Chip simulation of automotive ECUs

Jakob Mauss, Matthias Simons

Abstract

Modern ECUs contain ten thousands of engine parameters that need to be tuned. Calibration of all these parameters is time consuming and complex. Simulation on a PC could help to automate and speed up the calibration process, in particular if simulation runs much faster (e. g. 20 times) than real-time. However, engine calibration is typically performed by an OEM, while the ECU code is owned by the supplier of the ECU. Therefore, the OEM is typically unable to set up a ECU simulation based on the original C code of the ECU. Instead, to set up a simulation, time consuming and error prone reverse engineering is needed to develop an 'equivalent model' of the ECU function of interest. To improve this situation, we have integrated a chip simulator into the virtual ECU tool Silver. This is used to simulate hex files compiled for TriCore targets directly on PC. Simulation requires

1. a hex file that contains program code and parameters of the simulated functions
2. start addresses of the functions to be simulated
3. an ASAP2/a2l file that defines the conversion rules for the involved inputs, outputs, and characteristics, as well as corresponding addresses

The start addresses of functions can e. g. be extracted from a map file generated together with the hex file. Silver uses the a2l file to automatically convert scaled integer values to physical values and vice versa during simulation. A TriCore simulation can also be exported as SFunction (mexw32 file) for use in MATLAB/Simulink. On a standard PC, hex simulation runs with about 40 MIPS. If only simulating selected functions of an ECU, this is fast enough to run a simulation much faster than real-time. In this paper, we also report how such simulations are used today to support the development of gasoline engines at Daimler.

1. Introduction: Virtual ECUs in the development process

Simulation has great potential to improve the development process for ECUs. Simulation helps to move development tasks to PC, where they often can be performed faster, cheaper or better in some respect. Examples that illustrate this point:

- on a PC, an engineer can easily 'freeze time', i. e. stop simulation and inspect the call stack and all variables of a virtual (i. e. simulated) ECU without band width limitation and repeat a simulation deterministically as often as needed. In contrast, real ECU as used in HiL settings or test rigs must run in real time. Stopping and stepping is impossible or requires considerable extra effort, e.g. based on the JTAG debug interface. Exact reproduction of experiments is difficult or impossible on a HiL or test rig, due to non-deterministic effects.

- on a PC, a calibration tool like INCA (ETAS) or CANape (Vector) can be connected to a virtual ECU via XCP to measure into a running simulation and to tune characteristics online. This way, many parameters of a ECU can be tuned using a relatively cheap and highly available PC platform, without blocking rare and more expensive resources like real prototypes and test rigs.
- A virtual ECU might run on PC 20 times faster than real time. When used in combination with test automation, a simple PC gives then 20 time higher test throughput than a much more expensive HiL test system.
- On a PC, a development engineer can rebuild the entire ECU within 5 minutes after modification of a module, thanks to incremental build, and test drive the result in a simulated environment. This helps to detect problems early on the developer's PC, and decreases the number of problems that show up late, when integrating all modules. As experience shows, such early checks speed up development.

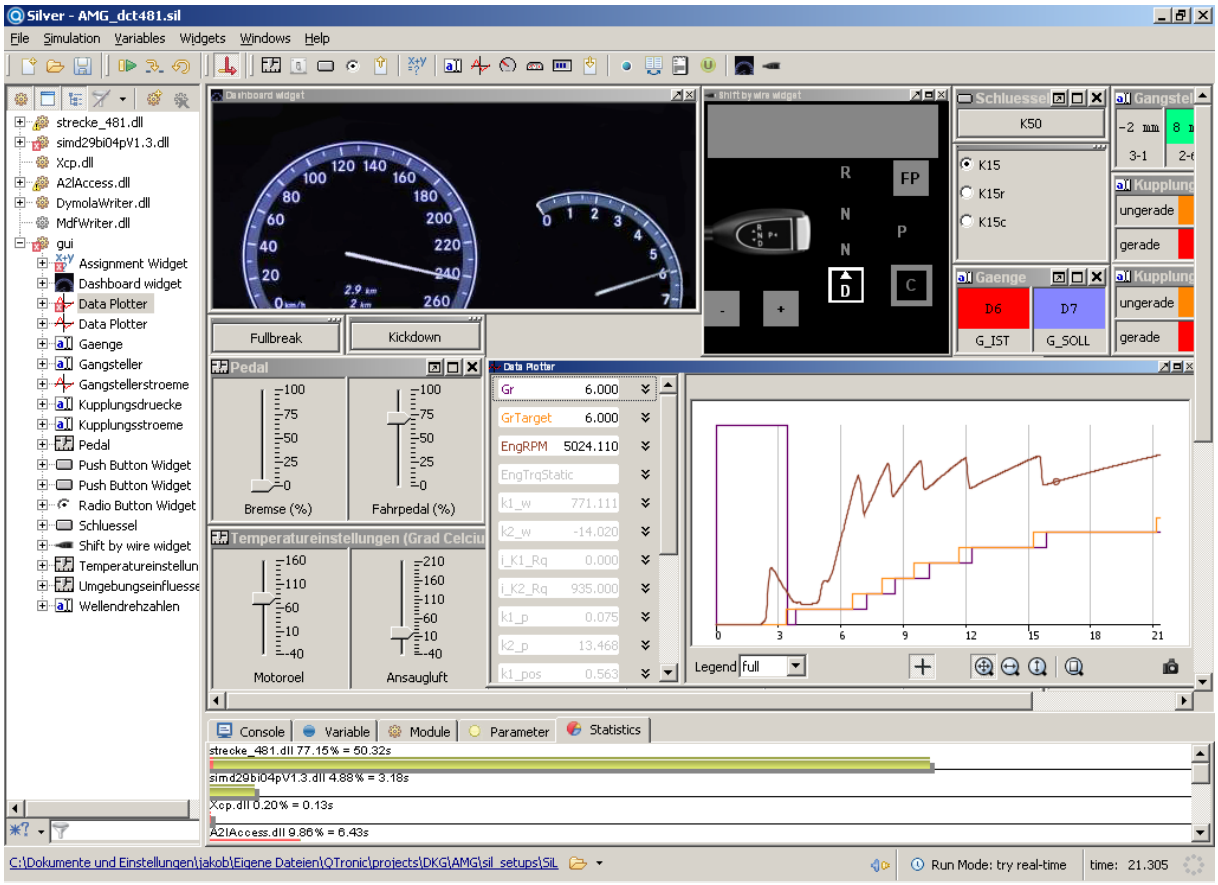

Fig 1: Transmission control unit running in Silver (taken from [6])

To exploit these and other benefits when developing an ECU, the ECU must first be ported to PC. This is typically done based on the C code of the ECU, which is either hand coded, or generated by tools such as Ascet (ETAS), TargetLink (dSPACE) or Embedded Coder (MathWorks). For example, QTronic's virtual ECU tool Silver [1] provides a framework to

- compile given ECU tasks for Windows PC,
- emulate the underlying RTOS and other services (CAN, XCP),
- run the resulting virtual ECU closed-loop with a simulated vehicle.

Typical applications are [2, 6], where a virtual ECU is used to develop the controller for an automatic transmission. For closed-loop simulation (shown in Fig. 1), vehicle

models can be imported from many simulation tools into Silver, including MATLAB/Simulink, Dymola, SimulationX and MapleSim, e.g. through the FMI format for model exchange [4].

However, sometimes C code is not available for implementing a virtual ECU. There are two main sources for such a situation:

- *Protection of intellectual property*: All or major parts of the ECU have been developed by a supplier and the OEM interested in building a virtual ECU (e.g. to support calibration, a task typically performed by an OEM) has therefore no access to the C code.
- *Target-specific C code*: C code is available but the C code uses pragmas and other target or compiler specific constructs, which prevents compilation for other targets, such as the Windows x86 platform.

To deal with such situations, we have recently integrated a chip simulator into the virtual ECU tool Silver. This way, a virtual ECU can be build based on a hex file compiled for the target processor of the ECU. No access to C code is needed in this case. Instead of compiling C code for the Windows x86 platform, the chip simulator takes the binary compiled for the target processor and simulates the execution of the instructions by the target processor on Windows PC.

The remaining paper is structured as follows: in section 2, we describe how to use this feature to build and run a virtual ECU on PC. In section 3, we report how this is currently used by Daimler to support the development of gasoline engines.

2. Chip simulation for TriCore targets

Many automotive controllers are based on processors of Infineon's TriCore family, in particular in the power train domain. Examples are engine controllers of the MED and EDC family by Bosch and transmission controllers by Continental. Since the initial release of TriCore AUDO (AUtomotive UnifieD-ProcessOr) in 1999, Infineon has released four updates of the TriCore architecture, named AUDO NG (e.g. TC1796), AUDO Future (e.g. TC1797), AUDO MAX (e.g. TC1798), and AURIS. All members of the TriCore family are based on the same instruction set. Individual chips differ in memory maps, kind of memory, on-chip devices, such as CAN controllers, and interfaces to external devices. This section describes the support for TriCore chip simulation as provided by Silver 2.5.

2.1 Turning a hex file into a virtual ECU

The software of an ECU consists of a real-time operating system (RTOS) that runs functions (tasks) at specified times, either initially, periodically or at certain events, such as angle positions reached by the crankshaft. Three kinds of tasks can be distinguished

1. tasks that generate signals, e.g. by reading sensors or CAN messages
2. tasks that compute output signals from input signals
3. tasks that use signals to command actuators or to create CAN messages

The tasks of categories 1 and 3 typically depend on details of the particular chip, and on the ECU hardware. In contrast, tasks of category 2 are fairly independent from such hardware-specific details. To simulate ECU code, it is therefore convenient to run only tasks of category 2. The required inputs for these tasks can either be taken from measurement files (open-loop simulation), or they are computed online by some plant model (closed-loop simulation), bypassing the tasks of category 1. Likewise, the outputs of category 2 tasks can be directly compared to measurements (open loop) or fed into the plant model (closed loop), bypassing the category 3 tasks. The signal interface between categories 1-2 and 2-3 is typically well documented and available, e.g. from the CAN specification (DBC file) of the ECU.

This modelling strategy has a very good cost-benefit ratio. In order to simulate also the tasks of categories 1 and 3, one has to model hundreds or peripheral and chip specific registers, and to build state-machine models for low-level peripherals, such as CAN controllers. Technically, this is possible, e. g. with SystemC [5], but hardly justified by the added value, at least for the applications considered here.

Silver 2.5 uses a specification file (similar to the OIL file used to configure OSEK) to specify, which tasks of a hex file to simulate. Silver automatically turns such a spec file into an executable Silver module (dll) or SFunction. A typical spec file looks as follows:

```
01 # specification of sfunction or Silver module
02 hex_file(m12345.hex, TriCore_1.3.1)
03 a2l_file(m12345.a2l)
04 map_file(m12345.map)     # a TASKING or GNU map file
05 frame_file(frame.s)       # assembler code to emulate RTOS
06 frame_set(STEP_SIZE, 10)  # Silver step size in ms
07 frame_set(TEXT_START, 0xa0000000) # location of frame code
08
09 # functions to be simulated, in order of execution
10 task_initial(ABCDE_ini)
11 task_initial(ABCDE_inisyn)
12 task_triggered(ABCDE_syn, trigger_ABCDE_syn)
13 task_periodic(ABCDE_20ms, 20, 0)
14 task_periodic(ABCDE_200ms, 200, 0)
15
16 # interface of the generated sfunction or Silver module
17 a2l_function_inputs(ABCDE)
18 a2l_function_outputs(ABCDE)
19 a2l_function_parameters_defined(ABCDE)
```

The hash `#` character starts a comment, which is ignored by Silver. The spec file first lists the required files (line 2-5). The map file is optional. If a map file is given, the spec file may use symbolic names for functions (such as `ABCDE_20ms`). Otherwise, addresses (such as `0x80081cde`) must be used. File frame.s (line 5) contains startup code and the generic part of the RTOS emulation used to run the tasks. As usual, the startup code sets up stacks, registers, timer and other resources.

Lines 10 - 14 lists the functions to run, and specifies when and in which order to run these functions. Silver uses this to generate the application-specific part of the RTOS

emulation. For event triggered tasks, Silver offers two alternative event models. Line 12 shows a function that is executed *n* times at each Silver step, where *n* is the value of the input variable `trigger_ABCDE_syn` at the beginning of the step. Typically, *n* is 0 or 1 during simulation. Higher values occur only, when more than one trigger event occurs during one step. Silver also offers a more accurate event model, that allows execution of an event triggered task at exact event time, not just at the beginning of a step.

Finally, lines 17-19 define the inputs, outputs and parameters of the generated module or SFunction. In this case, we just reuse the interface of a FUNCTION element of the a2l file, for a function called ABCDE. It is also possible, to list individual variables here by name, as long as their properties (such as address, conversion rule, data type) are described in the a2l file.

In addition, the spec file offers means to specify

- properties of the XCP emulation, if any, to support online calibration and measurement using tools such as INCA and CANape
- data sections to be included into the generated Silver module or SFunction. This way, initial loading of the hex file into simulated memory can be avoided, to speed up simulation.
- memory areas to be copied to other (faster) memory by the start-up code
- functions to be replaced by other functions. This way, a function called by a task of category 1 or 3 to access sensors or actuators can be replaced by a function that directly accesses a plant model or measured values instead.
- logging options, e.g. to track memory access during simulation

The Silver module or SFunction generated this way performs exactly the same computations on PC, as on the real target, since the effect of every machine instruction on memory and chip registers is exactly simulated on PC. However:

- simulation is just instruction accurate, not cycle accurate. This means, the simulation on PC cannot be used to exactly predict execution time on the real target. For example, pipeline effects of different access times to memory (e.g. fast on-chip RAM vs. external RAM) are not modelled.
- conceptually, simulated tasks execute infinitely fast. This means that the emulated RTOS never interrupts a task. The corresponding effects cannot be analysed using the generated model.
- Silicon bugs are not simulated. If a compiler for the real target does not work around a silicon bug correctly, this is likely to be invisible in the simulation: simulated behaviour and behaviour on the real target might differ in such cases.

2.2 Debugging the virtual ECU

The spec file used to port selected parts of a hex file to PC might contain bugs. To locate bugs, Silver integrates a debugger based on the instruction set simulator tsim, developed by Infineon. This debugger is used whenever a simulation does not perform as expected, i.e. differs from measured behaviour. Silver can be switched to step mode. In this mode, Silver uses tsim to run just one TriCore instruction per step, allowing a user to inspect register content before and after execution of an

instruction. It is also possible to set code and data breakpoints, for example to pause a simulation whenever a certain variable is accessed.

2.3 Running times

In order to measure the execution speed of chip simulation, we have ported a complex ECU function implemented by 5 different C functions that run initially, every 10 and 200 ms, and synchronous to the crankshaft. The spec file is very similar to the one shown in section 2.1. The function has 114 scalar inputs, 102 scalar outputs and 108 parameters (characteristics), many of them axes and maps. We have then measured all inputs and outputs of the function on an engine test rig for a scenario of 3.5 minutes and used the resulting measurement (mdf/dat) file to drive simulations in Silver, using either tsim or a generated Silver module. Each simulation executed 380.205256 million instructions (counted by tsim) and has been repeated 5 times on a Windows PC with Intel i5 processor at 2.4 GHz and 2.92 GB RAM. Average execution times found this way are shown in Table 1.

simulator	execution time on PC	MIPS
Infineon tsim	919.15 sec	0.41
Silver module	9.30 sec	40.80

Table 1: Performance of chip simulation for the BGLWM example

The ECU considered here (MED17 with TC1797) runs at 200 MHz and has a performance of about 300 MIPS. Nevertheless, on the ECU, the execution time for the 3.5 minutes scenario is of course exactly 3.5 minutes, due to the real time constraint. On a PC, this function runs 20 times faster.

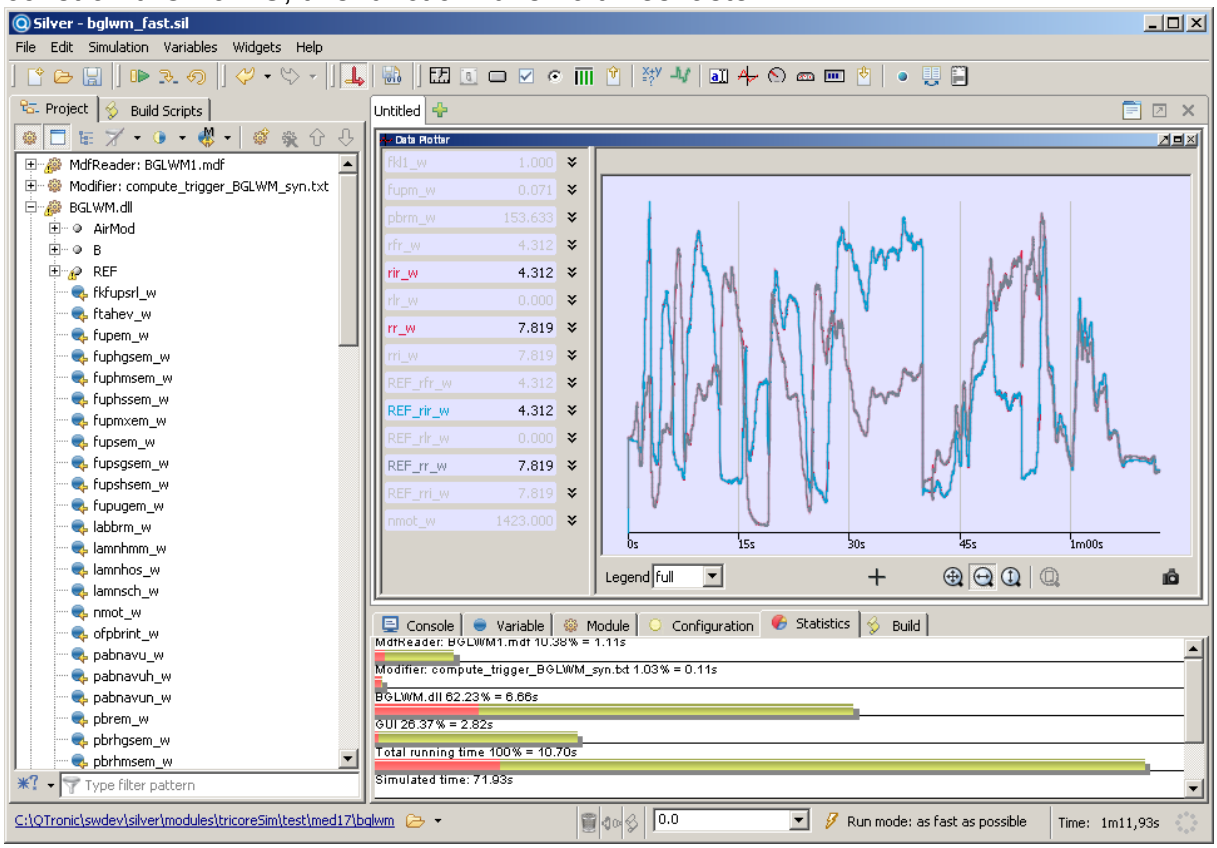

Fig 2: The BGLWM function running in Silver, driven by measurement file

2.4 Exporting a simulation to MATLAB/Simulink

Silver can also turn a spec file as described in section 2.1 into a SFunction, i.e. a mexw32 file that runs in Simulink. This is particularly interesting when using chip simulation to support automated optimization of parameters, because many optimization tools are implemented on top of MATLAB/Simulink. The generated SFunction accepts all characteristics listed in the spec file as SFunction parameters. This makes it easy to connect the generated SFunction with an optimization procedure. For example, the SFunction can be called with workspace variables that are then automatically varied by the optimization procedure between SFunction calls. The performance of a generated SFunction is again about 40 MIPS.

3. Applications of chip simulation

In this section, we shortly sketch current applications of the presented approach at Daimler.

3.1 Bypass hooks on Windows PC

During development of an engine controller, a developer might want to replace a certain function of the ECU by its own version of that function, bypassing the original function. For real ECUs, this can be done with tools such as EHOOKS (ETAS) or No-Hooks (ATI). These tools manipulate the original hex file, such that the bypassed function is not executed any more, but just calls the new function instead. The new function is e. g. developed with MATLAB/Simulink in conjunction with a code generator and a compiler for the target processor. This methodology still requires access to the real ECU: the manipulated hex file needs to be flashed into the ECU, and the ECU needs to run the new function, such that its behaviour can be assessed. In order to further simplify the assessment of the new function, we execute the manipulated hex file in Silver on PC using chip simulation as described above. Such simulations are typically driven open loop by measurement files (MDF).

The placing of bypass hooks by direct manipulation of the hex file is a mighty but error-prone tool. Sometimes a hooked function is not called at all or only some variables are overwritten and some not. Normally, such errors are only detected after the manipulated hex-file was flashed on the ECU and then run on the test bench or in a car. With the possibility of instruction accurate simulation of the patched hex file, we can detect these errors much faster and without any risk to car or engine.

3.2 Numerical optimization of engine parameters

Modern ECUs contain ten thousands of engine parameters that need to be tuned. Calibration of all these parameters is time consuming and complex. Therefore, engine developers all over the world try to automate this task, see for example the proceedings of the biannual conference 'Design of Experiments' [3].
We have combined chip simulation as described above with a procedure for numerical optimization to compute optimal values for certain engine parameters. These computations require an accurate and fast model of the engine function of interest. In the past, we have used hand-coded models of ECU functions, developed

with MATLAB/Simulink. This has been time consuming and error prone. We have now partially replaced these hand-coded models with SFunctions generated automatically by Silver from a given hex file. The generated SFunctions proofed to run as fast as their hand coded counterparts. The replacement of hand-coded floating-point models by generated fix-point SFunctions raises the following problem: Some optimization procedures require gradient information to guide the search for optimal parameter values. For example, when searching for an x that minimizes $f(x)$, the derivative df/dx is to be computed during optimization for different values of x. Finite differences are often used here, i.e. df/dx is computed as $(f(x + h) - f(x)) / h$ for small h, say $h = 10^{-6}$. If f is computed using chip simulation, x and $x+h$ are often both mapped to the same integer, resulting in a zero gradient. As a consequence, the optimization procedure is lacking guidance, and might return a suboptimal solution. There are also so-called derivative-free procedures for optimization. Obviously, these are not affected by the above problem.

4. Conclusions

As demonstrated above, an ECU hex file compiled for some target processor can be executed by the virtual ECU tool Silver on Windows PC, either open-loop driven by measurements or in closed-loop with a vehicle model. Depending on the application, selected ECU functions are simulated, or nearly the entire ECU.

This kind of simulation opens new possibilities to move development tasks from road, test rig or HiL to PCs, where they can be processed faster, cheaper or better in some respect, without requiring access to the underlying C code. Daimler currently uses this innovative simulation approach to support controls development for gasoline engines. Other applications, such as online calibration on PC via XCP seem to be doable as well.

References

[1] A. Junghanns, R. Serway, T. Liebezeit, M. Bonin: Building Virtual ECUs Quickly and Economically, ATZ elektronik 03/2012, Juni 2012. See www.ATZonline.de or http://qtronic.de/doc/ATZe_2012_en.pdf

[2] H. Brückmann, J. Strenkert, U. Keller, B. Wiesner, A. Junghanns: Model-based Development of a Dual-Clutch Transmission using Rapid Prototyping and SiL. International VDI Congress Transmissions in Vehicles 2009, Friedrichshafen, Germany, 30.06.-01-07.2009. http://qtronic.de/doc/DCT_2009.pdf

[3] K. Röpke (ed.): Design of Experiments (DoE) in Engine Development - Innovative Development Methods for Vehicle Engines. Expert Verlag, 2011.

[4] T. Blochwitz, M. Otter et. al.: Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. 9th International Modelica Conference, Munich, 2012.

[5] SystemC, Language for System-Level Modeling, Design and Verification, see www.systemc.org

[6] M. Tatar, R. Schaich, T. Breitinger: Automated test of the AMG Speedshift DCT control software. 9th International CTI Symposium Innovative Automotive Transmissions, Berlin, 2010. http://qtronic.de/doc/TestWeaver_CTI_2010_paper.pdf

The Authors:

Dr. Jakob Mauss, QTronic GmbH, Alt-Moabit 92, 10559 Berlin
Matthias Simons, Daimler AG, 70546 Stuttgart