PORSCHE

Guidelines for Creating the Software Architecture Documentation

Change History

Version	Date	Description	Author
5.1	14.04.2023	Adaption to Posix systems	Otmane El Azzati (translate only)
4.6 (EN)	12.12.2022	Formal adaptions	Michael Schmidt, Alkesh Solanki
4.6	21.07.2022	Chapter 8.6 „Timing Behaviour“ added	Michael Schmidt, Nils Asmussen
4.5 (EN)	15.06.2020	Translation of the German ‘Leitfaden’ version 4.5 to English.	Michael Schmidt, Jean-Marc Meyer
4.5	14.01.2019	Critical architecture-relevant requirements, behaviour during startup, shutdown and reset. Adaptation Chapter 4.3.	Kevin Zange, Michael Schmidt, Nick Walther
4.4	19.01.2018	Corrections and rephrasing, form & provision, concretization Multicore/-CPU	Kevin Zange, Michael Schmidt
4.3	30.07.2015	Safety, security, application data set download, Flexray, operation modes, multicore, watchdog concept, organisational structure	Ulf Vatter
4.0	12.12.2013	Adaptions to AUTOSAR	Ulf Vatter
3.2	22.02.2010	Information about flash times added	Christian Andersch
3.1	19.05.2009	Various error corrections	Christian Andersch
3.0	29.04.2009	Extensive streamlining and restructuring of content	Christian Andersch
2.1	29.06.2007	Detailing and refinement	Christian Andersch
1.0	25.07.2005	Revision of the previous template document based on version 1.14.	Christian Andersch

Table of Contents

1 General Requirements

1.1 Objective of these Guidelines

This document is a guideline that describes Porsche's requirements for the software architecture documentation of embedded control units (ECUs).

The structure of this guide, in particular chapters 3 to 7, may be used as a basis for the software architecture documentation to be created.

1.2 Aim and Purpose of the Software Architecture Documentation

The documentation of the software architecture is the basis for the architecture review and formal architecture approval by Porsche AG, which are part of the Porsche development process. The information it contains shall therefore be comprehensive, complete and consistent.

Furthermore, the software architecture documentation serves to provide Porsche AG with an insight into the system on a technical level in order to be able to carry out various measures quickly and efficiently during the development process (e.g. risk assessment of changes, carrying out code reviews).

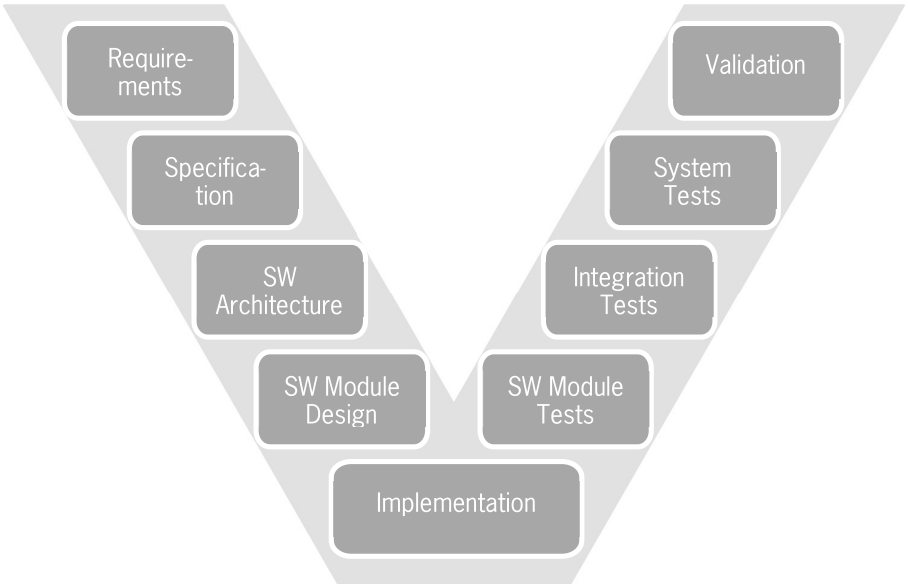


Figure 1: V-Model

It makes sense to carry out the review at the end of the architecture phase (see Figure 1). Earlier, the documents are not (completely) available, whereas later changes will lead to increased efforts. With timely implementation, errors can be avoided in the first place.

1.3 Use of Existing Information and Documents

In organizations whose software development process complies with Automotive SPICE Level 2 (alternatively CMM(I) Level 3) or higher, the content required by Porsche AG is usually a subset of the documentation created during the design phase. Therefore, the creation of the software architecture documentation should essentially consist of a compilation of already existing information.

A further option is to adopt the design documentation available from the supplier as software architecture documentation, in agreement with Porsche AG - even if it differs in individual points from Porsche requirements. However, the information described in these guidelines shall be available.

1.4 Creation, Design and Cover Page

Since the software architecture documentation is to be provided by the supplier to Porsche AG, document templates from the respective supplier can be used. The software architecture documentation shall be prepared in either German or English. The structure of the cover sheet is described in Chapter 3 Cover Sheet.

1.5 Form and Provision

The SW architecture documentation shall be provided to Porsche in PDF or HTML format. Format deviations shall be agreed with the person responsible for the software architecture review.

The SW architecture documentation should consist of only one document. If external documents cannot be integrated, they shall be supplied in full as a PDF or HTML document.

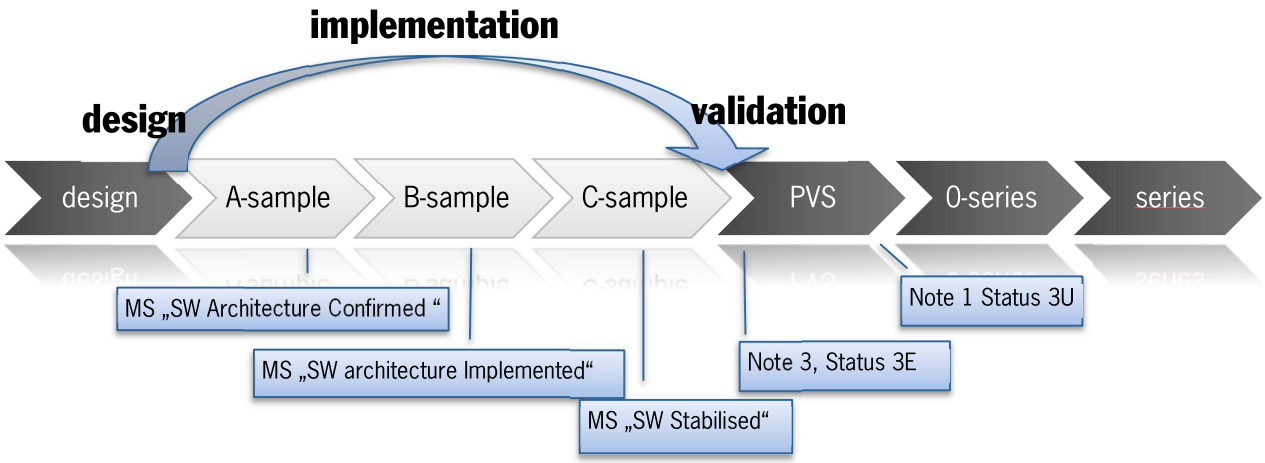
2 Milestones

2.1 Target Architecture and Project Livecycle

The software architecture document shall represent the target architecture of the system to be developed, as well as any significant changes that are already known at the start of the project (e.g. development starts with processor derivative A, a change to processor derivative B is planned for release xy).

It is not intended to describe the current project status or the ongoing project progress in the software architecture document during development via the planned releases (the feature release plan or the project requirements specification is used for this purpose).

There are three major milestones during the project lifecycle, which represent the progressive maturity of the architecture. These milestones shall be planned in the release plan. In general, all milestones shall be reached



as early as possible.

Figure 2: Time schedule of the milestones (schematic representation)

2.2 Milestone "SW Architecture Confirmed"

Since the B sample implements the essential architectural features of the system, the formal confirmation of the software architecture takes place during the A sample phase. The exact date is project-specific and is determined together with the supplier at the beginning of the project. The software architecture documentation must be available in its final version by this time.

The achievement of the milestone is documented by the distribution of a formal software architecture confirmation by PAG. This formal confirmation of the architecture contains among other things:

- An overall evaluation of the architecture according to the PAG traffic light scheme
- If necessary, a list of open issues and corresponding risk reduction measures
- If the architecture changes nevertheless in the subsequent development phase, an updated version of the documentation, including a change history, shall also be delivered

2.3 Milestone „SW Architecture Implemented“

The point in time at which the architecture is first implemented in the control unit:

- All structural elements of the architecture are existing. The layer model is implemented and interfaces are defined
- All hardware related SW components (e.g., device-drivers, operating system, networking) are integrated into the system and are functional. This also applies to the relevant components of the standard software.
- The task structure and communication are implemented, but not yet optimized.
- The fulfilment of the system-specific critical functional requirements and the compliance with the planned resource provisions (RAM, ROM, CPU utilization) were demonstrated
- Variant and data handling are fully implemented and approved
- The Functional Safety architecture is shown

The confirmation of this milestone is carried out by a random inspection of the source code including the development documentation at the supplier's site.

2.4 Milestone „SW Stabilised“

The point in time when the architecture is fully implemented, but possibly not yet optimized:

- All software components of the system are integrated
- All interfaces are implemented but not yet optimized
- All functions of the system are implemented but not yet optimized
- Task/process structure and -communication are optimized and confirmed
- The achievement of the system-specific critical functional requirements and the compliance with the planned resource provisions (RAM, ROM, CPU utilization) are verified and documented in the target system based on measurements

This milestone will be reached at:

- Feature Freeze or
- Grade 3 or
- Test Series

Depending on the overall vehicle project, the earliest of these dates is valid.

3 Cover Sheet

A cover sheet should be provided.

The following data shall be included:

- Control Unit
- Target Product Line
- Vehicle Architecture
- Other future product lines and/or vehicle architecture with planned deployment
- Author and Contact Person
- Date
- Version
- Technical Boundary Conditions:
  - Terminal supply (Terminal 15 or 30)
  - Communication buses used
  - Deployed operating systems (supplier, name, version)
  - Deployed MCU(s)/Controller, available RAM, ROM, EEPROM
  - Deployed Flash bootloader (supplier, name, version)
- Further special requirements
  - Maximum Functional Safety level of the system
  - Special security requirements (component protection, SWaP, flash data security)
  - OBD relevance
  - Participation in partial network operation (active, passive, none)
  - Diagnostic class

4 Architectural Concept View

The conceptual view represents the essential functions and features of the system, as well as the integration of the system in the environment or vehicle.

4.1 Overview

This view is intended to describe which functions the system includes and which not. A brief introduction to the system shall be provided. The purpose is to make the functionality understandable to readers who are experts in software architecture but are not familiar with this system. Ideally, the following items are listed:

- Key features
- Most important use cases
- Most important components
- Most important parts delivered (deliverables)
- Added value

A textual description is sufficient; sketches and graphics can be added for illustration.

4.2 Takeover and Further Development Based on Other Projects

To minimize effort and risks, often platform concepts are applied, or existing projects (the so-called reference project or lead project) are used as a starting point for further development.

In this section, it shall be explained whether and to what extent the existing project is based on already existing products and developments. This applies not only to the software and its configuration but also to the hardware used.

When describing a carry-over scope, a distinction must be made between the carry-over of concepts and the carry-over of implemented components.

Furthermore, the development stage of the lead project(s) shall be shown, e.g.

- already in current series production at one or more other OEMs
- currently under development, planned series production at one or more OEMs at an earlier time than PAG series production
- under development, planned series production at about the same time as PAG

4.3 Use of New Technologies or Concepts

In case new concepts or technologies are applied in the project, these shall be mentioned. In addition, the impact of these changes on the software shall be described. This includes for example:

- Migration from hand coding to model-based development with subsequent code generation
- Change of the programming language
- Changing the system's communication bus technology
- First-time use of AUTOSAR or switch in Major-Version
- Changing ECU type from 'terminal 15' to 'terminal 30'
- New ASIL level or first-time application of ISO26262

4.4 Critical Architectural Requirements

Critical architecture relevant requirements are those requirements that must be considered in the design of the system architecture. These shall be derived from the risk management plan and explicitly listed here, for example:

- maximum admissible control time of a chassis or safety systems
- required data throughput between hardware components (e.g. data transfer between main processor and one or more DSP / FPGA) and its effect on the system, especially with regard to initialization times and load peaks
- maximum allowed reaction time of a control unit

Are there strong dependencies on ECU external functions, such as

- of the latency of a client server communication
- of the quality of received data (e.g., object lists from a radar sensor)
- takeover diagnostic functions for a slave control unit
- takeover of central vehicle functions in the system (e.g., ETH switch, firewall)
- interface to externally provided partial functions (e.g., antennas, actuators, sensors)

4.5 Safety Related Aspects

In case of safety relevance (ASIL level), the safety goals to be achieved shall be listed. Safety goals are abstract requirements for the system to minimize hazards to the user, such as

- The operation of the system may not cause any injuries to the user

4.6 Security Related Aspects

Any requirements for tamper resistance, data security, etc. shall be listed. In addition, existing security objectives shall be listed and based on that measures that are relevant for the architecture shall be derived.

4.7 System Interfaces

In this chapter it shall be shown which external devices and tools (e.g., development or debug tools, Porsche Diagnostic Testers, XCP Master, flash tools, production tools etc.) can be connected directly or indirectly to the ECU and via which interfaces they communicate. A tabular overview appears suitable for this purpose.

If special interfaces and features are available in the ECU for supporting the development and debugging process, these shall also be shown. In addition, it shall be explained whether, how, and at what time the corresponding functionality is deactivated or removed in the production software of the ECU.

In the case of software functions that can be optionally activated, the activation concept shall be described.

4.8 System Behaviour in the Network

An overview of the networking requirements shall be provided. This includes at least:

- Logical Terminal (terminal 15 or 30), Sleep Latency
- Power supply (terminal 15 or 30)
- Wake-up
- Reasons to stay awake
- Reasons for communication request
- Wakeup authorisation
- Participants immobilizer, component protection, SWaP

If relevant:

- Subnetwork operation (active, passive, none)
- BAP role
- Diagnostic class
- OBD relevance (master, primary, secondary, not OBD relevant)
- For Ethernet: Use of service discovery, SomeIP, remote procedure calls, etc
- For Flexray: syncnode, cold starter
- Send/receive Functional Safety relevant messages with indication of the ASIL level

In addition, the representation can also be done graphically in the form of a state machine.

4.9 Flexray Operation

When participating in Flexray operation, the following information shall be provided:

- Flexray channel used, cold starter, syncnode
- Flexray job lists
- Job List Scheduling Algorithm
- Synchronization of the Flexray tasks to the application

4.10 Software Updates

Furthermore, it shall be shown in which way the software can be updated - both during the development process and in series production, e.g.

- via the CAN bus based on the flash protocol described in the diagnostic specification (in development and series production)
- via the CAN bus based on the XCP protocol (during development and series production)
- via the CAN bus based on a proprietary protocol
- via the Trace32 Debugger (during development)
- via other data media (e.g., system DVD, memory card)

- via special development interfaces (e.g., RS232, USB)
- via special bus (e.g., Ethernet, WLAN)
- via OTA control unit
- special protocols (deviating from standard implementation VW80127)
- special tools that deviate from Porsche standard tool chain (PIDT, PASDT, ODIS)
    - example for such a special tool: vFlash

For the update process via the Porsche standard tool chain, an estimation shall be given of how long the software update of the system will take. For this purpose, the target data rate shall be specified, and the amount of data estimated.

The security mechanisms used (e.g., flash data security, flash data encryption, ASN.1 parser) shall be demonstrated.

In addition, it shall be listed which program parts/SWCs can be updated independently of each other, and which conditions/restrictions must be considered.

4.11 Fail handling

A description of the system reaction in case of an error (emergency operation, mitigation, fail-safe) shall be given.

In addition, the mechanisms for reaching the safe state shall be given. The following aspects shall be taken into account:

- Maximum allowed time to reach the safe state
- Necessity of switching of the MPU
- Entry from:
    - Task level, i.e. Application Process
    - ISR level, i.e. Kernel Mode
    - OS hook, i.e. Signal Handler
    - Error handler, if present (e.g. handler for ECC-, Lockstep-, or MPU- error handling)

4.12 Special Operation Modes

A list of special operating modes and the expected functionality in these modes shall be provided.

Examples of special operating modes are:

- Production mode
- Customer Service mode
- Transport Mode
- Roller Rig Mode

4.13 Behaviour at Startup, Shutdown and Reset

The behaviour of the system during startup, shutdown and reset shall be described.

In particular:

- Which (customer) requirements apply regarding the maximum permitted startup and shutdown times?

- Which effects arise from security requirements, e.g., due to Secure Boot / Trusted Boot?
- Which general steps must be carried out by the system in the startup and shutdown phase? (e.g., HW initialization, execution of bootloader, reading RAM mirror, saving NVM data, ...)
  - Which hardware components (controller, CPU cores, peripheral modules, ...) are involved?
  - In which order the individual steps are executed?
  - Which dependencies exist between the components and the external periphery (e.g., synchronicity, blocking, waiting for signalling of a state via CAN)?
  - Which time frames are planned for the individual steps?
- A representation of the sequences, e.g., in the form of a table, a state machine or a sequence diagram (see Figure 3).
- If there are different startup/shutdown paths, these shall also be documented (e.g., depending on the reason of a reset).
- Which conditions lead to a reset and which additional reactions are performed?
- How are reset cycles prevented and what reaction is executed in case a reset cycle was detected?

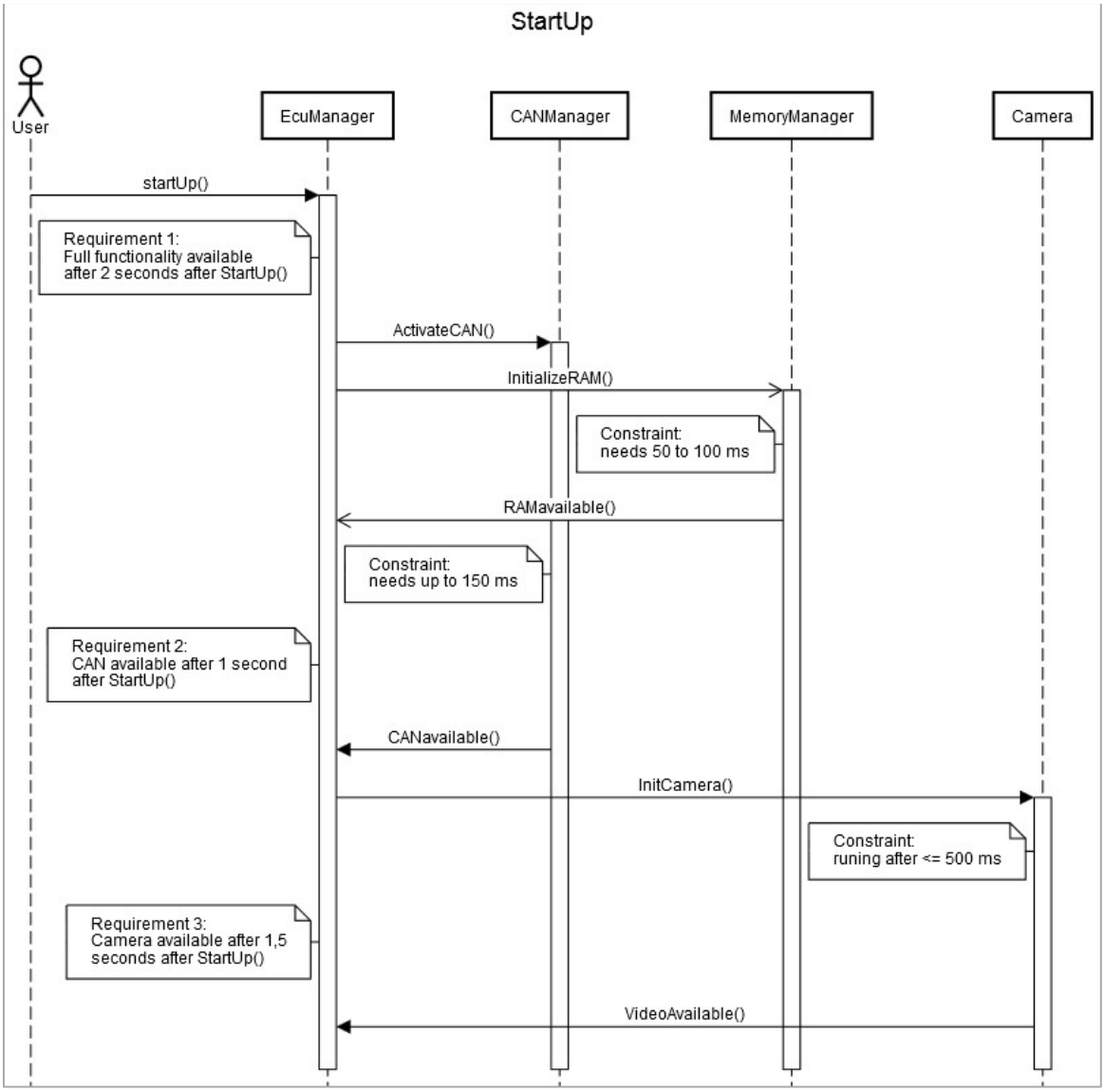


Figure 3: Exemplary representation of the startup-behaviour as sequence diagram

5 Infrastructure View

5.1 Hardware Components and Block Diagram

The infrastructure view uses a table to describe the runtime environment of the system in form of hardware components (such as processors, all types of memory, CAN controllers, additional I/O, external watchdogs, networks, external plausibility units, fallback measures) and all protocols involved. In addition, the performance data and parameters of the elements involved are described (e.g., manufacturer, type, clock frequency, memory size).

This information is needed because the hardware components usually have an influence on the software and its performance. Furthermore, it must be ensured that critical semiconductor components are available for a sufficiently long time.

A block diagram is used to show how the hardware components described above are connected. Block diagram and table must be consistent with each other.

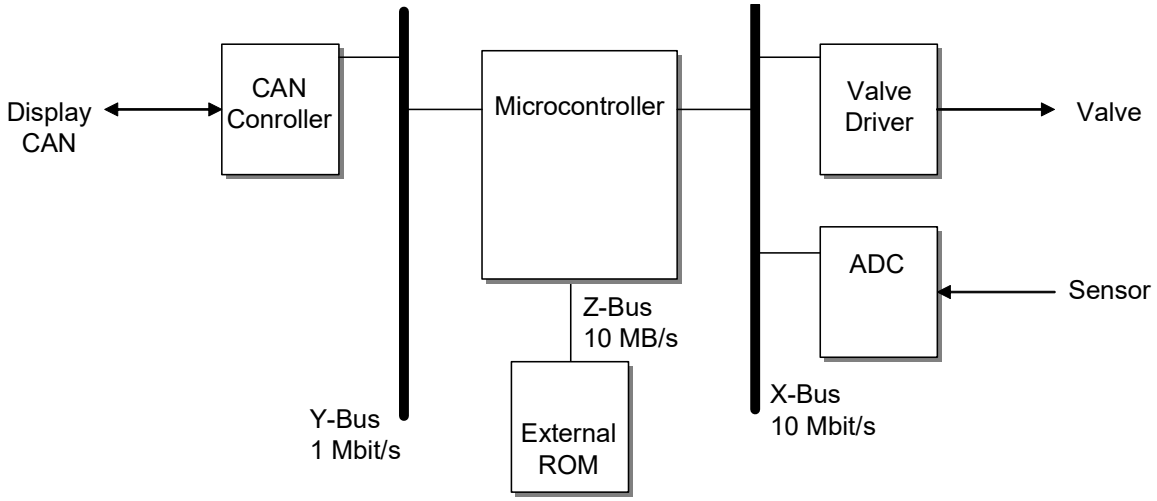


Figure 4: Example hardware components block diagram

5.2 Description of used microcontrollers/-processors

The capabilities and limitations of the used microcontrollers/-processors shall be listed individually. In addition, it shall be documented which of these functions are utilized Examples are:

- Number of cores
- Task frequency
- Safety features: Lockstep, ECC
- Security features, e.g., SHE, HSM, random number generator (TRNG or PRNG)
- Interrupt controller, latencies until ISR entry (caused by HW and OS), duration of setting and releasing an interrupt lock (see OS description)
- Virtualization capabilities (Hypervisor)
- MPU, MMU. What restrictions do the protectable memory windows have? (e.g., size at least 4k; start address must be aligned to log2 of the size; number of monitorable windows is 4)
- HW stack check
- Address space (Linear, Tiled, Havard)

Furthermore, for microcontrollers:

- Flash size, flash window size, connection to memory bus
- Maximum memory cycles of the flash, which maximum flash times are given after degradation of the flash
- RAM size
- EEPROM (or flash emulation)
- At Multicore:
    - Description CPU local RAM
    - Connection of the shared RAM including access timings (in cycles)
    - Connection of the shared ROM (Flash) including access timings (in cycles)

5.3 Peripherals

The capabilities and limitations of the peripheral modules that influence the system behaviour or the controller itself in terms of their function type or connection to the controller(s) shall be listed:

- What types of peripherals?
    - (FPGA, DSP, SBC, ASIC, RAM, Flash, IRC)
- What function?
    - Examples
        - Voltage measurement / protection circuit,
        - Image processing,
        - Controller,
        - ...
    - ASIL?
- Which interaction
    - Communication to other peripherals/MCU/MPU
        - Physical connection, protocols
    - Diagnosable
    - Programmable, if so, how?
    - Startup/shutdown/reset

External peripherals that have no functional influence on the MCU, such as a simple voltage divider, can be neglected.

5.4 Planned Resource Consumption at SOP

Please indicate the planned resource consumption of the system for series production and the basis for this estimation (e.g., reference project with the same controller):

- ROM consumption (absolute and in % of the available storage space)
- RAM consumption (absolute and in % of available memory)
- EEPROM consumption (absolute and in % of available memory)
- CPU performance (in % of the available CPU performance, broken down into base load and peak load)
- Stack consumption (absolute, per task and ISR, and OS hooks)
- Example table, possibly broken down by platform and project-specific scope

In addition, the determination method used for measuring the idle time shall be specified (e.g., measurement with Gliwa T1, measurement with self-implemented measurement methods, etc.) It is also particularly important to specify the reference base (e.g., longest running time slice).

The method for monitoring the resource consumption during development shall be demonstrated.

In case of a change from Single to Multi Core, the assumptions for the expected performance increase shall be shown in detail. In particular, the consideration of the following aspects shall be described:

- Overhead necessary for synchronization
- Losses due to slow Flash/RAM connection (crossbar switch)
- Changes of the latency caused by synchronization
- Potentially duplicated code (for code that runs on multiple cores)

In case the memory is managed dynamically, it should be illustrated which concept the management is based on. In particular:

- whether the allocation of memory is organized via a memory pool
- which data mainly contribute to dynamic memory consumption?
- which measures are used to prevent critical memory usage?

5.5 Hardware Scalability Strategy

In this chapter it is shown which provisions are made to be secured against resources becoming scarce in the course of the project. Please indicate explicitly which measures are planned to avoid the following scenarios:

- Insufficient computing power of the microcontroller/-processor (e.g., by replacing with a pin-compatible controller of higher performance, by increasing the clock frequency, etc.)
- Insufficient ROM, RAM or EEPROM memory of the microcontroller (e.g., by replacing the controller with a pin-compatible controller with larger memory, by using external memory, etc.)
- Discontinuation of relevant hardware components by the semiconductor industry (e.g., by second sources)

Within the scope of the measures, it shall also be shown whether and which changes to the software architecture are required as a result, e.g.

- Modification of existing components
- Use of other drivers etc.

5.6 Development Environment

In addition to the hardware components, the infrastructure also includes the development environment. The description shall show which development environment (compiler, IDE, debugger etc.) and which other tools are used, e.g. code generators, graphical modelling tools, case tools, resource measurement and analysis tools.

5.7 Prevention against Flash Corruption during Operation

To protect the flash memory from corruption, it shall be ensured that no unintentional writing to the flash memory can occur during normal operation. For example, damage to the program code stored at this memory area could cause the ECU to fail. The following shall be described:

- Whether the ECU has write access to its own flash memory during normal operation (i.e. whether the normal application contains flash drivers or whether they can only be used from the bootloader).
- whether and how these access options are used
- how these accesses are secured against unauthorized execution or address range overruns

If unintentional flash overwriting cannot be prevented causally, it shall be noted:

- whether and how the data concerned is checked for plausibility (in order to prevent using corrupted data)
- which measures are triggered in case of an error is detected (apart from possible entries in the error memory)

5.8 Watchdog Concept

A description of the watchdog concept should be given. The following information is required:

- External and internal watchdog
- To which clock are they connected
- Timeout watchdog, window watchdog, etc.
- Window sizes
- Offset of the windows to each other
- Under what circumstances is a watchdog switched off?
- Conditions to be fulfilled for triggering the watchdog
- Anchoring of the watchdog trigger in the system (name of the task/ISR, name of the SW module)
- Starting the Watchdogs after Power-On
- Status of the watchdogs when a regular shutdown is initiated
- Determination of the reset cause after a watchdog reset
- Watchdog operation in the Flash bootloader
- Watchdog handling during the transition from Flash bootloader to application and vice versa

6 Logical View

The logical view maps the functionality of the system to software components. Software components represent abstractions of source code, such as functions, classes, modules, packages, or subsystems. The logical view describes the central software components and their static and dynamic dependencies.

6.1 Static Architecture Model of the Software Components

The logical software components and their structural dependencies shall be represented using a static architecture model. For example, components, layers or services need to be described. The granularity of the logical software components shall be comparable to the granularity used in the Software Release Plan.

If a multicore or multiprocessor system is used, the static architecture model shall be displayed separately for each core and CPU. A common architecture can be used if the distribution to CPU cores is managed dynamically.

6.2 Description of the Software Components

The following information shall be compiled for each of the previously illustrated software components (a table is suitable for the illustration. In the listing below, the term “component” (Classic AUTOSAR) is used synonymously with the term “cluster” (Adaptive AUTOSAR)):

- Name of component
- Type of component (SW-C, CDD, BSW, MCAL, foundation- or service-component)
- Function(s) of component
- Origin of component
  - completely new developed component
  - purchased standard component (e.g., CAN driver)
  - component developed by third party
  - component provided by Porsche
  - component reused from other projects without changes
  - component reused from other projects with changes
  - for reused components, indicate when the reference project has gone or will go into series production.
  - in case of purchased components the manufacturer or supplier must be indicated
- Method of implementation
  - hand coded
  - supplied object code that is integrated in binary format
  - Model-based or automatically generated (please specify the tool, e.g., TargetLink or CANGen)
- ASIL level of the component

In addition to or instead of a tabular representation, it may also be useful to depict the above listed points by means of various schemes such as color, shape or similar, including corresponding legends/references in the layer model.

SW modules that contain open-source software shall be explicitly marked and the corresponding license shall be stated.

Standard software modules provided as Single Source by PAG shall be explicitly labelled (e.g., SFD, ID, VKMS, etc.)

6.3 Organisational Structure

The structure of the development teams and the project management incl. their location shall be specified for each component.

6.4 Data Flow

A representation of the essential data flows between the components (within the control unit) shall be provided. This can be done on a very abstract or architectural level.

Safety-relevant data flows shall be marked separately.

Central mechanisms shall be shown. This includes, for example:

- RTE- or ARA::COM
- Sockets
- Safety mechanisms, e.g., double-inverse storage, MPU secured data exchange memory, etc.
- IPC and the mechanisms used for data exchange between cores
- Ensuring data integrity in case of concurrent accesses by different tasks, interrupts and cores.

6.5 Safety Architecture

A description of the mechanisms used in the software to ensure data integrity, sequencing and monitoring shall be provided, e.g.:

- Program execution monitoring
- HW measures: MPU, ECC (RAM, bus, ALU, etc.), lockstep, stack protection
- Double-inverse data storage
- Checksum secured areas
- RAM/Periphery testing
- Diversified code
- Monitors (monitoring of SW components)

An assignment of the measures to safety levels shall be shown. For this purpose, the concept of the safety levels used shall also be specified. Example:

- Level 0 (prevention): Static checks, reviews, and so on.
- Level 1 (incident detection): Validation parameters, double-inverse storage, etc.
- Level 2 (prevention of faults): MPU, Program execution monitoring, ECC-RAM, Lockstep, etc.
- Level 3 (monitors): Safety modules for result validation
- Level 4 (external validation): external watchdog, safety circuit in HW, ...

7 Data View

The data view shows which variants of the system exist, how they are created and managed, how the persistent data is structured and which fall back concepts are used.

7.1 ROM Memory Map

For each CPU in the system, please illustrate the memory map of the assigned ROM and EEPROM memory and highlight the following factors:

- The number, size and location of the memory areas
- The information whether and which memory areas can be updated individually, e.g., a separately flashable data set
- The contents of each memory area, e.g., program code, data set, boot loader, constants, BSS
- The reserved memory space for each memory area (please ensure consistency with the specifications in Section 5.2)
- Memory areas that require special measures against overwriting/manipulation (for example keys for component protection, Functional Safety requirement, other types of security requirements, etc.)

Please make sure that the target configuration *planned for SOP* is displayed, and not the current development status. A representation on byte level is not required.

Please pay particular attention to the following elements:

- Development key for flash data security
- Serial key for flash data security
- Areas for download of data sets

7.2 RAM Memory Map

For each CPU in the system, please illustrate the memory map of the assigned RAM memory and highlight the following factors, if these are specified:

- Variables of the individual components
- Global variables
- Variables that are shared between
    - Interrupts and tasks
    - Various components
    - Tasks with different priorities
    - Different cores (for multicore)
    - Various OS Applications (for Memory Protection)
- Number and size of stacks
- EEPROM Buffer
- Safeguards (e.g., MPU, checksum, ECC, etc.)

If an MPU is used, the protected areas and the corresponding access rights shall be displayed.

Please make sure that the target configuration planned for SOP is described and not the current development status. A representation on byte level is not necessary.

7.3 Variants

Please describe how many and which variants of the control unit software exist. A distinction shall be made between functional software and data records. The number of variants shall also consider coding variants and in particular, their effects on data records and memory space consumption. The following figure represents a suggestion of how this information shall be documented:

There are two variants of the function software:

- one for the sports car
- one for the offroad vehicle

There are also six different data sets:

- Sports Car Suction Engine
- Sports car Turbo engine
- Offroad vehicle Suction engine
- SUV Turbo engine
- Provision of two unused data sets for further engine variants

Figure 5: Exemplary representation of the variants

Furthermore, the following shall be presented,

- how the variants of the function software are created and switched over, e.g:
  - common code base; generation of the variants at compile time by pre-processor switch and configuration of the ECU by flashing in the production of the supplier or at Porsche.
  - common code base; configuration of the variants at run time via diagnostic coding
  - common code basis; configuration of variants at runtime depending on information available on the vehicle networks or external interfaces (e.g., coding by connector pins)
  - different code base; configuration of the variants by flashing during production at supplier site or at Porsche.
  - dynamically reloadable or activable functions
- how variants of the data sets are stored and configured, e.g.:
  - there is one data set stored in the ECU. The configuration of the data record is flashed into the ECU via the diagnostic tester/production machine.
  - there are several data sets stored in the control unit. The selection/configuration of the data set is done by coding via the diagnostic tester/production machine or at runtime.
  - mixed forms of the above scenarios (please describe).
- A distinction shall be made between coding, data set download, and application data set downloads and file transfer.

7.4 Data Storage and Data Assurance

Based on the memory areas listed in 5.1 and 5.2 (usually EEPROM, Flash-ROM, hard disk, etc.), in this section it shall be described what kind of data is persistently stored in these areas. For reasons of clarity, it is advisable to create a table (see following example) in order to specify the information required by PAG in detail.

Storage Media	Type of Data	Size	is written when …	Is read when …	Data Assurance	Remarks
Flash-ROM	Default application data set	1 kB	Never during operation, can only modified by software update.	This data set is used as fallback in case of no valid variant coding is present or the EEPROM content is corrupted.	No protection during runtime, is checked for integrity during the flash process (same as for program code)	
EEPROM	Application data set for variant “USA”	1 kB	Never during operation, is calibrated during production at supplier site.	Is copied to RAM during system startup due to performance reasons.	CRC Checksum	If data validation failed the default, data set is read from Flash-ROM. The check is running cyclic in task XY with low priority.
EEPROM	Application data set for variant “ROW”	1 kB	Never during operation, is calibrated during production at supplier site.	Is copied to RAM during system startup due to performance reasons.	CRC Checksum	If data validation failed the default, data set is read from Flash-ROM. The check is running cyclic in task XY with low priority.
EEPROM	User data (original)	256 Byte	At initial operation, default data is copied from ROM. During operation, changed values are stored immediately.	During operation the data is requested via EEPROM Manager	CRC Checksum and redundant data storage	For a detailed description of the data mirror concept see chapter X.Y
EEPROM	User data (mirror copy)	256 Byte	Copy of original data is written during system shutdown.	Only if validation of original data failed	See above	For a detailed description of the data mirror concept see chapter X.Y
…	…	…	…	…		…

Figure 6: Example table of persisting data

In columns "Type of data" and "Scope", for example, the following entries are conceivable:

- Coding values, size 100 bytes
- Variant dependent data set, size 200 bytes each
- Calibration values (e.g., end stops), size 64 bytes
- User data (e.g., seat position, telephone book entries), size 1 kB

In columns, "is written when..." and "is read when..." it shall be specified when this data is written and read, e.g.:

- immediately after each change of value
- cyclically every “n“ seconds
- during control unit overrun

- only on request via diagnostic tester
- in production, e.g., through a teach-in function during assembly

In column "Data assurance" it shall be described which measures are taken to secure the corresponding data against corruption and what actions are performed in case of inconsistencies are detected, e.g.

- no data backup
- simple data storage with checksum over the entire data area, fall back to default data set in the Flash-ROM as soon as an unrecoverable error is detected
- redundant data storage with checksum over the entire data area, reverting to redundantly stored date in case an unrecoverable error is detected in the original area
- triple data storage and "two out of three" selection over the entire data range
- double data storage with checksum for each individual element of the data area
- other data protection mechanisms (please explain the concept)

Further information can be given in the column "Remarks". In the case of more complex data backup mechanisms, it is advisable to document these separately in a separate section and to specify the corresponding cross-reference here.

7.5 Data Buffers

If persistent data is buffered in RAM (especially concerning RAM-mirror of EEPROM memories), the following shall be described in detail:

- Times of alignment between buffer and non-volatile memory
- Protection of the data of the buffer memory and the non-volatile memory against corruption
- Actions in case of detected errors

7.6 Reset Safe Data

It shall be shown how data is exchanged between the application and the flash bootloader.

It should also be shown how a reset reason is detected after booting.

8 Process View Classic AUTOSAR

The operating system configuration is described in the process view.

8.1 Operating System

The manufacturer of the operating system shall be specified. Furthermore, the following list of ConformanceClass shall be specified, as long as it is relevant for the system:

- OSEK ConformanceClass (BCC1, BCC2, ECC1, ECC2)
- OSEK ScalabilityClass (SC1, SC2, SC3, SC4)
- AUTOSAR Implementation Conformance Class (ICC1, ICC2, ICC3)

If a proprietary operating system is used, its essential properties and functions shall be described, if necessary, by reference to the relevant documentation.

With reference to Section 4.2, it shall be shown to what extent the configuration of the operating system is taken over or derived from a possible existing lead project.

If 'Multiple Activations' are used, a short justification shall be given.

8.2 Runnable

An overview of the configured runnable shall be prepared, describing the essential properties:

- Function
- Start conditions, cycle time
- Start sequence within the task
- Assignment to Task
- Preemptivity
- Maximum permissible term
- Maximum ASIL Level
- Assigned MPU setting (only if a separate MPU setting is used)

8.3 Tasks

An overview of all configured tasks is to be prepared, which describes the essential properties:

- Function
- Cycle time
- Priority
- Preemptivity
- Extended Task or Basic Task
- Maximum permissible term
- Deadline, maximum permissible latency
- Assigned OSApplication (only for SC3 or SC4)
- Stack consumption
- Which stack is used (e.g.: own stack, shared stack, on the SingleStack)

- How is the task started (e.g., periodically by OS; by event EvA set by task A; by auto-start; by task A using Chaintask)
- Maximum ASIL level
- Assigned MPU setting

If available, startup tasks and background/idle tasks in particular shall also be described.

To illustrate the dynamic design, a graphical overview of the planned scheduling is required.

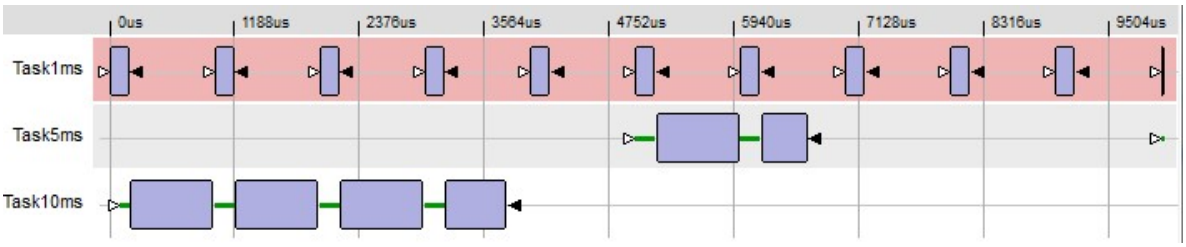


Figure 7: Example of the task scheduling

8.4 ISRs

An overview of all configured ISRs shall be prepared, describing the essential characteristics of each ISR:

- Function
- Cycle time (expected and maximum)
- Priority
- Category 1 or 2
- Self-interruptible by other ISRs (if so, which ones)
- Maximum permissible term
- Deadlines, maximum permissible latency
- Assigned OSApplication (only for SC3 or SC4)
- Which stack is used?
- Stack consumption
- Maximum ASIL level
- Assigned MPU setting

8.5 OS Hooks

If OS hooks (e.g. ErrorHook, StartupHook, ShutdownHook, PostTaskHook, PreTaskHook, etc) are used, the following considerations shall be documented:

- Function
- Reaction of the system to a call
- Maximum duration
- Which stack is used?
- Stack consumption
- Assigned OS Application (only for SC3 or SC4)
- Maximum ASIL level
- Assigned MPU setting

8.6 Timing Behaviour

8.6.1 Dynamic Design

To illustrate the dynamic design, a graphical overview of the planned scheduling is required.

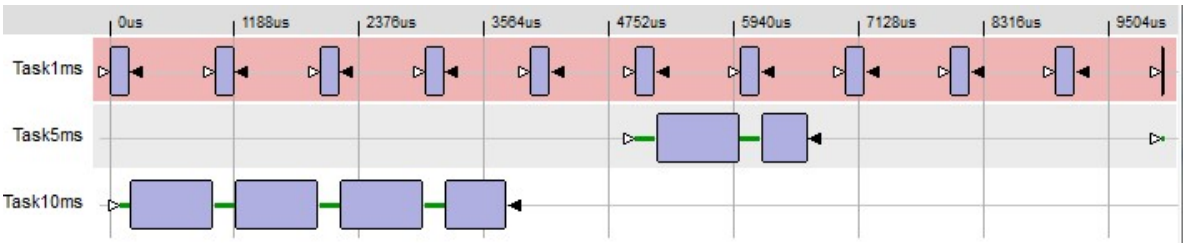


Figure 8: Example of the task scheduling

8.6.2 Real-time Requirements

Real-time requirements shall be documented and evaluated according to their criticality. The criticality rating results from the effects of its violation, e.g., causing a reset or delayed reaction of the control unit, and the expected tolerances. Such real-time requirements can, for example, are derived from customer requirements, originate from FuSa functions (programme sequence control / watchdogs) or result from experience from previous projects.

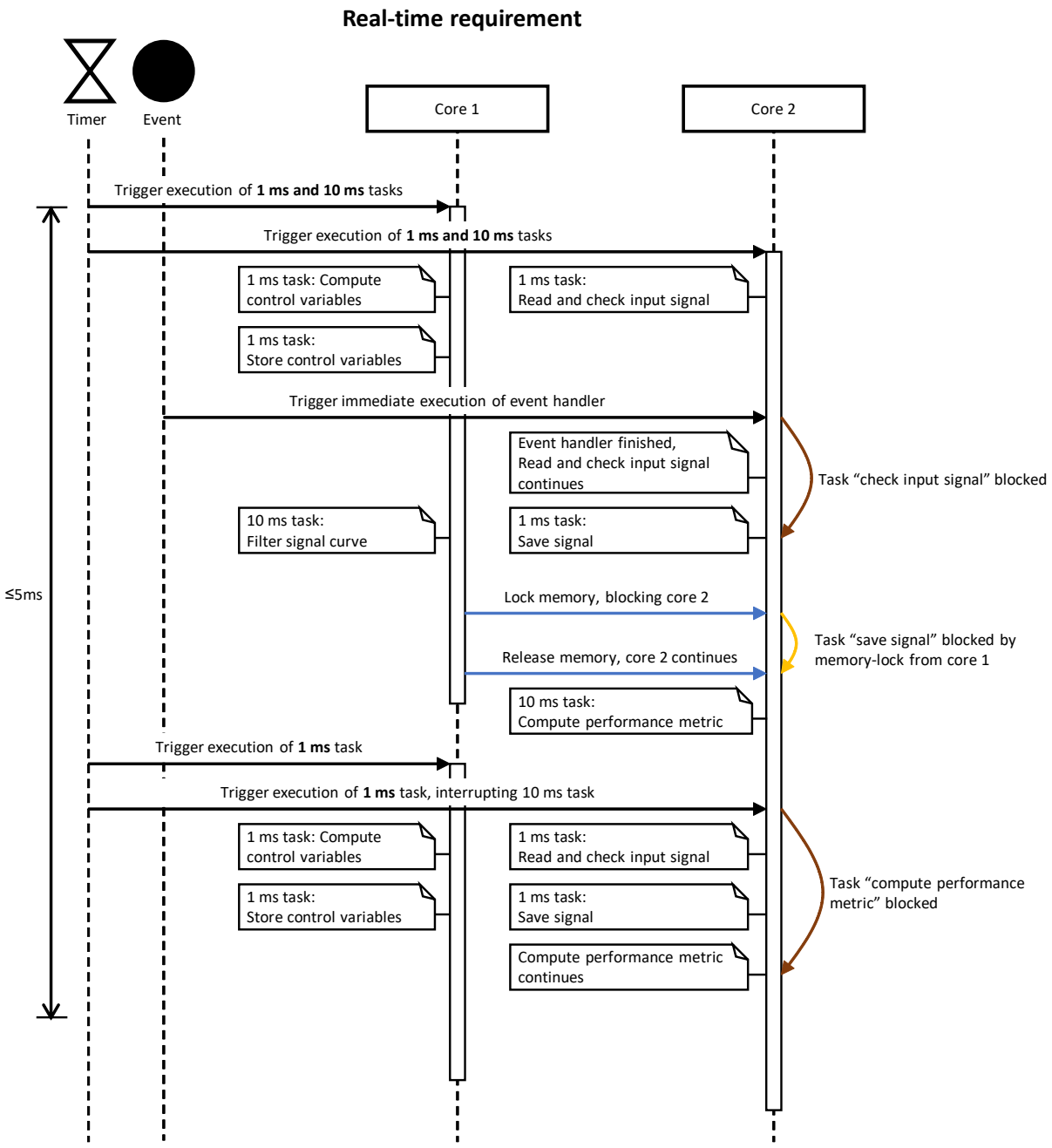


Figure 9: Example of a real-time requirement - sequence diagram

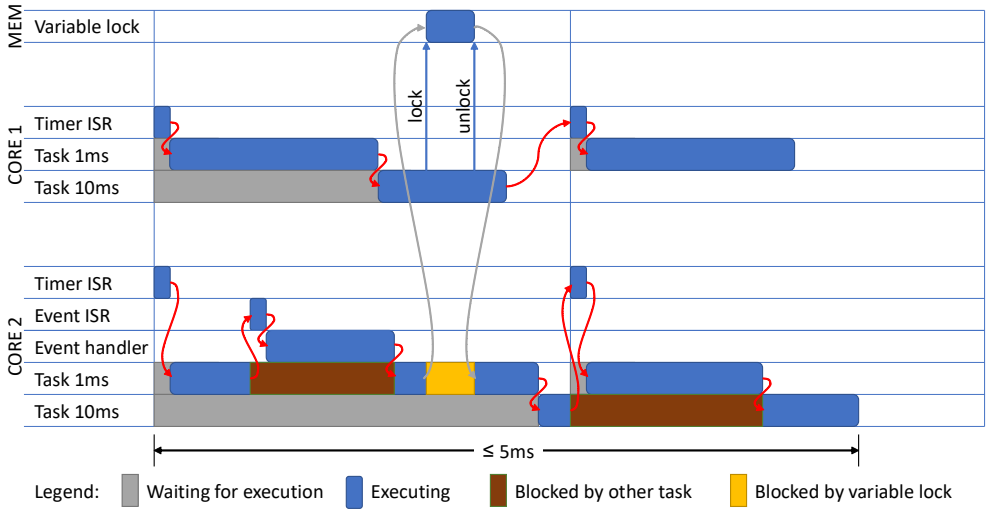


Figure 10: Example of a real-time requirement - graphical representation

8.6.3 Influencing Factors

Influence factors regarding execution timings between components shall be listed and considered, such as:

- Possible interruptions by higher priority Tasks and ISRs.
- Possible waiting times of runnables/tasks for the release of resources
- Possible waiting times for the availability of data
- Possible saturation of a data bus

8.6.4 Load Tests

Based on this, potential high load scenarios for the later test are to be systematically identified and documented.

9 Process View POSIX-OS based ECUs

The process view describes the operating system configuration.

9.1 Operating System

The manufacturer of the operating system shall be specified.

The manufacturer of utilized SDKs shall be also specified.

If a proprietary operating system is used, its essential properties and functionalities shall be described, if necessary, also by a reference to the corresponding documentation.

With regard to Section 4.2, the extent to which the configuration of the operating system is adopted or derived from any existing lead project, shall be described.

9.2 Processes

An overview of all configured processes shall be provided, describing the main characteristics:

- Functionality
- Cycle time, if known
- Number of threads
- Priority
- Real-time requirements
- Dependencies on other processes or events
- Expected CPU load
- Assignment to a CPU core
- Expected RAM consumption (stack and heap memory)
- How is the process started (by init-process; through another process)
- Maximum ASIL level

To illustrate the dynamic design, a graphical overview of the planned scheduling is required.

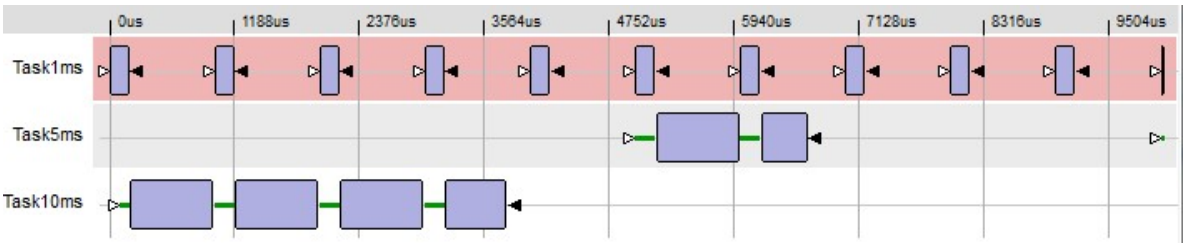


Figure 11: Example of the task scheduling

9.3 Operating System Services

An overview of all running operating system services shall be provided, describing the main characteristics:

- Functionality
- Priority
- Real-time requirements
- Dependencies on other processes or events
- Expected CPU load
- Assignment to a CPU core
- Expected RAM consumption (stack and heap memory)
- Maximum ASIL level

Appendix A: Abbreviations

ASIL	Automotive Software Integrity Level (FuSa)
ASPICE	Automotive Software Process Improvement Capability dEtermination (Process Model)
AUTOSAR	AUTomotive Open System ARchitecture
BSW	Basic Software
CAN	Controller Area Network
CDD	Complex Device Driver
CPU	Central Processing Unit
ECC	Error-Correcting Code (RAM)
ECU	Embedded Control Unit
EEPROM	Electrically Erasable Programmable Read Only Memory
FuSa	Functional Safety
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
MCAL	Microcontroller Access Layer
MMU	Memory Management Unit

MPU	Memory Protection Unit
OBD	On Board Diagnostic
ODIS	Original Diagnosis System
OEM	Original Equipment Manufacturer
OTA	Over-The-Air (Update)
PAG	Porsche AG
PASDT	Porsche Automated ECU Diagnostic Tester
PIDT	Porsche Interactive Diagnosis Tester
PVS	Production test series
RAM	Random Access Memory
SBC	System Basis Chip
SOP	Start Of Production
SWaP	SW as a Product
SW-C	Software Component
XCP	Universal Measurement and Calibration Protocol