DEVELOPMENT OF A COMPLETE AUTOSAR 4.0 SOFTWARE PROJECT WITH THE AUTOSAR EDUCATION ENVIRONMENT

Peter Will, Parthib Khound, Jaysheel Mehta and Robert Mayr

Universität Siegen, Department Elektrotechnik und Informatik, Institut für Regelungs- und Steuerungstechnik, Siegen, Nordrhein-Westfalen (North Rhine-Westphalia) – Germany

peter.will@uni-siegen.de, parthib.khound@student.uni-siegen.de, jaysheel.mehta@student.uni-siegen.de, robert.mayr@uni-siegen.de

Abstract: The automotive industry introduced the AUTOSAR standard for the development of software for automotive electronic control units. With the availability of the AUTOSAR Education Environment, students have the possibility to work with AUTOSAR-related hard- and software. This article gives information about the realisation of a first complete AUTOSAR 4.0 Software Project at Institut für Regelungs- und Steuerungstechnik (Institute of Automatic Control Engineering) of Universität Siegen.
Keywords: AUTOSAR Education Environment, AUTOSAR 4.0 Software Project

Introduction

The term AUTOSAR stands for **AUT**omotive **O**pen **S**ystem **AR**chitecture and is an international standard of the automotive industry. AUTOSAR describes a reference architecture for the software of electronic control units. The goal of this standardisation is to keep the growing amount of software with increasing complexity manageable (Kindel & Friedrich, 2009).

Besides the standard itself, the term AUTOSAR is used for the international development partnership that develops the standard as well. Vehicle manufacturers, their suppliers and other companies from the electronics, semiconductor and software industry are members of this partnership.

This development partnership maintains the website https://www.autosar.org/ where the standard can be downloaded for informational purposes. The commercial use of this information is restricted to AUTOSAR-members.

In order to give students the possibility to work with AUTOSAR-related hard- and software the company Vector Informatik (from Stuttgart, Germany), an AUTOSAR-member, has developed the AUTOSAR Education Environment. The hardware included in the Education Package is an automotive electronic control unit (ECU) and a Hardware Debugger for transferring the developed software to the microcontroller in the ECU and of course for debugging the developed program code. The customer can either directly connect his own hardware or has the choice between two different breakout-boxes. On the software side the Education Environment includes parts of Vector Informatik's own AUTOSAR implementation and several software tools for ECU software development and debugging.

We at the Institut für Regelungs- und Steuerungstechnik (Institute of Automatic Control Engineering) of Universität Siegen bought the AUTOSAR Education Environment. Software development for microcontrollers is not something new to us – we did it several times for microcontrollers of different manufacturers. However, since the development of software according to the AUTOSAR standard is very domain specific, several things were new to us. Therefore, the goal of our first student project was to develop a complete AUTOSAR 4.0 Software Project and to document the complete workflow that is necessary to set up the project. The created documentation cannot replace the support if one has a specific problem but it is good basis for further projects.

The first part of this document will cover information that is more general and gives a brief overview of the basic ideas and concepts of the AUTOSAR standard. It introduces the reader to the workflow that is established with the standard – the so-called AUTOSAR-Methodology as well.

The second part of this document will give an overview of the realised AUTOSAR 4.0 Software Project at Universität Siegen.

Part I: Overview of the basic ideas and concepts of the AUTOSAR standard

This part of the document has no claim to give a complete overview of the AUTOSAR standard. Instead, a brief explanation of some basic ideas and concepts are shared for a good understanding.

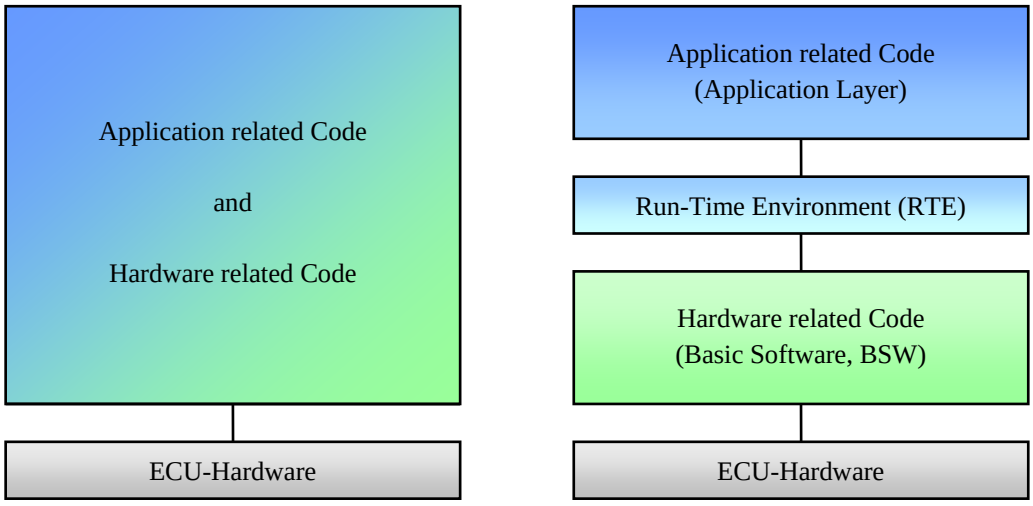
The implementation of a layered architecture

AUTOSAR implements the so-called AUTOSAR Layered Software Architecture (Kindel & Friedrich, 2009). This architecture describes several abstraction layers for software of an ECU: The Application Layer, the Run-Time Environment and the Basic Software. The Basic Software is the layer directly above the ECU-Hardware. Figure 1 (b) shows the structure of this architecture. In such architecture, the interfaces between the layers are well defined. One layer uses the functionality provided by the layer below and itself provides functionality to the layer above. If an error has to be corrected in one of the layers, only the implementation of that specific layer needs correction since the interfaces between the layers remain unchanged. The layer architecture therefore supports the modularity of the software.

Preliminary to the introduction of AUTOSAR, there was not necessarily a layered architecture present in the software of an ECU. This is illustrated in Figure 1 (a).

The layered architecture may remind one of the OSI reference model from the International Organization for Standardization (ISO) (Meinel & Sack, 2012). The OSI reference model is commonly used for explaining the structure of the Internet protocol (TCP/IP). The intension behind both reference models is comparable: The Interfaces between the different layers are defined and hardware dependence decreases as you move up the layers.

The Basic Software (BSW) can be regarded as an operating system that is responsible for hardware abstraction. The Application Layer contains the Software Components with the Application related Code and the Run-Time Environment (RTE) implements the communication between the different Software Components and between Software Components and the Basic Software.



a) Classical ECU b) AUTOSAR-ECU

Figure 1. Comparison of the software structure in a Classical ECU (a) and an AUTOSAR-ECU (b) (see also Figure 3-2 in Kindel & Friedrich (2009)).

The procedure to get executable code for an ECU: The AUTOSAR-Methodology

Several companies deliver the electronic control units in a vehicle. In order to manage and to parallelise the development of these control units for the vehicle the AUTOSAR-Methodology was introduced (Kindel & Friedrich, 2009), (AUTOSAR, 2008). The Methodology describes a procedure or a "Work Product Flow" to get executable code for an ECU. The necessary information for the work product flow is usually stored in xml-files. Normally software tools intensively support the work steps and a by hand modification of the xml-files should not be necessary. Figure 2 gives a simplified overview of the AUTOSAR-Methodology.

The Methodology consists of three views: The System View (red), the ECU View (blue) and the Component View (green). The System View with the System Configuration (1) contains descriptions about all Software Components, ECUs, communications links and system restrictions of a vehicle. The Software Components will be assigned to a certain ECU when the system is configured (2). From the resulting System Configuration Description (3), one can conclude which data has to be exchanged between the ECUs. Since not all ECUs and Software Components come from the same supplier, ECU Views and Component Views can be extracted (4) from the System View. The different views allow a parallelization of the complete development process for all ECUs and Software Components. Software Components that have been developed with the help of the Component View and in parallel to everything else can be integrated into an ECU (14) as binary code.

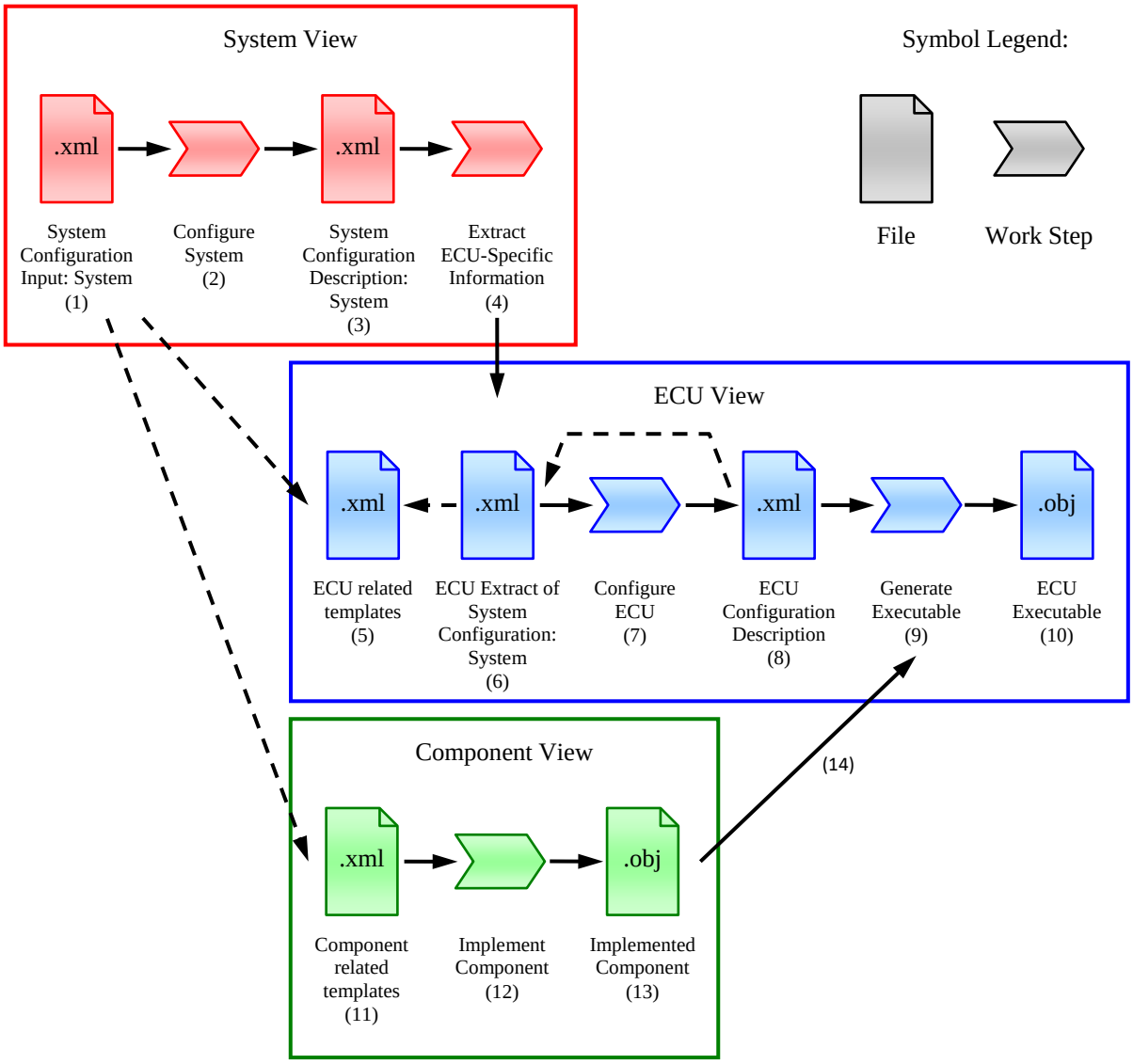


Figure 2. Simplified overview of the AUTOSAR-Methodology (see also Figure 5-10 in Kindel & Friedrich (2009)).

With the help of the ECU related templates (5) and the ECU specific extract of System Configuration (6) one ECU can be developed. If this information is given, the ECU can be configured (7) in the next step. This configuration step includes for example the selection and configuration of the modules of the Basic Software that are needed to realise the functionality and the configuration of the task scheduler of the operating system. The result of the configuration step (7) is the ECU Configuration Description (8). The work step generate executable (9) includes the generation of source code (for example for the Run-Time Environment or the Basic Software), the compilation of the source code (for example of the generated source code or of the Software Components that are given form of source code) and finally linking all the object files together to an executable program (10).

With the help of the Component related templates (11) a Software Component or several versions of a Software Component can be implemented (12). The implemented Software Component is an object file (13) that can be integrated into an ECU (14) in link stage of the executable generation (9).

The Communication between Software Components and the Virtual Functional Bus

The Virtual Functional Bus (VFB) describes abstract communication relations between Software Components (SW-C) in the System View (Kindel & Friedrich, 2009), (AUTOSAR, 2010). During the System Configuration, all Software Components will be assigned to the respective ECUs. According to the AUTOSAR-Methodology, the Configurations of all ECUs (No. 6 in Figure 2) are extracted (No. 4 in Figure 2) from the System Configuration (No. 3 in Figure 2). Figure 3 illustrates this step of development that is carried out with the help of software tools.

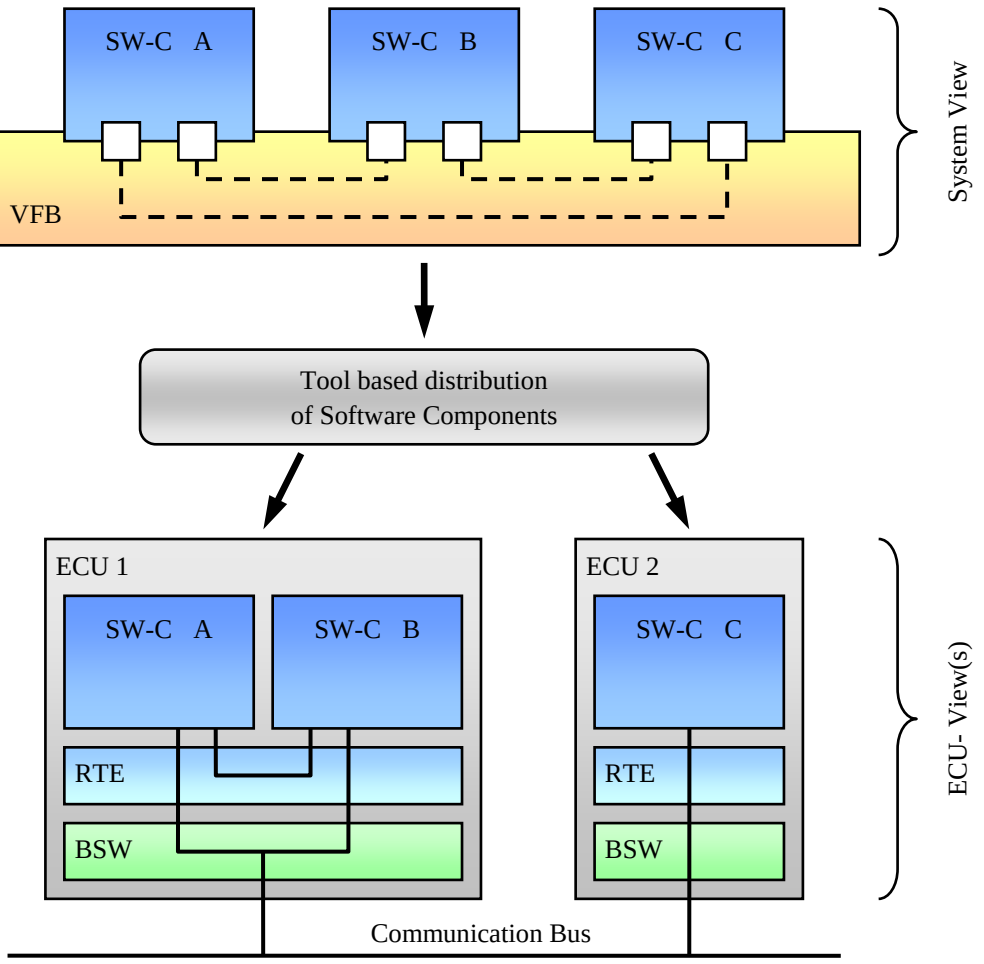


Figure 3. Tool based distribution of Software Components across ECUs (see also Figure 6-2 in Kindel & Friedrich (2009)).

When the ECU Views are generated, the abstract communication relations between Software Components will be replaced by concrete communication connections. For this step, it does not matter if the communication takes place between two Software Components on the same ECU or if the communication has to be established across a Communication Bus. This flexibility is reached by the auto-generation of the Run-Time Environment that realises the communication connections between the Software Components.

If there are problems in one ECU regarding the computational power of the CPU, the System Configuration can be modified and the extraction of the ECU Configurations can be repeated. A prerequisite for this flexibility within the distribution of Software Components is that there are enough free communication resources on the communication bus. A constrain that two Software Components have to be assigned to the same ECU can restrict this flexibility.

The structure of the Basic Software

Within the layered architecture the Basic Software is located below the Run-Time Environment and above the ECU-Hardware. In addition to that, the Basic Software itself is subdivided into three horizontal abstraction layers and six vertical sections (Kindel & Friedrich, 2009). Figure 4 illustrates the location of the Basic Software in the layered architecture and the structure of the Basic Software itself.

The vertical subdivision of the Basic Software results in the following six sections (from left to right): System Services, Device Stack, Memory Stack, Communication Stack, I/O-Stack and Complex Device Drivers. The three horizontal Abstraction Layers are (from bottom to top): Microcontroller Abstraction Layer (MCAL), ECU Abstraction Layer and Service Layer.

The section System Services provides basic functionality as operating system-, diagnostic- and ECU State Management functions to Application Software Components and to other modules of the Basic Software. The Device Stack abstracts Microcontroller internal Hardware like the Watchdog and the Timers. The Memory Stack provides unified access to non-volatile memories as Flash-Memories and EEPROMs.

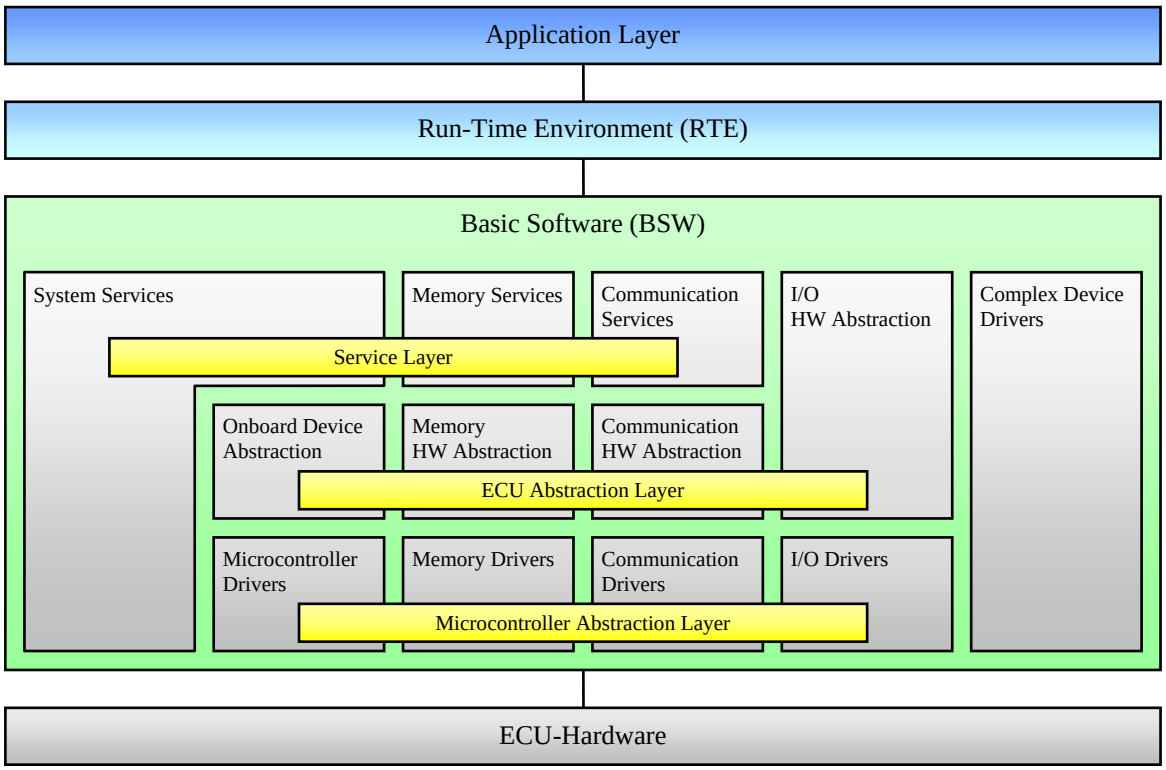


Figure 4. Structure of the layered architecture and substructure of the Basic Software (see also Figure 9-2 in Kindel & Friedrich (2009)).

The Communication Stack provides communication services to the Application- and the Basic Software in order to exchange data with other electronic control units. The abstraction within this stack is done in a way that applications can be developed independently from the type of communication bus they use. The I/O-Stack provides access to analog-, digital- and PWM I/O-Pins of the ECU.

The section Complex Device Drivers does not belong to any layer of the Basic Software and is not specified in detail. A Complex Device Driver provides the possibility to integrate new modules into the AUTOSAR system. There can be three reasons why a Complex Device Driver is implemented: 1. A module for a special functionality is missing, 2. There are timing requirements that cannot be fulfilled with AUTOSAR and 3. Existing software shall be ported to AUTOSAR step by step.

Part II: Overview of the realised AUTOSAR 4.0 Software Project

This second part of this article will give an overview of the complete AUTOSAR 4.0 Software Project (Khound, 2017) realised by Mr. Parthib Khound, student in the master course Mechatronics at Universität Siegen. A second work of Mr. Jaysheel Mehta, also student in the master course Mechatronics, is still in progress.

Overview of the Hardware in the System

The block diagram in Figure 5 gives an overview of all hardware components that were used during the development of the AUTOSAR 4.0 Software Project. The ECU (1), for which the Software Project was developed, can be seen on the left side of the diagram. The ECU is mounted on a Breakout Box (2). The Breakout Box contains some input/output elements like switches, adjustable resistors, LEDs and connectors for several types of busses. The three LEDs numbered with 10 up to 12 and the connector for the CAN-Bus are used in this project. The Hardware Debugger (3) which is mounted on top of printed board of the ECU is used to flash the generated program code to the ECU and can of course be used to debug the developed program code. The Hardware Debugger is connected via USB link to a PC (4) that is used for software development and debugging.

The primary data source in our system is a 3D Acceleration Sensor with an additional temperature sensor (5), that sends its measured data periodically on the CAN Bus. The CAN Bus Interface (6) is used to monitor the data on the CAN Bus. The CAN Bus Interface is connected to the PC via USB link.

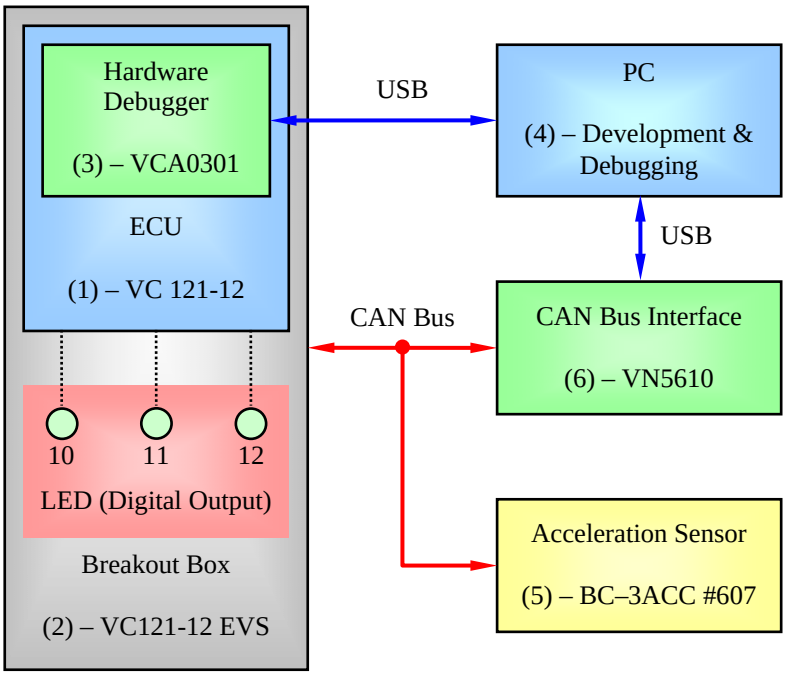


Figure 5. Block diagram of the relevant components in the overall system.

Most of the hardware such as the ECU (Type: VC 121-12), the Breakout Box (Type: VC121 12 EVS) and the CAN Bus Interface (Type: VN5610) come directly from Vector Informatik. The Hardware Debugger is a product of the company iSYSTEM AG and is sold by Vector Informatik (Type: VCA0301). The Acceleration Sensor (Type: BC–3ACC #607) was thankfully provided by 2D Debus & Diebold Meßsysteme GmbH.

Overview of the software used to develop the project

All software that was used to develop the code for the ECU is listed in Table 1. In the project described here, only one ECU is present. Therefore, the work steps within the ECU View (see Figure 2) of the AUTOSAR-Methodology were important for us. In our case, there is no System Configuration or System Configuration Description. Therefore, the configuration for the ECU could not be extracted from it. We had to create the ECU Configuration by ourselves. For configuration purposes, the DaVinci Configurator Pro (1) was used.

In order to implement the functionality described in the next chapter, the ECU needs to communicate. It receives the messages from the Acceleration Sensor and transmits messages after processing the data of the sensor. The easiest way to set up the communication is to describe the structure of the messages on the bus in a database file and add this file to the configuration. The DaVinci Configurator Pro adjusts many settings in the project according to the CAN database files. The CAN messages database files themselves can be created with CANdb++ (2).

The creation of Software Components and skeletons for Application Code has been done with DaVinci Developer (3).

CANoe (4) is a tool for monitoring and analysation of data on a bus. In order to bring the collected data to the user's eyes self-defined virtual panels can be created in CANoe. Beyond this, the software has the ability to simulate complete ECUs or complete networks of ECUs (residual bus simulation). What type of bus can be accessed by CANoe depends on the type of bus interface you have.

winIDEA (5) is the software that gives you access to the Hardware Debugger mounted on top of the board of the ECU. The developed program code can be flashed to the ECU with the help of this software and it can be of cause used for debugging.

A GNU toolchain (6) is used to compile and link the developed software for the microcontroller in the ECU.

No.	Software	Developer	Description
1	DaVinci Configurator Pro	Vector Informatik	Configuration and Generation of the Basic Software and the Run-Time Environment
2	CANdb++	Vector Informatik	Creation of CAN Bus communication database files
3	DaVinci Developer	Vector Informatik	Creation of software components and skeletons for application code
4	CANoe	Vector Informatik	Monitoring, Analysation and Simulation of communication bus signals
5	winIDEA	iSYSTEM	Software for flashing the generated program code to the ECU and for debugging of the developed software
6	GNU toolchain	Free Software Foundation, Red Hat (Software)	Compiler and Linker for the ECU-Hardware

Table 1. List of Software used in this project.

Overview of the developed functionality

The implemented functionality of the ECU can be subdivided into two periods. The first period is the first six seconds after the ECU is started. The second period is the time after the first six seconds.

Within the first six seconds, the ECU will let the LEDs 10, 11 and 12 glow one after another. This can be regarded as a dummy replacement for a self-test function that is executed after the ECU is started.

Six seconds after the ECU has been powered, the main algorithm is started. Within the main algorithm, the ECU receives the data from the Acceleration Sensor via CAN Bus, evaluates the received data, retransmits a CAN frame and controls the LEDs 10, 11 and 12. The data of the Acceleration Sensor and the data coming from the ECU can both be evaluated and viewed in the software CANoe on the PC.

The Acceleration Sensor transmits its measured data periodically on the CAN Bus. This data includes the accelerations in x-, y- and z-direction plus the temperature of the sensor. The ECU checks all four values against threshold values and transmits a CAN message which contains information if the measured values are is below or above their corresponding threshold values. Additionally, the ECU will let the LEDs 10, 11 and 12 glow if the acceleration in x-, y- or z-direction exceeds the corresponding threshold. As already mentioned, the data of the Acceleration Sensor as well as the status values transmitted by the ECU are visualised by CANoe.

Summary

Vehicle manufacturers require that the software of new developed automotive electronic control units are developed according to the AUTOSAR standard. The standard itself introduces a Layered Software Architecture and a work product flow called AUTOSAR-Methodology that describes how software for an ECU can be developed according to the standard. The included concept of the Virtual Functional Bus provides the possibility to move software components between ECUs and increases thereby flexibility. The Basic software provides many services and functionality to the Run-Time Environment and the application software. Therefore, it can be regarded as something like an operating system. Unlike an operating system for a personal computer, the Basic software is not delivered preconfigured. This is a task the user has to perform. Since a lot of flexibility is requested, this task can be difficult. With the help of the AUTOSAR Education Environment, a first complete AUTOSAR 4.0 Software Project could be developed. The created documentation is good basis for further AUTOSAR projects.

References

Kindel O. & Friedrich, M. (2009). Softwareentwicklung mit AUTOSAR. Grundlagen, Engineering, Management in der Praxis. Heidelberg, Baden-Württemberg: dpunkt.verlag.

Meinel C. & Sack H. (2012). Internetworking. Technische Grundlagen und Anwendungen. Heidelberg, Baden-Württemberg: Springer.

AUTOSAR. (2008, February 14). AUTOSAR Methodology. [PDF]. Retrieved from https://www.autosar.org/fileadmin/files/standards/classic/3-0/methodology-templates/methodology/auxiliary/AUTOSAR_Methodology.pdf

AUTOSAR. (2010, October 14). Specification of the Virtual Functional Bus. [PDF]. Retrieved from https://www.autosar.org/fileadmin/files/standards/classic/3-0/main/standard/AUTOSAR_SWS_VFB.pdf

Khound, P. (2017, June 7). Development of a complete AUTOSAR 4.0 Software Project for VC121-12 Automotive Electronic Control Unit (Unpublished project work). Universität Siegen, Nordrhein-Westfalen, Germany.

