in: Getriebe in Fahrzeugen 2009, Friedrichshafen 30.06.-01.07.2009

Model-based Development of a Dual-Clutch Transmission using Rapid Prototyping and SiL

H. Brückmann, J. Strenkert, Dr. U. Keller, Daimler AG, Stuttgart;
B. Wiesner, Dr. A. Junghanns, QTronic GmbH, Berlin

Abstract

Since several years Mercedes-Benz integrates simulation and comprehensive tests with a high degree of automation in the development process of automatic transmissions. This process has been continuously improved and extended. Recently also first suppliers and engineering service providers have been integrated in this process. In this paper we present the current state of the development process and the corresponding tool chain. As an application example, we use a dual-clutch transmission (DCT) for passenger cars currently under development at Mercedes-Benz.

1. Introduction

The complexity of transmission systems is steadily increasing, due to growing market expectations regarding transmission efficiency, agility, and fun to drive. Mercedes-Benz addresses these demands with a growing number of vehicle models and configurations, and with additional functions of the transmission systems, many of them realized using TCU software. The corresponding development times are constantly shortened, while simultaneously keeping high quality standards.

System development, and in particular system evaluation and test with limited resources (time window and costs) is therefore a great challenge for the development teams. Conventional development and test processes rely mainly on (often model-based) development, hardware-in-the-loop (HiL) tests, and validation and calibration using physical prototypes. Growing complexity and limited resources impose an increasing pressure on both OEM and suppliers to further improve this process, to make it more reliable and more cost-effective.

According to these goals, a few years ago, Mercedes-Benz introduced a rapid integration of TCU functions based on software-in-the-loop simulation [1, 2] and comprehensive system validation based on automated test generation [6, 3, 4]. In this paper, we present the current state of this development process and the corresponding tool chain. As an application example, we use a dual-clutch transmission (DCT) for passenger cars currently under development at Mercedes-Benz.

The DCT development environment integrates the following components (partly shown in Fig. 1):

- A multi-domain simulation environment used to build a model of the physical world around the TCU, i.e. transmission components and car simulation. We use the modelling language Modelica [7], and Dymola as a modelling and code generation tool for the simulation model.
- MATLAB/Simulink is used for model-based development of the TCU 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 rapid prototyping environment is used to validate the DCT prototype and the TCU in a real vehicle and on HiL.
- Silver is the tool for virtual integration of modules based on SiL simulation. Silver imports both the transmission and car model generated by Dymola and the TCU software generated by TargetLink as DLLs and runs them in a co-simulation. 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, much like in a real car.
- CANape is used as measurement and calibration tool in both, the real car and the SiL environment.
- TestWeaver [3, 4] automatically generates, runs and assesses tens of thousands of different driving manoeuvres for comprehensive system test during TCU development.
- A HiL setup includes a script-based test automation solution
- Tools to perform static analysis on module and source code level and script-based tests on module level.

2. Goals and Motivation

Our development process makes use of rapid system development using SiL-integration and systematic test with a high degree of automation. The goals are to improve the development speed and costs while keeping and often improving the quality of the resulting products. The development process relies on the availability of a simulation model of the power train and a SiL integration platform to integrate the TCU control algorithms with the simulated car. Such virtual integration platforms have the following advantages:

- Early system validation: With early availability of executable system behavior, system behavior can be validated against specifications and requirements. This is the traditional „front-loading“ argument : engineers are able to test, debug and/or optimize their own modules in a system context and are not restricted to module tests.
- High availability: Virtual integration platforms and setups are relatively cheap, easily available and setups can be replicated exactly with little effort because they run on the engineers' laptops. This makes it possible to automatically generate, run and assess tens of thousands of simulation runs at virtually no additional cost, producing orders of magnitudes of coverage improvement.
- Reproducibility: Once a problem has been identified, it can be shared and reproduced efficiently for analyses. SiL simulations are deterministic. HiL simulations are not always deterministic.
- Handling: Virtual integration platforms allow for comfortable analyses and debugging, because the engineer can: (a) view and manipulate all internal variables (control algorithm and car simulation), (b) start/stop/step through time, including the use of breakpoints on specific signal properties, (c) attach source-code debuggers such as Visual Studio to step through the control code line by line, and (d) deterministically reproduce problems encountered earlier.
- Fidelity vs. speed: Without real-time constraints, if needed, power train models can be simulated in SiL with higher precision than possible on HiL platforms. Conversely, simple simulation models, like those used on HiL, run many times faster than real time on conventional PC hardware. This allows engineers to freely trade off the speed versus the fidelity of the model according to the project needs.
- Parallelization of work: Synchronisation points in the work flow can cause wait times. We have taken great care to avoid synchronisation of work results wherever possible. For example, each supplier, team, or even engineer can incrementally build the complete control software for the SiL platform to include modifications to „his“ local module(s) without waiting for a central built.
- IP protection: The SiL/Silver integration platform couples executable models in binary form. This allows each participant to share work results without sharing sources and proprietary know-how – an increasingly important feature with todays complex development structures with multiple suppliers and engineering-service providers.

3. Development Environment

In the following we explain some of the components in more detail.

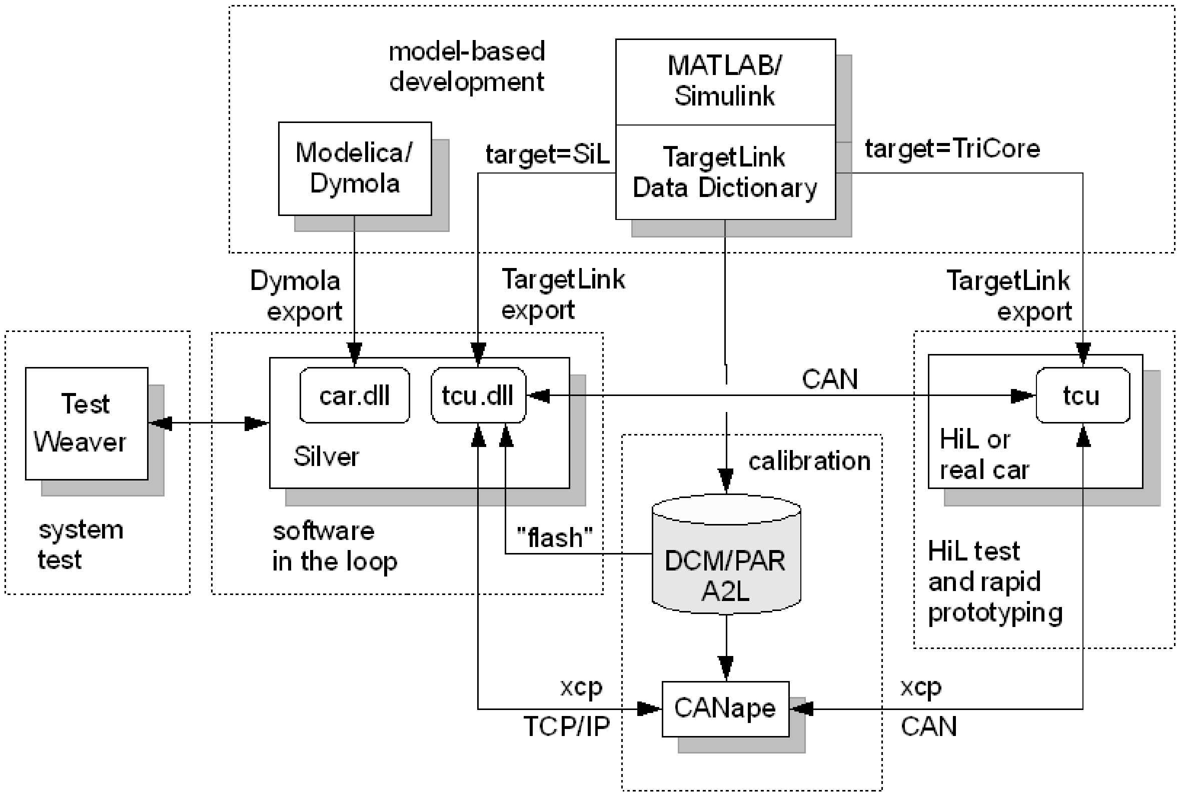

Figure 1: The Development Environment

3.1 Model-Based Development

For the DCT, a full model-based development process was used, based on MATLAB/Simulink as modeling environment and the dSPACE TargetLink block set with data dictionary (DD) and the TargetLink code generator. The DCT TCU integrates over 150 software modules, developed in parallel by several teams, suppliers and engineering-service providers. Compiled binary versions of the modules are shared using the PVCS versioning tool. This way every engineer has access to the results of all others, without sharing IP unnecessarily. Additionally, executable system models can be build from the stock of existing binaries from other contributers plus the modified modules each engineer improved. Such an incremental linking also speeds up the build process from about 2 hours (complete build) down to a few minutes. The build tool supports two execution targets: the real TCU with its TriCore processor, and the SiL/Silver environment that runs on Windows PC. In effect, every developer can build the latest version of the DCT TCU control software within less than three minutes and explore the resulting TCU behavior by driving a virtual car via SiL/Silver on its laptop. Note: The code running on the laptop is the final code with fix-point arithmetics.

3.2 Reusable Transmission and Car Model with Modelica

Modelica is a vendor-neutral language for modeling of physical systems. The Modelica language has been developed since 1997 by the non-profit Modelica Association [7]. Due to its multi-domain concepts, Modelica offers outstanding support for the modeling of mechatronic systems, such as automatic transmissions. High quality simulators for Modelica are offered by several tool vendors. For the DCT, Dymola was used to build a Modelica model of the DCT (without the TCU control software), the entire vehicle (including engine and its interactions with the DCT), driver and road. Dymola is also used to generate high quality simulation code from the model, to be executed in the SiL environment. In the Mercedes power train departments, Modelica transmission and car models have been used for TCU development for several years, e.g for the 7G-Tronic transmission family [1,2]. Over the years, a Mercedes internal Modelica library of reusable modeling components has been build up. In the case of the DCT, this has lead to a significant reduction of the modelling effort, because large parts of the car simulation model could be composed from the existing library.
Another point worth discussing is related to the model calibration. This is often a time consuming step. Model calibration requires measurement data from prototypes and adjusting model parameters until a given accuracy (e.g. less than 10% error) is reached. Fortunately, TCU software contains more and more adaptation algorithms. The control software adapts to the real car using parameter learning algorithms. For the DCT, this was exploited to lower the demands for model calibration. Automated adaptation in the SiL is used to adapt the TCU software to the simulated car. Of course, this requires to save the adapted parameter values (typically stored in EEPROM) to file and to reuse them (by "flashing" the parameters from file in the SiL/Silver simulation) at the beginning of each subsequent simulation run.

3.3 Virtual System Integration with SiL/Silver

A SiL platform is used for integrating all TCU software modules (C code with fix-point integer variables) with the transmission and car simulation. During DCT development, all developers of the TCU software have access to this co-simulation on their laptops. This allows the engineers to assess changes to a TCU-module with respect to the effects on the entire system within minutes.
As SiL integration platform, Silver [5] has been used. Silver offers a configurable GUI for interactive and script-based control and analysis of simulations. TCU functions and

simulation modules are bounded to Silver as compiled modules (wrapped as dynamic linked libraries, DLL) by a set of export tools. At running time, the compiled modules are cyclically executed by Silver in a single process using fixed macro step width. The macro step width depends on the sampling rate of the involved ECUs, 5 ms in the case of the DCT TCU. The modules exchange signal values at each macro step. Within a macro step, the transmission simulation module(s) may use much finer time scales for the numerical integration.

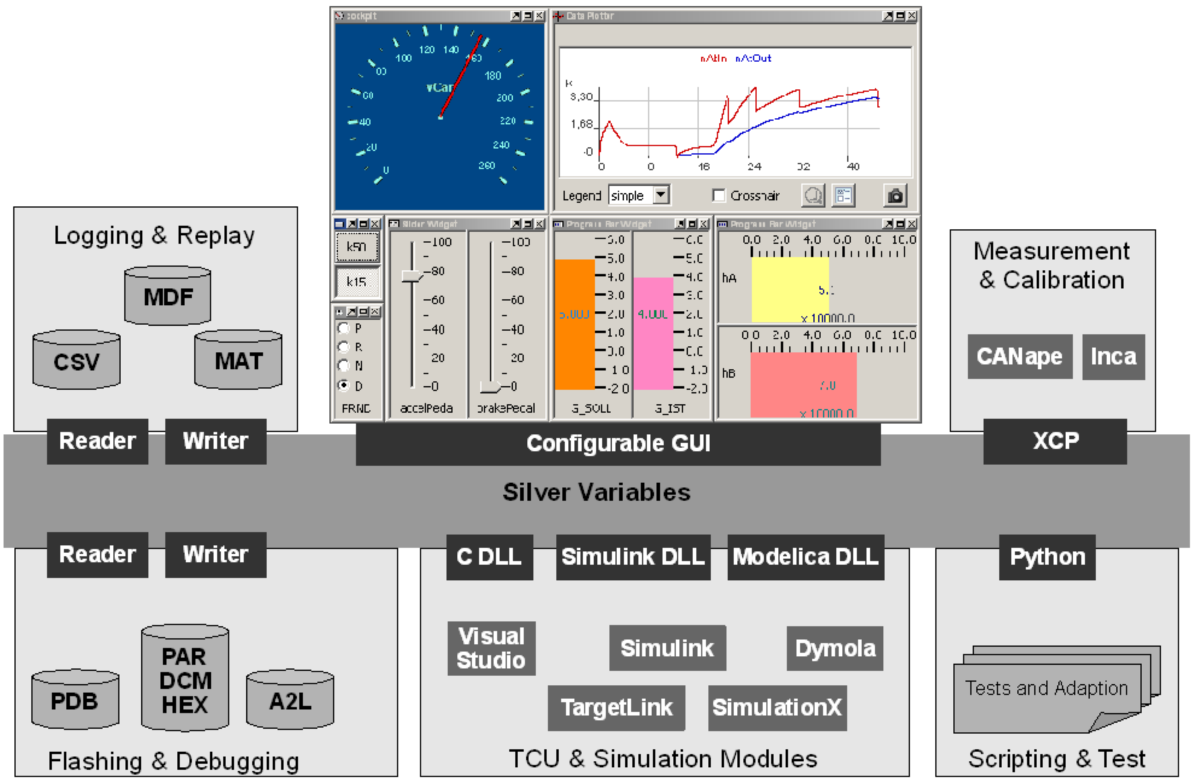

Figure 2: Virtual Integration with Silver

Silver also provides support for several utility services and for certain interfaces and standards used for automotive software development (see Figure 2):

- XCP support: Standard calibration tools (such as CANape or INCA) can connect to Silver using XCP on TCP/IP for measurement and calibration using the same setups that are used in the real car.
- Emulation of read or write of EEPROM memory: Used to save or read the results of calibration and model adaptation as described in 3.2.
- ASAP2 ECU description (A2L): Used by Silver to obtain address information for every variable (CHARACTERISTICs and MEASUREMENTs) of the control software. In effect, every static variable can be plotted or set during simulation.

- Calibration parameters: Silver can read and write calibration data in DCM, PAR or HEX format. Values can be written to files or „flashed“ from files into the simulation. This way, the scope of system properties that can be tested in SiL is considerably broadened. Often, problems can be caused by faulty or inappropriate calibration parameters.
- Python scripting: Simulation can be driven by Python scripts, e.g. for test automation or for automating the adaptation of the TCU software to the simulation model.
- Debugging support: Silver supports breakpoints at signal level and also connection with external source-code debuggers, for instance with Microsoft Visual Studio for step-by-step module execution.
- Reading and writing of measurements, for instance in MDF or CSV format.

The simulation can either be driven interactively using the GUI, or it can be driven by measurements, scripts or other tools, e.g. for test automation. The simulation results can be plotted, recorded and compared with reference signals. Silver is used by each development engineer for independent individual tests. The Silver simulation is also used by the comprehensive tests with TestWeaver.

3.4 Rapid Prototyping and HiL

Many development tasks traditionally performed using real hardware (prototypes, test rigs, or HiL) can be performed earlier, often with a high degree of parallelism, also on the SiL platform - leading to shorter development cycles because of all the advantages described in section 2. Of course, several system aspects can only be validated using real hardware.
The DCT development environment integrates rapid prototyping and HiL as follows: The TCU control software offers a special 'slave' mode. When in this mode, the TCU is remote controlled via CAN effectively bypassing the control logic and only using the power electronics and sensors. This way, the TCU can be controlled from a Windows PC using the same TCU software DLL that is used for co-simulation with Silver. For use in HiL, the TCU is connected to a PC via CAN and switched to slave mode, passing TCU control to the Windows PC. Likewise for rapid prototyping with a real car, a laptop running the TCU software DLL is connected to the car using a CAN card, and TCU control is passed to the laptop by switching the TCU to 'slave' mode.

3.5 Comprehensive Test with TestWeaver

Test automation is traditionally implemented by running hand-written test scripts on a MiL/SiL or HiL platform. Unfortunately, this approach does not scale very well with increasing system complexity. In particular, automatic transmissions need to be tested in a huge number of differing situations: differing shifts, differing engine torque and speed domains, road slope, temperature, driver commands, and differing car configurations. For that reason, a few years ago, Mercedes-Benz started to complement traditional scripts-based testing with a new method for system validation based on automated test generation [6].

For automated test generation, TestWeaver [3, 4] is used. TestWeaver does not require pre-fabricated test scripts. Instead, TestWeaver generates, runs and evaluates thousands of tests automatically. The tests are not generated randomly, but in a reactive, informed way, trying to maximize the coverage of the states reached by the system and to actively worsen scenarios that are already sub-optimal until system behavior violates a requirement. The following kinds of problems can be found:

- low-level coding errors, e.g. division by zero, access violations, integer overflow, pre-defined range exceeded, index out of bounds,
- algorithmic errors, e.g. state estimation significantly different from values in the model for long periods of time, oscillating and non-converging controllers, etc.
- system-level problems, e.g. clutch overheating, over-speed of engine and transmission components, inadequate fault reaction, engine stalled, etc.
- quality indicators, e.g. worst-case and average measurements for shift durations, power losses, and other shift quality indicators.

To test the TCU control software of the DCT, TestWeaver is connected to the SiL/Silver platform. TestWeaver can control certain system inputs (acceleration pedal, brake pedal, PRND lever, road profile etc.) and can observe certain outputs (states of the TCU control software, key variables of the car simulation) as shown in Fig 3. All problems found by TestWeaver can be recreated in simulation with Silver for detailed analysis and debugging.

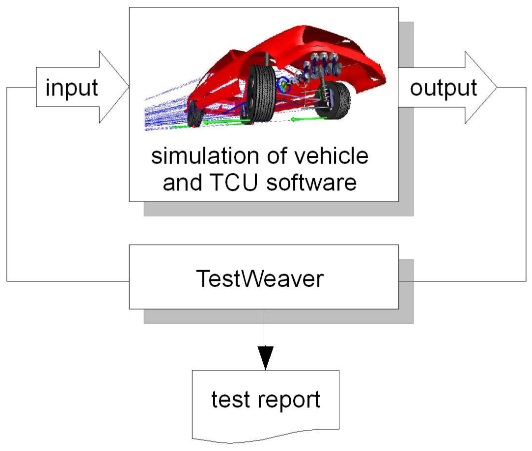

Figure 3: System Test with TestWeaver

5. Application to the DCT

The build process for the SiL/Silver target is a modified version of the build process for the TriCore processor. Because compiled module versions are stored and shared in the PVCS version management system, an incremental build after only a few modules have been modified takes only a few minutes. As opposed, a complete build takes about two hours.

Also the TCU modules contributed by external suppliers are integrated in the SiL/Silver target. Thus, all development engineers have a confortable and rapid access to the SiL/Silver simulation of the complete system. Thus they can test their own modules and the interaction with the rest of the system in parallel and independently of each other. Suppliers and engineering service providers that co-operate in the project also start to use the SiL/Silver platform for integration and tests. Several potential problems are directly shown by Silver, for instance: mismatching signal names, violation of the min-max bounds from A2L, unexpected system behavior visible by plotting signals, etc.

In addition, extensive tests with TestWeaver are run each week. During a typical test, for instance over the weekend, over 2000 test scenarios are automatically generated, classified and assessed. As the project is still in a relatively early phase, we concentrate more on software errors and algorithmic errors. But also more and more quality criteria are added to the testing goals. Many of these criteria can be reused from the TestWeaver configuration for the 7G-Tronic transmission. At the end of a test several coverage and overview reports are available for showing what has been tested, and what problems have been found. The problems found are then assigned to the responsible developers. For the detailed problem analysis and debugging the test scenarios can be replayed with Silver, where additional signals can be plotted, breakpoints can be set, etc.

6. Summary and Outlook

We presented the tool chain and process currently used at Mercedes-Benz to develop the control software for a dual-clutch transmission. The work process is centered around a virtual integration (SiL) platform, here Silver [5]. This enables us to perform significant validation, test and analysis steps earlier than in traditional test development setups and that on highly available standard PCs available for each engineer participating in the project. Organising processes around sharing object files removed significant synchronisation points in the development process and allows engineers to assess their improved modules in a system context. When problems are found, the SiL platform provides a comfortable analysis and debugging environment. The investment in building and maintaining the SiL platform proved to be well justified by speed-ups due to shorter development cycles. The presented approach to system validation based on automated test generation with TestWeaver [3, 4] proved to be particularly useful. Over the entire project, the number of different test cases used to validate the system has been increased by 2 or 3 orders of magnitude, without increasing the work load for test engineers. On the contrary, we estimate that the effort spent for test setup and maintenance is now only a fraction of the effort required for setting up and maintaining the script-based approach [1].

The current economic trends continue to put a high pressure on OEM and suppliers to further improve their development process, to make it more reliable and cost effective. The AUTOSAR standardization of the software architectures [10] will hopefully contribute in this direction. We expect and support also more steps toward a standardisation of the development tools and processes, including the SiL virtual integration platforms – an effort also followed in the Modelisar project [9]. OEM and suppliers will have to work more together for achieving these goals.

The SiL virtual integration complements in an excellent way the HiL and the physical prototype integration and test efforts. The SiL technology has matured to a point where development can be supported even up to pre-calibrating the power train. This allows significant time savings in a flexible and robust development process, across teams and companies.

References

[1] Karlheinz Braun, Jochen Strenkert: Simulation und Test eines Drehmomenten-koordinators für das AUTOTRONIC-Getriebe von Mercedes-Benz. In: C. Gühmann: Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik II, Expert Verlag, 2008.

[2] Andreas Stiegelmeyr, Uwe Keller, Dieter Hans: Simulationsmodell zur Analyse der Getriebedynamik. In: C. Gühmann: Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik II, Expert Verlag, 2008.

[3] A. Junghanns, J. Mauss, M. Tatar: Testautomatisierung nach dem Schachspielerprinzip. AutoTest 2008, Test von Hard- und Software in der Automobilentwicklung, Stuttgart, October 2008.

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

[5] Silver 1.1 Product Information, QTronic GmbH, www.qtronic.de/doc/Silver_en.pdf

[6] Zug um Zug zum perfekten Zusammenspiel – Mechatronische Bauteile virtuell getestet. In „DaimlerChrysler Hightech Report“ 1 / 2006, pp. 54-57.

[7] Modelica Association, see www.modelica.org

[8] EuroSysLib project: http://www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf

[9] Modelisar project: http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf

[10] AUTOSAR partnership: www.**autosar**.org