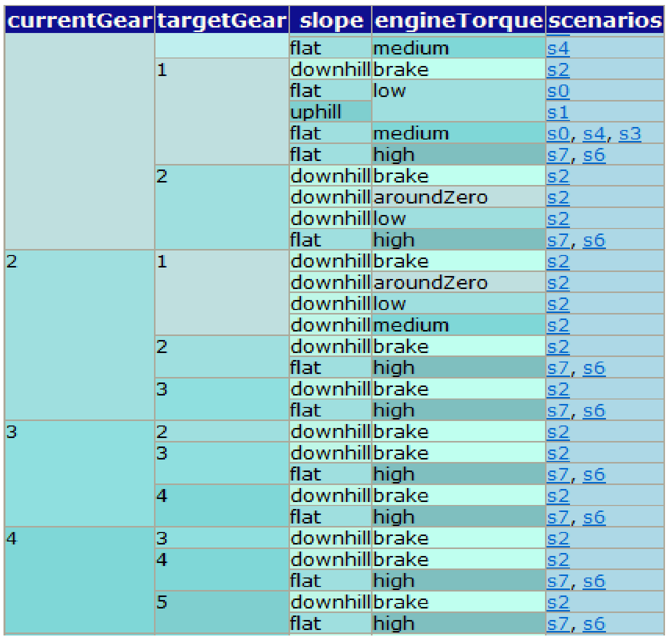
Since TestWeaver generates 1000’s of scenarios, it is not practical to evaluate the scenarii results by hand. TestWeaver provides an automatic reporting/analysis system that needs to be configured according to our project requirements. The reporting system will explore and dig through the huge data generated from the 1000’s of scenarios generated by TestWeaver. This introduces a shift of paradigm in testing. The usual way of thinking for testing is script based : „how to write a script that will go from 0% accelerator position to 100% accelerator position during a 2-3 gearshift?“. In TestWeaver, instead, we write a query which will find *all* generated scenarios where such case happen. To summarize, instead of focusing on *how to generate* various states by scripting, we *look for* the states that we have interest in. A positive side effect is that TestWeaver will generate scenarios and sequences that were not thought of by the test engineer, increasing drastically the chances to find possible hidden bugs and issues.

Among variables of interest that we need to report, we monitor the Actual Gear, the Target Gear, the engine speed, the speed of each shaft and the gearshift time length. Monitoring these variables allows to find issues in the gearshift logic such as a jammed/blocked gearshift due to a TCU bug.

Finally, we also want to assess how well the SUT has been tested. The test coverage is a complex metric that should be evaluated from different angles :

(1) Code coverage : Did we test all functions, branches ? Did we reach all meaningful input/output values ?
(2) System states coverage : Did we reach all possible gearshifts ? Did we consider all possible environment conditions ?

The code coverage and system states coverage are not equivalent: It is possible to reach a high code coverage without reaching a high state coverage for instance. For a TestWeaver/Simulink setup, the code coverage is usually ignored since the final production c-code is not integrated into the simulation. We thus estimate test coverage by using the system state coverage only. The system state coverage is a metric that has to be defined according to the system specification. For automatic gearboxes, a relevant metric is the gearshift reached during testing. We illustrate an example of system state coverage in the following figure. When doing scenario generation, TestWeaver will try to maximize the state coverage.



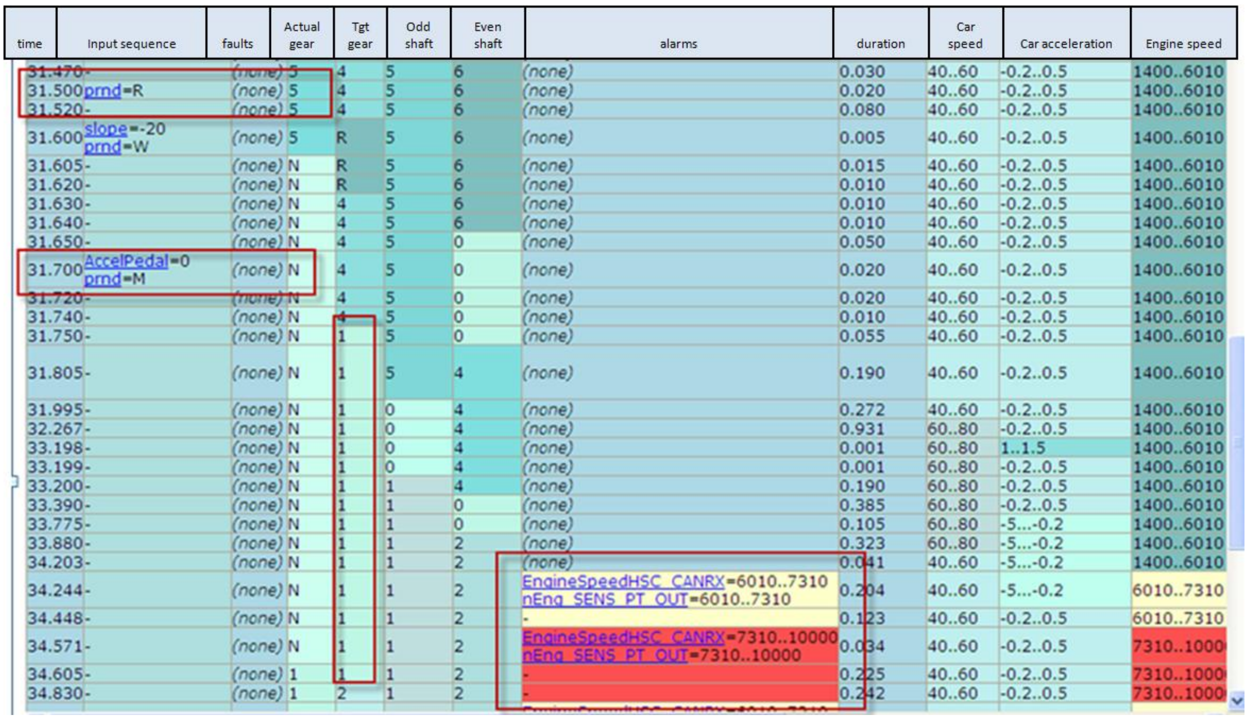
currentGear	targetGear	slope	engineTorque	scenarios
		flat	medium	s4
	1	downhill	brake	s2
		flat	low	s0
		uphill		s1
		flat	medium	s0, s4, s3
		flat	high	s7, s6
	2	downhill	brake	s2
		downhill	aroundZero	s2
		downhill	low	s2
		flat	high	s7, s6
2	1	downhill	brake	s2
		downhill	aroundZero	s2
		downhill	low	s2
		downhill	medium	s2
	2	downhill	brake	s2
		flat	high	s7, s6
	3	downhill	brake	s2
		flat	high	s7, s6
3	2	downhill	brake	s2
	3	downhill	brake	s2
		flat	high	s7, s6
	4	downhill	brake	s2
		flat	high	s7, s6
4	3	downhill	brake	s2
	4	downhill	brake	s2
		flat	high	s7, s6
	5	downhill	brake	s2
		flat	high	s7, s6

Figure 3: System state coverage : Reach gearshifts with various slopes and engine torques

APPLICATION AND USE CASES

For TCU testing, we generally first focus on generating scenarios where all possible gearshifts are performed, under various environment conditions (slope). TestWeaver will automatically report generic issues such as run-time exceptions, division by zero or memory errors. System specific requirements and criteria are checked through the implementation of instruments and reports queries. These can include variables such as shift durations, clutch temperature, shafts speeds, and Actual Gear upshift/downshift repeated oscillations.

We give in the following paragraph an example of how a bug was found by TestWeaver and how it was solved. During generation of scenarios, TestWeaver reported a case where engine speed is above the prescribed limit of 7300 rpm. We could open the report of this specific test scenario report to examine the input sequence. We provide the scenario report in the following figure.



time	Input sequence	faults	Actual gear	Tgt gear	Odd shaft	Even shaft	alarms	duration	Car speed	Car acceleration	Engine speed
31.470	-	(none)	5	4	5	6	(none)	0.030	40..60	-0.2..0.5	1400..6010
31.500	prnd=R	(none)	5	4	5	6	(none)	0.020	40..60	-0.2..0.5	1400..6010
31.520	-	(none)	5	4	5	6	(none)	0.080	40..60	-0.2..0.5	1400..6010
31.600	slope=-20 prnd=W	(none)	5	R	5	6	(none)	0.005	40..60	-0.2..0.5	1400..6010
31.605	-	(none)	N	R	5	6	(none)	0.015	40..60	-0.2..0.5	1400..6010
31.620	-	(none)	N	R	5	6	(none)	0.010	40..60	-0.2..0.5	1400..6010
31.630	-	(none)	N	4	5	6	(none)	0.010	40..60	-0.2..0.5	1400..6010
31.640	-	(none)	N	4	5	6	(none)	0.010	40..60	-0.2..0.5	1400..6010
31.650	-	(none)	N	4	5	0	(none)	0.050	40..60	-0.2..0.5	1400..6010
31.700	AccelPedal=0 prnd=M	(none)	N	4	5	0	(none)	0.020	40..60	-0.2..0.5	1400..6010
31.720	-	(none)	N	4	5	0	(none)	0.020	40..60	-0.2..0.5	1400..6010
31.740	-	(none)	N	4	5	0	(none)	0.010	40..60	-0.2..0.5	1400..6010
31.750	-	(none)	N	1	5	0	(none)	0.055	40..60	-0.2..0.5	1400..6010
31.805	-	(none)	N	1	5	4	(none)	0.190	40..60	-0.2..0.5	1400..6010
31.995	-	(none)	N	1	0	4	(none)	0.272	40..60	-0.2..0.5	1400..6010
32.267	-	(none)	N	1	0	4	(none)	0.931	60..80	-0.2..0.5	1400..6010
33.198	-	(none)	N	1	0	4	(none)	0.001	60..80	1..1.5	1400..6010
33.199	-	(none)	N	1	0	4	(none)	0.001	60..80	-0.2..0.5	1400..6010
33.200	-	(none)	N	1	1	4	(none)	0.190	60..80	-0.2..0.5	1400..6010
33.390	-	(none)	N	1	1	0	(none)	0.385	60..80	-0.2..0.5	1400..6010
33.775	-	(none)	N	1	1	0	(none)	0.105	60..80	-5...-0.2	1400..6010
33.880	-	(none)	N	1	1	2	(none)	0.323	60..80	-5...-0.2	1400..6010
34.203	-	(none)	N	1	1	2	(none)	0.041	40..60	-5...-0.2	1400..6010
34.244	-	(none)	N	1	1	2	EngineSpeedHSC_CANRX=6010..7310 nEng_SENS_PT_OUT=6010..7310	0.204	40..60	-5...-0.2	6010..7310
34.448	-	(none)	N	1	1	2	-	0.123	40..60	-0.2..0.5	6010..7310
34.571	-	(none)	N	1	1	2	EngineSpeedHSC_CANRX=7310..10000 nEng_SENS_PT_OUT=7310..10000	0.034	40..60	-0.2..0.5	7310..1000
34.605	-	(none)	1	1	1	2	-	0.225	40..60	-0.2..0.5	7310..1000
34.830	-	(none)	1	2	1	2	-	0.242	40..60	-0.2..0.5	7310..1000

Figure 4 : Scenario report with abnormally high engine speed

The examination of this scenario seems to point out that the gear shift logic is broken when shiftlever changes from R(Rear) to W(winter) and back to D(Drive). The target gear is set to 1 whereas the vehicle speed is already quite high (>60 km/h). We can notice that this bug appears with a downhill slope and with a complex unusual gearlever sequence. It would probably have been omitted from traditional script-based testing.

To degub this situation, TestWeaver can replay the scenario in the Simulink enviromnent so that the TCU logic can be followed and debugged. In the Stateflow subsystem defining the gearshift logic, the problem was isolated as an issue in a state transition from D to R and M to R, which sets a wrong value for a variable (t_gear_creep). We give an overview of the stateflow subsystem in the following figure.

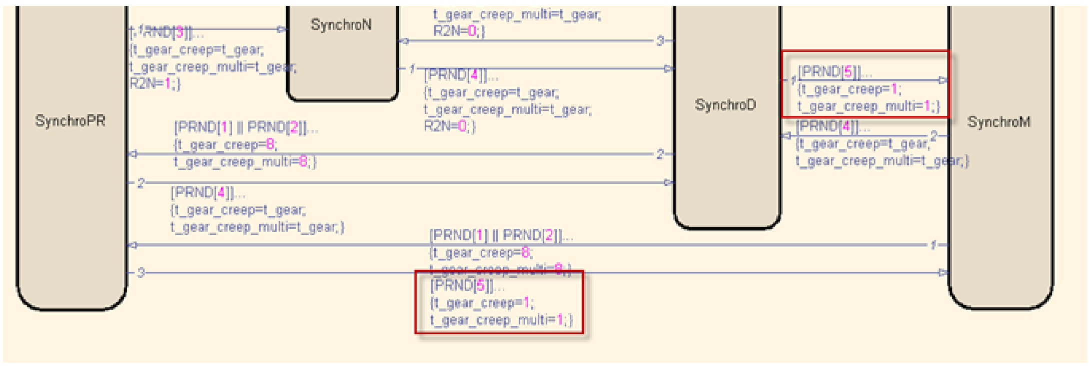


Figure 5 : bug isolated in the Stateflow model

A correction is applied to this transition and we can replay again the scenario to make sure that the problem is fixed. The model is then compiled with RTW and we let TestWeaver generate again scenarios to make sure that this modification does not introduce side effects. A TestWeaver scenario test database can also be used for non-regression testing.

Using TestWeaver/Simulink to debug logical bugs is much easier than debugging on HiL because replaying a scenario with TestWeaver/Simulink will highlight on which line the stateflow jamming occurs. If a first guess is made on where the bug is, then replaying with TestWeaver/Simulink can easily confirm of infirm this first guess, without wasting time on adding flags, compiling and building model to hex, then flashing to TCU.

TESTWEAVER AND HiL SYSTEMS

Hardware-in-the-Loop test is a well established technology and process in the automotive industry. TestWeaver fully supports connection to HiL systems such as dSPACE simulator, ETAS Labcar or NI Veristand. TestWeaver also supports CAN hardware and XCP protocol so that TestWeaver can automatically collect and reset fault codes on the ECU under test. HiL systems are powerful for validating controllers network integration and software/hardware integration. However, from a software testing point of view, HiL systems suffer from several drawbacks such as :

- Poor features for supporting software debugging
- Complexity and cost of the overall system.
- Real-time constraints. A HiL *must* run real-time, not slower, not faster.
- Undeterministic behavior. Repeating the same input sequence could lead to different outputs.
- Complexity and time needed for evaluating a software modification : code generation, compilation, flashing to TCU, flashing calibration, evaluating scenarios.

A TestWeaver setup for HiL will suffer from the same issues. Automatic scenario generation for reaching a meaningful coverage could take hours instead of minutes. However, there is still interest in *replaying* on HiL selected scenarios generated from a TestWeaver/Simulink setup. The motivation for HiL replay of scenario is :

(1) Verification of the real TCU behavior (instead of model) for specific scenarios
(2) The HiL setup replays much faster than Simulink. Simulink could take 10 mins to replay a 60s scenario while HiL, as a real time system, only takes 60s.
(3) Analyzing data on CANape is much easier than looking at several scope on Simulink, and CANape measurement file can be sent and shared by different software engineers.

Based on the above, it was decided not to connect TestWeaver directly to the dSPACE HiL simulator but to allow replay on HiL.

To replay a scenario on HiL, the first step is to export a csv file from TestWeaver, the CSV file describes the input sequence in time. The csv file is converted to a MAT file that the HiL can read. Each row of the MAT file contains several columns, which are shifter lever, accelerate pedal, brake pressure and slope respectively.We built a simulink model which can convert such csv files to MAT files. The simulink model makes sure the sample time between each row is 1 ms sperately, i.e. a 60s driving scenario will be converted to a 60,000 row MAT file. Then the MAT file will be loaded by the HiL plant model. In the plant model, the rows are read one by one, and all inputs will be set identically as in the Simulink generated scenario.

CONCLUSION AND LIMITATIONS

A method for automatic large coverage testing of a DCT Transmission Control Unit has been described in detail. The TestWeaver/Simulink setup introduced in this paper is an helpful tool for debugging and validating complex controls for automatic gearbox. However, this setup has several limitations :

(1) Calibration parameters are difficult to handle and to implement in the Simulink model. Overall, the configuration in the Simulink model is slightly different than the configuration tested in HiL or in the car.
(2) The TCU model is tested, instead of the TCU production c-code. Possible issues and bugs due to fixed point arithmetics / scaling will be missed.
(3) System state coverage is available but code coverage is not available
(4) Replaying scenario is slow due to the interpreted Simulink model.
(5) No connection with measurement tools such as CANAPE, no convenient writing/reading of measurements files (.mdf)
(6) CAN configurations (dbc files) are not included in the TestWeaver/Simulink setup

The above issues can be fixed with the implementation of a TestWeaver/Silver setup where the production c-code of the TCU is integrated within a Virtual ECU. Silver is a Virtual-ECU/SiL platform that provides the missing services and features described above. In terms of usage, the TestWeaver/Silver setup would be somehow a „virtual“ HiL, focusing of software testing and software debugging, without the hardware constraints of HiL simulators.

REFERENCES

[1] Test-driven development of DCT Control Software, *N.Papkonstantinou,S.Klinger,M.Tatar,* 8th International CTI Symposium Innovative Automotive Transmissions, 2009, Berlin.

[2] Automated test of the AMG speedshift DCT Control software, *R.Schaich, M.Tatar,* 9th International CTI Symposium Innovative Automotive Transmissions, 2010, Berlin.

[3] Model-based Development of Dual-Clutch Transmission using Rapid Protoptying and SiL,*H.Bruckmann,J.Strenkert,B.Wiesner,A.Junghanns,* Getriebe in Fahrzeugen, 2009, Friedrichshafen

[4] Test automation based on Computer Chess Principles, *A.Junghanns,J.Mauss, M.Tatar*,7th International CTI Symposium Innovative Automotive Transmissions, 2008, Berlin.