Virtual ECUs for Developing Automotive Transmission Software

Dr. Thomas Liebezeit[1], Jakob Bräuer[1], Roland Serway[1], Dr. Andreas Junghanns[2]
[1]IAV GmbH, Carnotstraße 1, 10587 Berlin
[2]QTronic GmbH, Alt-Moabit 92, 10559 Berlin
thomas.liebezeit@iav.de, andreas.junghanns@qtronic.de

1 Introduction

The development of controller software for transmissions for production use is characterized by high demands on algorithm and code quality. Extensive validation measures are required to achieve these demands. In addition to independent validation by a test team, developers are required to test functional behavior of their modules in the system context.

For a long time now, IAV has used dSPACE-HiL systems and test vehicles for this purpose, as complex open-loop and closed-loop control features can only be verified with the help of a controlled system or model thereof with regard to system characteristics, going beyond the module test capabilities. However, both approaches only offer limited possibilities for troubleshooting and analysis, especially because real code debugging is impossible in such environments.

For this reason, a software-in-the-loop system has been commissioned using QTronic's Silver, giving every developer the possibility for source-code debugging on their development computer in the system context (with a plant model). To this end, a virtual control unit in SiL was created, complementing the real ECU in SiL testing/validation.

2 Boundary Conditions

IAV is developing function software for transmissions on behalf of customers for dual-clutch transmissions. The development is based on an established, customized development process, subject to constant efforts towards quality and efficiency improvements. This process identified debugging of controller software as a major area for potential improvements. An evaluation of possible solutions swiftly revealed the potential offered by software-in-the-loop (SiL) for this particular purpose.

A number of conditions had to be fulfilled before finally introducing another tool into the development process:

- **Possibility of line-by-line debugging in C-code**
  Full-featured debugging must be supported, as this was the main purpose of the improvement efforts. It is presumed that the normal application development features also apply here, such as fixed and conditional breakpoints, reading and changing of run-time variables and a convenient GUI.
- **Deployable at any time**
  It should be possible for every function developer to use the SiL system with his current controller code.
- **No code changes in the function software just for the SiL**
  The existing code should be kept unchanged. Accordingly, special pre-compiler switches etc. for the SiL are not permitted.

- **Reuse where possible**
  Existing work results should be reused to prevent unnecessary extra work.
- **Consistency**
  It should be possible to re-use the data files consistently in X-in-the-loop and in the vehicle. For example, it must be possible to integrate the SiL in the existing build process as an additional option and to incorporate it in version management.
- **Minimum possible extra workload for SiL**
  As a general rule, the necessary additional workload for the SiL system should be kept to the necessary minimum. Improving the debugging possibilities may not result in increased workload for configuration, familiarization or adaptation.
- **Full SiL control**
  It should be possible for IAV to provide all necessary know-how for setting up and updating the SiL.

This results in the following points relevant to concrete SiL implementation:

- Support of standard formats and techniques such as:
  - Parameter and mesurement signal definition (ASAP2)
  - Parameter files (PAR)
  - CAN bus definition and use (DBC)
  - Recording and replaying of measured signals (MDF)
- Support for C-code and Simulink models (reusing plant models from the HiL domain)
- Simple tool handling
- Flexible licenses (dongle or network license) according to current usage requirements
- Easy configuration for adapting to specific needs

3 Silver

Silver is a product of QTronic GmbH and has been used for some time now at IAV in various transmission development projects. Silver provides a simulation environment for software-in-the-loop simulations. It supports many automotive standards (A2L, DCM, PAR, XCP, HEX, MDF, DAT, ...).

However, up to now it has been used in IAV transmission projects with another focus. For example, Silver is used for virtual integration. This entails integrating the software functions developed by IAV in the customer code that has not been revealed to IAV and then checking for its correct behavior in the overall system context.

The positive experience gained with Silver so far, together with an additional evaluation in terms of the specific criteria defined above led to the selection of this particular SiL implementation.

4 SiL setup

To achieve the abbility for debugging function software, a SiL system must be created. It consists of four components (see Fig. 1):

1. Transmission function software
2. Virtual electronic control unit (ECU)
3. Environment model (plant model plus environment)
4. Silver with silver configuration (e.g. GUI, scripts)

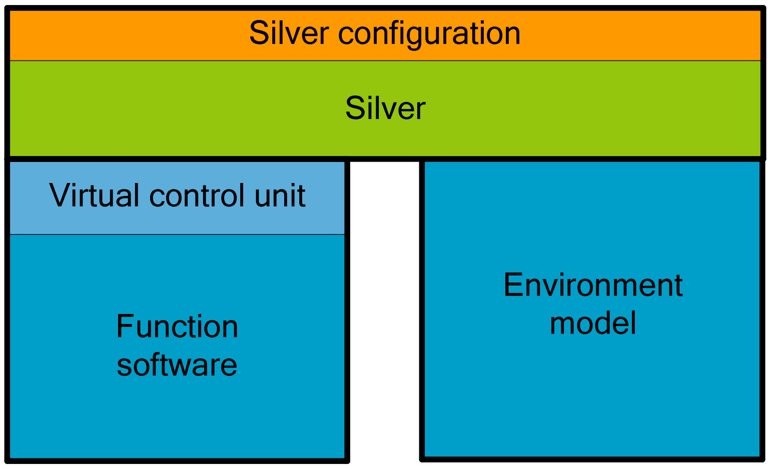


Figure 1: Components of the software-in-the-loop system

The function software and the virtual control unit constitute the ECU side of simulation, while the environment model simulates the relevant vehicle behavior on the other side. Silver is responsible for linking the two sides, each in the form of a DLL file.

The DLL files are generated with the Microsoft Visual Studio C compiler. On the ECU side, after linking, an A2L file is updated with the current memory addresses. This makes it possible to use A2L variables in the SiL system and to flash parameter files. The build call has been integrated in the programming environment (IDE) for the algorithm developers. It therefore only takes one click to generate a DLL on the basis of the updated function software.

The environment model is compiled using Real-Time Workshop with a Silver target. The result then consists of one DLL for every parametrization of the controlled system (creating a virtual fleet of vehicle variants) because "flashing" of vehicle parameters is currently not possible. The DLLs are available to the algorithm developers via the version management.

Similarly, a project-specific standard Silver configuration is also provided.

The following section takes a closer look at the individual components and the adjustments necessary for creating the SiL system.

4.1 Function Software

The function software is mostly hand-written C-code. Parts of the higher levels of the software are generated with Targetlink. The interface to the BIOS of the control unit is provided by the BIOS API.

The code of the function software is available to every developer almost completely in a daily updated version. Smaller parts of the function software are supplied as pre-compiled LIB. These parts have to be linked with the compiled C-code for the SiL system.

No relevant changes to the C-code were necessary for compiling the SiL system.

4.2 Virtual Control Unit

The virtual control unit simulates the relevant aspects of the real control unit. For example, it connects the function requests of the function software made to the BIOS with the corresponding input and output signals of the environment model so that the function software can for example read sensor values and set actuators.

The parts of the ECU relevant for the SiL system are:

- **Timing**
  Timing typically encompasses definition of when to call each task and with which priority.
- **Input/output interfaces**
  In order to simulate the inputs and outputs of a real ECU, the respective signals and CAN messages have to be transmitted to the environment model via a software interface.
- **BIOS functionality**
  It was possible to adopt parts of the BIOS, such as valve polynomials and application systems, but as the BIOS code is not available, these BIOS functions have been simulated by the SiL system.
- **Non-volatile memory**
  The control unit ROM is simulated in the SiL system by means of an external text file. This makes it possible to save and load earlier ROM statuses together with ROM manipulation in a text editor.

With the Silver Basis Software (SBS), Silver offers a simple template for creating the parts that are typical for a control unit. Input/output signals and task timing, for example, can be defined with just a few lines of C-code. Corresponding helper functions for sending and receiving CAN signals are also available.

Already when configuring the virtual control unit, consideration has to be given to weighing up which parts of the function software should remain active in the SiL and can therefore be tested and debugged. Two examples should illustrate the need for deliberate considerations and appraisals in the design of the individual interfaces. Normally, SiL systems frequently exchange individual physical signals between function software and environment model. But here this would bypass the lower levels in the function software so that it would then no longer be possible to test these with the SiL system.

One example in this context consists in the processing of CAN messages. Normally, the individual values from the environment model would be linked directly to corresponding variables in the function software. The entire CAN handler would not be involved, thus bypassing an essential part of the software.
Instead, the environment model generates and receives complete CAN messages (including CRC) and transfers these to the virtual ECU via a virtual CAN bus. In turn, this makes the raw byte sequences available to the function software. Consequently, the CAN handler is active in the SiL simulation.

Simulation of all functions is not always appropriate, as shown by the second example. The function software computes speed signals from the raw signals of the speed sensors. The environment model contains these speeds as numerical signals. Simulating the sensor raw signal in this case is too complicated and too error-prone to justify the higher test coverage. Accordingly, processing of the raw signals has been bypassed: the values are coupled directly into the function software.

4.3 Environment Model

In SiL, the environment model performs the same task as in HiL: it supplies the control unit / function software with sufficiently accurate vehicle behavior in terms of longitudinal dynamic behavior, CAN rest bus etc. On the basis of the model, parametrization is then used to simulate various vehicles with different engines.

The behavior and accuracy of the existing HiL model are adequately assured thanks to the frequent use of the HiL by the function developers and testers. This makes it highly beneficial to keep using it for the SiL.

Minor model adjustments are necessary for use with the SiL:

- The dSPACE blocks have to be replaced with Silver library blocks
  The HiL model contains dSPACE RTI blocks for triggering the HiL hardware (digital and analog IOs, CAN controllers etc.). These blocks are replaced by compatible blocks from the Silver block library. Silver offers blocks for input and output signals, together with blocks for reading and writing CAN messages.
- Restructuring the timing
  The HiL model contains various hardware timers that trigger the subsystems. The SiL system has to have a small shared time constant from which all other timers are derived as integral multiples. The SiL model must run with the smallest time constant and with a fixed step solver.

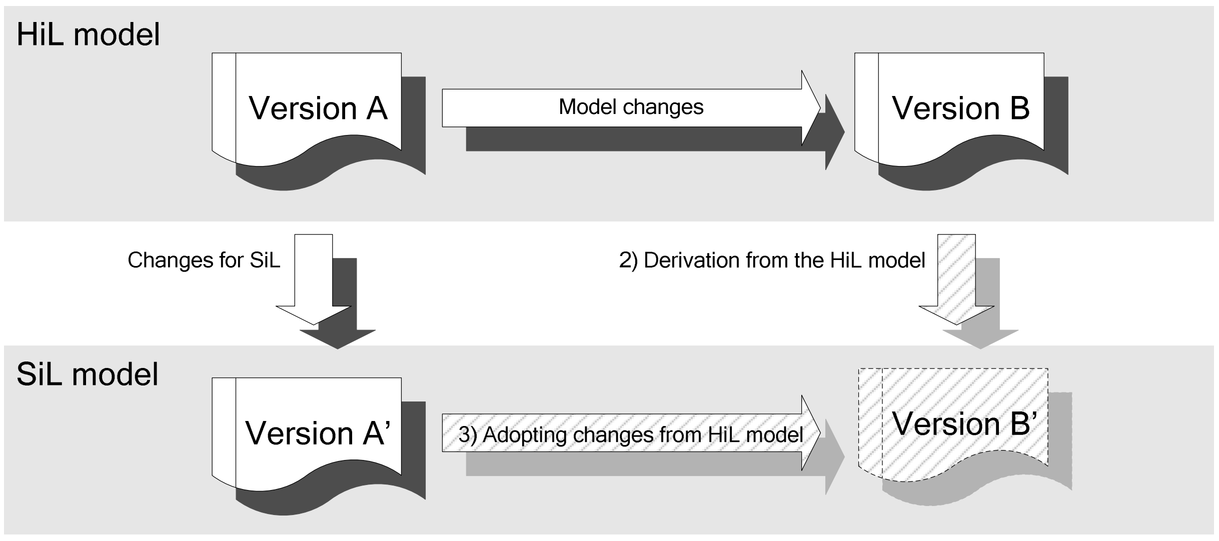


Figure 2: Strategies for adopting model adaptations

Model updating is a central aspect in using the HiL model for the SiL system, i.e. incorporating HiL model changes in the SiL model. Basically there are three different strategies here:

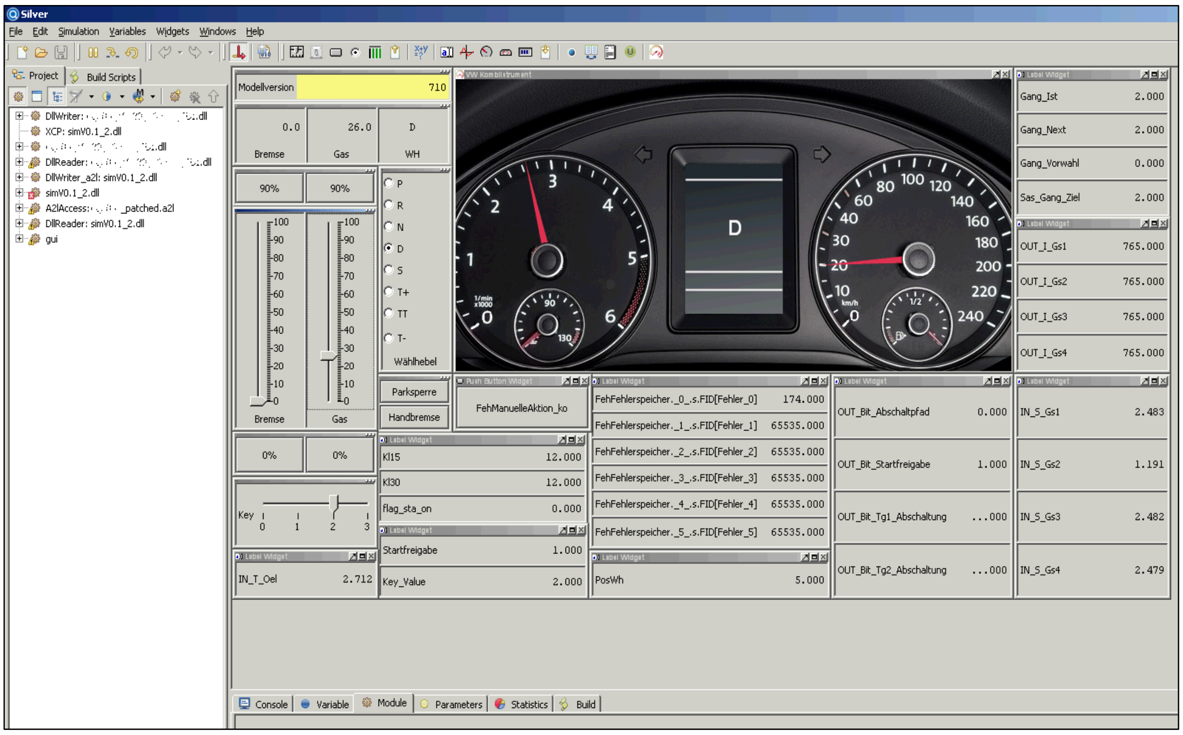
1. Shared model core
   In this case, the changes are adopted automatically. Unfortunately, this is not possible here as IAV does not proceed with further development of the HiL model.
2. Derivation from the HiL model
   A new SiL model is derived after every change to the HiL model. While entailing a great workload, this approach ensures that all changes are adopted.
3. Adopting changes from the HiL model
   The changes from the HiL model are adopted accordingly in the SiL model. This does not automatically ensure that all changes are adopted.

Strategy 3 has proven the most effective as most changes were only minimal. Here it makes sense to use a diff-/merge tool for Simulink, such as Medini Unite, permitting specific identification and adoption of the changes in a graphic view of the models.

4.4 Silver Configuration

The Silver configuration puts the SiL system together, ensures the system is initialized and defines the user interface (see Fig. 3).

Silver links the virtual control unit with the environment model. Both sides run parallel and the input and output variables are synchronized with each other every 10 ms (period of the fastest task). Linking takes place automatically using identical signal names.



```
XCP: simV0.1_2.dll
DllWriter_a2l: simV0.1_2.dll
simV0.1_2.dll
DllReader: simV0.1_2.dll
gui

Modellversion   710
Bremse 0.0   Gas 26.0   WH D
Bremse 90%   Gas 90%
Wählhebel: D
Bremse 0%   Gas 0%
Key 2

Gang_Ist       2.000
Gang_Next      2.000
Gang_Vorwahl   0.000
Sas_Gang_Ziel  2.000
OUT_I_Gs1    765.000
OUT_I_Gs2    765.000
OUT_I_Gs3    765.000
OUT_I_Gs4    765.000

FehManuelleAktion_ko
Kl15                  12.000
Kl30                  12.000
flag_sta_on            0.000
Startfreigabe          1.000
Key_Value              2.000
IN_T_Oel               2.712

FehFehlerspeicher._0_.s.FID[Fehler_0]     174.000
FehFehlerspeicher._1_.s.FID[Fehler_1]   65535.000
FehFehlerspeicher._2_.s.FID[Fehler_2]   65535.000
FehFehlerspeicher._3_.s.FID[Fehler_3]   65535.000
FehFehlerspeicher._4_.s.FID[Fehler_4]   65535.000
FehFehlerspeicher._5_.s.FID[Fehler_5]   65535.000
PosWh                                       5.000

OUT_Bit_Abschaltpfad       0.000
OUT_Bit_Startfreigabe      1.000
OUT_Bit_Tg1_Abschaltung    ...000
OUT_Bit_Tg2_Abschaltung    ...000

IN_S_Gs1   2.483
IN_S_Gs2   1.191
IN_S_Gs3   2.482
IN_S_Gs4   2.479
```

Figure 3: Silver GUI

The current configuration contains the following parts:

- Flashing the parameters from PAR file to the software at the start of simulation
- Graphical user interface (GUI) with the main elements
  - Selector lever, brake pedal, accelerator pedal
  - Display of elementary information (vehicle speed, current gear, fault memory contents)
- Direct export and setting of values from/in the function software and from the environment model
- Via XCP, measurement tools (such as CANape or INCA) can be used with user-specific measurement screens to read control unit variables and to read or write parameters – just as in the real vehicle

The SiL system runs on the developer laptop in real time without fully utilizing the computer. The exact performance is influenced by the number of control unit parameters and variables to be read or written, so that user-defined restrictions are appropriate. Only a small number of signals is needed for the general display in the GUI.

5 Debugging

The main aim in introducing the SiL system was to simplify debugging. The SiL setup presented above is an efficient solution for providing convenient debugging.

With the normal build process, firstly the function developer builds the current function software for the SiL (including the virtual control unit) and retrieves the current versions of the environment model and the Silver configuration from the version management tool. After opening the Silver configuration, the Microsoft Visual Studio Express Edition has to be started as debug IDE. This is where the function developer sets breakpoints at the source-code lines requiring analysis. The SiL system can be started once Visual Studio is connected with the Silver process. Using the GUI, a relevant driving situation can now be started, leading to the code with the breakpoint. On reaching the breakpoint, the whole system stops, i.e. the function software, virtual control unit and environment model are stopped.

Visual Studio then permits debugging and step-by-step execution of the code. At the same time, all common features are available, such as read and write access to any variable. This procedure permits swift, direct debugging without additional definition of measured variables or similar.

Debugging can also be used in the following scenarios:

- **Situations that are difficult to produce**
  Manual debugging (e.g. changing internal status variables) generates a specific driving situation for the function software. Correct processing of the corresponding code parts can be tracked in debugging.
  This procedure is suitable particularly for checking rarely achieved boundary conditions.
- **Timing errors**
  It is easy to specifically adjust and analyze timing errors by means of debugging.
- **Re-simulation of vehicle measurements**
  Recorded vehicle measurements can be used as defaults for the driver. However, this can lead to differences in run-time behavior which possibly conflict with replication of faults found in the vehicle. The differences result from replicating the vehicle behavior for simulation and are defined by the quality of the environment model.
- **Fault simulation**
  Specific activation of hardware faults in the environment model results in easy and safe testing of fault detection and reaction. In the easiest case, sensor or actuator values are overwritten such that it is not even necessary to change the environment model.
- **"Living code"**
  Debugging helps function developers to familiarize themselves more quickly with the code as they can actively follow it step by step during the process. An approach like this does not replace documentation of the code, but improves an understanding of how the functionalities are currently being implemented.

6 Experience and conclusion

The software-in-the-loop system presented here has been used successfully for months by IAV transmission function developers.
The experience they have gained is summarized briefly below.

- **Time required for implementation**
  Two IAV developers were responsible for initially setting up the SiL system and incorporating it in the existing development process, taking about six man-months. The experience gained in this process means that future projects can be set up more quickly.
- **Time required for version updating**
  The time required for updating depends on the changes to the structure of the function software and the environment model. Both parts need minimum updates and

adaptations that are to be completed promptly. Fast reaction is important to warrant high availability of the SiL for function developers.
Consequent reuse of existing work and integration in the existing development and build process keeps the additional workload to a minimum.

- **Added value from debugging**
  For the function developers, debugging puts a new quality into the analysis of the run-time behavior of function software. It can be seen that the analysis times are drastically reduced by faster change-analysis-change cycles.
- **User acceptance**
  Debugging the function software in the SiL system is well on the way to asserting itself like the HiL systems years ago. At the moment it constitutes a supplement to established work methods.

7 Outlook

An operational and well established software-in-the-loop system offers additional possibilities over and beyond debugging.
These include:

- **Measuring resources**
  Silver offers further analysis options. It is possible, for example, to measure the usage of resources (time and stack) for code sections (e.g. tasks) in order to identify optimization potential.
- **Reset analysis**
  The specific triggering of control unit resets (in the virtual control unit) also makes it possible to examine the robustness of the control unit software.
- **SiL application**
  Calibration of the function software is conceivable when the quality of the environment model is sufficient. Especially where safety-critical parameters are concerned, it is possible to test various calibration values without any danger.
  Silver supports XCP for the connection of calibration tools such as CANape and INCA.
- **SiL testing**
  Unlike HiL, the SiL is not hardware-compatible with the vehicle environment (other compiler, simulation of control unit). Test results are not directly transferable one-to-one. Even though, the SiL system can still be put to effective use in developing HiL test cases or many other functional aspects.
  Silver is already connected to test tools such as TestWearer and TPT and can also be connected quite easily as execution platform for other test tools, such as ECU-Test or EXAM. It is also possible to run Python scripts in Silver allowing repeatable, reactive tests to be executed.