文章编号：0253-374X(2022)S1-0151-05

DOI：10.11908/j.issn.0253-374x.23717

基于 MPSoC 平台的自动驾驶车辆故障运行控制

Dennis NIEDBALLA，Hans-Christian REUSS

（斯图加特大学 汽车工程学院(IFS)，斯图加特 70569，德国）

摘要：在德国联邦研发项目UNICARagil中，开发了一个ECU，作为UNICARagil新颖的大脑架构的一部分。应用所谓的脑干部件，通过面向服务的架构，对全自动驾驶车辆进行安全关键的实时控制。本文介绍了该设备的硬件架构和基本软件架构，并列举了一些安全措施。重点是MPSoC和OS配置、ECU的更新过程和启动过程。回退实例系统可以保护它免受引导加载程序、内核或根文件系统以及开发过程中和应用现场中的严重错误的影响。最后，列举了设备在车辆环境中的功能。在众多的科学内容出版物和社交媒体上，笔者展示了以使用脑干为核心部件的车辆通过试验场的测试情况。

关键词：MPSoC；故障运行；嵌入式实时系统；UNICARagil；PetaLinux

中图分类号：U461　　**文献标志码**：A

MPSoC-Based Platform for Fail-Operational Control of an Automated Research Vehicle

Dennis NIEDBALLA，*Hans-Christian REUSS*

(Institute of Automotive Engineering(IFS)，University of Stuttgart，70569 Stuttgart，Germany)

Abstract：In the project UNICARagil，an ECU was developed that is part of the novel UNICARagil brain architecture[1]. The so-called brainstem was dedicated for safety-critical real-time control of a fully automated vehicle via a service-oriented architecture. This paper describes the hardware and basic software architecture of the device and points out some of the safety measures. The focus is on the MPSoC and OS configuration，the update process of the ECU，and the boot process. A system of fallback instances protects the ECU against critical errors in the bootloader，kernel or root file system，as well during the development process as in the field. Finally，this paper points out the functionality of the device in the vehicle context. In numerous publications in science and on social media，it presents the vehicle going over the proving ground with the brainstem as a central component.

Key words：MPSoC；fail-operational；embedded real-time system；automotive；UNICARagil；PetaLinux

The automotive domain today faces great challenges of which automated driving，electric drive and novel E/E-architectures are just some examples. To take a step to the future and unify all these technological trends into experimental real-life vehicles we started the UNICARagil project. There the Institute of Automotive Engineering (IFS) of the University of Stuttgart developed the fail-operational real-time ECU “brainstem” which belongs to the novel brain-based E/E-architecture of our vehicles [1]. Fig. 1 shows the latest version of the brainstem with opened cover. It presents the two redundant instances，the power and communication interfaces and the maintenance cable.

In this paper we will describe its hardware and software structure and make a deep dive into the boot and update process，as it gives a good insight into the technological structure of this system.

1　Hardware architecture

The UNICARagil brainstem is a fail-operational embedded real-time system with two redundant

收稿日期：2022-08-15
基金项目：德国联邦教育和研究部资助项目(FKZ 16EMO0289)
第一作者：Dennis Niedballa(1986—)，男，工程学硕士，主要研究方向为故障操作实时嵌入式系统。
E-mail：dennis.niedballa@ifs.uni-stuttgart.de
通信作者：Hans-Christian REUSS (1959—)，男，教授，工程学博士，主要研究方向为汽车机电控制技术。
E-mail：hans-christian.reuss@ifs.uni-stuttgart.de

Fig. 1 Brainstem hardware V5.0

internal instances. The main application is trajectory control[2] and a safe stop functionality as a fallback system in case of an error in the main vehicle control system[3-4]. Fig. 2 shows the hardware structure of one brainstem instance based on a Zynq Ultrascale+ ZU3EG with a quad-core ARM Cortex A53 application processing unit (APU) and a dual-core ARM Cortex R5 real time processing unit (RPU) with lockstep functionality. A dual 5–60 V wide input power supply guarantees full functionality even in cases of extreme power instability. Both the APU and the RPU possess an own Gbit Ethernet interface for full access to the whole vehicle network. The brainstem can additionally access peripheral units via two CAN interfaces.

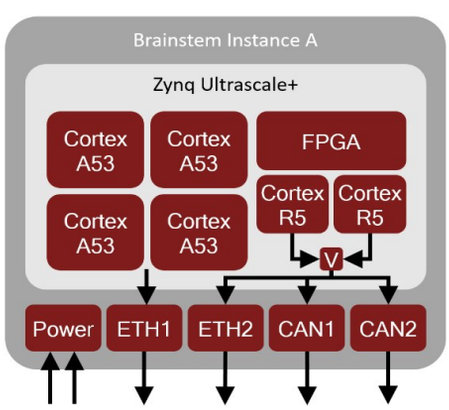

Fig.2 Hardware structure of one brainstem instance

2 Software architecture and toolchain

Fig. 3 describes the toolchain that is required for Xilinx MPSoC development. The workflow starts with the MPSoC hardware configuration. The system enables flexible pin configuration and activation of interfaces like CAN and SPI. The integrated programmable logic (PL) supports the implementation of ultra-fast low-level functionality like hardware support for communication protocols. The Xilinx tool Vivado offers pre-implemented IP-cores for the PL, e. g. for integration of a second GbE MAC on the chip.

The APU software is based on an embedded Linux with PREEMPT_RT kernel. The command line tool "PetaLinux Tools" generates complete embedded Linux boot images, containing bootloader, kernel, device tree and root file system. As a next software layer the Automotive Service Oriented Architecture (ASOA) serves as a middleware for the equal communication between all ASOA services in the vehicle, independently if they are located on the same ECU or not. The application software on the APU, especially the ASOA services, can be developed with general purpose code editors like Visual Studio Code. Examples for ASOA services on the APU are a vehicle boot and shutdown service, a driving corridor monitoring (DCM), a platform sensor adapter (PSA) and the trajectory preprocessing (TP) services[3-4]. Additionally the APU hosts basic software like the ASOA orchestrator, PTP and system management tools for update, boot & shutdown of the brainstem itself.

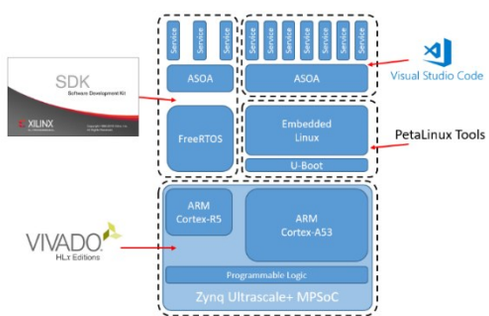

Fig.3 Toolchain for brainstem development

The RPU is the host for our most safety and time critical application: the vehicle trajectory control. It is also the ideal place for a system monitoring service. The applications on the RPU are, like all software applications in the UNICARagil vehicles, implemented as ASOA services. The corresponding ASOA middleware for the RPU was

implemented on the FreeRTOS real-time OS[5]. The RPU has its own GbE network interface with PTP time synchronization. For software development on the RPU the Xilinx SDK (XSDK) is required.

3 Memory partitioning

The brainstem has two persistent flash memories: a QSPI flash for the boot loader and the kernel and an eMMC flash for the root file system.

The QSPI flash has a physical size of 128 MB of which 78.5 MB are used, separated into 6 partitions. Tab. 1 shows the partitioning of the QSPI flash memory. The first 3 partitions are dedicated to the bootloader, the boot environment and the embedded Linux kernel. Each of those partitions has a redundant partition for robustness reasons. The bootloader and the kernel are protected by two different redundancy concepts: backup partitioning and A/B symmetry. The Bootloader and its boot environment each have one backup partition. For technical reasons the active bootloader always needs to be placed in the first partition of the QSPI flash, but a backup can be copied from another partition in the case of faults. The Kernel partitions implement an A/B symmetry: both partitions are equivalent, one of them is active and the other one inactive. We can switch the active partition as described in chapter 5. The QSPI flash is exclusively accessible for the bootloader, the embedded Linux system operates on the eMMC flash memory.

Tab.1　Partitioning of QSPI flash memory

Bootloader (BOOT.BIN)	Boot environment	Kernel A (image.ub)
7 MB	256 KB	32 MB
Bootl. backup (BOOT.BIN)	Boot env. backup	Kernel B (image.ub)
7 MB	256 KB	32 MB

Tab. 2 describes the partitioning of the eMMC flash memory. It has a physical size of 8 GB of which we use 7 GB, separated into 3 partitions. The first two partitions are reserved for the root file system, in an A/B symmetrical design: This allows the developers to update the complete file system in a resilient way.

When an update fails due to a structural error in the new file system, the bootloader automatically switches back to the last one. The third partition is meant for common data like log files[6].

Tab.2　Partitioning of eMMC flash memory

Rootfs A	Rootfs B	Log partition
3,3 GB	3,3 GB	400 MB

4 Update mechanism from embedded Linux side

We will now describe the boot and update process of the brainstem, because this example points out some of the characteristic features of its basic software structure. Fig. 4 shows the update process from the embedded Linux side. For every software type on the brainstem the update process is different since they intervene at different depths in the system. All update processes start with calling the update program from the embedded Linux shell. It downloads the new image files either from a local download server via SSH or from a remote GIT repository. After the files are on board, the brainstem places them in a specific directory, dependent on the image type.

The update functionality of the brainstem supports a one-step update of the complete root file system. For this purpose it downloads the corresponding image file from the server and copies it into the inactive partition on the eMMC flash (A or B). Subsequently the definition of the active and inactive root file system partition has to be saved into the boot environment. Finally, after a reboot the bootloader loads the new file system in resilient way, see chapter 5.

The RPU firmware automatically loads its image from the directory /lib/firmware. So the brainstem just has to download the boot image, stop RPU, copy the image to the directory and start RPU again. In UNICARagil embedded Linux applications such as services are managed via git. We reproduced relevant parts of the folder structure such as /usr/bin and /usr/lib in a repository, so the brainstem can clone its content directly to the root directory "/". All services

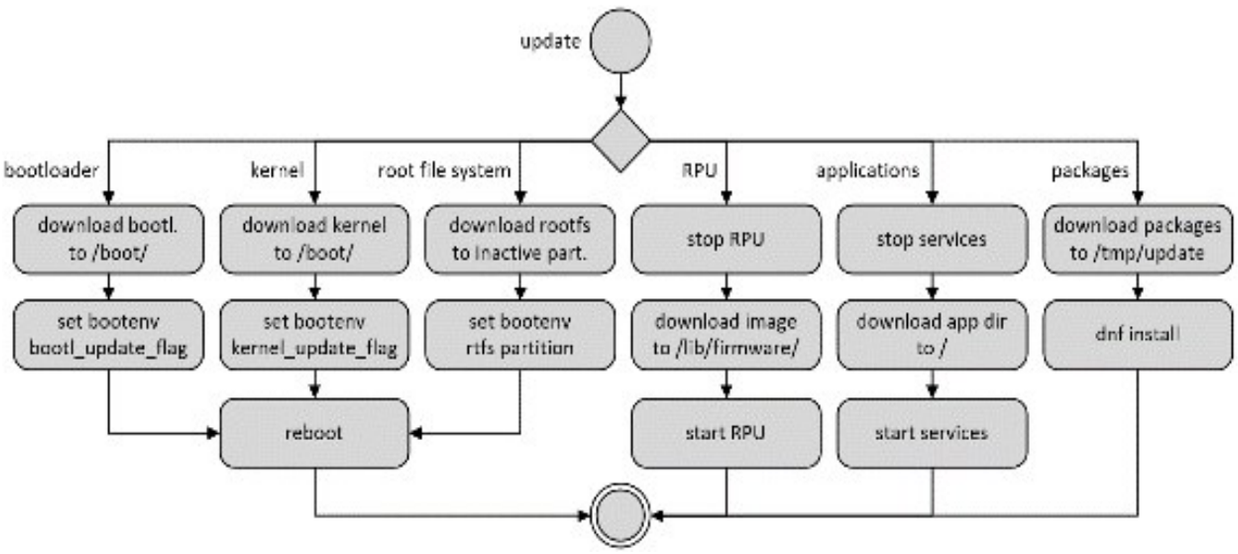

Fig.4 Update process from the embedded Linux side

have to be stopped during that process.

Our embedded Linux supports package management via the DNF tool, which is deeply integrated into the automatic update process to install, update and delete system packages in a defined manner.

The update processes of bootloader and kernel represent special cases, as they need access to the QSPI flash. This memory device is not directly accessible for embedded Linux - apart from discrete variables in the boot environment over the commands fw_setenv and fw_printenv. So first the update program places the corresponding images in the directory "/boot" in the root file system. Then it sets the corresponding bootloader or kernel update flag in boot environment and initializes a reboot. After the system reset the current bootloader performs the final installation of the image on the QSPI-Flash, which will be described in detail in the next chapter[6].

5 Resilient update support in boot process

The update of basic software components like bootloader, kernel or root file system is a critical process, as the installation of faulty images can put the whole ECU out of commission and make it hard to recover. So we implemented a resilient update support in the boot process of the brainstem, described in Fig. 5. After each boot the bootloader, kernel and rootfs are "approved", i. e. marked as bootable. If an image has already been booted but was not approved, the bootloader assumes a system crash, e. g. kernel panic, and reactivates the last working state. We implemented 3 different update strategies for bootloader, kernel and rootfs.

Bootloader: The bootloader copies itself and its boot environment to the backup partition, loads the new bootloader image from the /boot directory on the eMMC flash to the bootloader partition on the QSPI flash and performs a reset. The new bootloader is per default marked as not executed and not approved.

Kernel: The bootloader copies the kernel image from the /boot directory on the eMMC flash to the inactive kernel partition, switches the active partition, marks the image as not executed and not approved and performs a reset.

Rootfs: As the update of the rootfs is performed from the embedded Linux side, the bootloader just checks, whether it was approved after the last boot process and switches the active partition in case of a kernel panic.

As a last step of the bootloader process, the bootloader approves itself. So, if the boot or update process crashes, after a reboot the bootloader will load its own backup and make the system bootable again[6].

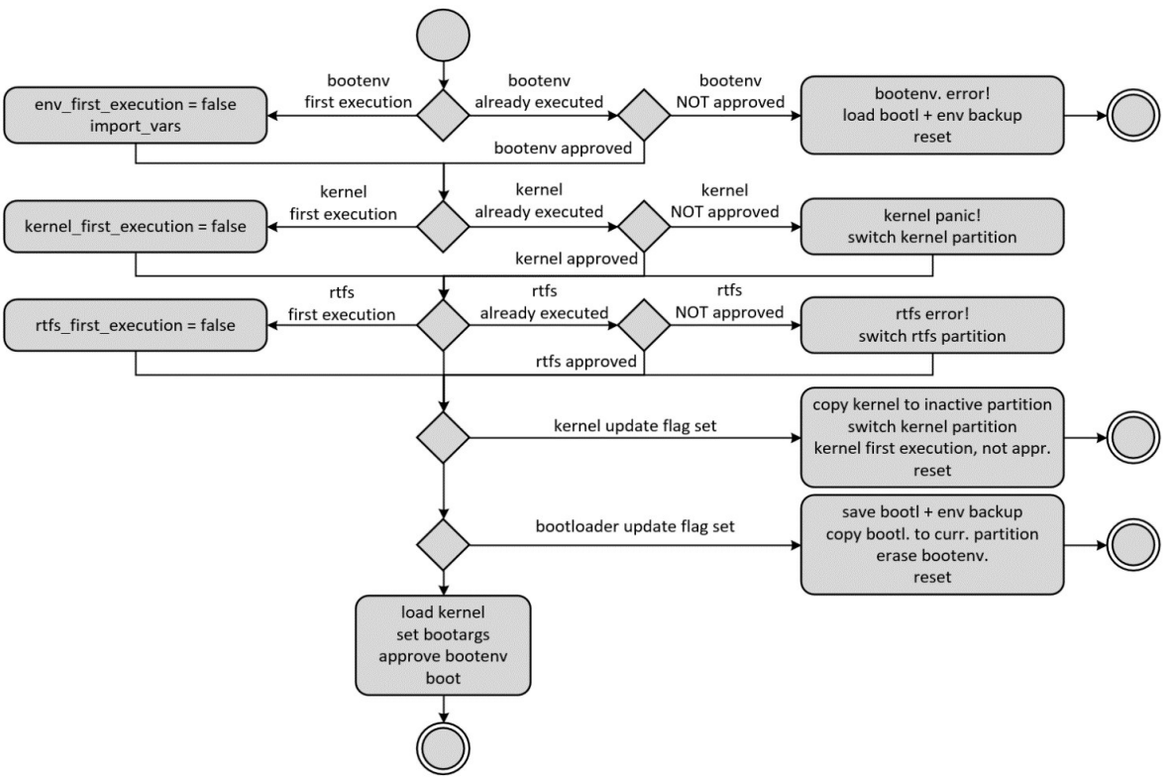

Fig.5　Bootloader process with robust update support

6　Test and conclusion

As the UNICARagil project is still going on we continue to develop the brainstem basic software and its applications. The resilient boot and update process supports us in our work and enables a fluent and agile development process as we can quickly and easily test new implementations and recover the last working state in case of mistakes. This allowed us finally to test the main functions of the brainstem like trajectory control in the real vehicle and prove our concept on the test route(see Fig. 6).

Fig.6　UNICARagil vehicle AUTOtaxi on test route[7]

The coming months will be characterized by the final commissioning of the four vehicles in the fully automated mode. We will demonstrate it on the test route in our final event on May 11, 2023 in Aachen.

Reference:

[1] KEILHOFF D, *et al*. UNICARagil—New architectures for disruptive vehicle concepts [C]//Proceedings of 19 Internationales Stuttgarter Symposium. Wiesbaden: Springer Vieweg, 2019.

[2] HOMOLLA T, *et al*. Verfahren zur korrektur von inkonsistenten Lokalisierungsdaten in modularen technischen Systemen [C]//13 Workshop Fahrerassistenzsysteme und Automatisiertes Fahren. Walting: TUbiblio, 2020.

[3] ACKERMANN S, *et al*. Modul und verfahren zur absicherung von solltrajektorien für automatisiertes fahren: Deutsche Patentanmeldung Anmeldenummer: 10 2019 125 401.9[S].2019.

[4] ACKERMANN S, WINNER H. Systemarchitektur und Fahrmanöver zum sicheren Anhalten modularer automatisierter Fahrzeuge [C]//13 Workshop Fahrerassistenzsysteme und Automatisiertes Fahren. Walting: TUbiblio, 2020.

[5] www.unicaragil.de.

[6] MOKHTARIAN A, et al. The dynamic service-oriented software architecture for the UNICARagil project[R]. Aachen: Universitätsbibliothek der RWTH Aachen, 2020.

[7] DECKER E. Optimierung des boot-und shutdownvorgangs in einem steuergerät für automatisierte fahrzeuge[D]. Stuttgart: IFS, University of Stuttgart, 2021.