Author/s:
Grubmüller, S;Stettinger, G;Nešić, D;Watzenig, D

Title:
Concepts for improved availability and computational power in automated driving

Date:
2018-08-01

Citation:
Grubmüller, S., Stettinger, G., Nešić, D. & Watzenig, D. (2018). Concepts for improved availability and computational power in automated driving. Elektrotechnik Und Informationstechnik, 135 (4-5), pp.316-321. https://doi.org/10.1007/s00502-018-0625-4.

Persistent Link:
https://hdl.handle.net/11343/297926

Noname manuscript No.
(will be inserted by the editor)

Concepts for improved availability and computational power in automated driving

Stephanie Grubmüller · Georg Stettinger · Dragan Nešić · Daniel Watzenig

Received: date / Accepted: date

Abstract Automated vehicles are required to operate on highways and in complex urban scenarios. To handle these complex environmental influences, sophisticated automated driving functions demand a high computational power. Multi-core platforms are deployed to cope with this demand. Key enablers of fully automated vehicles are the availability of driving functions in combination with increased computing power. To achieve higher system availability for SAE level 3 and higher, fail operational concepts from system level to Microcontroller Unit (MCU) level are needed. These concepts include hardware as well as software requirements. For an increased computing performance, a parallel computation method for driving functions and their control algorithms is introduced. First a stabilizing controller is designed following the emulation design method. Afterwards, the controller is implemented on different cores of the multi-core processor. Finally, this resulting closed loop system is modeled as a hybrid system for stability analysis purposes.

Keywords fail-operational · parallel computation · waveform relaxation · multi-core processor

S. Grubmüller · G. Stettinger
Virtual Vehicle Research Center, Inffeldgasse 21a, Graz, Austria
E-mail: Stephanie.Grubmueller@v2c2.at

D. Nešić
Department of Electrical and Electronic Engineering, The University of Melbourne, Parkville, 3010 VIC, Australia

D. Watzenig
Institute of Automation and Control, University of Technology Graz, Inffeldgasse 21b, Graz, Austria

1 Introduction

Automated driving will significantly influence our future mobility to make it cleaner, more comfortable and safer. According to [7], it is assumed that vehicles are fully automated by 2050. To achieve a full automation of vehicles, the authors in [21] identified a number of research challenges, two of them are tackled in this paper: 1) availability, reliability and robustness and 2) constraint computing power.

In terms of end-user acceptance and passenger safety, a high availability [3], reliability and robustness [21] of (highly) automated systems even under adverse conditions is mandatory. To achieve these requirements, errors need to be detected entirely as well as properly handled and a redundancy of all involved components has to be provided. This leads to a fail-operational behavior, where the automated driving function operates even when faults occur [11]. At the moment, fail-operational concepts exist for X-by-wire systems (e.g. [17]) but not for automated driving functions and their active safety systems, which we discuss here.

For a successful implementation of automated vehicles three building blocks are mandatory [21]: i) Sensing, ii) processing and decision making and iii) vehicle control. Based on this functional architecture, an autonomous driving function represents a control system. Such control functions consists of complex algorithms, rich software component interfaces and has to process amounts of data [13]. Thus, the higher the level of automation, the more computational power is required by the control function implemented on an Electronic Control Unit (ECU). A conservative implementation approach is the one function per ECU, which leads to a decentralized architecture and a high number of ECUs [15]. Consequently, the vehicle's power

consumption will significantly increase. To solve this problem, multi-core ECUs are considered to replace single-core platforms in automotive domains [15]. The deployment of multi-core ECUs will improve the computing performance by parallel computation of different driving functions [13]. To further increase the computing performance, a distribution of the computing workload of one control function on a number of processor cores can be implemented. This idea is inspired by software development, where one task is partitioned into smaller sub-tasks and allocated to different computing resources [2, 15].

In this work we consider dynamic controllers. A decomposition of the controller state-space model and parallel computation may lead to instabilities of the closed-loop system. Thus, to address this issue an appropriate analysis method is needed. In this paper we present a first modeling approach for parallel computation of controllers which is the basis for further stability analysis.

First we discuss the fail-operational concepts in Section 2. In Section 3 parallel computing and the modeling framework is presented. Finally, the future work and conclusions are given.

2 Fail-operability in automated driving

This section presents fail-operational strategies on hardware and software level and shows the importance of fault-tolerant systems in automated driving, especially for active safety systems.

Different levels of fault tolerance exist that range from a high fault tolerance to a safe state. A system is called fail-operational, when after one internal component failure the system still operates with full or degraded functionality until the system enters a safe state [11]. For example, if an error is detected and the vehicle is brought to a safe standstill by the remaining active components, then the automated vehicle shows fail-operational behavior. Different fault-tolerant concepts for hardware, such as sensors, actuators, ECUs etc., exist and are given by different structures as presented in [11]. One is the static redundancy structure, where the system is implemented redundantly including a majority voter. In Figure 1a) is shown a triple modular redundancy, where the 2-out-of-3 voting allows a single point of failure. This concept leads to a high number of components and increases the weight of the vehicle.

A dynamic redundancy decreases the amount of systems, but a raise in data processing may be expected. A dynamic redundancy system consists of a primary and backup system. In case of a failure detected by the

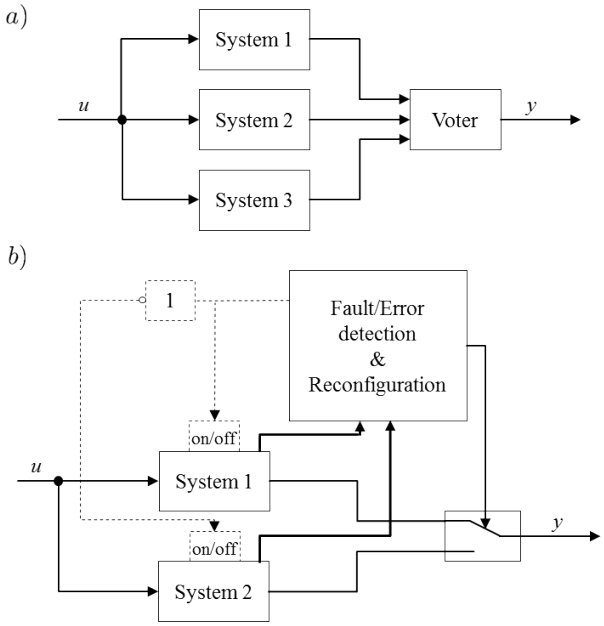

Fig. 1 a) Triple modular redundancy (2-out-of-3 voting); b) Dynamic redundancy with hot standby (solid lines) and cold standby (including dashed lines) according to [11]

fault detection module, the primary system goes offline and the backup system takes over control. According to [11], faults are either detected by model-based or signal-based methods. In Figure 1b) two dynamic redundancy structures for a duplex system are presented. In case the backup and primary system are active all the time (solid lines), the backup system which may have a degraded functionality is a hot standby. Therefore, the backup is always ready to take over but the wear out rate is higher. Is the backup system deactivated but active if the primary system fails (dashed lines), then it works as a cold standby, where the reconfiguration may take some time. For example, for a fail-operational 360° environmental awareness designed for nearly all weather conditions, the field of view of the sensors is needed to be redundant. On the other hand, for actuators, such as braking systems, duo duplex systems consisting of two duplex systems with fault detection are appropriate.

When we consider automated driving functions, the control function is implemented on an ECU. In fault-tolerant control systems, failures are detected in sensors, actuators and the plant which lead to a reconfiguration of the controller [18]. But the controller itself is not fail-operational. To reach a fail-operational state of the controller, the ECU and the controller implementation need to meet a number of requirements. The authors in [12] differentiate between fail-operational concepts on ECU and on MCU level, where the MCU represents the central component of the ECU. On ECU level

the following concepts are presented: (i) Triple modular redundancy. (ii) Symmetric two-out-of-two diagnosis fail safe (2oo2DFS), where the system is designed either as fully redundant system or as a system consisting of a primary system with a backup system. Both systems are fail-safe, detect their own errors and monitor the other system. (iii) Asymmetric 2oo2DFS: where the architecture works as a master-slave strategy. One ECU runs the full functionality of the software and the backup system a reduced version of it.

Depending on the performance, the MCU host critical, safety-related and comfort functions. For a fail-operational concept, the MCU needs to provide fault tolerance, fault containment and fault diagnosis. A multi-core MCU presented by [12] meets these requirements.

But not only the hardware also the software need to be redundant. Software redundancy concepts cover fault-tolerance on bit and code level. On bit level, checksums detect errors in data (e.g. bit flips) [19]. On code level different redundancy strategies exist. One static redundancy is the repeated execution of code segments [19]. The most popular static redundancy concept is N-version programming. The algorithms are developed and implemented independently, provide the same functionality, run in parallel and the output is provided by a majority voting [10]. The recovery block concept is a dynamic redundancy structure. The safety critical code has got alternative implementations which are executed subsequently. If an error in a code is detected, the previous states will be restored and an alternative will be executed [10].

3 Parallelism of control functions

According to [2], parallel computing is defined as the concurrent usage of a number of processors to solve one computational problem. Therefore, the problem is decomposed in chunks of work and then allocated to several computing resources. The processor communication architecture, the partitioning and finally the modeling of such problems is described in this section.

3.1 Processing structure and essentials

A parallel computing system consists of several processors (cores) which are located in small distance of each other and therefore the exchange of data between them is reliable and predictable [5]. Different processor interconnection architectures exist. One possibility is to exchange data via a shared memory. Another option is a message-passing architecture, where the network can either consist of directly connected links or of a shared bus. A combination of both architectures is also possible and called a hybrid architecture. In our work we assume to have a simplified architecture of synchronized processor cores as shown in Figure 2. Each core is directly connected to a buffer which is directly connected to each buffer of the other cores.

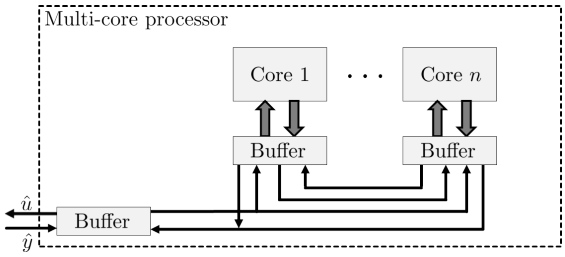

Fig. 2 Processor communication architecture

After the architecture is identified, the control function can be decomposed. The author of [2] distinguishes between domain and functional decomposition. In domain decomposition, the data set manipulated by the problem is decomposed and every sub-task works on one sub-data set. On the other hand, in functional decomposition the computational workload is partitioned into less computing resource consuming sub-tasks. Due to the structure of the controller state space model and the dependencies between the states herein, we concentrate on functional decomposition.

Due to a poor decomposition, an unbalanced workload may lead to a decreased performance. Therefore it is recommended to equally, as far as possible, split up the workload between processors to avoid long delay times and minimize the inter-processor communication [1].

To ensure that for every task a time interval for computation is allocated, a task scheduling on multi-core level is needed. Static scheduling policies where the execution order is determined off-line and unchanged during run-time (e.g. Round Robin) are used in real-time systems, due to their predictability. But dynamic schedulers, where the execution order is defined on-line are under research currently [15]. In [8] some parameters are identified important for the scheduling and shown in Figure 3. The time instant t_{r_i} indicates the release time

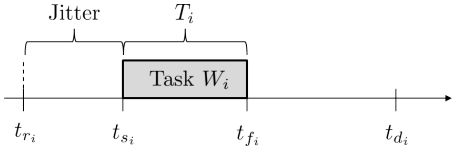

Fig. 3 Parameters for task scheduling according to [8]

of the task W_i. After some Jitter, W_i actually starts at t_{s_i} and ends at t_{f_i}. The computing time T_i is the estimate of the worst case execution time. In hard real-time systems, such as control systems, W_i must be finished by its deadline t_{d_i} and its execution is repeated periodically [6]. The continuous time signals to and from the plant, such as the control input and the plant output, are sampled periodically with a sampling period T_s and then held constant between sampling time instants [6].

3.2 Modeling and system description

For the modeling of the parallel computation of controllers we use the idea of the emulation design as in Networked Control Systems (NCS) [20]. First a stabilizing controller for the plant is designed while the effects of parallel computation are ignored. The continuous time control system is shown in Figure 4a). Then the controller is distributed over several processors as shown in Figure 4b). The continuous time plant and a

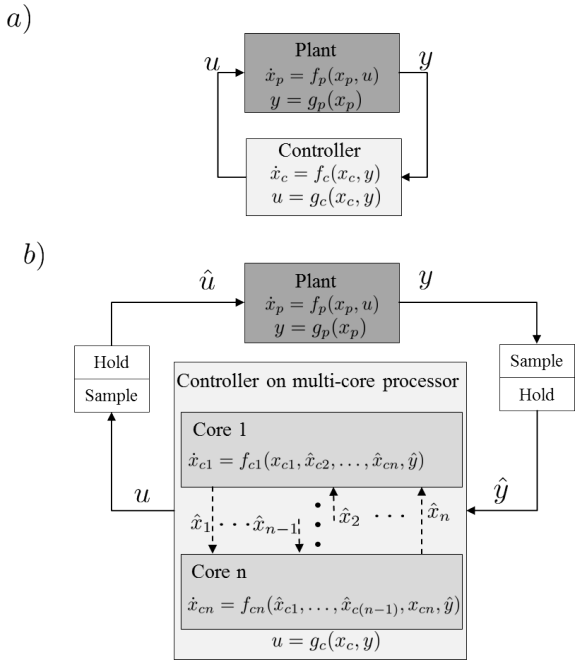

Fig. 4 a) Closed-loop control system; b) Closed-loop control system including a parallel computed controller

stabilizing controller is given by:

$$\dot{x}_p = f_p(x_p, u), \qquad y = g_p(x_p), \tag{1}$$

$$\dot{x}_c = f_c(x_c, y), \qquad u = g_c(x_c, y), \tag{2}$$

where the plant state is $x_p \in \mathbb{R}^{n_p}$, the controller state is $x_c \in \mathbb{R}^{n_c}$, the control input is $u \in \mathbb{R}^{n_u}$ and the plant output is $y \in \mathbb{R}^{n_y}$, where the dimensions n_c, n_u, n_y and n_p are positive real numbers. The functions f_p, g_p, f_c and g_c are assumed to be continuous and sufficiently smooth.

For our purpose the controller is decomposed into n subsystems $x_c = [x_{c1}^T, \ldots, x_{cn}^T]^T$ and allocated to n different processor cores, where $\hat{u}$, $\hat{y}$ and $\hat{x}_{ci}$ indicate buffers which operate as Zero-Order-Hold (ZOH) equivalents and satisfies $\dot{\hat{y}} = 0$, $\dot{\hat{u}} = 0$ and $\dot{\hat{x}}_{ci} = 0$ between update time instants. u and y are periodically sampled every T_s and subsequently stored in $\hat{u}$ and $\hat{y}$; $\hat{x}_{ci}$ are the values of states exchanged with other processors at certain time instants. This leads to the following system

$$\begin{aligned}
\dot{x}_p &= f_p(x_p, \hat{u}) \\
y &= g_p(x_p) \\
\dot{x}_{c1} &= f_{c,1}(x_{c1}, \hat{x}_{c2}, \ldots, \hat{x}_{cn}, \hat{y}) \\
\dot{x}_{cn} &= f_{c,n}(\hat{x}_{c1}, \ldots, \hat{x}_{c(n-1)}, x_{cn}, \hat{y}) \\
u &= g_c(x_{c1}, \ldots, x_{cn}, \hat{y}).
\end{aligned} \tag{3}$$

The interval T is referred to as one computational interval and T_s to the computing cycle. The time sequence t_k is defined by $t_0 < t_1 < \ldots$ with $T = t_{k+1} - t_k$ and satisfies $T > 0$. T_s is discretized in N_{stages} time instants with $N_{stages} \in \mathbb{Z}_{>0}$ and $T_s = t_{k+Nstages} - t_k$.

3.2.1 Task scheduling

The next step is the choice of an appropriate task scheduling. Here a static scheduling as Round Robin [20] is considered. The schedule policy itself is inspired by methods of parallel numerical computation of Ordinary Differential Equations (ODEs), especially by the parallelism across the system [4]. It is a way of decoupling the system's state equations to allow a parallel numerical integration [9]. Several methods were introduced over the years. Important for our work is the Waveform Relaxation (WR), which was first mentioned in [14]. It iteratively solves the ODEs based on the iterative Picard method. The system of ODEs are decomposed into coupled subsystems. Each subsystem is solved by its own numerical integration method. The solution of each subsystem for one iteration is called wave. After one iteration the waves are exchanged and the solver starts again until the numerical solution converges to the unique solution [4].

Two iterative methods are distinguishable: Jacobi WR and Gauss-Seidel WR [5]. These two methods differ in the way how and when the waves of the subsystems are exchanged. The Jacobi WR is similar to a parallel computing scheme, where waves generated an iteration before are used for integration. Therefore, waves at specified time instants are exchanged and need to be stored for the whole run-time or simulation time.

On the other hand, the Gauss-Seidel WR behaves like an sequential computing scheme where waves are sequentially generated and exchanged. To use the WR, multiple buffers for all controller states and specified exchange time instants would be necessary.

Hence, the scheduling policies used here are motivated by Jacobi WR and Gauss-Seidel WR. First we assume that after one integration the waves converge to the unique solutions and numerical integration effects can be ignored. Furthermore, the buffers $\hat{x}_{ci}$ contain the values of x_{ci} at the latest exchange time instant.

The resulting idea of the task scheduling for a test system with controller states x_{c1} and x_{c2} is depicted in Figure 5.

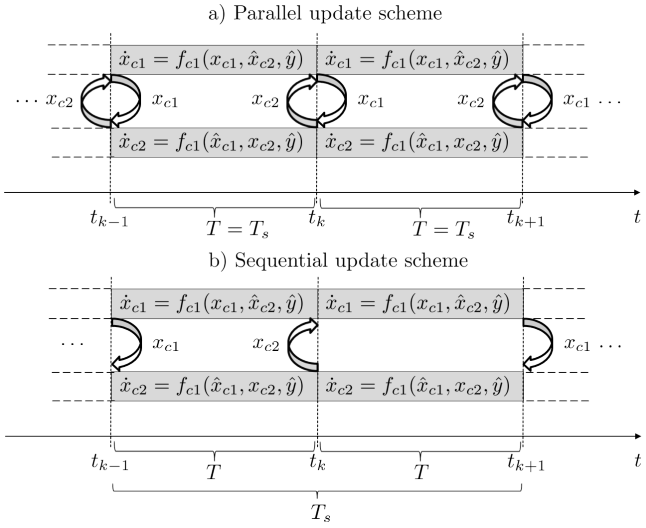

Fig. 5 a) Parallel task scheduling policies for controller

The resulting parallel computing scheme shown by Figure 5a) is inspired by the Jacobi WR, where the latest values of x_{c1} and x_{c2} are exchanged simultaneously after every computational period T and instantaneously stored in the buffers $\hat{x}_{c1}$ and $\hat{x}_{c2}$. The processor cores are busy with computing all the time. It is assumed that read and write actions on the buffers are sufficiently fast, so that the caused delays can be ignored. After every $T_s = T$, u and y are sampled and stored in the buffers $\hat{u}$ and $\hat{y}$.

The sequential computing scheme given in Figure 5b) is inspired by the Gauss-Seidel WR. Within one computing period T_s, $\hat{x}_{c1}$ and $\hat{x}_{c2}$ are updated sequentially. This leads to the definition of computational nodes. If the subsystems update their buffers in parallel, then they are assigned to the same computational node and $N_{stages} = 1$. Otherwise, the subsystems are assigned to different computational nodes and $N_{stages} = n$, where n is the number of subsystems.

Consequently, each scheduling policy discussed before can be expressed similar to the update function defined in [20]. Based on the considerations before, an impulsive system similar to that one presented in [16] can be introduced.

4 Conclusions

Two research challenges in automated driving are tackled in this paper: Fail-operability and parallelism of the driving functions.

Fail-operational concepts from system level to component level are discussed. These concepts include software as well as hardware. Fail-operability is a main driver for user acceptance of highly automated driving functions due to the increase in availability, reliability and robustness.

To increase the computational performance and decrease the number of ECUs in a vehicle, a method for parallel computation of the driving functions and its inherent modeling approach was presented here. The method and the model depend on the processors architecture, the decomposition of the workload and the scheduling of the tasks. For a first consideration, a simplified processor and communication architecture is assumed. At this stage of the work, a static scheduling is considered, where the scheduling order relies on specific realisations of WR. The main outcome is an impulsive model of the the closed-loop system including a parallel computed controller. That model can be considered in further stability analysis to find an upper bound on T which guarantees stability.

5 Future work

Currently we are working on a specification for a fail-operational active safety systems for SAE level 3 and higher. Our specific use case is an automated emergency brake assistant that works under adverse weather conditions. It includes the fail-operational field of view of the sensors, sensor fusion and decision making as well as fail-operational actuators. The goal is to demonstrate a fail-operational behavior of the active safety system under real road conditions.

The second part of the future work involves the parallel computing of control functions. The decomposition and allocation of the controller may lead to instabilities. Therefore a stability analysis is under investigation at the moment. It will provide a maximum allowable computational interval for each task so that the closed-loop system is uniformly asymptotically stable. It is further planned to integrate idle states of the processors in the

model which will also cause an extension of the stability analysis. Besides, only static scheduling is considered here. It would be beneficial to investigate the deployment of dynamic scheduling in such parallel control systems.

Acknowledgements This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737469 (AutoDrive Project). This Joint Undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Germany, Austria, Spain, Italy, Latvia, Belgium, Netherlands, Sweden, Finland, Lithuania, Czech Republic, Romania, Norway. In Austria the project was also funded by the program IKT der Zukunft and the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit). The publication was written at VIRTUAL VEHICLE Research Center in Graz and partially funded by the COMET K2 Competence Centers for Excellent Technologies Programme of the Federal Ministry for Transport, Innovation and Technology (bmvit), the Federal Ministry for Digital, Business and Enterprise (bmdw), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).
D. Nešić work was supported under the Australian Research Council under the Discovery Project DP170104099.

References

1. Alt, A., Wilsey, P.A.: Profile driven partitioning of parallel simulation models. In: Proceedings of the Winter Simulation Conference 2014, pp. 2750–2761 (2014). DOI 10.1109/WSC.2014.7020118
2. Barney, B.: Introduction to parallel computing. (2017). URL https://computing.llnl.gov/tutorials/-parallel_comp/
3. Becker, J., Helmle, M., Pink, O.: System Architecture and Safety Requirements for Automated Driving, pp. 3–16. Springer International Publishing, Cham (2017). DOI 10.1007/978-3-319-31895-0_11. URL https://doi.org/10.1007/978-3-319-31895-0_11
4. Ben Khaled, A., Ben Gaïd, M.E.M., Pernet, N., Simon, D.: Fast multi-core co-simulation of Cyber-Physical Systems : application to internal combustion engines. Simulation Modelling Practice and Theory **47**(September), 79–91 (2014). DOI 10.1016/j.simpat.2014.05.002. URL https://hal-ifp.archives-ouvertes.fr/hal-01018348
5. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and distributed computation: Numerical methods. Athena Scientific, Belmont (1997)
6. Derler, P., Lee, E.A., Toerngren, M., Tripakis, S.: Cyber-physical system design contracts. In: Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on, pp. 109–118
7. European Road Transport Research Advisory Council (ERTRAC): Automated Driving Roadmap (2017)
8. Feki, A.K.E.: Distributed real-time simulation of numerical models: application to power-train. Ph.D. thesis, Automatic, Universite Grenoble Alpes (2014)
9. Gear, C.W.: Massive parallelism across space in odes. Applied Numerical Mathematics **11**(1), 27 – 43 (1993). DOI http://dx.doi.org/10.1016/0168-9274(93)90038-S. URL http://www.sciencedirect.com/science/article/pii/016892749390038S
10. Hecht, H.: Fault-tolerant software. IEEE Transactions on Reliability **R-28**(3), 227–232 (1979). DOI 10.1109/TR.1979.5220573
11. Isermann, R., Schwarz, R., Stolzl, S.: Fault-tolerant drive-by-wire systems. IEEE Control Systems **22**(5), 64–81 (2002). DOI 10.1109/MCS.2002.1035218
12. Kohn, A., Schneider, R., Vilela, A., Roger, A., Dannebaum, U.: Architectural concepts for fail-operational automotive systems. In: SAE 2016 World Congress and Exhibition. SAE International (2016). DOI https://doi.org/10.4271/2016-01-0131. URL https://doi.org/10.4271/2016-01-0131
13. Leitner, A., Ochs, T., Bulwahn, L., Watzenig, D.: Open Dependable Power Computing Platform for Automated Driving, pp. 3–16. Springer International Publishing, Cham (2017). DOI 10.1007/978-3-319-31895-0_14. URL https://doi.org/10.1007/978-3-319-31895-0_14
14. Lelarasmee, E.: The waveform relaxation method for time domain analysis of large scale integrated circuits: Theory and applications. Ph.D. thesis, EECS Department, University of California, Berkeley (1982). URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/1982/9614.html
15. Monot, A., Navet, N., Bavoux, B., Simonot-Lion, F.: Multisource software on multicore automotive ecus - combining runnable sequencing with task scheduling. IEEE Transactions on Industrial Electronics **59**(10), 3934–3942 (2012). DOI 10.1109/TIE.2012.2185913
16. Nešić, D., Teel, A.R.: Input-output stability properties of networked control systems. IEEE Transactions on Automatic Control **49**(10), 1650–1667 (2004). DOI 10.1109/TAC.2004.835360
17. Sinha, P.: Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives. Reliability Engineering & System Safety **96**(10), 1349 – 1359 (2011). DOI https://doi.org/10.1016/j.ress.2011.03.013. URL http://www.sciencedirect.com/science/article/pii/S095183201100041X
18. Teixeira, A.: Toward cyber-secure and resilient networked control systems. Ph.D. thesis, KTH Royal Institute of Technology, School of Electrical Engineering, Department of Automatic Control, Stockholm, Schweden (2014)
19. Ulbrich, P.M.: Ganzheitliche fehlertoleranz in eingebetteten softwaresystemen. Ph.D. thesis, Technische Fakultt der Friedrich-Alexander-Universitt Erlangen-Nrnberg (2014)
20. Walsh, G.C., Ye, H., Bushnell, L.G.: Stability analysis of networked control systems. IEEE Transactions on Control Systems Technology **10**(3), 438–446 (2002). DOI 10.1109/87.998034
21. Watzenig, D., Horn, M.: Introduction to Automated Driving, pp. 3–16. Springer International Publishing, Cham (2017). DOI 10.1007/978-3-319-31895-0_1. URL https://doi.org/10.1007/978-3-319-31895-0_1