IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

A Conversion Concept for a Legacy Software Model towards AUTOSAR Compliance

Vimal Sivashanmugam
Institute of Technical Informatics
Graz University of Technology
Graz, Austria

Tobias Scheipel, Marcel Baunach
Institute of Technical Informatics
Graz University of Technology
Graz, Austria

Bhargav Adabala
Powertrain Engineering
AVL List GmbH
Graz, Austria

***Abstract*—Automotive Open System Architecture (AUTOSAR) is a commonly established standard for automotive software development. Over the last decade, the usage of AUTOSAR methodology for Electronic Control Unit (ECU) software development has gained popularity among the industries. While AUTOSAR has put forth an efficient methodology for the stepwise development of the software from the system design phase until the ECU integration phase, the guidelines for the conversion of a legacy software model into an AUTOSAR compliant software are not covered by the standard. This work investigates the direct conversion of legacy software into AUTOSAR compliant software. A suitable AUTOSAR conversion methodology has been identified and applied to the non-AUTOSAR legacy software by examining the deviations from the standard along the V-model development workflow. After the software has been converted to include the missing AUTOSAR features, it has been analyzed whether a complete AUTOSAR conformance is achievable.**

***Keywords*—AUTOSAR, Electronic Control Unit, V-model.**

I. Introduction

AUTOSAR has been set up in 2003 to promulgate a common standard in automotive software development. The standard proposes a unique layered architecture and a unique software development methodology. The benefits of setting a common standard are not limited to the reduced effort in the development process, but also to drive cost efficiency, quality requirements, easier work-sharing, and software reusability [1]. Although AUTOSAR encourages software reusability, the problem of converting fully non-AUTOSAR legacy software to AUTOSAR compliant software is not well addressed by the standard. As the automotive industries have been increasingly adopting AUTOSAR methodology over the last decade, the conversion from a legacy software model to standard-compliant software also gains importance. Through the direct conversion approach, automotive companies can focus on the reusability of the non-AUTOSAR software for AUTOSAR projects rather than setting up the project from scratch. In this work, an approach to convert a proprietary legacy software to meet the standard compliance is discussed. The legacy software is the Hybrid Control Unit (HCU) software proprietary to AVL List GmbH. Firstly, the deviations present in the software to the AUTOSAR standard have been studied. Secondly, the steps for conversion have been put forth which also the toolchain and methodologies for conversion. Once the software is converted, the performance of the standard-compliant software in comparison with the non-AUTOSAR proprietary software is evaluated on a Hardware-in-the-Loop (HIL) setup. In this step, different versions of the converted software are also generated with different optimization settings, and the performance is compared with the original legacy software. Furthermore, the degree of conformance of the converted software to the standard will also be examined. The rest of the paper is organized as follows: Section II discusses some of the related works already presented in the topics of migration of legacy software to AUTOSAR standard. Section III provides a short overview of the software architecture of the AUTOSAR layered architecture and that of the legacy software. Section IV presents the deviations in the legacy software to the AUTOSAR standard. In Section V, the AUTOSAR conversion approach has been discussed. Section VI explores the evaluation results of different versions of the converted software.

II. Related Work

This section discusses some of the related work already published in the field of migration from legacy software to AUTOSAR compliant software. Daehyun et al. [2] propose a migration concept of a legacy software model of an interior lighting system to the AUTOSAR platform. The authors discuss the need to decompose the legacy application among various Software Component (SWC) types and use a separate AUTOSAR specific Basic Software (BSW) stack for integration through Run Time Environment (RTE) generation. The work by James et al. [3] use MATLAB scripts for the conversion of legacy applications to AUTOSAR format and dSpace SystemDesk for the SWC design and RTE generation. In both the works discussed above, the authors generated the RTE using AUTOSAR specific tools and integrated with the AUTOSAR specific BSW. In contrast, the work described in this paper is not migration to the standard platform in its entirety, but rather the incorporation of AUTOSAR concepts in the existing legacy software. It has been investigated whether complete standard compliance is possible when the Application Software (ASW) of the legacy software is converted to AUTOSAR format and integrated with the typical non-AUTOSAR BSW of the legacy software. The RTE layer, in this case, has been tailored to suit the integration needs.

III. OVERVIEW OF SOFTWARE ARCHITECTURES

In this section, the software architecture of the legacy software and the AUTOSAR standard are discussed. Referring to Figure. 1, the layered architecture proposed by AUTOSAR can be divided into three layers: the ASW, the RTE, and the BSW. The ASW is coded as individual SWCs that communicate with each other via dedicated ports. The RTE layer is the interfacing portion that is used to glue the ASW with the BSW. The RTE also aids in resolving the port communication between various SWCs and also the signal

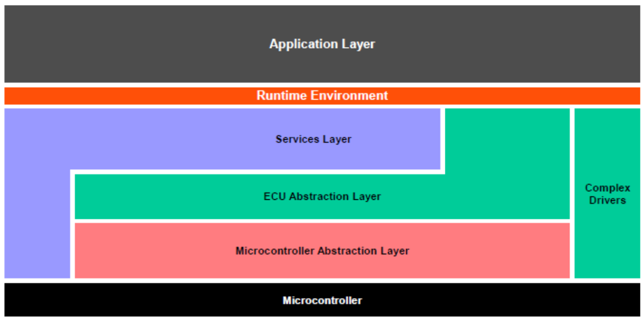


Figure 1. AUTOSAR layered architecture.

transfer between the ASW and the BSW. Additionally, the task bodies which are used to schedule the runnable-entities are also housed within the RTE. The BSW is the core software portion that contains the Operating System (OS) and other software drivers. The BSW can be hierarchically subdivided into various sublayers as shown in Figure 1. The topmost Service Layer is used to provide various BSW services to the ASW. The ECU Abstraction layer interfaces the Service Layer and the Microcontroller Abstraction Layer (MCAL). The MCAL is the software layer written directly over the hardware. In comparison, the software architecture of the legacy software is more or less similar to that of the AUTOSAR architecture (Figure. 2). The software can be classified into the ASW, the Customer Interface Layer (CIL), and the BSW. The CIL is the interfacing portion, which is used to manage the communication (COM) and diagnostic signal transfer between the ASW and the BSW. This layer is not similar to the RTE. Unlike the RTE, which includes the RTE function calls, the ASW and the BSW are interfaced via variable mappings. Additionally, the BSW used in the legacy software is not AUTOSAR compliant, as the BSW supplier has used their proprietary architecture in its implementation.

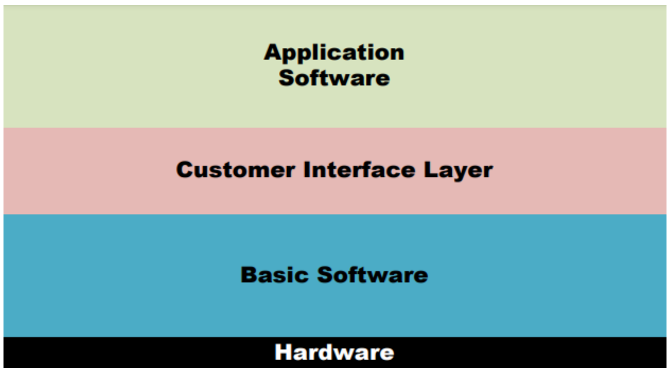


Figure 2. Legacy software architecture [5].

IV. DEVIATIONS FROM AUTOSAR

The software deviations from AUTOSAR have been investigated not only based on the software architecture but also based on the software development process. Hence, the V-Model flow of Software Development Lifecycle (SDLC) has been used to identify deviations in the legacy software from AUTOSAR. The software deviations have been categorized as follows as per the V-model workflow:
A) Deviations at the requirements level.
B) Deviations at the design level.
C) Deviations at the implementation level.
D) Deviations at the integration level.
These deviations will be introduced as follows:

A. Deviations at the Requirements Level

Considering the deviations at the requirements level, the requirements, in this case, are the BSW requirements provided to the BSW supplier by the proprietor of the legacy software. A predefined architecture for the BSW is not specified in the requirements provided to the BSW supplier. Therefore, the BSW supplier for the legacy software follows proprietary implementation for the BSW architecture. The AUTOSAR consortium, in contrast, defines a standard architecture for the BSW [4].

B. Deviations at the Design Level

This section discusses the deviations in the interface handling and scheduling concepts. Figure. 3 shows an overview of the interface handling in the legacy software. The application SWCs communicate via globally defined port variables. The diagnostic and communication signals from the ASW are mapped to variable interfaces at the CIL level which in turn are mapped to the interfaces provided by the BSW. AUTOSAR architecture (Figure. 4), on the other hand, proposes three types of interfaces for realizing the communication between the ASW and the BSW. The interface types are AUTOSAR interfaces, standardized AUTOSAR interfaces, and standardized interfaces [4]. The AUTOSAR interfaces are the port types that can be used for interaction between the software components. The AUTOSAR port types are further classifiable into Sender/Receiver ports, Client/Server ports, etc.

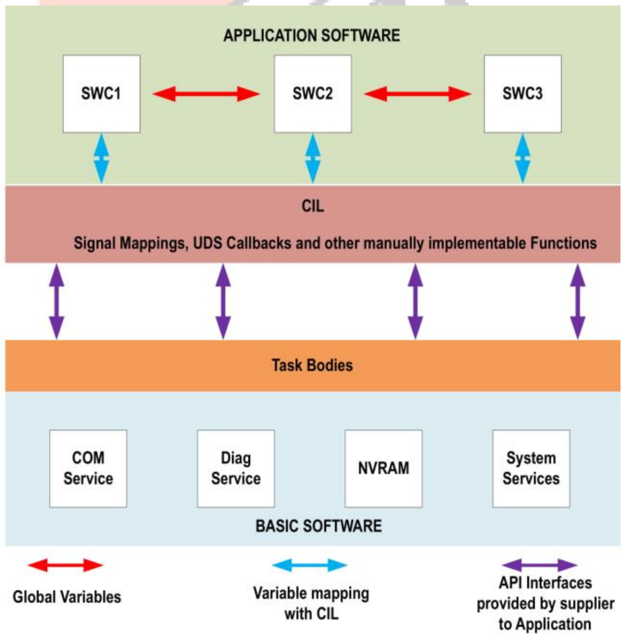

APPLICATION SOFTWARE
SWC1
SWC2
SWC3
CIL
Signal Mappings, UDS Callbacks and other manually implementable Functions
Task Bodies
COM Service
Diag Service
NVRAM
System Services
BASIC SOFTWARE
Global Variables
Variable mapping with CIL
API Interfaces provided by supplier to Application

Figure 3. Interface handling in legacy software [5].

All the port communications are resolved via the RTE. The standardized AUTOSAR interfaces are the standardized ports that can be used for providing BSW services to the upper layers. A notable use case for the standardized AUTOSAR interfaces is the diagnostic data transfer between the application software components and the Diagnostic Communication Manager (DCM) module in the BSW. The third category of interfaces, the standardized interfaces, are standardized API calls used for interaction among the BSW modules as well as the RTE. In terms of the scheduling concept, both the legacy software and the AUTOSAR architecture use Operating System (OS) tasks to schedule the runnable entities. However, only one basic task of periodicity 10ms is used by the legacy software to schedule all the ASW runnable entities as shown in Figure. 5.

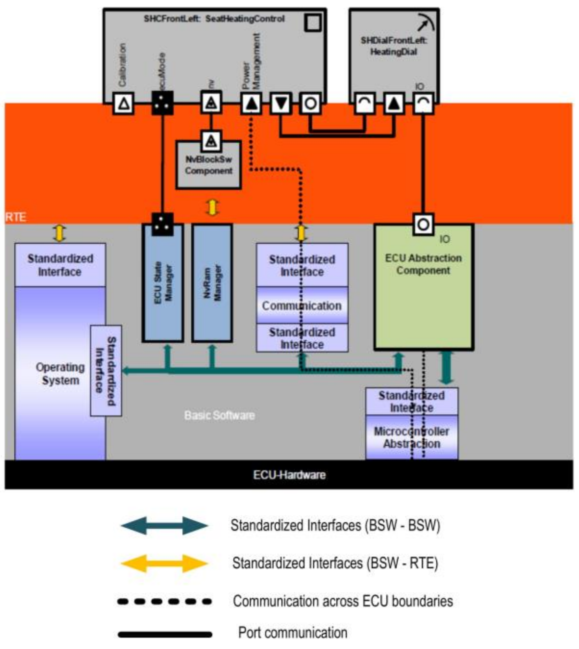


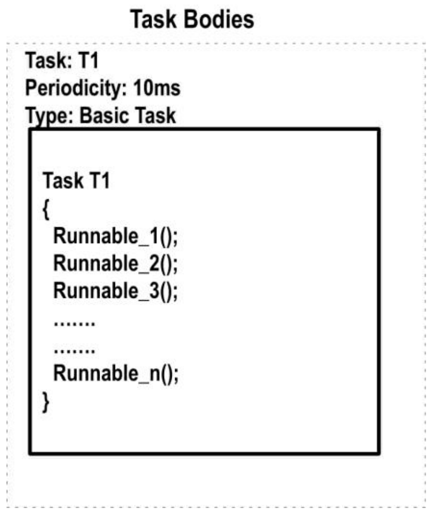
Figure 4. Interface handling in AUTOSAR environment

The task bodies are provided as empty functions by the BSW supplier. The calls to the runnable entities are made using one of these task bodies. The runnable entities are invoked sequentially so that the complete execution of the task body finishes within the next period of the task. As there is no synchronized execution among the ASW runnable entities, the OS events are not used.

In the AUTOSAR case, there is a provision to use more than one task type (basic or extended) depending on the application context. The basic tasks are used for runnable entity with no wait points and extended tasks are used for runnable entity with one or more wait points which have to be resolved by the occurrence of the particular RTE event. Additionally, the task bodies are generated as part of the RTE generation.

C. Deviations at the Model Development and Code Generation Level

The software development in the case of legacy software begins with the definition of system requirements. From the system requirements, the software requirements for the HCU are filtered out and the HCU software system is designed. In this phase, the list of SWCs and the Input/Output (I/O) and parameter signals present in the HCU software system for each SWC are defined. Automotive Data Dictionary (ADD) [8] has been used to define model I/O and parameter signals, which are stored as database files for each SWC. In the SWC design phase, the model algorithm is developed using MATLAB version 2013b and dSpace TargetLink.



```
Task T1
{
 Runnable_1();
 Runnable_2();
 Runnable_3();
 .......
 .......
 Runnable_n();
}
```

Figure 5. Scheduling concept in legacy software [5].

In the AUTOSAR case, the AUTOSAR Extensible Markup Language (ARXML) files [4], which are also the description files defined in AUTOSAR schema, play an important role in every phase of development. AUTOSAR software development begins with the definition of the System Description [4], which includes information about the complete vehicle ECU system. The System Description is prepared by the automotive Original Equipment Manufacturers (OEMs). The information about a particular ECU (e.g. HCU) is extracted as an ECU Extract [4] and delivered to the respective ECU manufacturers. The ECU suppliers can then extract the SWC information as an SWC description and port it into a model-based development tool such as MATLAB and can implement the SWC model algorithm.

D. Deviations at Integration Level

In the case of legacy software, the generated model SWC code files, the CIL code, and the BSW code files from the supplier are built in a build environment and the ECU executable can be generated in binary format. In the AUTOSAR case, the configuration of BSW modules and the RTE is a necessary step in the software integration phase. The BSW modules must be individually configured using standard configuration tools and the ECU Configuration Description has to be created. The result of the configuration step is the BSW configuration code files. The configuration code files have to be built along with the BSW static files and the RTE code files during the ECU build process.

V. AUTOSAR CONVERSION APPROACH

This section discusses the conversion approach followed to incorporate AUTOSAR features in the legacy software architecture shown in Figure. 3. Referring to Figure. 6, the ASW models of the legacy software are converted to AUTOSAR format and integrated with the original BSW of legacy software. As the objective of this work is to incorporate AUTOSAR features by reusing the software modules of the legacy software, the BSW has been reused for AUTOSAR conversion. As indicated earlier, the BSW of the legacy software is not compliant with the AUTOSAR standard. Besides, the CIL has also been retained since completely replacing it with the RTE can involve huge manual modifications.

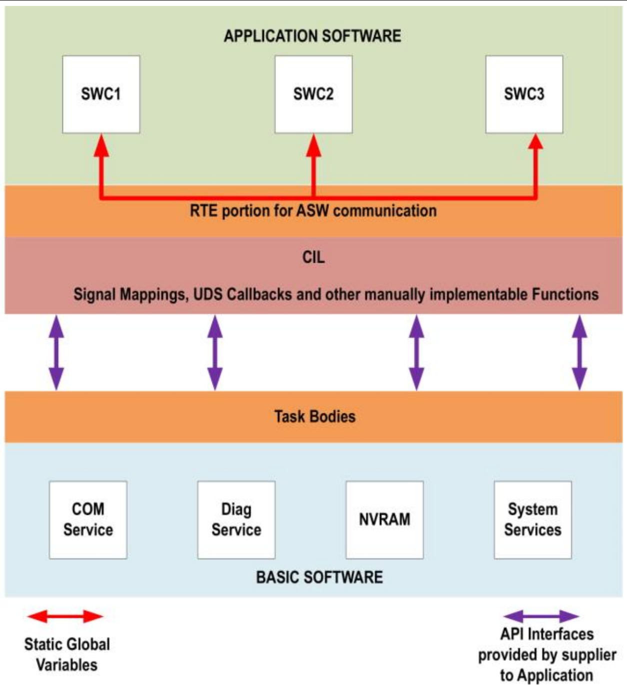

APPLICATION SOFTWARE
SWC1
SWC2
SWC3
RTE portion for ASW communication
CIL
Signal Mappings, UDS Callbacks and other manually implementable Functions
Task Bodies
COM Service
Diag Service
NVRAM
System Services
BASIC SOFTWARE
Static Global Variables
API Interfaces provided by supplier to Application

Figure 6. AUTOSAR conversion approach [5].

However, the portion of the RTE layer to support the SWC port communications is generated and integrated on the top of the CIL. The converted architecture, shown in Figure. 4, still meets ICC1 (Implementation Conformance Class) (p. 2-8, [1]) AUTOSAR compliance. The AUTOSAR ICC1 compliance requirement states that the BSW and the RTE can be considered as a single proprietary entity and that the interfaces between this BSW unit and the ASW shall be AUTOSAR conform. In the converted software architecture, this can be understood from Figure. 7. The BSW altogether with the RTE and the BSW is considered as a single proprietary implementation and the ASW is interfaced with this unit via AUTOSAR conform implementation.

VI. IMPLEMENTATION AND EVALUATION

The conversion of non-AUTOSAR models of the legacy software to AUTOSAR compliant models follows the bottom-up approach of the AUTOSAR workflow. For the conversion, MATLAB/Simulink version 2017b was employed. MATLAB introduces an additional support package to handle the models in the AUTOSAR development environment. MATLAB scripts were additionally developed to automate the model conversion to AUTOSAR format. Besides, a separate RTE generator engine (based on the work by Shiquan Piao et al. [6]) was also developed using MATLAB script to generate the RTE function definitions involving SWC port communication. Eventually, the code files for the RTE and the ASW models were generated for the AUTOSAR converted software. The generated code files are in turn subjected to ECU build. The result of the build process is the ECU executable in binary format. Additionally, different versions of the AUTOSAR converted software were also prepared based on different optimization settings as follows:

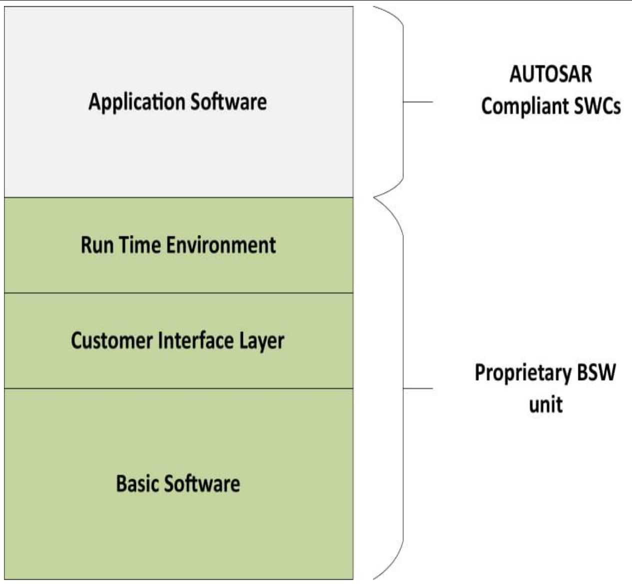


Figure 7. AUTOSAR compliance analysis for the converted software

- **AUTOSAR Normal Version** with no optimization settings. The compiler optimization flag is O0 in this case. It is the default optimization setting and indicates that zero optimization has been applied.
- **AUTOSAR Inline Version** in which the RTE function definitions are made inline. The optimization level is O0 for this software version.
- **AUTOSAR Optimized Version Level 1** which is compiled with the O1 optimization flag, which is the first level of optimization for reduced code size and execution time [7]. The RTE function calls are not made inline in this software version.
- **AUTOSAR Optimized Version Level 2** in which all the RTE function definitions are made inline, as well as the O1 optimization level, is applied.

The software versions were validated on a HIL setup and the performance metrics were analyzed in comparison with a non-AUTOSAR software used as a reference. A customizable real time simulation environment made of National Instruments VeriStand [10] was the HIL setup employed in the study. The software performances were compared in terms of memory consumption and execution time metrics. Figure. 8 shows the comparison of memory consumption analysis for different software versions. Although the trend of memory consumption is the same for all the software versions (both non-AUTOSAR and AUTOSAR compliant) across the. .caldata, .rodata, and. .bss memory sections, a significant difference could be analyzed in the .data and .text memory portions. The .text portion represents the code size and the AUTOSAR Normal and Inline versions (compiled with zero compiler optimization setting) tended to occupy more memory space than the non-AUTOSAR reference software. The reason can be the difference in code structure caused by the different code generator tools and also the presence of RTE function definitions in the AUTOSAR versions. Nevertheless, the code size improved dramatically for the AUTOSAR Optimized Level 1 and Level 2 software versions due to the compiler-induced optimization. A similar observation can be seen in the execution time metric (Table I). The execution time, in this case, is the total runtime of the OS task that is used to schedule all the ASW runnable entities. The mean execution time of the AUTOSAR Normal version had been about three times more than the reference software. However, with an increasing degree of optimizations for the

AUTOSAR versions, a significant improvement in the execution time could be observed. The average execution time of the AUTOSAR Optimized Level 2 version was nearly equal to that of the reference non-AUTOSAR software with only a marginal difference of about 11%.

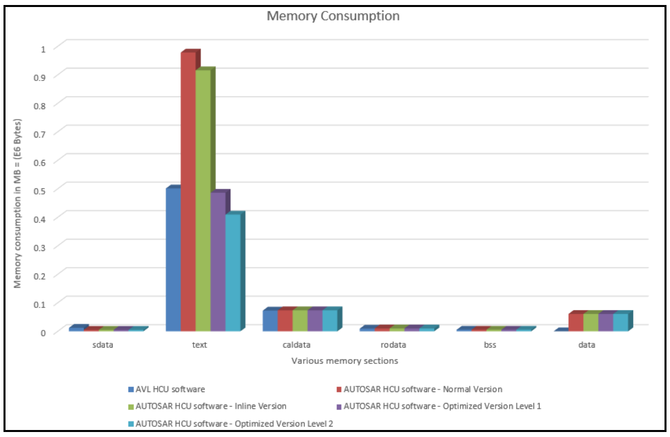


Figure 8. Analysis of memory consumption [5].

Table 1. Percentage Change in Average Execution Time

	Average Task Runtime (ms)	**Percentage Change**
Reference SW	2.483	0
AUTOSAR SW - Normal	6.384	+163%
AUTOSAR SW - Inline	5.347	+120%
AUTOSAR - Optimized Version L1	4.177	+72%
AUTOSAR - Optimized Version L2	2.698	+11%

VII. DISCUSSION

The AUTOSAR conversion concept has been established for the legacy software. The ASW of the legacy software has been converted to AUTOSAR format and integrated with the BSW via the RTE generation. Although the proposed conversion concept incorporates AUTOSAR features, an ICC3 level AUTOSAR compliance was not achievable due to the following factors:

- The usage of non-AUTOSAR BSW for AUTOSAR conversion.
- The presence of CIL and the supplier-specific interfaces from the BSW.

It is imperative to use the AUTOSAR specific BSW stacks and follow the AUTOSAR proposed development methodology from scratch to achieve ICC3 level compliance. Nevertheless, the proposed conversion approach could meet the ICC1 level considering the BSW, the CIL, and the RTE as a single proprietary BSW unit. The proposed conversion concept additionally cannot be applied to all types of legacy software. As mentioned by Daehyun et al. [2], in some cases, it is necessary to decompose the legacy software into different AUTOSAR SWC types before conversion. Considering the software performance, the original non-AUTOSAR legacy software outperformed the AUTOSAR versions in the memory consumption and task runtime metrics. The above observation is in accordance with the study in [2], which mentions that the incorporation of AUTOSAR concepts can bring out an increase in the code size and execution time. This is due to the presence of the RTE function definitions and the function call overheads introduced during software execution. However, it was also shown in this work that the additional function call overheads and the increase in the code size can be compensated by introducing various optimization options in the AUTOSAR converted software.

ACKNOWLEDGMENT

This work is carried out as a master's thesis project in collaboration with AVL List GmbH, Graz, Austria, and Institute of Technical Informatics, Graz University of Technology. We would like to express our gratitude to the Dept. of Powertrain Engineering, AVL List GmbH. for sponsoring this work.

REFERENCES

[1] Nicolas Navet and Francoise Simonot-Lion, Automotive Embedded Systems Handbook, CRC Press, Taylor & Francis, no. of pages 488, 2008.

[2] Daehyun Kum, Gwang-Min Park, Seonghun Lee andWooyoung Jung, " AUTOSAR Migration from Existing Automotive Software," in International Conference on Control, Automation and Systems 2008, Seoul, Korea, pp. 558-562, 2008

[3] James Joy, Anush G Nair," Automation framework for converting legacy application to AUTOSAR System using dSPACE SystemDesk," in dSPACE User Conference 2012 - India, September 14, 2012.

[4] AUTOSAR Classic Release 4.3.1, AUTOSAR Classic Platform, 2019.

[5] Vimal Sivashanmugam," Conversion of Control Unit Software towards AUTOSAR Compliance," Master's thesis in Information and Computer Engineering, Graz University of Technology, 2019.

[6] Shiquan Piao, Hyunchul Jo, Sungho Jin, and Wooyoung Jung," Design and Implementation of RTE Generator for Automotive Embedded Software," in 2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications, pp. 159-165, 2009.

[7] Sharang Kulkarni, Prof. Shafali Gupta, Rameez Tamboli, Anil Dake," Review of Techniques for Making Efficient Executable in GCC Compiler," in Imperial Journal of Interdisciplinary Research (IJIR), Vol-3, Issue-3, 2017.

[8] " ADD V19.1 Product Datasheet," VisuIT, 2019. [Online]. Available: https://www.visuit.de/vitdata/Download/pub/10_ADD/ADD_Info/Whats%20new%20in%2019.1.pdf [Accessed October 2, 2020]

[9] Hightec Compiler Suite Product Information, HighTec EDVSysteme GmbH, 2019. [Online]. Available: https://hightecrt.com/en/products/development-platform.html [Accessed May 24, 2020].

[10] VeriStand Product Information, National Instruments, 2019. [Online]. Available: http://www.ni.com/veristand/ [Accessed June 06, 2020].