Full Virtualization of Renault's Engine Management Software and Application to System Development

Yohan Jordan, Dirk von Wissel
Renault SAS - Centre Technique Lardy
1, Allée Cornuel, 91510 Lardy – France

Adrian Dolha, Jakob Mauss
QTronic GmbH
Alt-Moabit 92, 10559 Berlin – Germany

Abstract

Virtualization allows the simulation of automotive ECUs on a Windows PC executing in closed-loop with a vehicle simulation model. This approach enables to move certain development tasks from road or test rigs and HiL (Hardware in the loop) to PCs, where they can often be performed faster and cheaper. Renault has recently established such a virtualization process for powertrain control software based on Simulink models. If the number of runnables exceeds a threshold (about 1500) the execution of the virtual ECU is no longer straight forward and specific techniques are required. This paper describes the motivation behind a Simulink model based process, the virtualization process and applications of the resulting virtual ECUs.

Domain: Critical Transportation Systems
Topic: Processes, methods and tools, in particular: virtual engineering and simulation
Keywords: powertrain control, software development, frontloading, validation and test, virtual ECU

1. Motivation

Consequently, module developers get feedback about the behaviour of their modules late, i.e., weeks or months after releasing a specification (cf. Figure 1).

Fig. 1: Virtual ECU used to frontload system-level validation and test.

The cycle can be considerable shortened by frontloading system-level validation and test. Ideally, system-level test and validation should be performed interleaved with steps 1, 2 and 3 replacing the actual MiL, SiL validation process by a full ECU validation which would provide the software project team with the possibility to validate integrated software within minutes. Technically, such a frontloading of test activities can be achieved us

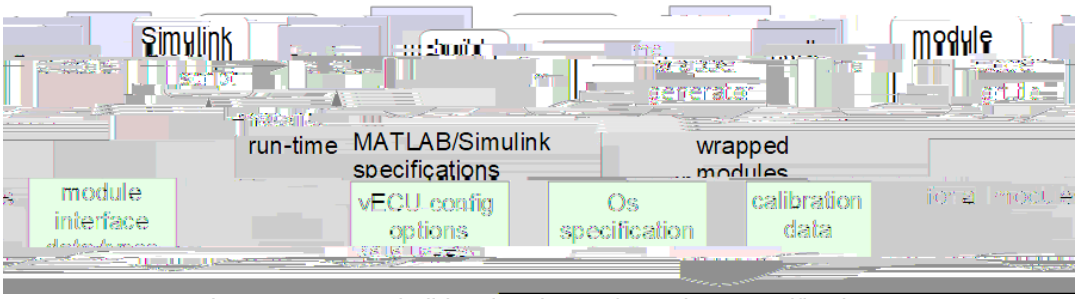

Fig. 2: Process to build a virtual ECU from given specifications

The virtualization process is structured as shown

and a generated model of the Os into Simulink and generate C code in one pass. This proved to be infeasible because initialization of the resulting model takes very long (> 10 hours) and the code generator tends to run out of memory during subsequent code generation, even with 64-bit Simulink.

2.2 Differences between real and virtual ECU

A virtual ECU is a model of the real ECU. Consequently, not all properties of the real ECU are visible in the virtual ECU. This means that not all tests that are relevant can also be moved to the PC. For some tests, access to real ECU hardware is needed.

The most crucial differences between real and virtual ECU as presented here are

Zero-time execution: the virtual ECU behaves like a device with unlimited computing speed. As a consequence, the Os of a virtual ECU cannot interrupt a running task (no pre-emptive multi-tasking, since a task takes no time to execute). All tasks run exactly as scheduled (for example, cyclically with 1, 2 or 10 ms period), no matter how long its execution may take on the real target hardware.

Missing basic software: Hand-coded basic software of the ECU platform (e.g. drivers for Can, Lin, FlexRay) is not part of the virtual ECU and can therefore not be validated. This is not a fundamental restriction of virtual ECUs, but a consequence of the EMS development workflow shown in Fig. 1.

Different production code: C code representation of calibration data and signals as generated with Simulink Coder is close to but not fully equivalent to the corresponding representations of the production code. For example, on both sides, the same scalar data types are used: `float`, `uint8`, `int16` etc. However, C code generated by Simulink Coder for grt represents all signals and tuneable parameters as members of a C struct, while in production C code, these are represented as global variables. Consequently, certain defects cannot be found using the virtual ECU.

Many of these differences can be avoided by switching to the SiL ([1],[3]) or vPiL (virtual processor, [2],[3]) type of virtual ECU. However, as explained above, this was not an option right now, due to the time constant for code generation and workflow issues, but might become feasible in the future.

Fig. 3: Virtual ECU of Renault EMS connected to engine model

Such models cannot be used in a real-time environment, but without problems in conjunction with virtual ECUs.

2.4 Runtime performance of the virtual ECU

A virtual ECU for a typical Renault EMS loads and initializes in less than 5 seconds on a Windows PC. In our experiments, ECU behaviour was sampled with 1 ms steps. Execution speed of the vECU on PC depends on the number of variables (outputs of the vECU) to be recorded during simulation.

When measuring 170 variables per ms, execution of the vECU is 4 times faster than real time.

When measuring 20.000 variables per ms, execution speed is 3 times slower than real time.

The engine model used has been developed for HiL applications and is therefore very fast. In a closed-loop simulation with a virtual ECU on PC, the engine model consumes less than 10% of the computing time.

3. Virtual ECUs: Related Work

In this section, we review existing work on the virtualization of ECUs and motivate the approach chosen here. As shown in Fig. 4, we distinguish three options to create a virtual ECU for execution on a PC.

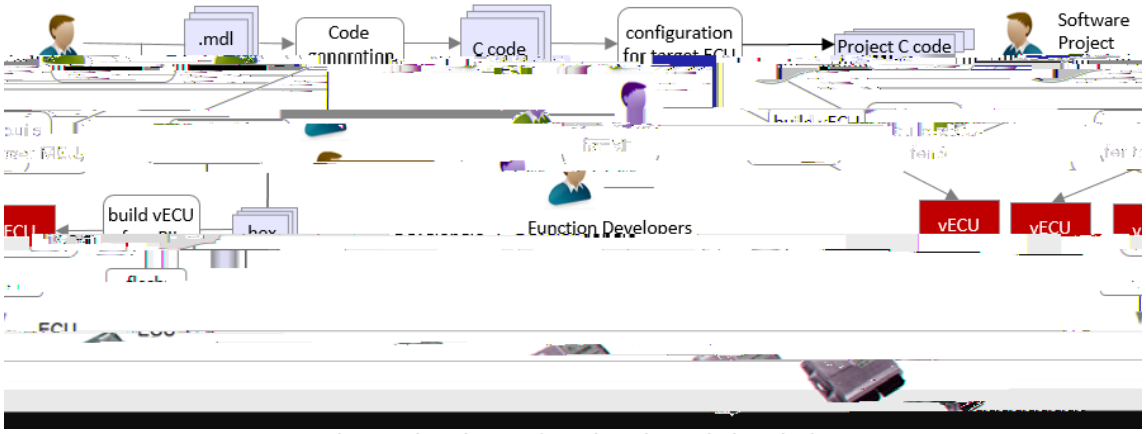

Fig. 4: Virtual ECU for MiL, SiL and virtual PiL

Depending on the use case to be supported, a virtual ECU is created from either

virtual PiL: the hex file resulting from compiling C code for the target processor

SiL: C code compiled for PC

MiL: models that generate the C code

3.1 Virtual PiL: Chip simulation to run a hex file on PC

At the heart of an engine control unit (ECU) is a microcontroller unit (MCU), i.e. a digital processor with additional peripherals such as CAN controllers, analog digital converters (ADCs) etc. integrated on a single chip. Wide spread MCU families for engine control are Tricore (Infineon), PowerPC (Freescale/NXP, STM), and v850 (Renesas). MCUs for engine control often have two or more cores for fast concurrent computing. A virtual ECU can be created by simulating the corresponding MCU. The hex file resulting from compiling the EMS software for the target MCU contains binary program code and calibration data. For example, for a Tricore MCU, the four bytes `0x50`, `0x1F`, `0xF1`, `0x8B` (given in hexadecimal notation) encode the assembler instruction

```
add %d5, %d1 -1
```

which instructs the MCU to take the 32-bit integer currently stored in data register d1, decrement the value by one and store the result in data register d5. A chip simulator reads instructions like this from simulated memory, decodes these and performs the corresponding computation, which may update register and memory content. Chip simulators come in different flavours: A chip simulator is *instruction accurate*, if it correctly simulates the entire instruction set of the target MCU. A chip simulator is *cycle accurate*, if the simulator also correctly predicts how many clock ticks it takes the MCU to run an instruction. Instruction accurate simulators at typically an order of magnitude faster than cycle accurate simulators because a cycle accurate simulator requires a far more detailed model of the underlying MCU, to take into account effects of multi-core processing, instruction pipelines, caching strategy, and speed of memory access on processor speed. Besides, chip simulators differ in the

coverage of their chip model: An *instruction set simulator*

3.4 Motivation for choosing the MiL type of virtual ECU

The objective of the project reported here has been to improve the quality of executable module specifications by providing a virtual ECU that runs all ECU modules simultaneously, only minutes after a module has been edited. It was therefore a natural choice to build the vECU from the module source (Simulink .mdl files), which lead us to a MiL type of vECU. The production C code generator is a shared and limited resource here, not easily available in the working environment of module developers. This currently rules out the SiL type of vECU. Hex files for the target MCU become only available weeks after a module edit (see Fig. 1), which rules out the vPiL type of vECU.

4. Applications of virtual ECUs at Renault engine development

In 2016, Renault created the first fully functional virtual EMS using the process described above. The process has been repeated for about 6 releases (updates and different platforms) of the EMS software since then. Besides, we focus work on setting up the tool chain for the most promising applications of virtual ECUs. These applications are briefly described here, ordered by their location on the time-line of the development process. For each such application, we also sketch the current state of the implementation, if applicable.

4.1 Definition of system requirements

In the requirement definition phase of a new function, a virtual ECU can be used to complete an environmental model when interactions between systems through control actions need to be taken into account to reflect the real system behaviour. This way, simulation can be used to derive (even quantify) requirements. We have not yet implemented this idea based on virtual ECUs as described above.

4.2 Module development in system context

As described above, a main use case for virtual ECUs is validation and test of module specifications in system context before the specifications are forwarded to the software project team. Such tests are best performed from within the development environment (IDE) for modules, which is Simulink here. To support this, virtual ECUs can be configured at compile time for co-simulation with Simulink. In the vECU configuration dialog, a module developer can select his or her module for execution in Simulink, bypassing (replacing) the implementation of the module provided by the virtual ECU. The module developer can then edit his module in Simulink, for example modify the Simulink block that implements a certain runnable. After such an edit, the developer can immediately run the virtual ECU, without rebuilding it. This runs all runnables of all modules of the vECU, except those of the bypassed module, which run in Simulink then. This way, the effect of an edit on behaviour can immediately be seen by the developer in the context of all other modules of the ECU. In such a co-simulation scenario, rebuild of the vECU is only required when the Os specification or the signal interface for the edited module changes, or to get more recent implementations of other modules into the validation loop. Thanks to incremental build, such a rebuild takes only minutes.

4.3 Pre calibration of the EMS

At least 50% of the development time for engine controllers is spent for engine calibration, i.e. tuning of software parameters of the EMS in order to simultaneously meet customer demands (for example, ‘fun to drive’), as well as legal requirements concerning emissions and fuel consumption. With virtual ECUs, Renault started to frontload calibration related activities as well. This is called pre-calibration. The objective is to develop a better starting point for calibration early, in order to gain more time for fine tuning in later phases of development. As pointed out before, a Silver virtual ECU can be configured to load calibration data at runtime from a human readable text file. This means that calibration data can be varied without rebuilding the vECU, which speeds up the calibration process. To support interactive on-line tuning of parameters, a virtual ECU can also be configured to expose all tuneable parameters of selected modules as permanent (dynamic) inputs of the vECU. This way, a calibration engineer can vary parameters even during simulation, typically using a slider provided by Silver’s user interface.

4.4 Virtual integration of modules before production C code is generated

Virtual ECUs enable us to virtually integrate all modules of an EMS and to test the resulting virtual ECU in closed loop with an engine simulation before turning the modules into production C code. This helps to shorten development cycles. Integration problems can now be detected more early, before integrating production C code on real ECU hardware. To exploit this idea, we are currently comparing simulation results recorded on a HiL test system with simulation results computed on PC using a virtual ECU for sufficiently similar versions of EMS software and calibration data. Unexpected differences quickly point us to potential problems.

4.5 Move selected tests from HiL to MiL

Many of the engine tests that run today on limited resources like HiL test systems, engine test rigs and real vehicles could actually be moved to highly available PCs, because they do not depend on properties of the underlying hardware platform. In such a process, only certain releases of the EMS software (in particular the final release) are tested using real target hardware, while all other intermediate releases are tested on virtual platforms. This can for example be used to increase test coverage. Virtual ECUs are a key tool to implement such a mixed strategy. This idea has not yet been explored using the virtual ECUs described here.

4.6 Support joint development within the Renault-Nissan partnership

References

[1] Andreas Junghanns, Jakob Mauss, Michael Seibt: Faster Development of AUTOSAR compliant ECUs through simulation. Full paper presented at ERTS 2014, Toulouse, 2014. http://qtronic.de/doc/ERTS_2014_autosar_sil.pdf

[2] Jakob Mauss: Chip simulation used to run automotive software on PC. Full paper presented at ERTS 2014, Toulouse, 2014. http://qtronic.de/doc/ERTS_2014_chip_simulation.pdf

[3] René Linssen, Frank Uphaus, Jakob Mauss: Simulation of Networked ECUs for Drivability Calibration. ATZ elektronik 4/2016. http://qtronic.de/doc/ATZe_04_2016_en.pdf

[4] Functional Mock-up Interface (FMI). https://www.fmi-standard.org/

[5] Dirk von Wissel, Pedro Moreno Lahore, et al.: Renault Model-Based Design - Powertrain control development process. 23rd Int. AVL Conference “Engine & Environment”, Graz, Austria, Sep.8th - 9th, 2011. https://www.researchgate.net/profile/Dirk_Von_Wissel/publications

[6] J. M. Dressler: A Walk through EMS 2010 Modular Software Development, 4th European Congress ERTS, Toulouse, 2008.

[7] Dirk von Wissel, Jean-Marie Quelin, et al.: Industrial use of HIL Engine Management System validation. 9th Symposium Automotive Powertrain Control Systems, Berlin, Sep. 20th – 21st, 2012. https://www.researchgate.net/profile/Dirk_Von_Wissel/publications