Development Solutions for Automotive Embedded Systems

Methods and Tools

Objectives and Requirements

Each development aims at creating a new function or enhancing an existing function of the vehicle. The functions are all functional features of the vehicle in this context. The user of the vehicle utilizes these functions; they represent a value or benefit for him. The technical implementation of a function, whether it is, in the end, a mechanical, hydraulic, electrical or electronic system in the vehicle, is of secondary importance.

Electronic components – in combination with mechanical, electrical or hydraulic components – offer many advantages in technical implementation, such as regarding the reliability that can be achieved, the weight, the required package space, and costs. That is why, like almost no other technology, electronics is today the key technology in many innovations in automobile construction. Almost all vehicle functions nowadays are controlled or monitored electronically. The constant leaps forward in electronics hardware technology and performance permit numerous new and increasingly powerful functions to be realized by software.

The increasing number of these software functions, their interconnection, increasing reliability and safety requirements, the rising number of vehicle variants, and varying lifecycles for software, hardware and vehicle represent requirements and constraints that have a considerable impact on the development of software for the electronic systems of a vehicle.

Mastering the resulting complexity is a challenge for vehicle manufacturers and suppliers. Safe handling of software and electronic systems must be ensured by appropriate measures during development. The methods supported by ETAS tools and software components presented in this catalog contribute to this goal. Powertrain, chassis and body are the main applications in this area.

Model-Based Development of Vehicle Functions

Inter-discipline cooperation in development (e.g., between powertrain and electronics development) requires a common and overall understanding of the problem. For example, when designing control functions for the vehicle, the reliability and safety requirements – as well as aspects of implementation by software in embedded systems – need to be viewed as a whole.

The basis for this common-function understanding can be a graphical function model covering all system components. Therefore, model-based design methods with notations such as block diagrams or state machines for data and behavior descriptions are increasingly replacing software specifications in text form.

This **modeling** of software functions offers additional advantages. If the specification model is formal, i.e. unambiguous and with no room for interpretation, the specification can be executed on a computer in a **simulation** and can be "experienced" in the vehicle by means of **rapid prototyping**.

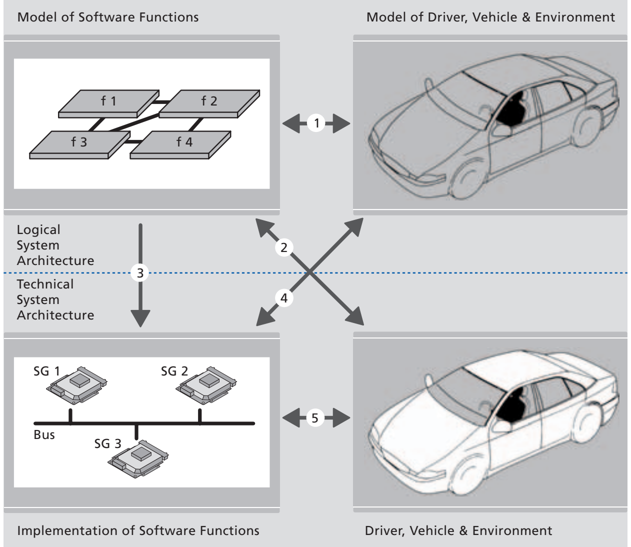
Using methods for automatic **code generation**, the specified function models can be transformed in software components for electronic control units (ECU). This may require that the function models be extended using the necessary optimization measures regarding the required product properties of the electronic system.

In the next step, **lab vehicles** simulate the environment of control units, thereby enabling the testing of the control units in the lab. Compared to bench tests and vehicle tests, this allows to achieve a greater degree of flexibility and easier reproducibility of test cases.

The **calibration** of software functions of electronic systems encompasses the vehicle-specific adjustment of the parameters of these functions that are implemented, e.g., in the form of characteristic values, curves or maps. This adjustment can often be performed rather late in the development process, frequently only directly in the vehicle with running systems, and must be supported by appropriate methods and tools.

Generally, the development methods shown in Figure 1.1 can be distinguished in this model-based development process for software functions.

Methods of a Model-Based Development of Software Functions


Model of Software Functions
Model of Driver, Vehicle & Environment
f 1
f 2
f 3
f 4
1
Logical System Architecture
Technical System Architecture
2
3
4
SG 1
SG 2
Bus
SG 3
5
Implementation of Software Functions
Driver, Vehicle & Environment

1 Modeling and simulation of software functions as well as of the vehicle, the driver and the environment
2 Rapid prototyping of software functions in the real vehicle
3 Design and implementation of software functions
4 Integration and test of software functions with lab vehicles and test benches
5 Test and calibration of software functions in the vehicle

Figure 1.1

This procedure can also be used for developing functional and control unit networks. This adds, however, two degrees of freedom, such as:

- combinations of modeled, virtual and realized functions, and
- combinations of modeled, virtual and realized technical components.

A consistent distinction between an abstract view of functions and a concrete view of their technical realization would therefore be beneficial.

Such an abstract and concrete view can be extended to all components of the vehicle, the driver, and the environment. The abstract view is called here **logical system architecture** (shown in gray in the figures), while the concrete view of the implementation is called **technical system architecture** (shown in white in the figures). The procedure described by the control functions can be applied in general – for example, for monitoring and diagnostic functions, as well.

Software Architecture and Standardized Software Components

Approaches for standardizing the software architecture for micro-controllers used in control units have been successfully introduced. They differentiate, e.g., between the "actual" software functions of the **application software** and a **platform software** that is partially dependent on the hardware (Figure 1.2).

Software Architecture for Microcontrollers and Standardized Software Components

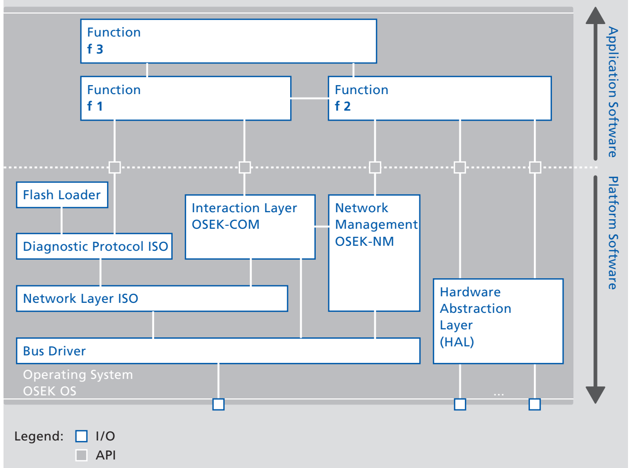


Figure 1.2

A layer of the platform software (Hardware Abstraction Layer, HAL) combines the software components covering the hardware-related aspects of the I/O devices of a micro-controller. As shown in the illustration, the description excludes the I/O devices necessary for communicating with other systems via buses from this "hardware abstraction layer". The bus drivers required for this area are discussed separately.

The platform software also includes upper-layer software components that are required for the communication with other control units in the network or for the communication with diagnostic testers.

Examples of standardized software components are real-time operating systems and the communication and network management according to OSEK (2), or diagnostic protocols according to ISO (3, 4). The software components provide standardized interfaces for the application software (Application Programming Interfaces, API). This allows the platform software to be standardized for various applications. The functions of the application software can be largely developed independent of the hardware.