

such as real ECUs, HiL (Hardware-in-the-Loop) test benches or prototype vehicles.

- Increased robustness: The automated generation of high-quality test cases enabled us to perform a much higher number of test cases than possible with the same effort using traditional test methods. Especially for extreme driving conditions and fault insertion tests the new methods we apply here found many problems that would have been difficult to find on the vehicle.

2 Virtual integration of the CVT software and of the plant model

The development environment for the CVT software contains a build system used to integrate and build the control software into a binary file for the TCU hardware. We extend this build system by adding a SiL target to it, so that developers can execute their code directly on PC first.

The developer can compile the module that he is currently developing, link it with all the other modules' object files and run the resulting, integrated control software as a DLL file on his PC immediately, to test the relative effects of his last modifications in a closed loop environment. The system is described in Figure 1. This compilation and build process takes less than a few minutes normally. The SiL setup can also provide access to thousands of variables of the control software that are listed in the A2L file, and also to every variable of the simulation model. The Silver simulation can also be attached to a debugger for step-by-step C code execution, and for injecting faults through changing values of variables. In addition, the simulation can be driven by a measurement MDF/CSV file, from vehicle test drive, or by Python scripts in order to trigger a specific driving condition of interest. Besides, the calibration data (DCM file for this project) can be 'flashed' into the controller of the simulation. This means we can perform some pre-calibration work based the accurate plant model.

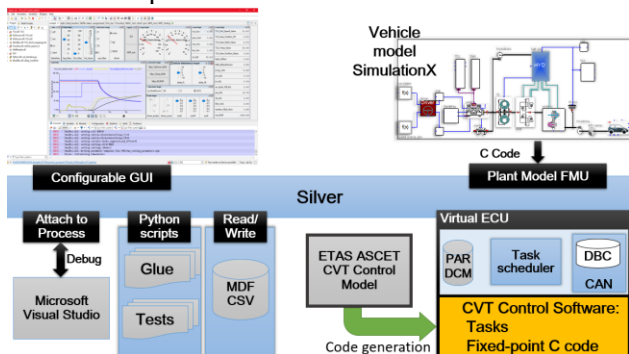


Figure 1 : Virtual integration of TCU and plant model

We execute the tasks for initialization as well as periodic tasks generated in the fixed-point C code of the CVT

control software. We also configure inputs and outputs variables needed for simulation and interaction between ASW (application software) and BSW (basic software). The original BSW is not included in the closed-loop simulation, it is emulated by SBS (Silver Basic Software). SBS emulates features such as task scheduling and non-volatile memory.

3 Vehicle model development and validation

3.1 Model development

To build a closed loop simulation environment, we need a vehicle model to match the CVT control software. For better simulation quality, we need a well calibrated simulation model, that can reproduce the vehicle behavior well, both in terms of logic, input/outputs dynamics and performances. Here the required vehicle model of the Xiaokang van was developed by Global Crown using SimulationX. It models the longitudinal dynamics of the vehicle and the following components are included (see Figure 2): a model of the combustion engine with engine maps, start/stop control logic and CAN, a torque converter based on a SimulationX library, a DNR clutch, a detailed CVT model with hydraulic and solenoid systems based on real parameters and diagrams, drive shafts and differential with stiffness, left/right tires model with slip characteristics, a car body with given air and driving resistance, a road model with different surface properties, a simple ABS model which can simulate blocking sequences with ABS control of the wheel speed. Detailed attention was paid to the push-belt and hydraulic models. The belt models can simulate slipping if the pressure of primary and secondary pulleys are not suitable with the respect to the input drive torque.

The model was also required to run on a HiL system, which means it had to satisfy real-time performance and uses a fixed-step solver.

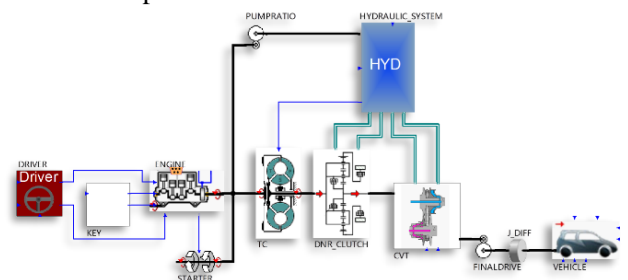


Figure 2 : Vehicle plant model schematic

The hydraulic system of the CVT has been modeled as detailed as possible based on real parameters and mechanical diagrams for every solenoid valves and oil channel. However, some high frequency physical effects are neglected because they are assumed to not influence the TCU. Such an assumption is also required

because the model must be solved with a fixed-step solver. The neglected high frequency effects include pressure dynamics inside feedback volumes of spool valves, pressure dynamics inside solenoid valves. The resulting model has 65 states.

One difficulty in running Modelica models for real-time application lies in the event management and the Differential Algebraic Equation (DAE) formulation of the models. Such DAE with events usually require solvers with iterative methods, which is in conflict with real-time requirement of predictable execution time for one time step. For real-time applications, explicit fixed-time step integration is commonly used in the industry for many years. SimulationX can generate a ODE system from the DAE system and a classical explicit fixed-step solver method can thus be applied. Concerning events, most of them are removed during the modeling, using Modelica NoEvent() function. Most of events indicators are not meaningful for a fixed-step solver and are only meaningful for zero-crossing finding algorithms of variable step solvers. Such events can thus be safely removed of the model.

Concerning the time step selection, the TCU having a 10ms sampling rate, a 1ms step or smaller was defined as a 'higher limit' on the time step. The 'lower limit' time step depends on the model execution time and simulator CPU speed. In the present case, the model could run fast enough to execute real-time with 0.25ms step. Finally a 0.5ms time step was used.

3.2 Model open-loop validation

Model open-loop validation is done to verify some properties such as hydraulic pressure control. In the model it is possible to directly control solenoid input currents and measure hydraulic output pressure on primary and secondary pulleys of the CVT. It is then possible to compare to measurements on the vehicle or on a hydraulic test bench. Some calibration of pressure-current (I-P) characteristics of solenoid valves is done.

3.3 Model export to FMI and integration

For integration within the Silver co-simulation, to achieve a closed-loop system with the TCU embedded software, we used the Functional Mock-Up Interface and exported the model as a FMU2.0 for co-simulation. Using "co-simulation" or "model exchange" should not make a significant difference in our case since both the TCU embedded software and the plant models are using fixed step methods. The TCU software is updated every 10ms, the plant model is computed every 0.5ms with a fixed step solver. All required inputs/outputs are configured and mapped to the TCU software inside Silver. Some debugging work is usually needed to make the system works properly. Once debugging is done and the simulation works (engine start-up, ratio control,

RND control, etc.), we proceed to closed-loop validation.

3.4 Closed-loop vehicle level validation

The closed-loop vehicle level validation is done by comparing simulation and prototype vehicle results for around 20 different drive situations, at various speeds, with various maneuvers. The maneuvers are selected so that we know the functional coverage of the model is good enough (DNR clutch, torque converter, hydraulic control, brakes, engine torque requests...).

When comparing vehicle results and simulation results, we must first make sure that TCU versions are close enough and that calibration parameters are the same, otherwise we might have very different behavior between vehicle and simulation. The simulation is executed in Silver with the plant model FMU, the TCU embedded software, calibration parameters and test sequences inputs read from measurements data files.

We give in Figure 3 an example of validation results obtained during the project. The key control variables are in good agreement between real vehicle and simulation. Such qualitative validation is done for the 20 drive test cases selected before. The time of building the initial vehicle model took about 6 weeks, the hydraulic system being the complex part of the model, and another 4 weeks for debugging and validation.

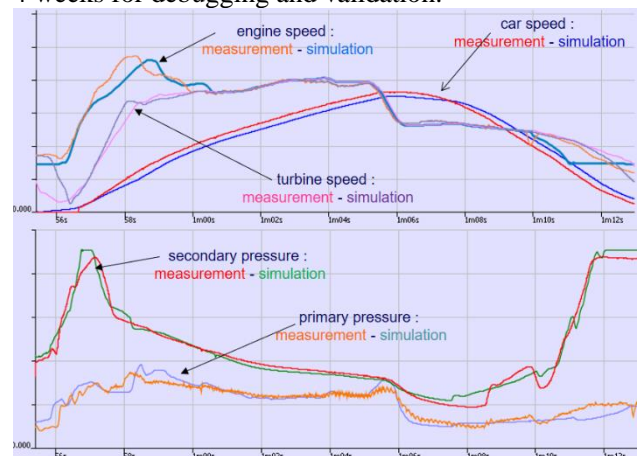


Figure 3 : Example of vehicle validation results

4 Automated testing for the CVT control software

It is possible to manually run tests in Silver, but this lacks the test automation and test reporting. Many simulations were done manually in Silver during debugging phases and for experimentation. For test automation and software validation, we applied TestWeaver, a test case generator, driving the SiL simulation through a sequence of inputs like acceleration and brake pedals, shift lever, road conditions, and fault insertions. As system states and system variables for monitoring, we selected meaningful signals, like engine speed, vehicle speed and

the current of solenoids. All these variables will be recorded during each test case generation and stored in a test database. TestWeaver tries to drive the vehicle to new states and new driving conditions not reached before, trying to maximize coverage of requirements system states and also executed code.

As shown in Fig. 5, the state space is made up by all inputs and outputs that connect the system for the test with TestWeaver. For example, TestWeaver cannot set the vehicle speed (a reporter/output here), but it can learn that pushing the acceleration pedal (a input/chooser here) for a given amount of time then the system gets a higher vehicle speed. And then TestWeaver stores this behavior in a test database for later tests. This way, TestWeaver successively learns how to ‘drive’ the vehicle to any specific condition.

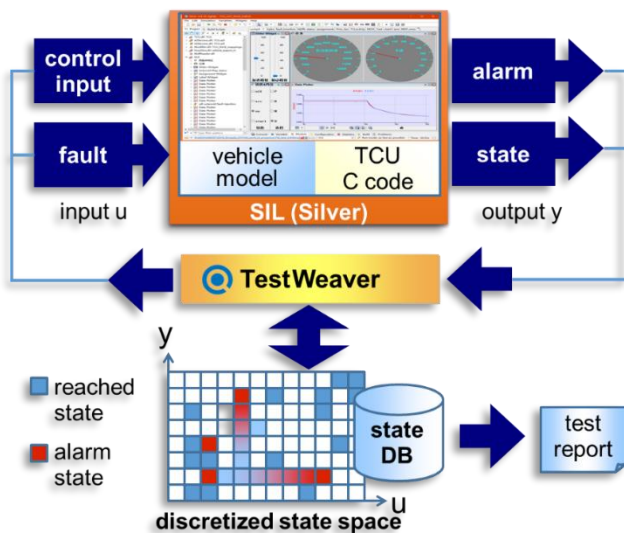


Figure 4 : Automated test of CVT control software by TestWeaver

Before we start automated system testing with TestWeaver, a test or development engineer will do the following work:

- ❑ Configuration of input signals choosers, partitions, occurrence definition for each partition. Fault injection variables are choosers too.
- ❑ Configuration of output signals reporters, partitions, severity definition for each partition, to support automated evaluation of generated scenarios during testing.
- ❑ Defining requirements monitoring using TestWeaver “watcher” instruments. Here complex logical requirements can be defined and TestWeaver can automatically report if the requirements have been tested, with success or

failure.

- ❑ Defining report templates for the variables we are interested in state coverage or code coverage or other specific testing results.
- ❑ General experiment configuration, such as the maximal duration of each scenario, maximal number for fault insertion per scenario, etc.

For testing the CVT control software, we need pay close attention to the following subjects:

- ❑ Ratio control of CVT: we need to monitor the difference between the actual ratio and target ratio to see if the CVT control software could control the ratio smoothly and precisely. Also if there are any oscillations in the ratio change, then there are probably issues on the ratio control.
- ❑ Diagnostic Trouble Code: for this part of the control code, we systematically check, using requirement watchers, that for a given fault condition the TCU executes a suitable fault reaction.
- ❑ Belt slip monitoring: for CVT, if belt slipping occurs, the lifetime would be shortened dramatically, even to the point of destroying the CVT. Therefore, we need monitor the slipping to analyze why this condition occurs.
- ❑ Code coverage: here the code coverage tool integrated in TestWeaver, was used. This feature is achieved by compiling the CVT control software with a special flag that tracks the C code coverage during execution in the Silver simulation.

In Figure 5, we show a report generated by TestWeaver that summarizes fault insertions on a solenoid and the monitoring of the fault reaction *Watcher_ffs_sesl*. TestWeaver detected several scenarios in which the TCU fails to apply a suitable fault reaction to the solenoid fault, which might lead to a dangerous situation for the CVT with a loss of transmission ratio control.

Secondary Solenoid Short to Ground Protection			
Watcher_ffs_sesl_2	injected faults	ises_out_A2L	
FAILED	fault sdi sesl BOB=SG	0.2	s38 3.33s s38
WAITING_CONDITION	(none)	0.2	s0 0s s0 0.01s
	KpKs_Offset=0.2	0.2	s23 0s s23 0s
	KpKs_Offset=0.2.0.4	0.2	s24 0s s24 0s
	ABS_Vehicle_speed_valid=1	0.2	s15 0s s15 0s
	ABS_fault=1	0.2	s16 0s s16 0s
	EMS_AccpedelError=1	0.2	s17 0s s17 0s
	EMS_CoolanTemperatureError=1	0.2	s18 0s s18 0s
	EMS_EngineSpeedError=1	0.2	s19 0s s19 0s
	EMS_ThrottleValvePositionError=1	0.2	s20 0s s20 0s
	EMS_Torque_Act_VD=1	0.2	s21 0s s21 0s
	EMS_Torque_DriverDesired_VD=1	0.2	s22 0s s22 0s
	fault ABSTO in=fault	0.2	s25 0s s25 0s
SUCCESS	fault sdi sesl BOB=SG	0.2	s38 5.02s s38
	fault sdi sesl BOB=SB	0.2	s40 40.2s s40
	fault sdi sesl BOB=UBATT	0.2	s38 20.4s s38

Figure 5: Fault reaction table for Solenoid Short

In the first column of the above table, we have the “state” of the “requirement watcher”.

Failed: the requirement has been “violated”.

Waiting_condition: requirement does not apply yet.

Success: the requirement has been tested and results are success.

The second column shows the injected faults.

At last for the state column, it lists some of the scenarios relevant to the state of the requirement watcher.

A scenario ([s38.3.33s](#) for example) listed in the table can be inspected in TestWeaver and replayed in Silver for detailed debugging by clicking on it.

After several test cycles, engineers removed most of the problems in the control software. For each cycle, there are thousands of test cases for different driving conditions, many usual but also many unusual conditions are included. This systematic analysis through Silver/TestWeaver is an important complement to other test methods, such as HiL tests, test benches and real vehicle tests.

5 Plant model for Hardware in the loop simulation

Since the plant model was developed from the beginning with the objective of satisfying real-time requirements, and was using a fixed-step solver, the adaptation to the HiL was rather straightforward.

The HiL system used in this project is an ETAS LabCar. The system was purchased several years ago and is an older version that does not support the FMI interface yet. To port the plant model on the HiL, we thus used the ETAS Labcar ‘export format’ of the SimulationX code export. SimulationX can generate the model c-code according to the ETAS LabCar module format. The SimulationX model can then be added to the Labcar project and compiled for its Linux real-time operating system. With newer ETAS Labcar versions, we could probably use directly the FMU generated for the SiL since it also contains the model c-code and can be recompiled for the ETAS Labcar target. The FMI standard would simplify interfacing of tools.

The model itself does not need to be changed from the SiL to the HiL, it was developed so that it can satisfy both cases, with a fixed-step solver. We only had to modify the code generation target from FMU to Labcar.

During this process, we found some minor bugs and compilation issues that were solved thanks to ESI/ITI SimulationX technical support.

6 Conclusion and future work

We presented an approach for an automated test of transmission control software based on SiL simulation (Silver) on standard PC and intelligent generation of test scenarios (TestWeaver) and how this testing method can be applied to the development of a CVT control software. For most of the state coverage, the intelligent automated test could generate thousands of scenarios, each scenario executing a 60 second drive maneuver. We also added specific test cases and test scripts to execute directly various standard maneuvers and fault injection.

To summarize, through the above approach of testing, we speed up the development cycle time for the CVT control software, and simulated most of the extreme driving conditions on a PC which is much safer than testing these conditions in real vehicles. After we solve most of the problems in the model/C code we found using SiL tests, then we move onto HiL tests and real vehicle tests using a more mature control software quality, reducing test time for the later, more expensive and less available platforms.

As future work, we plan to re-use TestWeaver “requirements watchers” and TestWeaver generated scenario database from SiL and port them on the ETAS HiL.

References

- [1] <http://www.fmi-standard.org/>
- [2] A.Abel, T.Blochwitz, A.Eichberger et al. Functional Mock-up Interface in Mechatronic gearshift simulation for commercial vehicles, 9th International Modelica Conference, 2012, Munich.
- [3] A.Junghanns, R.Serway, T.Liebezeit, M.Bonin. Building virtual ECUs quickly and economically. ATZ Elektronik, 03/2012, Volume7.
- [4] E.Chrisofakis, A.Junghanns. Simulation-based development of automotive control software with Modelica. Dresden: Modelica international conference, 2011.
- [5] M.Tatar, Schaich, Breitingner. Automated test of the AMG speedshift DCT control software. Berlin: 9th CTI Innovative Automotive Transmissions Symposium, 2010.
- [6] J.Mauss, M.Simons. Chip simulation of automotive ECUs. 9th symposium

Steuerungssysteme für automobile Antriebe,
2012, Berlin

- [7] N.Papakonstantinou, S.Klinger, M.Tatar. Test-driven development of DCT Control Software. 8th International CTI Symposium Innovative Automotive Transmissions, 2009, Berlin.
- [8] M.Neumann,M.Nass,M.Tatar. Absicherung von Steuerungssoftware für Hybridsysteme, Autoreg 2011, Friedrichshafen.
- [9] L.Belmon, J.Yan. Modeling and simulation of DCT gearshifting for real-time and high-fidelity analysis. SAE China-FISITA conference, F2012-C04-014, 2012
- [10] A.Abel, U.Schreiber, Valsania, Fornelli. Simulation based design of gearboxes for high-performance sports cars. Modena: 11th HTCES conference, 2005.
- [11] L.Belmon, Y.Geng, H.Qian, Virtual integration for hybrid powertrain development, using FMI and Modelica models, Lund, Modelica international conference, 2014