ANALYSIS AND EVALUATION OF INTER-CORE COMMUNICATION BASED ON A MULTI-CORE AUTOMOTIVE SOFTWARE AND HARDWARE

KRISTINA KRISTIANA ZUBAC

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Master of Science by Research

The University of Huddersfield in collaboration with Frankfurt University of Applied Sciences

March 2016

Abstract

The vehicles have become increasingly computerized in order to satisfy the strict safety requirements and to provide better driving experiences. Growing complexity of vehicle functionality requires higher performance and increases complexity of electronic control units (ECUs) software.

As a result, the AUTOSAR standard was established by several major vendors and manufactures in 2003. This foundation introduced a standard with the main goal to define a uniform ECU software architecture, which reduces the software complexity.

The standardisation of software along with the increasing requirements for resources brought the single core processors to their limit of performance. Hence, new techniques had to be introduced to increase computational power. A solution to this problem is a multi-core processor.

It is expected that the main focus for the automotive industry in the next decade will be to implement the existing single-core software to a multi-core architecture. One of the key challenges is to reduce the execution time of inter-core communication in order to maintain the response time in safety critical systems such as the electronic brake system.

As the need for multi-core processors became apparent in the automotive industry, AUTOSAR introduced the initial release of multi-core support in 2012 with version 4.0. The specification provides guidelines for implementing multi-core architecture. One of the key elements is the Inter OSApplication Communication (IOC), which provides the cross-core communication between applications located in different cores on a single ECU. The current multi-core software implementation is still in development and there are performance limitations that offer scope for improvement.

This thesis addresses the challenge of multi-core software in automotive systems and proposes possible approaches for inter-core communication. In order to identify possible improvements, a number of software benchmarking experiments were designed for intra-core and inter-core communication and implemented on an universal ECU to validate the performance in term of execution time and memory consumption. The results demonstrate that there are significant overheads caused by the design of the communication functions.

Table of contents

26 824 words

List of tables

List of figures

List of codes

Dedications and acknowledgements

I would like first and foremost to thank my supervisors Dr Violeta Holmes and Prof. Dr. Karsten Schmidt for the useful comments, remarks and engagement and they believe into my skills to provide me the opportunity to study in United Kingdom.

I would also like to thank my whole family for they strong believe that woman are dedicated for engineering.

Finally I would like to thank my new and old friends:
Silvia Stojćeva, Desika Divković, Elena Zänglein, Younes El Ouarti, Philipp Brocar,
Karina Zala, Rudo Nomhle, Emina Brown, Sree Nirjhor Kaysthagir ,Isuru Sendanayake, Savith Sendanayake, Noukhez Ahmed, Fajer Mohamed, Aadil Rafiq, Hardy Punglia, Joshua Higgins, Joseph Bradford and Matthew Newall for always being there when I needed to talk through ideas or just rant about my day.

Also I would like to say a big thank you to all supporters:
Dr. Colin Venters, Mrs Gwen Wood, Mr Chris Sentence, Miss Nicola Williams and Mrs Liz Rees.

List of abbreviations

A	Amper
ABS	Anit-lock braking systems
AMALTHEA	Advances of machine learning in theory and applications
AUTOSAR	Automotive open system architecture
BCM	Brake control module
BSW	Basic software
CAN	Controller area network
CANH	Controller area network High
CANL	Controller area network Low
CAS	Compare-and swap
DINx	Digital input number x
ECU	Electronic control unit
FIFO	First-in-first-out
Func	Function
GPRs	General purpose registers
IDE	Intergrated development environment
IOC	Inter- OS-application communication
IPO	Input-process-output
ITEA 2	International test and evaluation association
KB	Kilo byte
Kbit/s	Kilobits per second
KHz	Kilo Herz
KΩ	Kilo Ohm
L1	First level cache
L2	Second level cache
LIN	Local Interconnect Network
M	Master
MB	Mega byte
Mbit/s	Megabits per second
MCU	Microcontroller
Mictor	Matched Impedance Connector
MIPs	Million instructions per second
MMU	Memory protection unit
MPCP	Multi-core priority ceiling protocol
ms	Millisecond
MSRP	Multi-core stack resource policy
ns	Nanoseconds

ORTI	OSEK run time interface
OS	Operating system
OS-Application	Operating system application
OSEK	Open Systems and their interfaces for the electronics in motor vehicles
PC	Personal computer
PCM	Powertrain control module
PCP	Priority ceiling protocol
RAM	Random-access memory
ROM	Read-only memory
RTE	Runtime environment
S	Slave
SBC	System basis chip
SC	Safety controller
SchM	Basis software scheduler
SingleCoreV1_0	Single-core version 1.0
SingleCoreV2_0	Single-core version 2.0
SR	Share resource
SRAM	Static aandom-access memory
SWC	Software component
V	Volt
VDX	Vehicle distributed executive
VLE	Variable-length encoding
WCRT	Worst-case response time
μs	Microseconds

1. Introduction

The vehicles have become increasingly computerized in order to satisfy the strict safety requirements and to increase the driving comfort. It is the complexity of various requirements such as active driving safety and new powertrain concepts, that increase the need for controlling and programming additional electronic systems vehicles. Each of these innovations requires a platform to execute such software, the electronic control unit (ECU).

In order to define a standard for the development of new embedded software for automotive industry, a foundation called “AUTomotive Open System ARchitecture” (AUTOSAR) was founded by core partners, automobile manufacturers, suppliers, and tool developers in the year 2003, (AUTOSAR, 2003). The main goal of AUTOSAR is the definition of an uniform ECU software architecture, which separates hardware and software, (Fons & Fons, 2012). This decoupling reduces the software complexity by improving the reusability of software components as well as saving resources, time and costs.

The standardisation of software and the increasing requirements for resources brought the single core processors to their limit of performance. Hence, new techniques had to be introduced for increasing computational power in ECUs. One possible solution is to increase the frequency of the single-core microcontroller. However, this strategy caused high power consumption. The additional power leads to power dissipation that not only decreased the reliability, it also became too expensive for the automotive industry, (Freescale Semiconductor, 2009). Hence, the automotive industry decided to use multi-core processors, which provide the ability to reduce the design complexity of automotive control systems.

Although the multi-core systems have become the norm for desktop computers, the development of multi-core real-time solutions in automotive is still in the nascent stages. When the need for multi-core processors became apparent in the automotive industry, AUTOSAR introduced the first version of multi-core support in version 4.0, released in 2012. This version describes a new approach for handling microcontrollers with multiple cores especially the communication of applications among cores (inter-core communication). Although AUTOSAR released the latest version of AUTOSAR 4.2 in year 2015, the additional overhead that is caused by the inter-core communication, could not be limited.

Moreover, multi-core systems provide speedup gain through parallel coding. As the legacy code was written for a single- core system, as a sequential code. Even if fractions of this code are able to run in parallel, they are still requiring synchronization through communication over the cores in a multi-core system. Furthermore, embedded systems have strong requirements regarding the safety, reliability and real time constraints. To meet these conditions, the execution time of inter-core communication should be reduced as much as possible to maintain the response time in safety critical systems such as the electronic brake system.

This project is exploring the current state of inter-core communication on multi-core ECUs. It was discovered that no inter-core communication implementation is available for academic use. In order to ascertain possible approaches for inter-core communication, a number of experiments were designed to evaluate software for intra-core as well as inter-core communication. These experiments were implemented and validated on the universal ECU VC121-12 from Vector assessing the performance and memory consumption of software.

1.1. Aim and objectives

This Master project is investigating the current state of inter-core communication on multi-core ECUs. Therefore, the aim of the project is to analyse and evaluate the intra-core communication software implemented in current AUTOSAR single-core architecture regarding the performance in terms of execution time and memory consumption.

In order to meet the overarching project goal, the following objectives were identified:

1. Investigate current solution for inter-core communication
2. Identify appropriated hardware and software used in automotive industry for real-time systems
3. Design suitable experiments for intra-core and inter-core communication in ECUs by using the two different communication technics (explicit and implicit) defined in AUTOSAR
4. Implement these experiments by using the AUTOSAR methodology
5. Evaluate the experiments regarding the performance and memory consumption
6. Propose possible improvements in multi-core software design, especially for intra and inter-core communication software, in order to improve execution time and memory consumption

1.2. Structure of the thesis

This thesis presents the body of work, which has been carried out for this research study, and is organised as follows:

- Section 2 provides the fundamentals of multi-core systems deployed in automotive industry. It describes the requirements of embedded real-time systems and presents the platform on which this research is based, the Electronic Control Units. Furthermore, it introduces the AUTOSAR architecture and the multi-core microcontrollers used in automotive.
- Section 3 explains the challenges of multi-core microcontroller in automotive with a focus on inter-core communication. It also presents related work and possible mechanism to improve the inter-core communication for multi-core systems in automotive industry.
- Section 4 describes the experiments, which were designed, implemented and evaluated in this work.

- Section 5 presents the test and development environment on which the experiments were conducted.
- Section 6 explains implementation procedure of the experiments. It provides a brief overview of the hardware setup and the software development tool set. In addition, it describes the outcome of the implementation, and the generated source code.
- Section 7 contains the evaluation of the generated source code regarding the scheduling, performance and memory consumption.
- Section 8 concludes with the research findings providing a summary of the achievements. This section also discusses possible improvements that were determined the evaluation. It also proposes possible directions for future developments of inter-core communication software.

2. Fundamentals

This section explains fundamentals that are necessary to understand the scope of this work. First, the requirements of embedded real-time systems are described. Secondly, the electronic control unit and the AUTOSAR standard are presented. Finally, the multi-core microcontroller is introduced.

2.1. The requirements embedded real time

An Embedded system is a programmable computer device with specific functions, it has a minimum of one processor that implements a desired hardware function, (Noergaard, 2005). Real-time embedded systems are extremely dependent on time and costs. Each cent has to be justified according to functionality, production and reliability. Real-time systems operate in the real world and are therefore limited by the physical conditions of the controlled system.

Scheduling
In comparison with personal computers; real-time systems have to control many devices simultaneously, For instance, the requests of multiple users of a database, (Gallmeister, 1995 p. 13). To accomplish these challenges, these processes have to be scheduled in a way that each time constraint is fulfilled.

Resource management
The resources of embedded real-time systems have to be shared between tasks that are executed on the microcontroller (MCU). The need to share resources is due to the limited budget that influences the price, (Gallmeister, 1995). A shared resource such as memory has to be managed otherwise deadlocks can be caused. This is then the case, if a less important task gets the resource first and therefore blocks other more important tasks to access this resource. This would then lead to long latency, deadlocks.

Compatibility
To hold the selling price of a real-time system as low as possible the device should be portable through innovations, which means updates in software should be backwards compatible, (Gallmeister, 1995).

Reliability
During the execution of a movie at a personal computer some fails of sound can be tolerated, but a real-time system has to be reliable at every time, (Gallmeister, 1995 p. 13-15) .

2.2. Electronic control units

Software is no longer just represented in computing systems such as PCs. The amount of software implemented in vehicles increase since the introduction of the engine control unit in the 1970s (Ribbens, 2013 p. 178) . The estimation of (Scott, 2004) from General Motors illustrates that software that is implemented in a vehicle consists approximately 100 code lines in 1970. This amount of software increased to 100 000 code lines in 1990. It is the number of functionality such as active driving safety (e.g. anti-lock braking systems), new powertrain concepts (e.g. hybrid engines) and new infotainment functions (e.g. navigation system) that increase the electronic amount in vehicles. Each of these functionalities requires a platform to execute such software, the electronic control unit.

An ECU is a microprocessor-based system and uses the input-process-output (IPO) model to control electronic subsystems in a vehicle, (Ciulla, 2000) . Some of the ECUs are the Powertrain Control Module (PCM) and Brake Control Module (BCM), (Instruments, 2009). The PCM monitors and controls, for instance, the speed of the vehicle. Moreover, the BCM regulates the braking system. It optimises the braking mechanism by providing software solutions (funtionality) for safe driving. One of such funtionality is the anti-lock braking system (ABS), which provides full tractability while braking (Instruments, 2009).

The communication between such ECUs and sensor as well actuators are performed over the field bus. CAN, FlexRay and LIN are three of the most used field bus systems within the automotive domain. The following section is providing a brief overview of the CAN network, due to that this network is relevant for this work.

The Controller Area Network (CAN) is a serial communication technology, which is used to transfer data between ECUs, (Vector, 2010). It was introduced by Bosch in 1983 and standardised in 1994 (Standard, 1993). This CAN network connects maximal 32 CAN nodes, e.g. ECUs, through a physical transmission medium (CAN bus) with each other. The CAN bus consists of a unshielded twisted two-wire lines, CAN High (CANH) and CAN Low (CANL) to reduce the effect of external magnetic field. Furthermore, on each ends is a termination resistors (95 Ohm - 140 Ohm) set to avoid transient reactions by absorbing the energy of the data signals, ISO 11898-6 (ISO, 2013). CAN communication is gained through a CAN interface based on CAN-transceiver and CAN-Controller as seen in Figure 2-1. The CAN transceiver protects the CAN controller from overvoltage and handles the bus connection. The CAN controller contains all the necessary communication functions. A CAN Controller is often implemented as an on-chip module, therefore the communication between MCU and CAN controller is more reliable and faster (Vector, 2010).

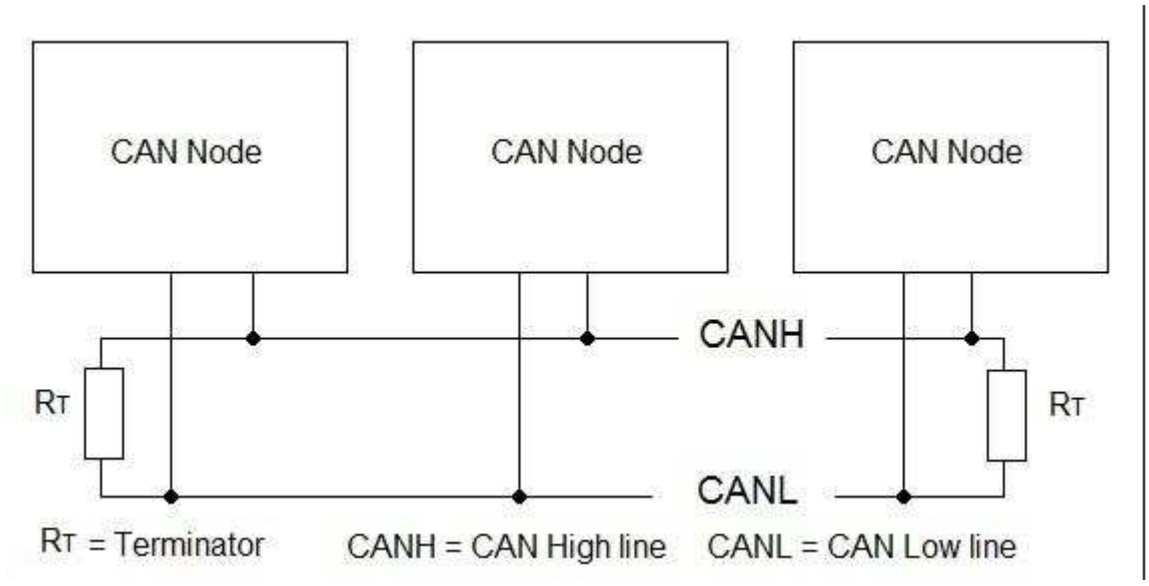


Figure 2-1: CAN network, (Vector, 2010)

CAN consists of two interfaces, high-speed CAN and low-speed CAN. High-speed CAN supports a data rate up to 1 Mbit/s and is used in safety critical applications like engine control unit. While Low-speed CAN supports a data rate up to 125 kbit/s and is used in non-safety critical applications like comfort services e.g. air conditioner. In additional the low-speed CAN does not require a specific termination resistor due to the reduced data rate data rate (Vector, 2010). The transmission of data between the CAN nodes is based on different voltages, which are distinguished by a dominant and recessive bus level. If therefore more than one CAN node sends CAN messages simultaneously through the CAN bus, the one with a dominant bus interface will overwrite the recessive bus levels. In the CAN network system every CAN node is authorized to place CAN messages on the bus and every CAN message is available for every CAN node. The CAN network provides three types of frames. Each CAN message is sent in a data frame. An error is acknowledged in an error frame and if a CAN node request data a remote frame is send.

2.3. AUTOSAR the approach of the automotive sector

The first software controlled application in vehicles, where written mostly in assembly language on a single microprocessor, (Knutsson, 2010). Therefore both hard- and software were tightly connected together. Also the hard- and software were provided from the same vendor supplier. Moreover, each function was implemented on its own ECU, (Knutsson, 2010). This ECU implementation strategy has led to increased software complexity. In addition, the number of ECUs increased as more and more functionalities had to be implemented on ECUs. Due to the high consumption of space and an increased fault rate, automotive manufacturers begin to merge functionalities from two ECUs into one to reduce the number of ECUs.

With the demand for more structure and reusability became apparent in the automotive industry, standardisation consortiums such as OSEK/VDX were introduced. OSEK/VDX was established 1994 from two joint projects, OSEK and VDX, to standardise embedded real-time operating systems for ECUs, (OSEK/VDX, 2005). This aim is continued and expended to the whole ECU software by “AUTomotive Open System Architecture” (AUTOSAR). Automobile manufacturers, suppliers and tool developers founded AUTOSAR in order to manage the software complexity in vehicles in the year 2003, (AUTOSAR, 2003). The main goal of AUTOSAR is the definition of uniform ECU software architecture and methodology.

The following section is first presenting the software architecture of AUTOSAR 3.2 for single-core ECUs. Furthermore, it describes briefly the AUTOSAR OS and illustrates the AUTOSAR methodology. Finally, the major communication mechanisms of the RTE are described.

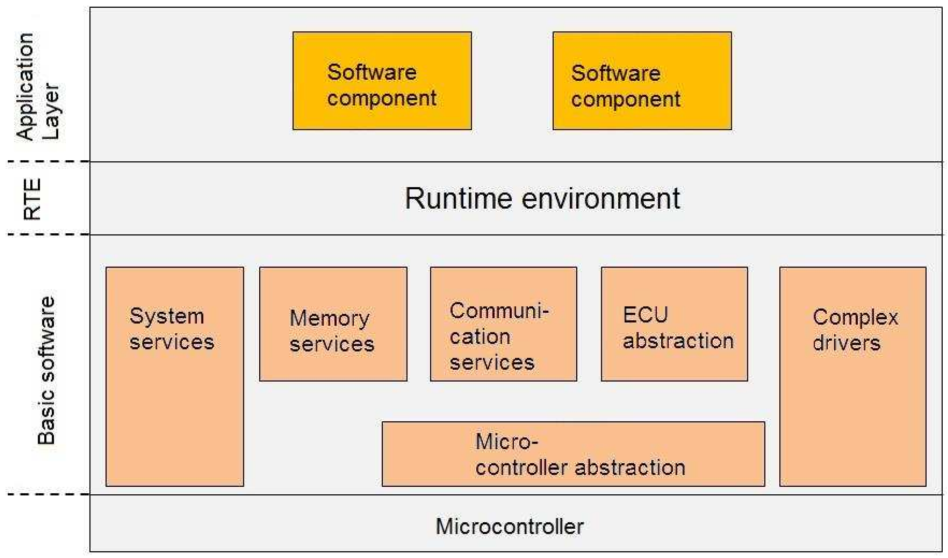


Figure 2-2: AUTOSAR ECU software architecture, (AUTOSAR, 2003)

The AUTOSAR software architecture is divided into four main layers, which are executed on a microcontroller, as shown in Figure 2-2, and described as follows:

Application software

The first layer is the application layer, which includes a set of ECU functionalities that are abstracted in software components (SWC). Each SWC contains one or more runnable. Runnables are small code fragments that call the standardised communication interfaces provided by the RTE. A runnable can be triggered through events or periodically.

Runtime environment

The second layer, the runtime environment (RTE), provides communication interfaces to the application layer, (Francisco & Fons, 2012). The isolation software application from the hardware dependent basic software, through the RTE ,provides the flexibility to exchange SWC without the consideration on the ECU hardware.

Basic software

Beneath the RTE is the basic software (BSW). The basic software contains BSW modules such as:

- the operating system (OS)
- services that handle different background e.g. memory and flash management
- communication that includes the communication via CAN and other networks
- microcontroller abstraction layer that abstracts hardware dependant features to avoid the direct access to the hardware

ECU hardware

At the bottom is the ECU hardware, which abstracts the MCU and some standard hardware peripherals, (AUTOSAR, 2003).

The layered architecture and defined interfaces between these layers reduces the software complexity by separated the hardware dependent component from the software, hardware independent, (Fons & Fons, 2012). In addition, it also provides the ability to reuse soft- and hardware components between different vehicle platforms.

2.3.1. The AUTOSAR OS

The Operating system of desktop PCs schedules each process of the BSW. In comparison to such operating systems, the operating system for automotive architecture provides only the services to schedule tasks, manage resources and events, (AUTOSAR, 2014a). Furthermore, the scheduling of the whole BSW is managed by the BSW scheduler, which will be explained on a practical example in section 6 on page 52, (AUTOSAR, 2008b). The following section describes briefly some major aspects of the AUTOSAR OS.

A task is the smallest schedulable framework of the AUTOSAR OS in which context the runnables are called (AUTOSAR, 2012a p. 81). Runables represents the implementation of a function in a SWC in the

Application layer. The AUTOSAR OS follows the static priority scheduling policy, where each task is assigned with static priorities, (AUTOSAR, 2014a). Furthermore, tasks can be classified as pre-emptive or non pre-emptive. AUTOSAR OS handles concurrent access to shared resources through the priority ceiling protocol (PCP), (Sha, 1990). This means that the task that operates with the shared resource is provided with the highest priority of all running tasks. In this way no other task is able to get the CPU to occupy the share resource, (Sha, 1990). AUTOSAR OS provides two different types of tasks: basic task and extended task, (AUTOSAR, 2014a). A basic task terminated after completion, whereas an extended task is able to wait for an event without termination in the state “WAIT”. Figure 2-3 on page 25 illustrates the possible states of both tasks types.

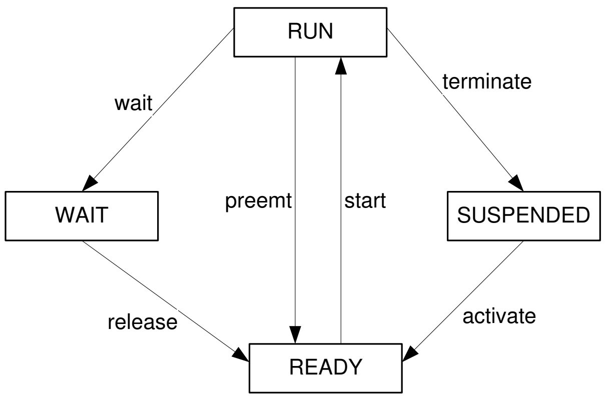


Figure 2-3: Task states of extended and basic tasks, (AUTOSAR, 2014a)

RUN	The task is currently being executed.
READY	The task has completed preparations for running and is ready to be executed
WAIT	The context of the task is saved and it waits until it is released to be transferred to the state “READY”.
SUSPENDED	The task is inactive in this state.

Tasks and therefore the included runnable calls can be activated periodically of by an event. The runnables contains the different communication interfaces that are provided by the RTE. Two of these communication interfaces are explicit and implicit communication. The explicit communication provides function that allows direct access, write and read, to the shared resource, here data. Whereas, in the implicit communication the data is modified from the RTE after the writing process is completed, (AUTOSAR, 2011). An implementation of these two communication types is demonstrated in section section 6.3 on page 58.

2.3.2. AUTOSAR methodology

Not only the ECU software architecture was standardized, also the process to implement the software is defined through AUTOSAR. The implementation of the ECU software is defined in a work product flow, the methodology. This methodology is divided into three configuration layouts: System configuration, ECU configuration and Component configuration, (AUTOSAR, 2008a). Each of these layouts begins with a configuration description in XML format and each configuration layout provides the necessary input for the upcoming layout.

The system configuration contains information about which software components are distributed on which ECU. The ECU configuration contains the configuration of the BSW and RTE. Furthermore, the software configuration contains the implementation of the runnables. These configurations are used to generate the source code that is executed on the ECU. The generation is provided by AUTOSAR development tool, see section 5.3.2 on page 49.

2.4. Multi-core microcontroller

The increase of functionalities that have to be implemented on a single-core ECU not only leads to an increase of software complexity, it also increased the need for more performance. This performance need was achieved through the improvement of the clock frequency of the processor, which caused an increase of power consumption. The additional power is generating heat that decreases the reliability and shortens the longevity of the ECU. Therefore, this mechanism to gain more performance got too expensive, (Freescale Semiconductor, 2009).

A solution to this problem is the multi-core processor, which reduce the design complexity of ECUs, (Senthilkumar & Ramadoss) . In addition, the design costs for an microcontroller with multiple number of cores is lower than to develop strategy to improve the performance of single-core microcontrollers, (Grave, Embedded Multi-Core Conference). In general multi-core microcontrollers has two types homogeneous and heterogeneous multi-core microcontroller, (Freescale Semiconductor, 2009).

Homogenous multi-core microcontroller has identical processors with the same features and instruction sets. Whereas, heterogeneous multi-core microcontroller has multiple processors with a set-up of different features and instruction sets, (Freescale Semiconductor, 2009). Heterogeneous multi-core microcontrollers provide the ability to merge functionalities that require less performance onto the processor that has less features.

Therefore, the automotive industry preferences the heterogeneous microcontroller, because of their efficient coverage of workload demand, (Wei, Qiu, Young, & Chang, 2011).

3. Literature review

This section highlights the challenges of multi-core in automotive with a focus on inter-core communication. It also presents related work and possible mechanism to improve inter-core communication for multi-core systems in automotive.

3.1. Multi-core challenge

Although the multi-core systems have become the norm for desktop computers, the development of multi-core real-time solutions in automotive is still in the early stages. As (Grave, Embedded Multi-Core Conference) presented at the Embedded Multi-Core Conference 2015 is the automotive industry at least five to eight years behind the technology used in IT systems. One possible reason for this regression are the requirements of embedded systems such as safety, reliability and real-time behaviour, see section 2.1 on page 20. Hence, the amount of performance improvement from the use of multi-core system is strongly dependant on the ability to run the legacy code parallel (Rudolf Grave 2014), the identification of possible parallel running code sections (tasks) is one of the main challenges for the automotive sector.

(Siebert, 2010) pointed out that source code that has 25 % of sequential code would decrease the performance gain of multi-core systems by a factor of four. Furthermore, new features such as advanced driver assistance systems increase the number of instructions and the complexity of the ECU software.

Therefore, it became essential to develop automated partitioning and mapping tools. Several studies were conducted to avoid the manual migration from single-core software to multi-core platform. Even projects such as the AMALTHEA platform (Amalthea, 2016), which is funded from a Europe-wide research and development inter-governmental network initiative the ITEA 2 (ITEA, 2016) was established in 2011 to face the challenge of parallelism in embedded multi-core systems by developing an open source tool platform for automotive embedded-system engineering.

One of the conducted criteria to gain an efficient task mapping to the additional core is the identification of the frequency and quantity of data transfer between the tasks (Höttger, Krawczyk, & Igel, 2015). These two aspects are significant, because an inefficient task allocation could require a high amount of data transfer, which would have an effect on the performance so that in the worst-case, the task would miss their time constraints.

3.2. **Challenges of inter-core communication**

The automated analysis tools are one possible way to face the time overhead of data transfers between processor cores (inter-core communication). Another possible solution would be to improve the inevitable inter-core communication.

To be able to improve such communication, it is necessary to analyse the possible obstacles first. Inter-core communication that is realized through shared memory could cause concurrency issues. In a system where two tasks have access to a shared resource, it is necessary to prevent a task from reading this shared resource (SR), when the SR is still being modified by another task (mutual exclusion) to avoid data inconsistencies,(Lakshmanan, Bhatia, & Rajkumar, 2011).

Furthermore, operating in a system with a static priority scheduling policy such as AUTOSAR conform operating systems could result to deadlocks. Deadlocks can occur in a system that uses lock-based mechanism to avoid data inconsistencies, (Lakshmanan et al., 2011). For example, a deadlock could occur if a high prioritised task interrupts a lower priority task, before the lower priority task is able to realises the lock of the SR. A request for the SR from the higher priority task would lead to an endless waiting loop (starvation), because the SR is still hold by the low priority task, (Lakshmanan et al., 2011). Such remote blocking is caused through inefficient resource management and a not existing synchronisation strategy. The consequence of such implementations are time overhead that could lead to not obey the time constraints, which is not acceptable for safety critical systems such as electronic braking systems.

3.3. **Introduction to AUTOSAR 4.0**

As the need for multi-core microcontrollers became apparent in the automotive industry, AUTOSAR introduced the first version of multi-core support in version 4.0 released in 2012. (AUTOSAR, 2012b) AUTOSAR 4.0 is differentiating between two communication types for SWCs running on the same core (intra-core communication) and the communication between SWCs running on different cores (inter-core communication).

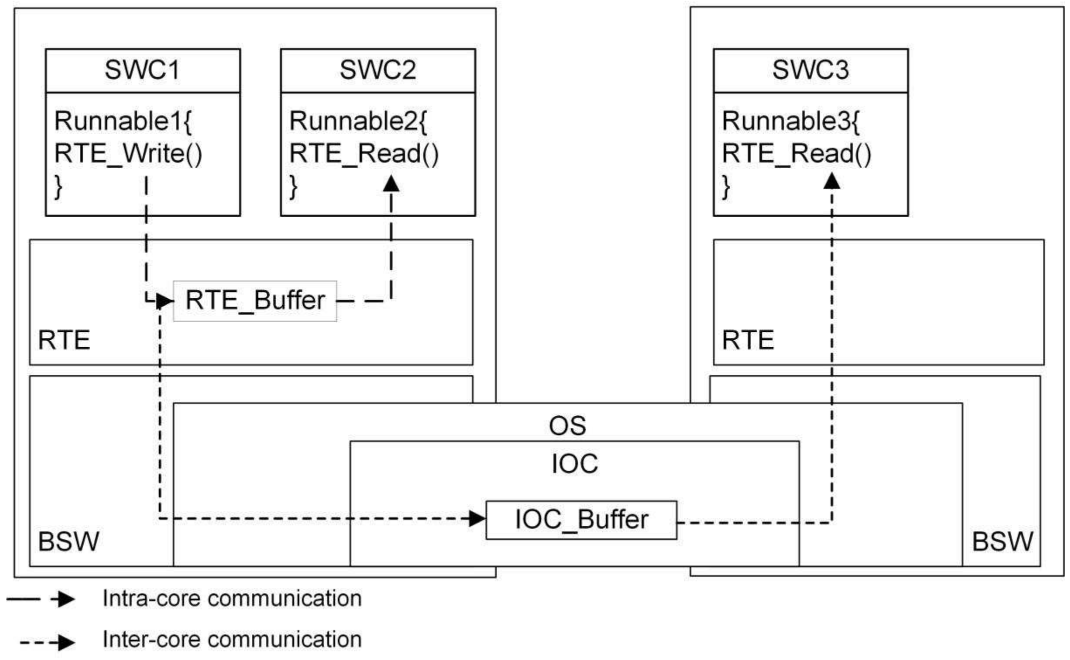


Figure 3-1: Intra-core and inter-core communication in AUTOSAR 4.0

Both communications are sharing particular resources between the tasks. In intra-core communication, the RTE is responsible to guarantee data consistency as described in section 2.3 on page 23. Figure 3-1 shows an abstraction of a possible implementation of intra-core communication according to AUTOSAR 4.0.

The multi-core OS in AUTOSAR is defined under the same configurations like AUTOSAR OS, but it operates on a different data structure (OS-Application) for each core with one operating system for all cores. An OS-Application is a collection of OS objects, e.g. task, resources, interrupt service routines and it is statically assigned to a core. Each of these objects has its own ID, which has to be unique across all cores (AUTOSAR, 2012c).

The inter-core communication is provided by the Inter- OS-Application Communication (IOC) and used by RTE. This way the upper layers of the software architecture will remain hardware independent by replacing the RTE communication buffer with the IOC buffer (AUTOSAR, 2011). AUTOSAR proposes to implement the IOC buffer as a share memory between the cores. Thereby, the IOC has to guarantee the data consistency, and is responsible for the notification for the other cores that new data is available after a successful write process.

Compared to intra-core communication could the PCP (priority ceiling protocol), see section 2.3 on page 23, not prevent deadlocks or mutual executions in multi-core processors since the highest priority just affects the task on the same core. For that reason, AUTOSAR recommends to use the so-called Spinlocks wrapped by a suspend-all-interrupt function. Spinlock is a lock mechanism, which polls a lock variable and performs a busy-wait[1] until it becomes available to enter the critical section[2] (AUTOSAR, 2014a).

Alternative to the PCP for intra-core communication is AUTOSAR suggesting several other mechanisms. Other possible mechanisms that guarantee data consistency are:

Sequential scheduling strategy
The write- and read-process will be executed in different time slots, so that no intervention is possible.

Task blocking strategy
This mechanism should avoid mutual pre-emption of tasks. Therefore, no task that has access to the critical section is allowed to pre-empt another task with the same access rights.

Copy strategy
In this mechanism each Buffer has an identical copy Buffer. In this case concurrent access for different tasks are possible (AUTOSAR, 2011).

The latest standard of AUTOSAR was released in 2015, AUTOSAR 4.2.1 and contained no new suggestions for inter-core communication. Moreover, it introduces new concepts for the data transfer through CAN, extended features for the infotainment systems and an extension for the basis software partitioning as presented from Mohamed Salah at the webinar “Migrating to AUTOSAR 4.2” (Salah, 2016).

3.4. Mechanism to improve inter-core communication

As AUTOSAR only provides a guideline for multi-core ECU developers, it is necessary to consider also other possible mechanism to avoid data inconsistencies, and still meet the time constrains as well as the ability to priorities safety critical functions over low priority tasks.
The following section is related to the possible bottlenecks of inter-core communication, such as synchronization strategy, resource management, notification and memory access.

[1] A time slot in which the processor is being busy without doing useful work
[2] The critical section is the piece of code, which produce incorrect result when executed concurrently

3.4.1. Synchronization strategy

The implementation of inter-core communication requires protection of shared resources to guarantee data consistency. (Zeng & Natale, 2011) discussed several protection mechanism by defining and evaluating the worst-case response time (WCRT) for each mechanism. The protection mechanism described in (Zeng & Natale, 2011) are:

Lock-based mechanisms

In a lock-based mechanism a task will block the SR through a lock as soon the SR is available. In case where the lock is hold by another task the requested task can be transferred to a waiting list or spin until the lock is released (Zeng & Natale, 2011), e.g. Spinlock defined in AUTOSAR.

Multi-processor extension of the Priority Ceiling Protocol

This mechanism is the multi-core extension of the Priority Ceiling Protocol (MPCP) defined in (Rajkumar, 1990). When a task fails to lock a SR, it will be added to a prioritized queue and ether be suspended, so that a lower prioritise task can be executed until the lock is released, or the task will continue spinning until the SR is available. When the task receives the lock for the SR, its scheduling priority will be raised to the sum of a priority higher than all normal tasks in the same core, and the highest priority of any task that has access to the SR.

Multi-processor stack resource policy

This lock-based mechanism is the multi-processor extension of Stack Resource Policy (MSRP) (Rajkumar, 1990), presented in (Gai, Lipari, & Natale, 2001), which is setting the task that holds the lock to non pre-emptive. Each task that fails to access the lock will be added to a FIFO queue and keeps spinning until the SR is available.

Wait-free mechanism

The wait-free mechanism is using a replication from the original communication buffer, which is comparable with the copy strategy mechanism from AUTOSAR. Through the replication a concurrent access is possible without causing data inconsistency (Zeng & Natale, 2011).

Evaluation of MCPC and MSRP

The evaluation in (Zeng & Natale, 2011) work is based on a theoretical scenario, where each core from a dual-core homogeneous system, executes three tasks with different priorities and shares two global SR. For the evaluation of the WCRT some parameters were defined such as a critical section that takes 0.5 millisecond to be executed.

The time analyse of H. Zeng showed that the MCPC mechanism was causing long remote blocking times because of the long critical section and 4 out of 6 tasks were not schedulable. While the results for MSRP mechanism show an improvement in remote blocking time, the application was not schedulable with this synchronisation strategy. Whereby, the wait-free mechanism did not lead to any remote blocking behaviour, but therefore it caused memory overhead through the additional buffer.

3.4.2. Resource management

In the previous section, we discussed several mechanisms to deal with failed requests for SR. In this section, we will focus on possible implementation to avoid data incoherence.

One of this mechanism is the explicit synchronization, which avoid data incoherence by executing the write- and read-process in different time slot (Zeng & Natale, 2011), which is compatible with the sequential scheduling strategy presented by AUTOSAR.

Compare-and swap
Another possibility is the compare-and swap (CAS) operation, which was used in the wait-free mechanism of (Zeng & Natale, 2011). This method uses atomic operation to compare the existing data in SR with an expected value. If the current data equals the expected value, then the current value will be updated with the new value . The atomic operation ensures that the body of the CAS function will be executed without any pre-emption from other tasks (Siebert, 2010).

Semaphore
Another method to CAS is a method called Semaphore designed by Edsger W. Dijkstra (Dijkstra, 1968), which is used in the MPCP and MSRP synchronisation strategy. A semaphore represents an operation, which locks the critical section for other tasks to modify the shared data in the critical section or to execute the same code segment (StMicroelectronics, 2013a).

Semaphore could be implemented as a shared counter, which content will be read whenever the shared resource has to be locked. If the value of the counter is positive (greater than zero), then the task will be able to execute the critical section and decrement the value. However, if the value of the semaphore equals to zero then the task has to wait until the task which hold the semaphore decreases the counter and therefore releases the lock (Alekseev, 2012).

3.4.3. Notification

The next possible optimization point for inter-core communication is the notification to acknowledge the read process that new data is available. A list of notification methods are discussed by E. W. Dijkstra (Dijkstra, 2001):

Non-blocking polling
One possible notification method is the non-blocking polling method, where the read process checks if new data is available. If new data was not written into the SR, the execution of the read process will be continued, (Devika K., 2013).

Blocking polling method

In contrast to the non-blocking polling method is the blocking polling method in which the receiver will be blocked until new data is available, (Devika K., 2013).

Interrupt-based notification

Whereby by the Interrupt-based notification the receiver will only be executed when it gets acknowledged through an interrupt that new data is available.

3.5. Conducted studies related to inter-core communication

With the introduction of the new version of AUTOSAR 4.0 several research studies have been conducted with a focus on inter-core communication.

One of these studies is the master thesis from A.S Moghaddam (Moghaddam, 2013). In his thesis, A.S Moghaddam investigates the possible overhead between intra-core communication and inter-core communication according to AUTOSAR 4.0 by implementing test scenarios on a heterogeneous microcontroller. The mutual exclusion was handled by using Spinlocks. It suggests a method to suspend all interrupts before requiring the SR, where for the notification an IVOR (Interrupt Vector Offset Register) was used. This study concluded that the inter-core communication is (56%) more expensive than intra-core communication.

The impact of intra-core and inter-core task communication on multi-core embedded systems is provided by Feljan (Feljan & Carlson, 2013). The focus of this research is the investigation of what kind of impact the allocation of software modules on the cores of a multi-core system has on the communication time. For this purpose J. Feljan and J. Carlson implemented a test scenario on an Intel Core 2 Duo E6700 processor (Intel, 2014), where each dual-core processor has a local L1 cache and a L2 cache shared between the two cores. The test scenarios consisted of two allocations using various data access pattern and several data sizes. The data size ranges from a size that fits into L1 cache, to a size that is too large for both L1 and L2 caches. The allocations include two tasks communicating on one core and two task distributed on different cores. In order to represent different data access patterns, the data was read from the buffer sequentially, non-sequentially and non-sequentially.

The results of this paper on one hand supports the conclusion of A.S Moghaddam (Moghaddam, 2013), that inter-core communication is 3 times slower than intra-core communication when the shared data fits into the local cache (L1) and when data is not read sequentially. On the other hand, it shows that if the transferred data is bigger than the cache size and therefore not fits in the local cache by reading it sequentially, that there is no significant difference between both communication types. These results were predictable by considering the fact that data which does not fit into the local cache has to be fetched through shared memory (L2 or RAM) and the access of RAM is slower compared to the access to L1. Shared L2 has a two or three times larger latency than L1 (Intel, 2014). This outcome proves that the number of available local caches and the way data is accessed has an important influence on the communication time.

3.6. Evaluation of previous work and gap in knowledge

It is evident that several methods can be used to improve the inter-core communication. However, the improvement of inter-core communication is not only dependent on one bottleneck. It is a combination of methods, which are optimally matched to one another for a particular purpose. The investigation of the related work has shown that MPCP and MSRP synchronisation with semaphore as lock-based method caused unacceptable time overheads and the wait-free mechanism with compare-and-swap method is favoured for the use for homogeneous embedded multi-core systems with the limitation of increased memory usage (Zeng & Natale, 2011).

Also, the AUTOSAR mechanism to avoid data corruption are not completely beneficial. The sequential scheduling strategy, for example, requires full understanding of the software, which is more than challenging since the ECU software got more and more complex since it was invented[3]. Also the task blocking strategy has its disadvantages as soon as safety-critical tasks occur that need to get access to SR. However, the copy strategy would require the largest amount of RAM memory compared to the other mechanism.

The main advantage of using multi-core microcontroller in ECUs is to join vehicle functionalities into less electronic control units. Processor manufacturers produce processors specialized for specific functions such as computationally-intensive operations or large amounts of signal processing, (Freescale Semiconductor, 2009). It would not be appropriate to execute a non-critical function on a processor designed for critical safety functions. A heterogeneous multi-core microcontroller can offer specialized cores for different applications, for example, to offload non-critical functions to less capable cores (Freescale Semiconductor, 2009). This ensures the most effective use of available resources. Therefore, it is questionable whether the same wait-free method presented by (Zeng & Natale, 2011) would have a positive impact on the communication time. Even if a practical implementation of the theoretical example would result in less time overhead of the wait-free method, the high amount of memory consumption would still lead to a rejection from ECU developer. Only implementations that satisfy the need for performance by reducing the costs at low power consumption will be produced.

Another bottleneck, which would influence the communication time, is the memory access. As presented by (Feljan & Carlson, 2013) a local cache on each core reduces the communication time difference between intra-core and inter-core communication. Hence, the test scenario of (Feljan & Carlson, 2013) was executed on a system that resembles processors for modern embedded systems, e.g. smartphones, but not for real-time systems like ECUs.

[3] Bosch released 1989 a hybrid control the ABS 2E generation (R. B. GmbH, 2008)

Therefore, it would be beneficial to put the theory analysed mechanism in (Zeng & Natale, 2011) into praxis and investigate the cause of possible high consumption of memory or low performance. In addition, the analyses of (Zeng & Natale, 2011) should be performed on test scenarios that are leaned on the real use case in automotive with a heterogeneous multi-core microcontroller. Heterogonous multi-core microcontroller provides the benefit to merge non-critical function on the additional core that is customized for such functionalities and is therefore performed to its full potential. In addition, these test scenarios should still be AUTOSAR conform or expand the suggested mechanism of AUTOSAR in a way that lead to improvements.

To be able to conclude possible speed up gain of a heterogonous multi-core ECU, it is necessary first to investigate the bottlenecks of AUTOSAR conform intra-core communication.
Therefore the aim of this project is to explore the current state of inter-core communication on multi-core ECUs. The analysis includes the design of test-scenarios (experiments) that are leaned on use cases in automotive. In additional, these experiments are not only suitable to evaluate inter-core communication, they also are designed to investigate the potential of intra-core communication.

The results that are gained after the implementation and execution of such experiments on a heterogonous multi-core ECU could then be evaluated according to their performance and memory consumption to assess whether the used methods fulfil the real-time requirements.

This contribution addresses the challenges of software for multi-core microcontrollers in automotive systems with the analysis of the methods used in intra-core communication.

4. Design of experiments according to automotive systems

The following section introduces the experiments, which are used to verify the intra-core communication regarding to performance and memory consumption. Firstly, a brief introduction into the automotive specific conditions are given. This introduction is then followed by the presentation of the experiments and their specific task mapping for each implementation.

The design of the experiments is based on the following conditions:

- Tasks trigger time constraint
  In their paper (Jena & Srinivas, 2012) verified the suitability of multi-core processors on the example of the Anit-Lock Braking Systems (ABS) . This paper demonstrates that the ABS functionality for single-core ECUs are divided into high, middle and low priorities. It also demonstrates that the tasks with the highest priority occur every millisecond, the tasks with a middle priority value occur every 5 milliseconds and the tasks with the lowest priority get called every 10 milliseconds.

- The different event types
  The AUTOSAR Specification of Timing divides events introduces three different categories: periodic triggered events, sporadic triggered events and events that are triggered through a pattern of different characteristics (length, minimum of occurrences) (AUTOSAR, 2014b p. 74 - 78). The experiments designed in this work focuses on the first two event types. The periodic events occur periodically. The events with a sporadic occurrence are used for example, to signify that the engine is overheated and the ECU should begin with the cooling process.

- The scheduling policy for two tasks with the same priority
  The AUTOSAR specification of the operating systems defines that tasks with the same priority should be called in order of their activation, (AUTOSAR, 2014a p. 82 - 83). This requirement is valid for the single-core as well as multi-core ECUs.

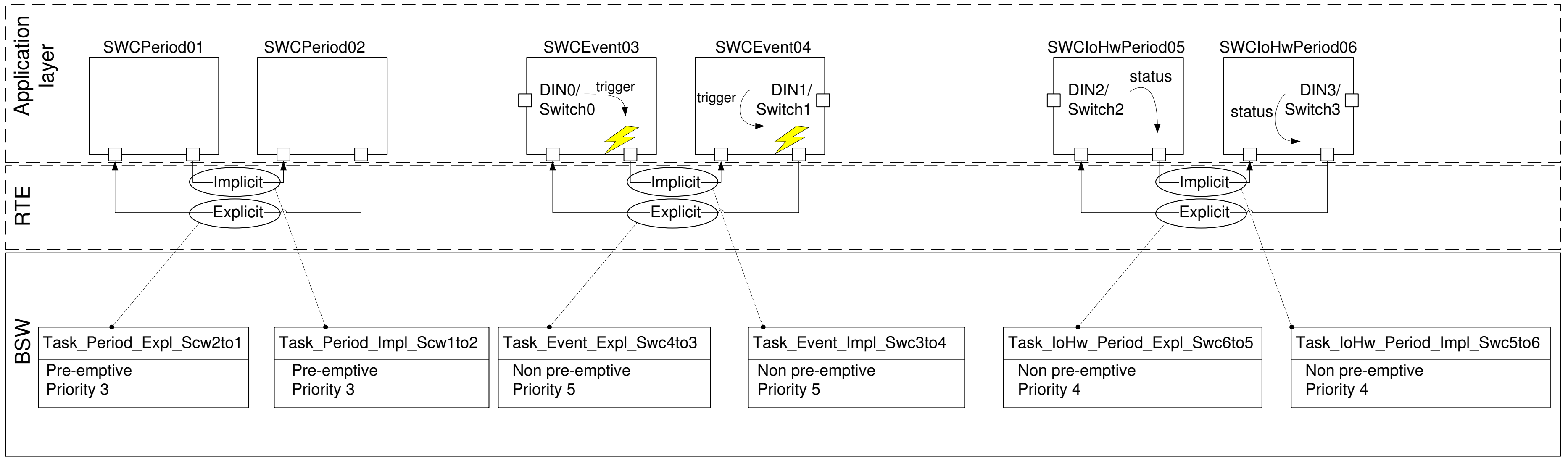


Figure 4-1: Graphical illustration of software components mapped to tasks of SingleCoreV1_0

As Figure 4-1 illustrates, the experiments of this work. These experiments consist 6 software components (SWC) and are realised in two implementations, SingleCoreV1_0 and SingleCoreV2_0. Two of these software components are communicating with each other whether through the implicit or the explicit communication, see section 2.3.1 on page 24. Each communication that begins from a software component, which is named with an odd number, communicates through the implicit communication. Whereby, each communication that begins from a software component, which is named with an even number, communicates through the explicit communication.

In the following, a description is giving for each pair of software components:

SwcEvent03 to SwcEvent04 and SwcEvent04 to SwcEvent03

SwcEvent03 and SwcEvent04 are exchanging a structure defined of three members of type integer and 8 members of type char. The communication between these software components is only executed when switch 0 for the communication between SwcEvent03 to SwcEvent04 and switch1 for the communication between SwcEvent04 to SwcEvent03 is closed. The overhead of the engine in a vehicle is simulated through the closed switch that triggers the communication.

SwcIoHwPeriod05 to SwcIoHwPeriod06 and SwcIoHwPeriod06 to SwcIoHwPeriod05

SwcIoHwPeriod05 and SwcIoHwPeriod06 are exchanging the status of the switch between each other with a data of type Boolean. The communication between these two software components simulates the reading process of a sensor and is not so safety critical as the previous case.

SwcPeriod01 to SwcPeriod02 and SwcPeriod02 to SwcPeriod01

SwcPeriod01 and SwcPeriod02 are exchanging the same data struct as SwcEvent03 and SwcEvent04 and are simulating an internal communication inside the ECU.

These communications are implemented as code segments inside the software components on the application layer, runnables. These runnables are called from the operating system through tasks. Therefore, it is mandatory to map these runnable to tasks of the operating system and to define the priority and scheduling policy of each task.

4.1. SingleCoreV1_0

The first implementation, SingleCoreV1_0, is designed for the single-core use, see Figure 4-1 on page 37. As a result, both read- and write-processes are mapped to the same tasks as followed:

Task_Event_Expl_Swc4to3 and Task_Event_Impl_Swc3to4

The task Task_Event_Expl_Swc4to3 contains the explicit write- and read-process of SwcEvent04 and SwcEvent03. Whereby, the task Task_Event_Impl_Swc3to4 includes the implicit write- and read-process of SwcEvent03 and SwcEvent04. Both tasks are non pre-emptive, because they are handling safety relevant mechanism. Due to the fact that an overheated engine represents a safety critical event, this event has to be handled immediately. Therefore, the communication between these software components not only has the highest software priority (5), the communication is triggered with the highest frequency of 1 ms.

Task_IoHw_Period_Expl_Swc6to5 and Task_IoHw_Period_Impl_Swc5to6

The task Task_IoHw_Period_Expl_Swc6to5 contains the explicit write- and read-process of SwcIoHwPeriod06 and SwcIoHwPeriod05. Whereby the task Task_IoHw_Period_Expl_Swc6to5 includes the implicit write- and read-process of SwcIoHwPeriod06 and SwcIoHwPeriod05. Both tasks are non pre-emptive. In addition, these tasks have a priority of 4 and are triggered each 5 milliseconds.

Task_Period_Expl_Scw2to1 and Task_Period_Impl_Scw1to2

The task Task_Period_Expl_Scw2to1 and Task_Period_Impl_Scw1to2 contains the implicit write- and read-process of the software components SwcPeriod02 and SwcPeriod01. These tasks are pre-emptive, because they contains the communication with the lowest priority. The communication between these software components have the lowest safety critical aspect and have therefore the lowest priority of 3 and are triggered each 10 milliseconds.

Such communication, where write- and read-process is combined in one task is also named as intra-task communication.

4.2. SingleCoreV2_0

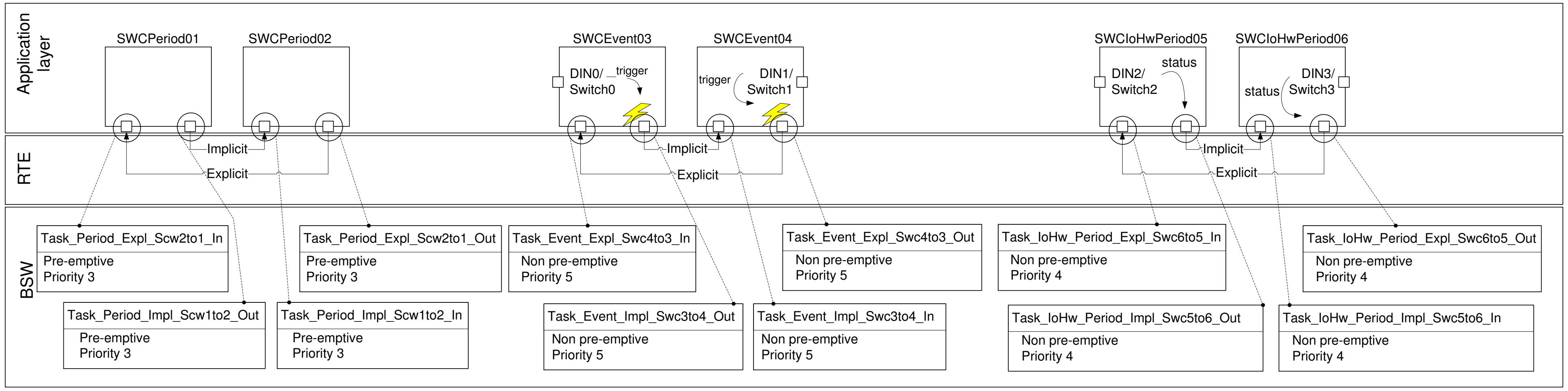

Application layer
RTE
BSW
SWCPeriod01
SWCPeriod02
SWCEvent03
SWCEvent04
SWCIoHwPeriod05
SWCIoHwPeriod06
DIN0/ Switch0
trigger
DIN1/ Switch1
trigger
DIN2/ Switch2
status
DIN3/ Switch3
status
Implicit
Explicit
Implicit
Explicit
Implicit
Explicit
Task_Period_Expl_Scw2to1_In Pre-emptive Priority 3
Task_Period_Expl_Scw2to1_Out Pre-emptive Priority 3
Task_Event_Expl_Swc4to3_In Non pre-emptive Priority 5
Task_Event_Expl_Swc4to3_Out Non pre-emptive Priority 5
Task_IoHw_Period_Expl_Swc6to5_In Non pre-emptive Priority 4
Task_IoHw_Period_Expl_Swc6to5_Out Non pre-emptive Priority 4
Task_Period_Impl_Scw1to2_Out Pre-emptive Priority 3
Task_Period_Impl_Scw1to2_In Pre-emptive Priority 3
Task_Event_Impl_Swc3to4_Out Non pre-emptive Priority 5
Task_Event_Impl_Swc3to4_In Non pre-emptive Priority 5
Task_IoHw_Period_Impl_Swc5to6_Out Non pre-emptive Priority 4
Task_IoHw_Period_Impl_Swc5to6_In Non pre-emptive Priority 4

Figure 4-2: Graphical illustration of software components mapped to tasks of SingleCoreV2

The second implementation, SingleCoreV2_0, is designed for multi-core use as illustrated in Figure 4-2. This implementation contains the same defined communication as the first implementation. SingleCoreV2_0 only differs from SingleCoreV1_0 in the fact that each read- and write-process is mapped to a separate task.

The AUTOSAR 4.0 specification of the operating system requires that OsApplications are distributed to processor cores instead of the tasks itself, (AUTOSAR, 2014a). Therefore are the write- and read-processes separated in individual tasks as followed:

Task_Event_Expl_Swc4to3_Out and Task_Event_Expl_Swc4to3_In
The write- and read-process of task Task_Event_Expl_Swc4to3 is separated in task Task_Event_Expl_Swc4to3_Out and task Task_Event_Expl_Swc4to3_In. Task_Event_Expl_Swc4to3_Out includes the explicit write-process, which is triggered each millisecond with a priority of 5 and non pre-emptive. Whereby, Task_Event_Expl_Swc4to3_In contains the explicit read process, which is triggered as soon as new data is available.

Task_Event_Impl_Swc3to4_Out and Task_Event_Impl_Swc3to4_In
The write- and read-process of task Task_Event_Expl_Swc3to4 is separated in task Task_Event_Expl_Swc3to4_Out and task Task_Event_Expl_Swc3to4_In. Task_Event_Expl_Swc3to4_Out includes the implicit write-process, which is triggered each millisecond with a priority of 5 and non pre-emptive. Whereby, Task_Event_Expl_Swc3to4_In contains the implicit read-process, which is triggered as soon as new data is available.

Task_IoHw_Period_Expl_Swc6to5_Out and Task_IoHw_Period_Expl_Swc6to5_In
The write- and read-process of task Task_IoHw_Period_Expl_Swc6to5 is separated in task Task_IoHw_Period_Expl_Swc6to5_Out and task Task_IoHw_Period_Expl_Swc6to5_In. Task_IoHw_Period_Expl_Swc6to5_Out includes the explicit write-process, which is triggered each 5 millisecond with a priority of 4 and non pre-emptive. Whereby, Task_IoHw_Period_Expl_Swc6to5_In contains the explicit read-process, which is triggered as soon as new data is available.

Task_IoHw_Period_Impl_Swc5to6_Out and Task_IoHw_Period_Impl_Swc5to6_In
The write- and read-process of task Task_IoHw_Period_Impl_Swc5to6 is separated in task Task_IoHw_Period_Impl_Swc5to6_Out and task Task_IoHw_Period_Impl_Swc5to6_In. Task_IoHw_Period_Impl_Swc5to6_Out includes the implicit write-process, which is triggered each 5 millisecond with a priority of 4 and non pre-emptive. Whereby, Task_IoHw_Period_Impl_Swc5to6_In contains the implicit read -rocess, which is triggered as soon as new data is available.

Task_Period_Expl_Scw2to1_Out and Task_Period_Expl_Scw2to1_In
The write- and read-process of task Task_Period_Expl_Scw2to1 is separated in task Task_Period_Expl_Scw2to1_Out and task Task_Period_Expl_Scw2to1_In. Task_Period_Expl_Scw2to1_Out includes the explicit write-process, which is triggered each 10 millisecond with a priority of 3 and pre-emptive. Whereby, Task_Period_Expl_Scw2to1_In contains the explicit read-process, which is triggered as soon as new data is available.

Task_Period_Impl_Scw1to2_Out and Task_Period_Impl_Scw1to2_In
The write- and read-process of task Task_Period_Impl_Scw1to2 is separated in task Task_Period_Impl_Scw1to2_Out and task Task_Period_Impl_Scw1to2_In. Task_Period_Impl_Scw1to2_Out includes the explicit write process, which is triggered each 10 millisecond with a priority of 3 and pre-emptive. Whereby, Task_Period_Impl_Scw1to2_In contains the explicit read-process, which is triggered as soon as new data is available.

Such communication, where write- and read-process is distributed in separated tasks is also named as inter-task communication.

5. Test and development environment

The following section describes the development environment of this project. The first section presents the requirements for the development environment and the verification of possible targets. The second part includes a detailed description of the software and hardware environment.

5.1. **Verification of possible targets**

In order to analyse the bottlenecks of AUTOSAR conform intra-core communication, suitable software and hardware are required.

For this project, the following aspects are essential:

The hardware has to:

- be equipped with a heterogonous multi-core microcontroller, which provide features as used in the automotive sector. Such features are summarised in section 11.3 on page 130.
- the development board on which the multi-core is integrated on, should be as similar as possible to the ECUs used in industry with network connection such as CAN, see section 2.2 on page 20.
- be versatile that many automotive projects could be carried out on it.

The software has to be at least AUTOSAR 3.2 and ideally, AUTOSAR 4.2.

	Experiment setup of (Moghaddam, 2013)	Experiment setup of (Feljan & Carlson, 2013)	Experiment setup of the current project
Hardware requirements			
Microcontroller	MPC5510EVB evaluation board with an MPC5517E microcontroller (Semiconductors, 2014)	Intel Core 2 Duo E6700 processor (Intel, 2014)	VC121-12 (Vector, 2014a)
Heterogonous multi-core microcontroller	✓	✗	✓
ECU similarities	$1/2$	✗	✓
Versatility	✓	✗	✓
Software requirement			
AUTOSAR 3.2	✗	✗	✓
AUTOSAR 4.0	$1/2$	✗	✗

Table 5-1: Comparison of different hardware environments

Table 5-1 shows the experimental setup of (Feljan & Carlson, 2013), which is the most inefficient hardware environment for the analysis of intra-core communication of automobile systems. The hardware used in (Feljan & Carlson, 2013) is provided with a homogenous microcontroller that is designed for the desktop PC. However, the hardware environment of (Moghaddam, 2013) seems more suitable for the work of this thesis. On the one hand it provides an heterogonous multi-core the MPC5517E microcontroller (Semiconductors, 2014) with several network connections (CAN, LIN) on the evaluation board MPC5510EVB (Semiconductors, 2007), as seen in Table 5-1. On the other hand, it only provides an AUTOSAR 3.1 and partial AUTOSAR 4.0 operating system.

The functionalities of version AUTOSAR 4.0 such as inter-core communication were "provided from Ecore" (Moghaddam, 2013 p.18) and are therefore not available for public use. In addition, the evaluation board MPC5510EVB does not provide analogue inputs or digital outputs to read signals, for example, from pressure sensors. Therefore, in limits its use in possible automotive projects. In comparison with the two presented environments is the universal ECU VC121-12 from Vector the device that is most similar to ECUs. It provides 20 digital inputs and 22 analogue inputs that can be used for several automotive projects, (Vector, 2014a). In addition, it features a dual-core heterogonous microcontroller with an AUTOSAR 3.2 operating system, which supports intra-core communication.

The VC121-12 is chosen for the investigation of intra-core communication based on the previous evaluation of possible development environment. The compatibility to the VC121-12 ECU is the decisive factor for the selection of appropriate AUTOSAR, compiling, debugging and analysis tools.

5.2. Hardware environment

The following sub section describes the hardware devices that are used through the project. These hardware devices are the universal VC121-12 ECU, the emulator IC5000 on-chip analyser and the development board STK600.

5.2.1. VC121-12 ECU

VC121-12 is a universal ECU for the automotive development sector. It contains a main controller, a system basis chip (SBC) and a safety controller (SC) as well as several communication ports such as inputs and outputs as seen in Figure 5-1.

Input and output ports

The VC121-12 ECU provides following input and output ports:

- 6 CAN interfaces
- 1 Ethernet interface
- 2 LIN interfaces
- 1 FlexRay interface
- 8 analog inputs to measure voltage between 0 and 5 V
- 6 analog impedance inputs for measuring resistances between 103 and 107 kΩ
- 8 analog inputs for measuring voltages between 0 and 18 V
- 20 digital inputs for switches
- 4 frequency inputs for digital signals, this signals may be high or low active and support frequencies up to 20 kHz
- 4 frequency inputs for reluctance sensors with a frequency up to 20 kHz
- 40 high-side outputs with differing current and PWM capabilities

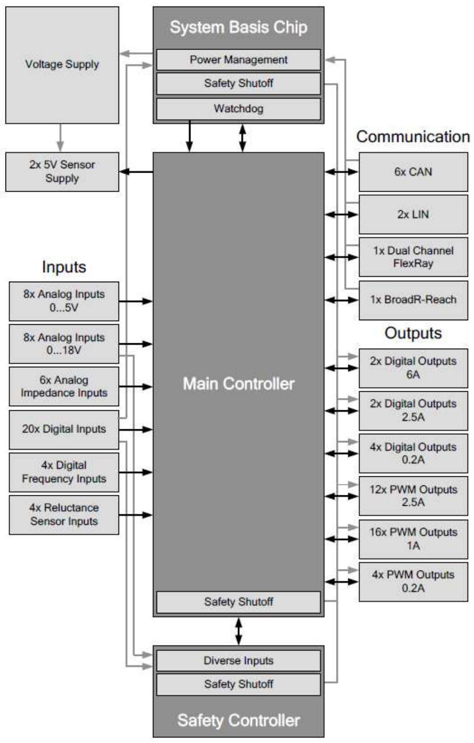


Figure 5-1: Block diagram of the VC121-12 ECU (Vector, 2011)

Four of the digital inputs (DIN0 – DIN3) are used for the conducted experiments in this project as described in section 6.1.4 on page 56. These digital inputs have a threshold of 4 V. If a switch is connected to one of these four digital inputs, then every voltage that is above 4 V will be accepted as "high" (switch is closed) and each voltage above 4 V up to 3.7 V is interpreted as "low" (switch is open). More information to the input and output ports can be found in (Vector, 2014b).

The main controller

The VC121-12 is equipped with a heterogeneous dual-core microcontroller SPC56EC64B3 (StMicroelectronics, 2013b) from StMicroelectronics as the main controller. The SPC56EC64B3 microcontroller is made for the main use in automotive vehicle body (StMicroelectronics, 2013a). It is a dual-core microcontroller with an e200z4d and an e200z0 processor as shown in Figure 5-2.

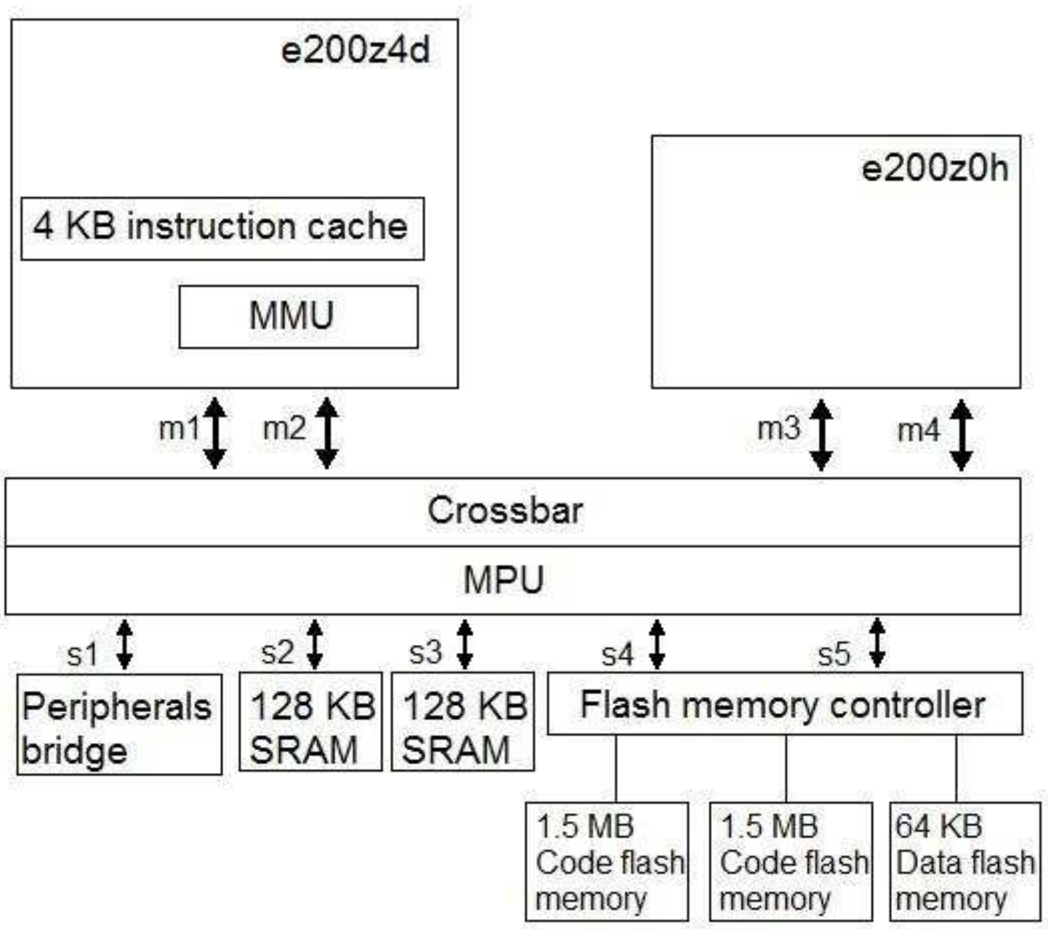


Figure 5-2: Block diagram of SPC56EC64B3 microcontroller

The e200z0 processor is one of the licensable e200 core of Freescale that had been open for licensing to other manufactures in April 2007 (JOSE, 2007). This processor is designed for embedded control application that does not require maximum performance. Therefore it is only able to execute one instruction per instruction cycle (single-issued) by 75 million instructions per second (MIPs). Additionally, it provides a 32-bit Power Architecture variable-length encoding (VLE) instruction set with 32-bit (GPRs) (F. Semiconductor, 2008).

By contrast, the e200z4d contains 64-bit general-purpose register, a double-issued architecture with an instruction cache of 4 KB, 200 MIPs and a Memory Management Unit (MMU), which is primary performing the translation of virtual memory addresses to physical addresses, (F. Semiconductor, 2008). As a result, the e200z4 processor is ideally suited for application that requires high performance.

Furthermore, both processors have two master (m) ports to the crossbar, one for instructions and one for data. The crossbar provides the ability to concurrently access modules, for instance, the peripheral bridge or memory through the slave (s) ports. Therefore, each master port has a unique priority through the registers of the Memory Protection Unit (MMU). The MMU evaluates each request, whereby the master port with the highest priority will get access to the slave ports.

Each slave port is assigned to a module. One of these modules is the memory, which is divided into SRAM and ROM (flash memory). The main controller is equipped with 256 KB SRAM in total, which is separated into 128 KB blocks. The flash memory is partitioned into two 1.5 MB code flash memory models and one 64 KB data flash memory. The access to these flash memory modules are controlled by the flash memory controller. Whereas, the access to the output pins of the main controller is guaranteed through the peripheral bridge (F. Semiconductor, 2008).

The main controller represents the head of the VC121-12 ECU. It executes the application and handles all inputs, outputs and communication channels (CAN, LIN, FlexRay and Ethernet). Some of this communication channels are connected straight to the main controller and other communicates with the main controller over a Serial Peripheral Interface (SPI) interface. The SPI interface is used to communicate between two microcontrollers or such as in this case, between a microcontroller and one or more peripheral devices, (ISO, 2004). This work will only focus on the CAN network as this communication channel is needed for power management.

Safety controller

The safety controller, STM8AF62 from StMicrocontrollers (StMicroelectronics, 2016) is a 8-bit controller that monitors the main controller and redundantly checks the input signals to verify their values. It also supervises the supply voltage and is able to set the output ports of the ECU to the state '"off" (Vector, 2011) .

System basis chip

To reduce the power consumption in vehicles, for instance when the vehicle is parking, it is necessary to turn the unused ECUs "off". Therefore, it is essential to define power modes for ECUs such as "sleep" mode or "wake-up" mode. For this reason, a System Basis Chip had been switched between the voltage supply of 12 V and the main microcontroller to control the voltage supply to the main microcontroller as seen in Figure 5-1 on page 44. In general, when the VC121 ECU is in "sleep" state, only the SBC is powered on to be able to observe all "wake-up" sources, (V. I. GmbH, 2011b). If a "wake-up" event occurs through an external interrupt such as CAN messages, the information of the "wake-up" reason are saved from the SBC and the main microcontroller will be supplied with 5 V.

The main microcontroller monitors this “wake-up” reason and switches to “sleep” mode as soon as the “wake-up” signal expires (Vector, 2011 p.21). A “wake-up” event is only valid if a particular pattern is recognized by the SBC. In order to wake-up the main controller a series of three dominant pulses with a duration longer than 500 ns in a time slot of 120 μs (N. Semiconductor, 2014 p. 57) has to be detected. A “wake-up” with a validation of the CAN pattern will only occur when the ECU has previously entered the “sleep” mode., which does not occur during the experiments. Therefore, it is not necessary to generate such patter for this work. Furthermore, the flashing process via a debugger requires permanent communication on the port CAN0. As a result, CAN messages must be provided during the flash process. A detailed description of procedure can be seen in section 6.1.4 on page 56.

5.2.2. The emulator IC5000 on-chip analyser

After the software has been designed and implemented, it is necessary to verify its correct or incorrect execution through debugging. Debugging is the ability to verify executive software by for instance, break pointing or single stepping through the application (Rouse, 2007). The debug mechanism is implemented as a program for software that is written for the use on PC. This dubug mechanism runs on the same PC on which the software has been implemented. Due to the limited resources of embedded systems, embedded real-time systems requires different debugging procedures compared to desktop PCs. Therefore, embedded systems are debugged by using remote debugging, where the debugging software is running on a host PC which controls the target through hardware (Rath, 2008).

Traditionally, an in-circuit emulator (ICE) is used to debug the developed software application. This ICE is plugged to the target MCU and it emulate all integrated functionality of the target MCU in real-time (Soffel, 2004). Today’s modern microcontroller integrate the debug functionality, on the chip itself so that the emulators are replaced by on-chip debuggers.

One of such on-chip debugger is the IC5000 analyser provided by iSystem, which supports the main controller used in VC121-12 ECU. The IC5000 analyser represents the interface between the analysis tool WinIDEA (on the PC) and the target VC121-12 ECU. The WinIDEA software provides not only the ability to upload (flash) the compiled software application into the target MCU, but it also supports real-time debugging and memory access as well as functionalities to trace the execution time of each running function with an accuracy of 10 ns (timestamps) and execution coverage (iSystem, 2009) . A detailed description of these functionalities can be found in section 5.3.4 on page 50.

5.2.3. STK600 development board

The STK600 is a development starter kit from Atmel for 8-bit and 32-bit AVR microcontroller and it is used in this work to generate the necessary CAN messages for the power management. It is set up with the STK600-TQFP64 socket (Atmel, 2016) and the 8-bit microcontroller AT90CAN128 (Atmel,

2008) . A detailed description of how this device is used in this work can be found in section 6.1.4 on page 56.

5.3. Software environment

This sub section describes the tools used to implement and evaluate the designed software. It begins with a description of the VC121-12 software, followed by the AUTOSAR development tools with which the source codes of the experiments have been generated. Afterwards, the compiler is presented, followed by the analyse tool WinIDEA. Finally, the software tool chain is illustrated.

5.3.1. VC121-12 software

The software of the VC121-12 ECU consists not only of the basis software of Vectors (MICROSAR) that is based on AUTOSAR 3.2 architecture; it also provides an input and output function library (I/O Function Library) to control all the communication ports of the hardware as seen in Figure 5-3. One component of the I/O library is the VC_IoHwAb, which provides interfaces to read and write out of the provided pins of the VC121-12 ECU, which has been used through this project to read the status of digital inputs. All these components of the VC121-12 ECU are generated through AUTOSAR development tools, which will be described in more detail in section 5.3.2 on page 48.

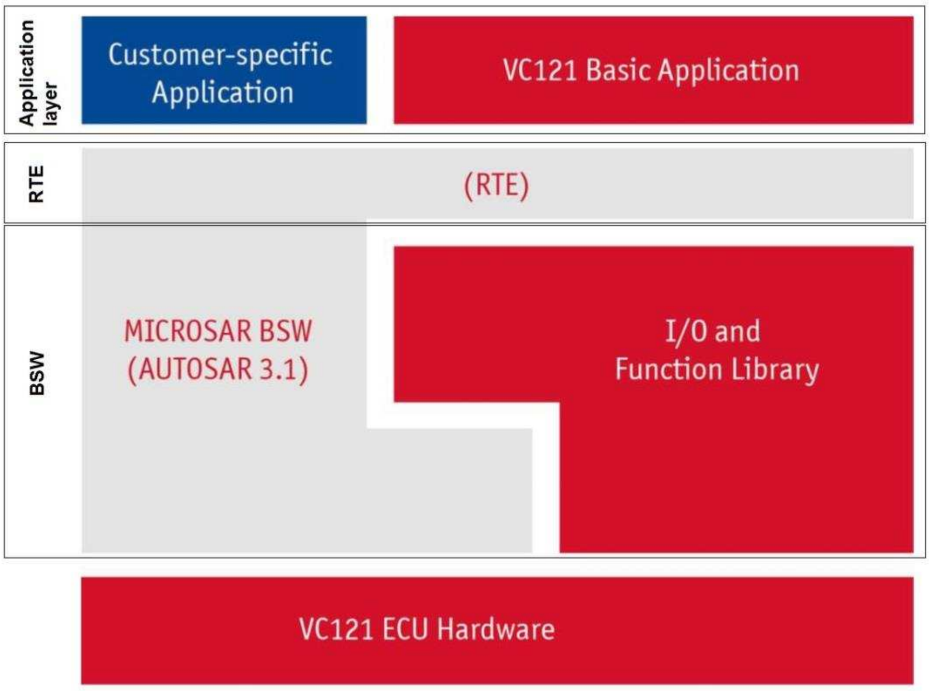


Figure 5-3: Overview of the software architecture of VC121-12 ECU, (Vector, 2012)

5.3.2. AUTOSAR development tool

The AUTOSAR methodology described in section 2.3 on page 23 is generated in three configuration tools. DaVinciDeveloper, DaVinciConfigurator and Geny.

The DaVinciDeveloper is used to design and generate the source code of the application layer and the RTE (.c and .h files) such as software components and runnables.
Whereas, the tool DaVinciConfigurator Pro is used to configure and generate all non-communication modules of the basic software layer.

The Geny tool supports the configuration of all the communication BSW modules such as FlexRay, LIN and CAN and defines the communication port for the “wake-up” mechanism. Through this project, two different configurations and implementation of the AUTOSAR RTE and application layer have been generated with all these three tools. A detailed description of these AUTOSAR development tools are can be found in section 6.1.4 on page 56.

5.3.3. Wind River Diab compiler 5.8

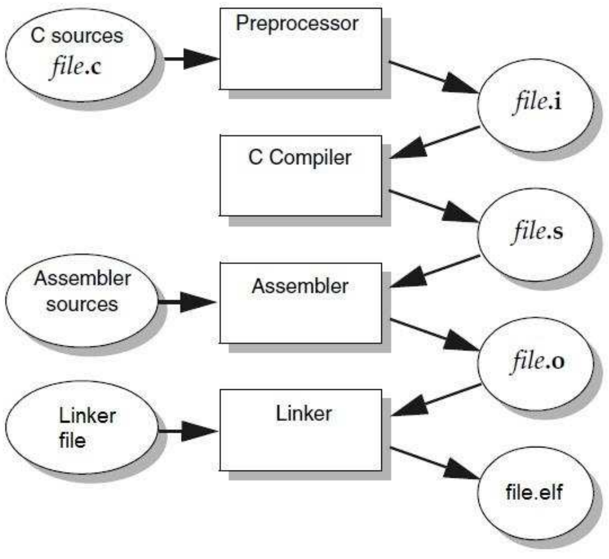


Figure 5-4: WindRiver Diab Compiler workflow

The WindRiver Diab compiler is an embedded cross-compiler that supports the main microcontroller of the VC121-12 ECU. Figure 5-4 shows the WindRiver Diab compiler workflow. It includes the pre-processor, assembler and linker in addition to the C compiler. Each stage of the processes provides the necessary input for the upcoming process. The pre-processor, for instance, parses the source code for macros and replaces them with the statements defined in the header files. The C compiler then translates the output files of the pre-processed source to assembly code (object files) that is transferred into machine code as the output from the assembler.

Before the final executable file *.elf is produced, the linker reads the defined sections out of the object files and maps those sections to memory space based on the instructions in the linker file. The linker file includes all information of the available RAM and ROM memory and where each section has to be placed (WindRiver, 2010 p. 311) . It is the responsibility of the programmer to create such linker file. The output file of the linker .map subsequently includes detailed information of the RAM and ROM consumption of each defined variable and function in the program .The memory consumption is based on this .map file and is evaluated in section 7.3 on page 100.

5.3.4. **Analysis tool** WinIDEA

WinIDEA is an Intergrated Development Environment (IDE) and debugger software by iSystem, (iSystem). It is provided along with the IC5000 hardware platform, see section 5.2.2 on page 47 from iSystem. Throughout this project, the WinIDEA is used for both for debugging and analysing the experiments that are described in section 4 on page 36.

5.3.4.1.1. Program trace

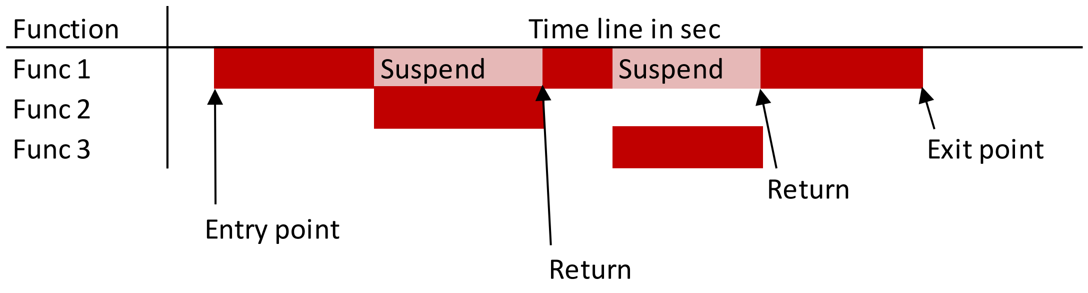


Figure 5-5: Abstract visualisation of the call - timeline

The performance analysis in section 7.2 on page 91 was based on the execution profiling (program trace) through the on- chip debugger IC5000. This program trace provides the ability to identify not only the execution time deviations, but also the program functions. To gain such a program trace, the profiler of WinIDEA records the executions of all instructions during runtime by recording the instruction address and the time of its execution (entry and exit point), (iSYSTEM, 2012).

The WinIDEA software will stop recording data manually (on user demand) or after the profiler buffer of the hardware component IC5000 becomes full. All of this information is then visualised in a call-timeline, where each function execution is visualised as bars that get interrupted when sub functions are called. Figure 5-5 shows the abstraction of a typical trace timeline. To be able to identify the entry and exit point of a function, it is necessary to know all sub function calls inside the measured function (Func1). Only with this knowledge, it is possible to identify task switches or occurred interruptions. Such interruptions should not be included into the execution time of Func 1.

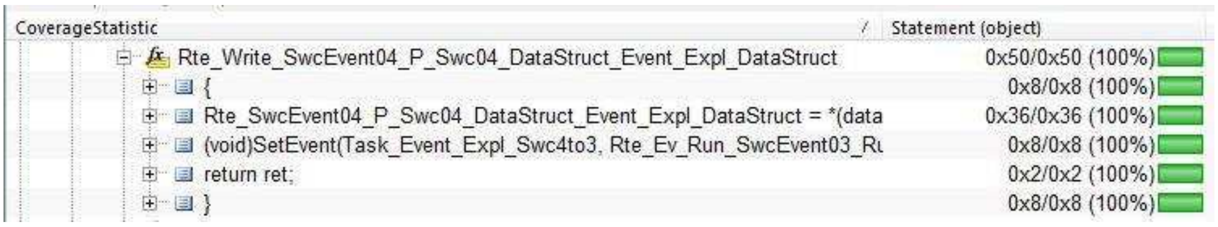
The example in Figure 5-5 does not include a task switch or interrupt, so that each sub function (Func2 and Func3) of Func1 is called and the execution time of Func 1 can be defined as the time between the entry and exit point. If an interrupt or task switch occurs during the execution of Func 1, then it is illustrated in the trace as a function call to an unexpected function for example, a function call to an interrupt-service routine. The time that is spent to call and execute the unexpected function has to be taken of the execution time of Func 1. Such approach guarantees to measure the exact execution time of Func 1.

5.3.4.1.2. Real- time debugger

Functions that are implemented as macros (inline function), as in section 6.3 on page 58, cannot be detected through a program trace. For this case, the high-level debugger of WinIDEA is used. Through the features such as assembler, memory and variable inspections, it is possible to observe if data was written into the memory space.

5.3.4.1.3. Execution coverage

The execution coverage of the real-time analyser provides the ability to detect which memory addresses have been gathered and executed by the microcontroller, (iSYSTEM, 2012). Therefore, it records all addresses that are executed and can be visualised line by line.



CoverageStatistic	Statement (object)
Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	0x50/0x50 (100%) 🟩
{	0x8/0x8 (100%) 🟩
Rte_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct = *(data	0x36/0x36 (100%) 🟩
(void)SetEvent(Task_Event_Expl_Swc4to3, Rte_Ev_Run_SwcEvent03_Rι	0x8/0x8 (100%) 🟩
return ret;	0x2/0x2 (100%) 🟩
}	0x8/0x8 (100%) 🟩

Figure 5-6: Coverage Statistic of function Rte_Write_SwcEvent04

Figure 5-6 shows such visualisation, where the column statement (object) represents the statement coverage on object code level in memory accessible units, (ISystem, Coverage). The first number defines the number of addresses that are executed, and the second number defines the number of addresses that are detected. This real-time analyse feature is used to detect the parts of a function, which has a main impact on the memory consumption, as seen in section 7.3 on page 100.

6. Implementation of experiments

This section describes the experimental procedures used to implement and evaluate the designed experiments. The experimental procedure is divided into two parts, the hardware setup and the software procedure. The hardware setup of the experimental procedures is presented first, followed by the software procedure and the generated source code. The details of the equipment used are described in section 5 on page 42.

6.1. Hardware setup

The following sub section represents the hardware setup of the test and development environment, which is illustrated in Figure 6-1.

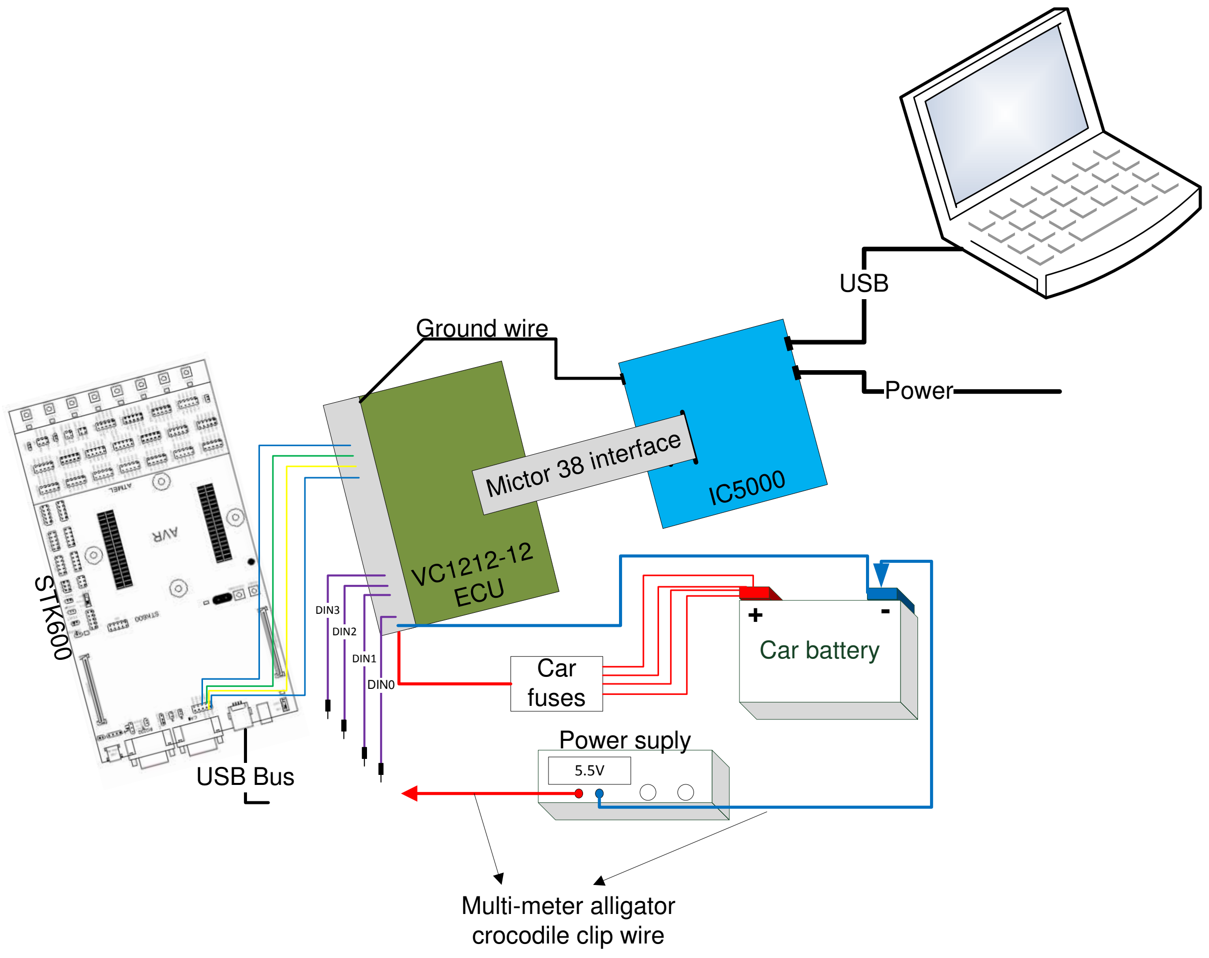


Figure 6-1: Layout of the test environment

6.1.1. Switch simulations

The switches that are required for the designed experiments in section 4 on page 36, are simulated through four pin-female-male jumper wires and a power supply. The female connection of those four wires is connected to the VC121-12 ECU according to Table 6-1 on page 53. The power supply provides the necessary 5.5 V to simulate a close switch and is connected to the male connector of the jumper wires through a multi-meter wire with crocodile clip. This wire is connected to the positive output of the power supply. Moreover, the negative output of the 5.5 V power supply has to be connected to common ground, provided by the car battery that is used to power up the VC121-12 ECU.

6.1.2. Connecting the VC121-12 ECU

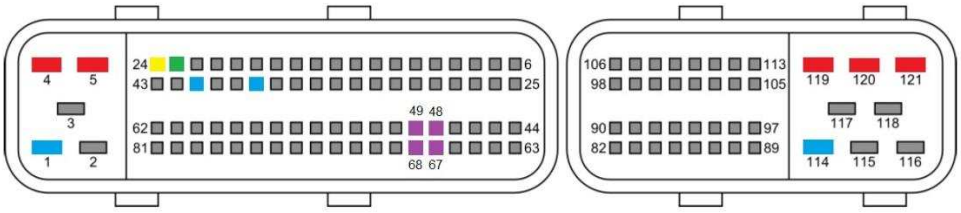


Figure 6-2: VC121-12 ECU pin connectors

Pin	Function	Colour	Connected to
1	Ground	Blue	Power supply (12 V)
4	Supply	Red	Power supply (12 V)
5	Supply	Red	Power supply (12 V)
23	CAN0 Low	Green	STK600
24	CAN0 High	Yellow	STK600
38	Sensor ground	Blue	STK600
41	Sensor ground	Blue	STK600
48	DIN2	Violet	Switch2
49	DIN0	Violet	Switch0
67	DIN3	Violet	Switch3
68	DIN1	Violet	Switch1
114	Ground	Blue	Power supply (12 V)
119	Supply	Red	Power supply (12 V)
120	Supply	Red	Power supply (12 V)
121	Supply	Red	Power supply (12 V)

Table 6-1: Pin assignment of VC121-12 ECU to other devices

The VC121-12 ECU possesses 5 power supply pins as illustrated in Figure 6-2 on page 53 and Table 6-1 on page 53 in order to be powered up by a the 12 V power system in a car (car battery). It is recommended to connect 4 car fuses of 10 A between the power supply and the VC121-12 ECU. To power on the ECU it is require to connect pin (4), pin (5), pin (119), pin (120) and pin (121) to the positive output of a 12 V power supply and pin (1), pin (114) to the negative output of the same power supply. Due to the fact that the sensor ground pins (38) and pin (41) are thinner than the pins connected to the positive power supply, it is recommended to connect pin (38) and pin (41) to the ground output of the STK600 board, see section 6.1.3 on page 55.

Moreover, to be able to flash the compiled software onto the main microcontroller of the VC121-12 ECUs, it is required that the ECU is in debug mode. Therefore, it is recommended by the user manual of the VC121 ECU (Vector, 2011 p.26) to set the SBC into debug mode. This mode is entered if a jumper is plugged into the X500 jumper, see section 11.4 on page 131 . It allows to flash and debug an application without causing a reset to be triggered. Furthermore, the communication on CAN0 is necessary to avoid that the SBC deactivates the supply voltage of the main controller (Vector, 2011 p.68). For this reason, the STK600 board is programmed with the required CAN messages, on which the pin (23) and pin (24) of the VC121-12 ECU are connected as described in the following section. The switches that are used in the designed experiments, see section 4 on page 36, are connected here with male-female jumper wires to the digital inputs DIN0 to DIN3 as shown in Table 6-1 on page 53.

6.1.3. Connecting the STK600

Due to the fact that the flashing process of the ECU requires permanent communication on the port CAN0, communication through the CAN network has to be provided. As a result, the STK600 development board is setup with the STK600-TQFP64 socket (Atmel, 2016) and the 8-bit microcontroller AT90CAN128 (Atmel, 2008).
The following pins have to be connected according to Figure 6-3. In order to be able to program the AT90CAN128 and to generate the necessary CAN messages, the following steps must be performed:

- Set “TERM” jumper
- Set “SLOPE CTRL” jumper
- Set “VTARGET” jumper
- Set “RESET” jumper
- Set “VBUS” jumper
- Connect the two-pins tx (yellow) and rx (green) with pin PD6 (green) and PD5 (yellow) of port D through two four-pin-female-male jumper wires
- Connect ISP Pins with STK600 ISP to be able to program the device with the one 6-wire cable for In-System programming
- Set the quartz crystal
- Connect the pins of the CAN HEADER according to Table 6-1 on page 53 with the VC121-12-ECU through four-pin-female-male jumper wires
- Connect the STK600 board with the USB bus to power on the system

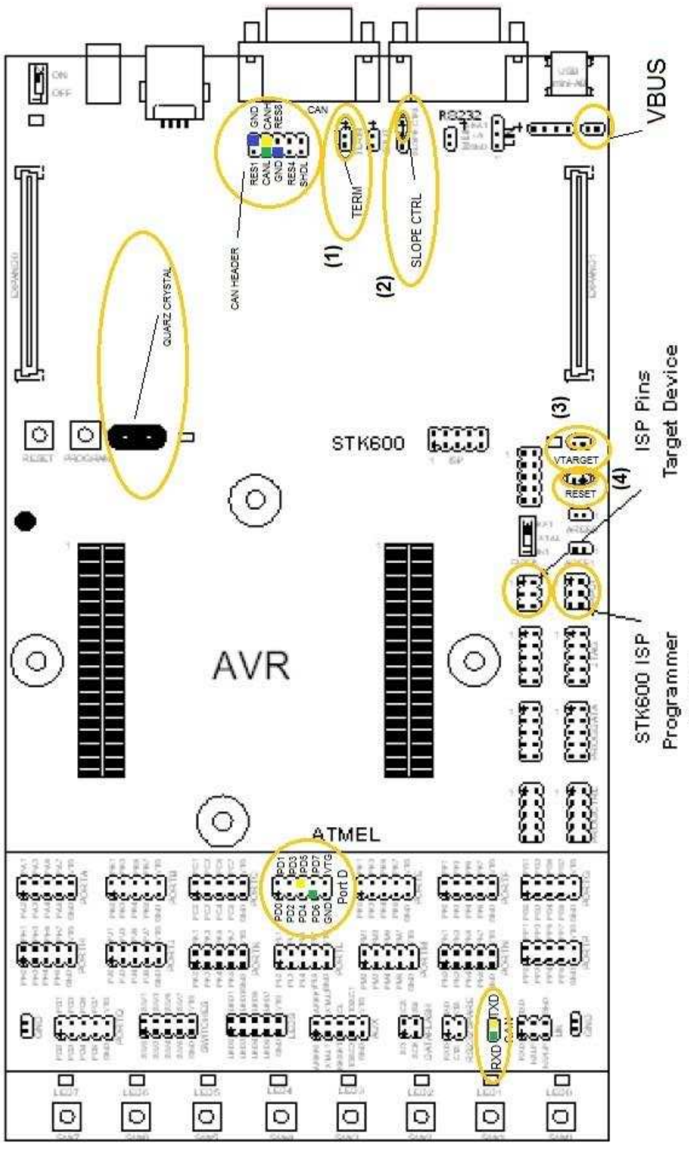


Figure 6-3: Pin assignment of the STK600 board, (Atmel, 2010)

6.1.4. Connecting the IC5000

The ECU provides two options to flash an executable file onto the main controller: flashing via a debugger, for example, with the analyzer IC5000, see section 5.2.2 on page 47 , and flashing via CAN. This project is using the first flashing option, which enables the ability to debug and to profile communication. For the flashing process via debugger it is mandatory to set the VC121-12 into debug mode by setting a jumper at connector X500, as described in section 6.1.2 on page 53. The communication between the host PC and the analyser IC5000 is established via the USB interface, see Figure 6-4. In this work the VC121-12 ECU is provided with a soldered Matched Impedance Connector (Mictor) 38 interface, (Arm), and is therefore connected with the IC5000 analyser through the Mictor 38 NEXUS connector provided from iSystem. Moreover, to avoid ground potential difference between the VC121-12 ECU and the analyser, it is recommended to connect both with a grounding wire as shown in Figure 6-5.

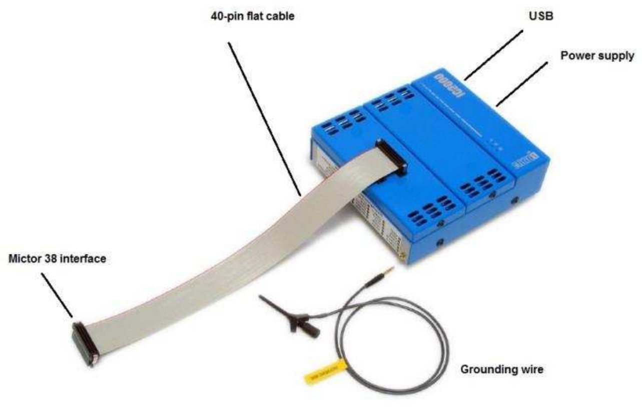


Figure 6-4: The IC5000 analyser and its connectors

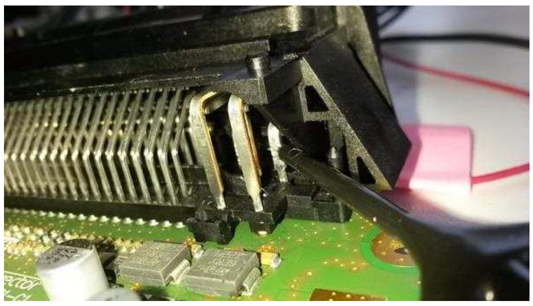


Figure 6-5: Illustration of the connection of the ground wire with the ECU

6.2. Software development tool chain

This section provides a brief explanation of the complete software development tool chain as illustrated in Figure 6-6.

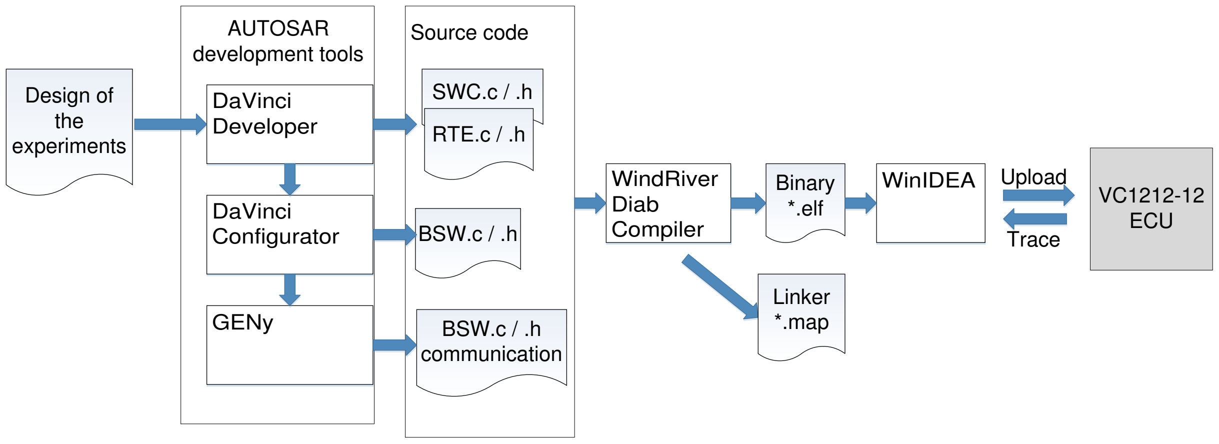


Figure 6-6: Overview of the software development tool chain

The complete software development tool chain includes the following steps:

1. The STK600 board is programmed to send CAN messages every 200 ms, which is connected to the CAN High and CAN Low pins of the ECU. The source code to this program can be found in section 11.1 on page 116.

2. The source code of the defined experiments, in section 4 on page 36, are generated with the AUTOSAR development tools, see section 5.3.2 on page 49**.** The source code of all non-communication modules, such as the operating system, is generated through the DaVinciConfigurator Pro. The source code for the RTE layer is generated by the DaVinciDeveloper. A detailed description how to configure software components can been found in section 11.2 on page 122. Furthermore, all configurations for the modules related to communication such as the communication network CAN are configured and generated with the AUTOSAR development tool GENY.

3. The WindRiver Compiler translates the generated source code to an executable file (***.**elf) as described in section 5.3.3 on page 49.

4. The executable file is uploaded and evaluated with the debugger IC5000, as described in section 5.3.4 on page 50.

The documentation of this work is inspired by (Silverio Miyashiro, Ferreira, & Sant'Anna, 2015).

6.3. **Generated source code**

This section describes the generated source code of the VC121-12 ECU. The source code is gained through the software and hardware described in section 5.2 on page 44 and section 5.3 on page 48. This source code includes the three main parts: initialisation process, BSW system and the application. This section begins with the description of the initialisation process, followed by the explanation of the scheduling of the BSW system. Finally, the application of both implementations are presented and the main differences between both implementations are highlighted.

Each execution of an application on the VC121-12 ECU starts with the execution of the start-up code. This code segment provides not only the entry point (start address) for the application, it also initialise the MMU and the RAM memory. The RAM areas are initialised with pre-defined values that are stored in the ROM memory and copied to RAM during the start-up. A detailed description of the start-up code can be found in section 0 on page 132 and in the technical reference, (Reinl, 2014)**.**

The initialisation process follows the start-up procedure. The initialisation process initialise all BSW modules, the RTE and OS. The initialisation process starts with the execution of the Ecum_init function and ends with the execution of the dispatcher, as seen in section 0 on page 132. The function dispatcher is elemental for the execution it:

- checks the activation queue of the scheduler
- saves the old context of the actual running task and
- starts the next task that has the highest priority and is in state “READY”.

The whole ECU software consists of several tasks. The defined tasks in section 4 on page 36 are implemented as application tasks that are scheduled through the operating system. Whereby, each BSW module has its main function. These main functions check for pending events and execute its specific functions, if the corresponding event occurs. Each BSW main function has its own cycle time.

The calling frequency of such main functions ranges from 1 ms to 100 ms in this work, as seen **in** section 11.5.2 on page 133 to section 11.5.5 on page 136. These main functions are scheduled from the BSW Scheduler (SchM). It is mandatory to consider that the dependencies between the BSW components and to choose the appropriate scheduling frequency for each BSW main function by configuring the SchM module through the GENy tool. The description of the specific cycle time of each main function can be found in the technical references of the individual modules that are delivered with the VC121-12 ECU (TechnicalReference_Asr_BSWmodule.pdf). In addition to the cycle time, also an activation offset can be configured to delay the initial triggering of a main function.

Both implementations in this work have the same system configuration. The SchM in both implementations have 5 offsets that are signified through the name of the time variable and are illustrate in the detailed program flow chart in section 11.5.2 on page 133 to section 11.5.5 on page 136.

The BSW Scheduler is implemented as two extended tasks, SchM_SyncTask and SchM_AsyncTask. SchM_SyncTask includes all BSW main functions that are dependant of the FLexRay network and therefore has to be executed in a specific point in the FlexRay cycle. All the BSW main function that are independent of the FlexRay network are implemented in the Sch_AsyncTask , (Vector, 2007) .

```
SchM_Task
{
 SchM_Timer_SchM_AsyncTask_1_0_10ms =10;
 SchM_Timer_SchM_AsyncTask_1_0_5ms =5;

 for( ; ;)
 {
  WaitEvent(CyclicEvent);
  GetEvent(&ev);
  ClearEvent(ev);
  If (0 == SchM_Timer_SchM_AsyncTask_1_0_10ms )
   {
    Can_MainFunction;
    Spi_MainFunction;
   }
  If (0 == SchM_Timer_SchM_AsyncTask_1_0_5ms)
   {
    Lin_MainFunction;
   }
  SchM_Timer_SchM_AsyncTask_1_0_10ms - -;
  SchM_Timer_SchM_AsyncTask_1_0_5ms - -;
 }
}
```

Source code 6-1: SchM task example

Each cycle time is implemented with a time variable that indicates the cycle period, such as SchM_Timer_SchM_AsyncTask_1_0_10ms. The time variable, in this case, is initialised with 10 with an offset of 0. Source code 6-1 illustrates an example of such SchM extended task. The SchM task is implemented as an endless for-loop. This task is set to the state “WAIT” with the execution of the WaitEvent function until the event CyclicEvent occurs. When the expected event occurs, the task is then transferred to the state “RUN”. The occurred event is then stored in the variable ev through the function GetEvent and disabled through the execution of the function ClearEvent. Both SchM tasks have a CyclicEvent that occures each millisecond. That means that each millisecond the task is set back to the state “RUN” and all time variables are checked if the initialised value equals zero. Each time variable is decremented after each execution of the SchM_Task. The decrementation of the time variables ensure that each BSW main functions is executed at its cycle period.

Each application task is represented as an entry in several variables. This variables defines the characteristics of tasks, such as:

- base priority, which represents the home priority before the execution of the priority ceiling protocol. Software-wise, is the lowest priority defined as zero, every value greater than zero will be declined as a higher priority, where 255 is the highest priority (AUTOSAR, 2014a p. 126-127). Conversely, in hardware the highest priorities is defined as zero and priorities greater then zero will be declined as lower priories.
- actual priority, which is used to store necessary priority changes during the priority ceiling protocol
- state, which defines the state of the task, see section 2.3.1 on page 24
- stack, which references to the starting address of the task stack. Each task has its own stack to store all local data, return addresses and the task context. The task context is a register in which all the information of a pre-emptive task is stored when it is transferred to the state "WAIT". If the task is transferred back to state "RUN", then the task can be executed on the point where it stopped previous (V. I. GmbH, 2011a p. 25)
- task type, which describes whether the task is extended or basic
- scheduling policy, it describes whether a task is pre-emptive or non-pre-emptive
- activation counter and the maximum amount of possible activations of the task
- events on which extended task are transferred to the state "READY" or in the state "WAIT". Is the task from type basic then this characteristic is set to zero.

This characteristics are available in the OSEK Run Time Interface (ORTI) file (<oil[4]_filename.>.ort), which contains debug information generated from the generator of the AUTOSAR tool**,** see section 5.3.2 on page 49**.** The characteristics for the application tasks can be found in section 6.3.1.1 on page 61 for SingleCoreV1_0 and in section 6.3.1.2 on page 67 for SingleCoreV2_0.

[4] OSEK Implementation Language (OIL)

6.3.1.1. SingleCoreV1_0

The first implementation SingleCoreV1_0, contains 6 application task of type extended, which include the write- and read-process as required from the designed experiments, see section 4 on page 36. Table 6-2 illustrates the characteristics of each application task and the two SchM tasks with the additional characteristic, priority mask. The priority mask is used to identify the hardware priority of each task in the scheduling process of the operating system, see section 7.1.2.2.2 on page 87. Whereby, the leading zeros of the priority mask is defining the hardware priority.

Name	Entry number/ identify number	Base software priority	Hardware priority	Priority mask	Task type	Scheduling policy
Task_Event_Expl_Swc4to3	0	5	1	0x4000 0000	Extended	Non-pre-emptive
Task_Event_Impl_Swc3to4	1					
Task_IoHw_Period_Expl_Swc6to5	2	4	2	0x2000 0000		
Task_IoHw_Period_Impl_Swc5to6	3					
Task_Period_Expl_Swc2to1	4	3	3	0x1000 0000		Full-pre-emptive
Task_Period_Impl_Swc1to2	5					
SchM_AsyncTask_1	6	2	4	0x0800 0000		Non-pre-emptive
SchM_SyncTask_1	7	1	5	0x0400 0000		

Table 6-2: Task characteristics of SingleCoreV1_0 Implementation

The extended application tasks of SingleCoreV1_0 are implemented with an endless loop such as the extended SchM tasks. In contrary to the SchM tasks are the extended application tasks here waiting for two events, an event for the writing process and an event for the reading process as illustrated in Source code 6-2. The event for the writing process occurs depending on the period that is defined for each task, see section 4 on page 36. Whereby, the reading process is evoked only if the corresponding event occurs.

```
Task_
{
  for( ; ;)
  {
    EventMaskType ev;
    WaitEvent(EvRunnableOut | EvRunnableIn);
    GetEvent(&ev);
    ClearEvent(ev);
    If (ev == EvRunnableOut )
      {
        RunnableOut( );
      }
    If (ev == EvRunnableIn)
    {
      RunnableIn();
    }
  }
}
```

Source code 6-2: Implementation of an extended task in SingleCoreV1_0

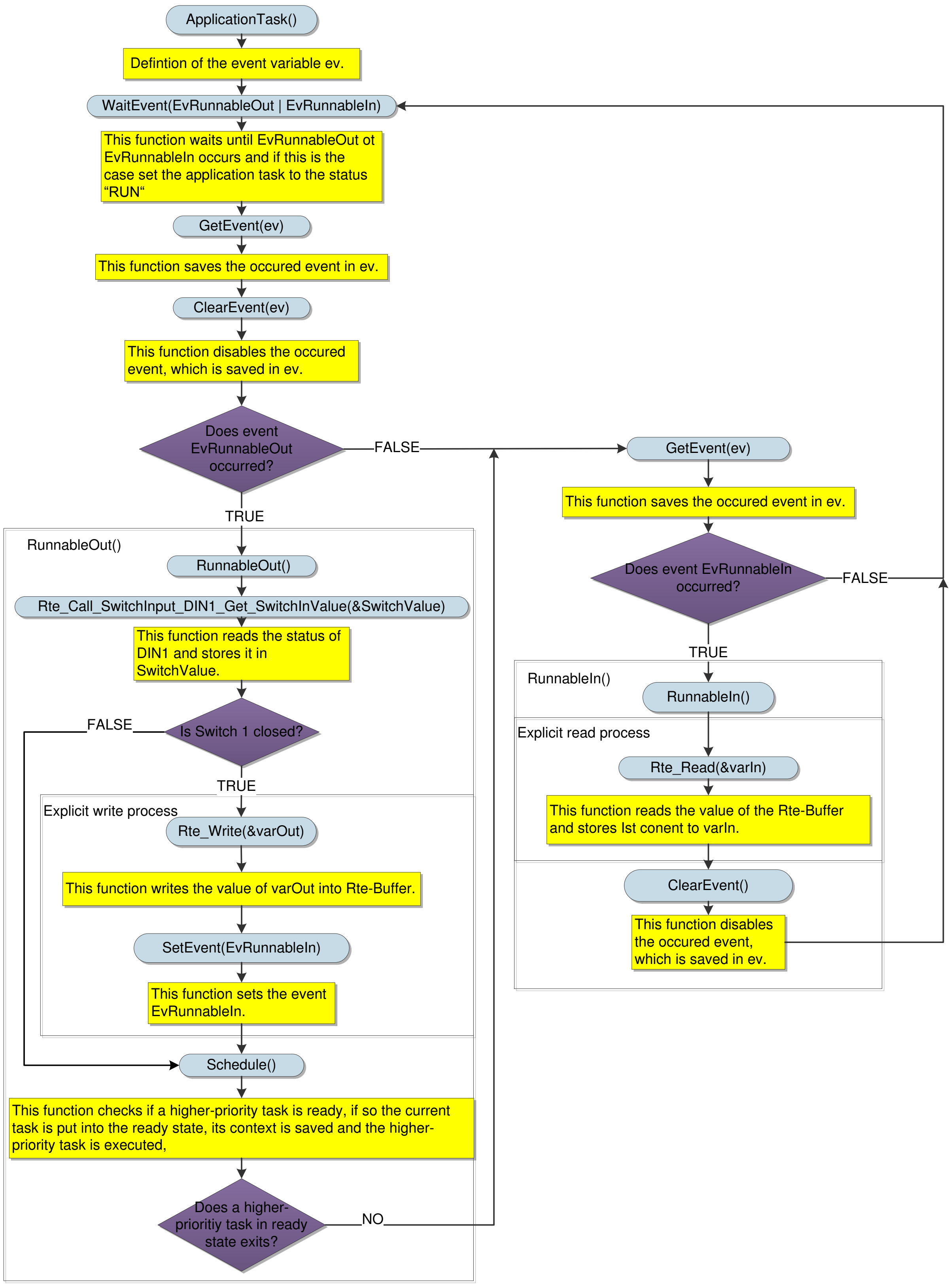
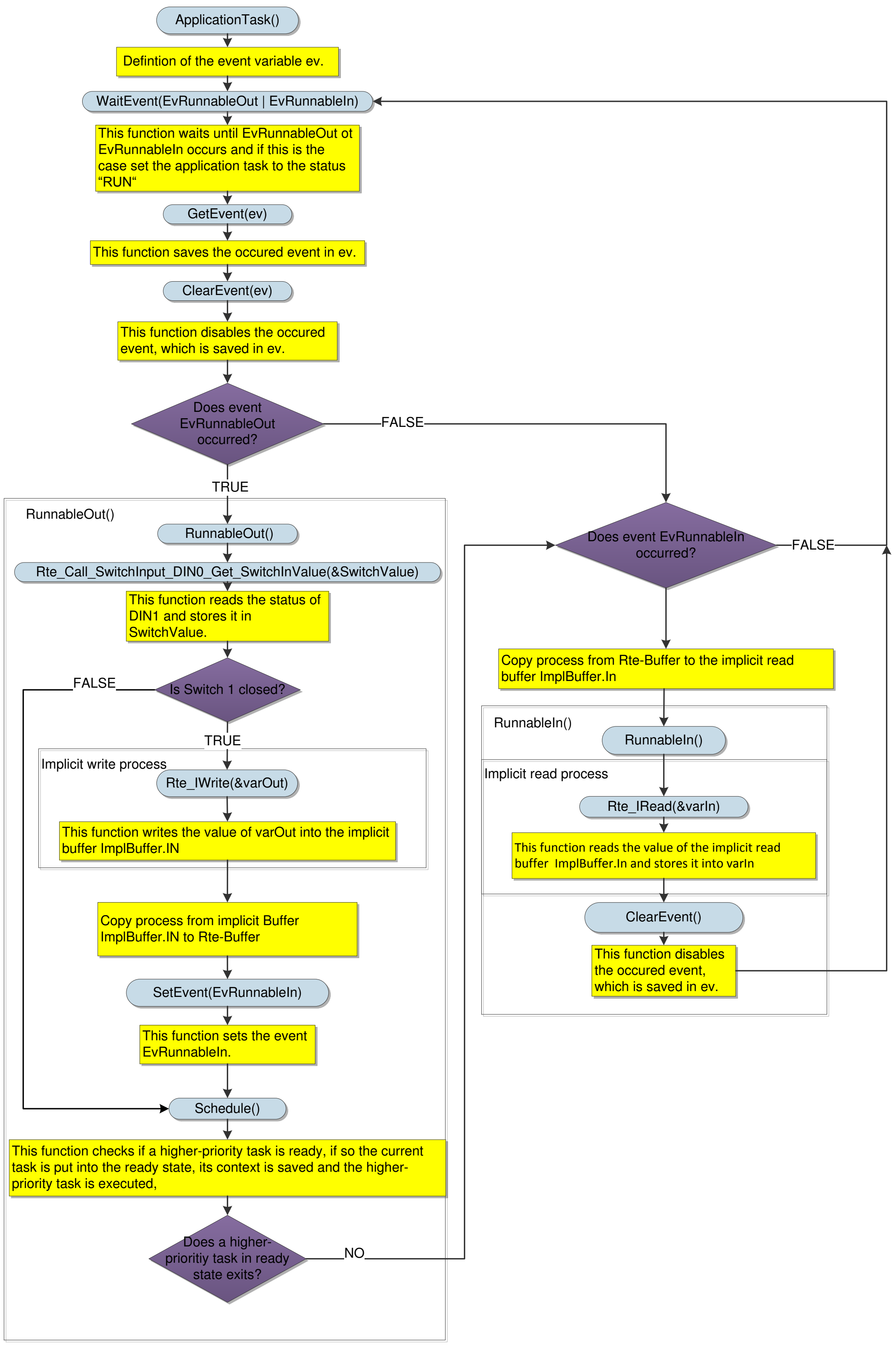

ApplicationTask()
Defintion of the event variable ev.
WaitEvent(EvRunnableOut | EvRunnableIn)
This function waits until EvRunnableOut ot EvRunnableIn occurs and if this is the case set the application task to the status “RUN“
GetEvent(ev)
This function saves the occured event in ev.
ClearEvent(ev)
This function disables the occured event, which is saved in ev.
Does event EvRunnableOut occurred?
FALSE
TRUE
RunnableOut()
RunnableOut()
Rte_Call_SwitchInput_DIN1_Get_SwitchInValue(&SwitchValue)
This function reads the status of DIN1 and stores it in SwitchValue.
FALSE
Is Switch 1 closed?
TRUE
Explicit write process
Rte_Write(&varOut)
This function writes the value of varOut into Rte-Buffer.
SetEvent(EvRunnableIn)
This function sets the event EvRunnableIn.
Schedule()
This function checks if a higher-priority task is ready, if so the current task is put into the ready state, its context is saved and the higher-priority task is executed,
Does a higher-prioritiy task in ready state exits?
NO
GetEvent(ev)
This function saves the occured event in ev.
Does event EvRunnableIn occurred?
FALSE
TRUE
RunnableIn()
RunnableIn()
Explicit read process
Rte_Read(&varIn)
This function reads the value of the Rte-Buffer and stores Ist conent to varIn.
ClearEvent()
This function disables the occured event, which is saved in ev.

Figure 6-7: Program flow chart for the explicit communication in SingleCoreV1_0

The program flow, see Figure 6-7 on page 63, illustrates the explicit communication of the SingleCoreV1_0 implementation. Each application task in SingleCoreV1_0 is implemented as an extended task to manage the write and read-process in with the same task, as seen in Table 6-2 on page 61. The following section describes the procedure of the explicit communication, which is illustrated in Figure 6-7 on page 63.

- Each application task is executed, firstly, with the occurrence of the event for the writing process (EvRunnableOut) through the functions WaitEvent, GetEvent and ClearEvent. Afterwards, the writing runnable (RunnableOut) is called.
- The software components, which are dependent on the status of the digital inputs (DIN0 to DIN3) are then calling the function Rte_Call_SwitchInput_DIN1_GetSwitchValue. This function saves the status of the corresponding switch, here switch1, into its parameter (SwitchValue). If switch 0 or switch 1 is closed then the writing process of SWCEvent03 and SWCEvent04 is executed. Whereby, the status of switch 2 or switch 3 is exchanged between the software components SWCIoHwPeriod05 and SWCIoHwPeriod06.
- The writing process for the explicit communication is implemented through the Rte_Write function. This function copies the content of its function parameter, here data into the RteBuffer. The RteBuffer represents a memory space that is allocated for the communication process.
- The notification process is responsible to acknowledge the reading process that new data is available. This process begins with the execution of the function SetEvent in the writing function Rte_Write and contains the following functions:
    - SetEvent(EventMask): This function sets the event corresponding to the EventMask, here the event for the reading process is set.
    - Schedule(): This function checks if a higher-task is in status "READY". If this is the case the actual task will be transferred to the status "WAIT", its context is saved and the higher-priority task is executed. In this work a rescheduling through the function Schedule is not taken place.
    - GetEvent(Event): This function copies the current occurred event to its parameter Event.
- The execution of the reading process (RunnableIn) is ensured, firstly, through the function SetEvent in the writing process and secondly, through the second GetEvent after RunnableOut is completed. The second GetEvent saves the occurred read event (EvRunnableIn) to its variable and enable the execution of the reading process. The explicit reading process is executed through the function Rte_Read that copies the content of the RteBuffer into its paramerter (Ivar).



ApplicationTask()
Defintion of the event variable ev.
WaitEvent(EvRunnableOut | EvRunnableIn)
This function waits until EvRunnableOut ot EvRunnableIn occurs and if this is the case set the application task to the status "RUN"
GetEvent(ev)
This function saves the occured event in ev.
ClearEvent(ev)
This function disables the occured event, which is saved in ev.
Does event EvRunnableOut occurred?
FALSE
TRUE
RunnableOut()
RunnableOut()
Rte_Call_SwitchInput_DIN0_Get_SwitchInValue(&SwitchValue)
This function reads the status of DIN1 and stores it in SwitchValue.
FALSE
Is Switch 1 closed?
TRUE
Implicit write process
Rte_IWrite(&varOut)
This function writes the value of varOut into the implicit buffer ImplBuffer.IN
Copy process from implicit Buffer ImplBuffer.IN to Rte-Buffer
SetEvent(EvRunnableIn)
This function sets the event EvRunnableIn.
Schedule()
This function checks if a higher-priority task is ready, if so the current task is put into the ready state, its context is saved and the higher-priority task is executed,
Does a higher-prioritiy task in ready state exits?
NO
Does event EvRunnableIn occurred?
FALSE
Copy process from Rte-Buffer to the implicit read buffer ImplBuffer.In
RunnableIn()
RunnableIn()
Implicit read process
Rte_IRead(&varIn)
This function reads the value of the implicit read buffer ImplBuffer.In and stores it into varIn
ClearEvent()
This function disables the occured event, which is saved in ev.

Figure 6-8: Program flow chart for the implicit communication in SingleCoreV1_0

The implementation of the implicit communication differs from the implementation of the explicit communication in the generated communication functions and the notification process.

The implicit communication functions Rte_IWrite and Rte_IRead are generated as macros that will be replaced with their identical copy of the RteBuffer (ImplBuffer.In and ImplBuffer.Out). Each implicit communication function, write and read, has its own RteBuffer, which is realised as a struct variable (ImpBuffer) with two members (In and Out), as seen in Figure 6-8 on page 65.

These three buffers are synchronised through a copy process to guarantee that each buffer contains the newest data. In addition, the implicit write function Rte_IWrite does not contain the notification function SetEvent. This function is called after the write process is completed and is therefore independent of whether the write process occurs or not. In addition, it is mandatory to acknowledge that the second GetEvent function is missing before the execution of the second if-statement, which can be seen in Figure 6-8 on page 65. Due to this missing GetEvent, the functions WaitEvent, GetEvent and ClearEvent have to be executed to acknowledge that the reading event (EvRunnableIn) occurred.

6.3.1.2. SingleCoreV2_0

In comparison with the previous implementation is the write- and read-process split into two separated tasks that are generated as basic tasks as seen in Table 6-3.

Name	Entry number/ identify number	Base software priority	Hardware priority	Priority mask	Task type	Scheduling policy
SchM_AsyncTask_1	0	2	4	0x0800 0000	Extended	Non-pre-emptive
SchM_SyncTask_1	1	1	5	0x0400 0000	Extended	
Task_Event_Expl_Swc4to3_In	2	5	1	0x4000 0000	Basic	
Task_Event_Expl_Swc4to3_Out	3			0x4000 0000		
Task_Event_Impl_Swc3to4_In	4			0x4000 0000		
Task_Event_Impl_Swc3to4_Out	5			0x4000 0000		
Task_IoHw_Period_Expl_Swc6to5_In	6	4	2	0x2000 0000		
Task_IoHw_Period_Expl_Swc6to5_Out	7			0x2000 0000		
Task_IoHw_Period_Impl_Swc5to6_In	8			0x2000 0000		
Task_IoHw_Period_Impl_Swc5to6_Out	9			0x2000 0000		
Task_Period_Expl_Swc2to1_In	10	3	3	0x1000 0000		Full-pre-emptive
Task_Period_Expl_Swc2to1_Out	11			0x1000 0000		
Task_Period_Impl_Swc1to2_In	12			0x1000 0000		
Task_Period_Impl_Swc1to2_Out	13			0x1000 0000		

Table 6-3: Task characteristics of SingleCoreV2_0 implementation

Each application task in SingleCoreV2_0 is generated as a basic task as seen in Source code 6-3 and Source code 6-4 . The basic tasks are not including any endless loops, due to the fact that basic tasks are not able to be transferred into the state “WAIT”. As a result, the basic tasks have to be terminated after its execution through the function TerminateTask.

```
Task_Out
{
  RunnableOut( );
  TerminateTask();
}
```

Source code 6-3: Implementation of the basic task of the write process in SingleCoreV2_0

```
Task_In
{
  RunnableIn( );
  TerminateTask();
}
```

Source code 6-4: Implementation of the basic task of the read process in SingleCoreV2_0

The explicit communication of the SingleCoreV2_0 implementation begins with the occurrence of the periodical event of the writing process. This event is the trigger to execute the writing task Task_Out, see Figure 6-9 on page 69.

- The writing task calls the RunnableOut, which differs from the explicit RunnableOut in only one aspect, the notification process.

- The notification process begins with the execution of the function ActivateTask instead of SetEvent as in SingleCoreV1_0. This function transfers the task with the task id (TaskID) to the state “READY”.

- The function TerminateTask transfers the actual task to the status “SUSPENDED” and identifies the next high prioritised task in status “READY” through the function SchedulePrio and and it calls the identified task through the functions Dispatcher.

- The function Dispatcher then activates the task for the reading process, if this task is the highest prioritised task in status “READY”.

- The reading task executes the reading process and is transferred into the state “SUSPENDED” after its execution through the function TerminateTask as seen in Figure 6-10 on page 69.

The implicit communication of the SingleCoreV2_0 includes the same IWrite and IRead function as the implicit communication of the SingleCoreV1_0. Also in SingleCoreV2_0 are two synchronisation points necessary to guarantee that each Rte-Buffer has the newest value. But the two implicit buffers of the write- and read-process are here generated as separate variables, ImplWriteBuffer and ImplReadBuffer. in comparision to the first implementation as seen in Figure 6-11 and Figure 6-12 on page 70.

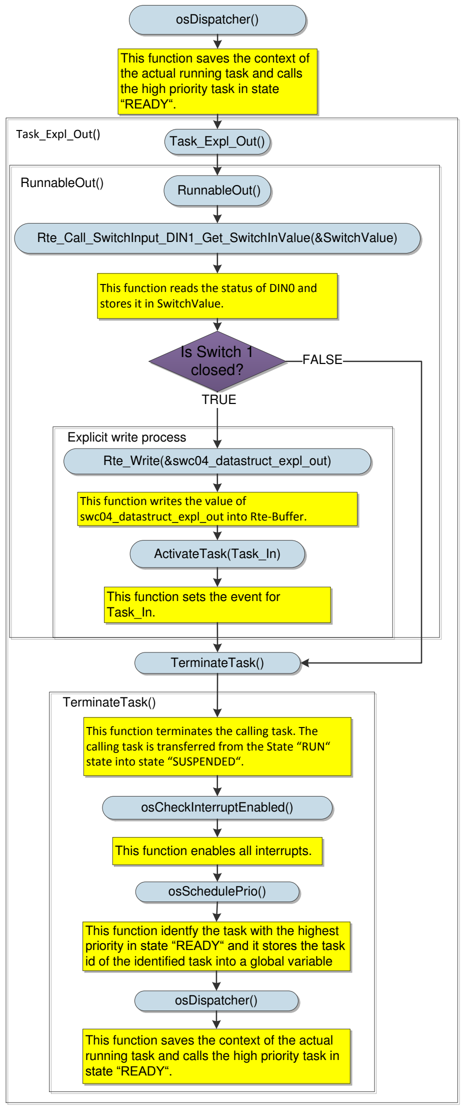

osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".
Task_Expl_Out()
Task_Expl_Out()
RunnableOut()
RunnableOut()
Rte_Call_SwitchInput_DIN1_Get_SwitchInValue(&SwitchValue)
This function reads the status of DIN0 and stores it in SwitchValue.
Is Switch 1 closed?
FALSE
TRUE
Explicit write process
Rte_Write(&swc04_datastruct_expl_out)
This function writes the value of swc04_datastruct_expl_out into Rte-Buffer.
ActivateTask(Task_In)
This function sets the event for Task_In.
TerminateTask()
TerminateTask()
This function terminates the calling task. The calling task is transferred from the State "RUN" state into state "SUSPENDED".
osCheckInterruptEnabled()
This function enables all interrupts.
osSchedulePrio()
This function identfy the task with the highest priority in state "READY" and it stores the task id of the identified task into a global variable
osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".

Figure 6-9: Program flow chart for the explicit communication of the writing task in SingleCoreV2_0

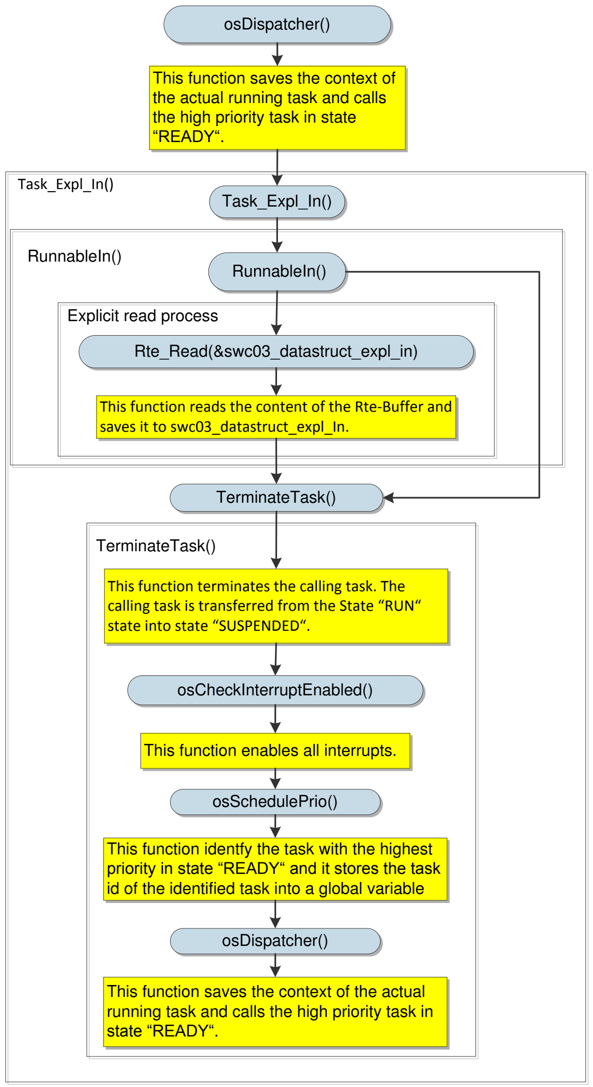
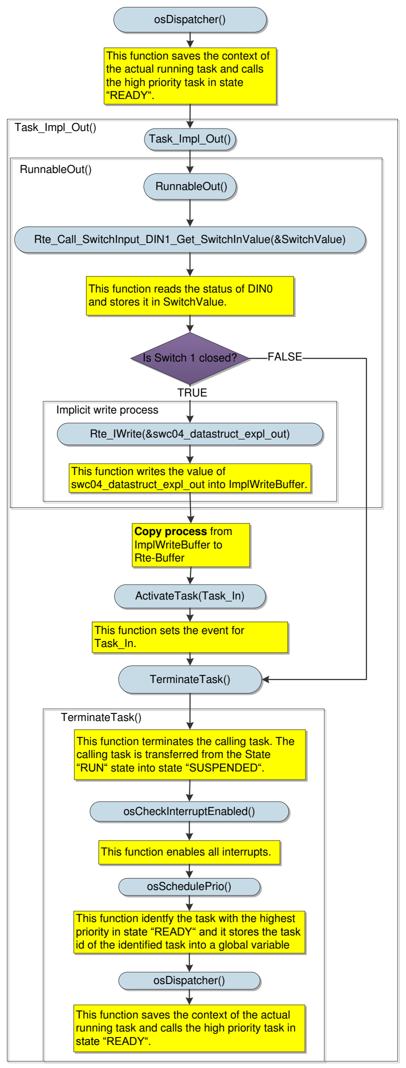
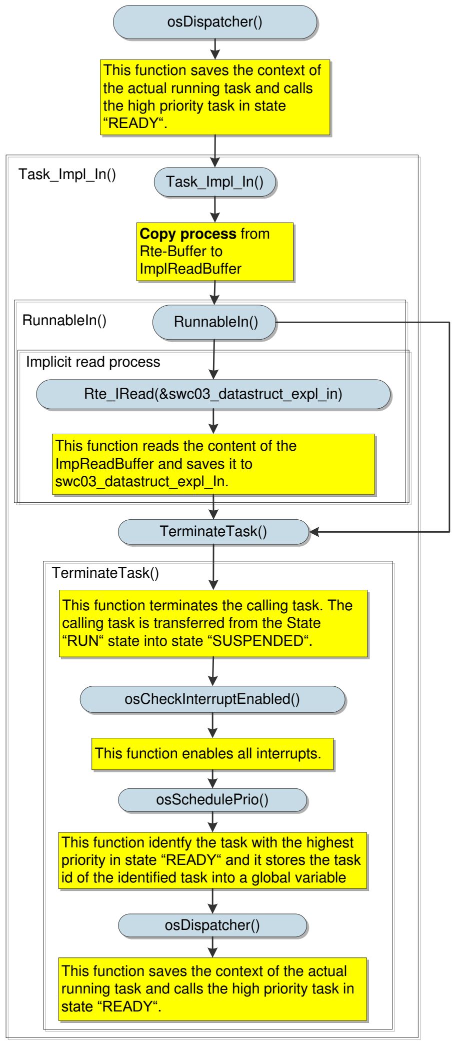

osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".
Task_Expl_In()
Task_Expl_In()
RunnableIn()
RunnableIn()
Explicit read process
Rte_Read(&swc03_datastruct_expl_in)
This function reads the content of the Rte-Buffer and saves it to swc03_datastruct_expl_In.
TerminateTask()
TerminateTask()
This function terminates the calling task. The calling task is transferred from the State "RUN" state into state "SUSPENDED".
osCheckInterruptEnabled()
This function enables all interrupts.
osSchedulePrio()
This function identfy the task with the highest priority in state "READY" and it stores the task id of the identified task into a global variable
osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".

Figure 6-10: Program flow chart for the explicit communication of the reading task in SingleCoreV2_0



osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".
Task_Impl_Out()
Task_Impl_Out()
RunnableOut()
RunnableOut()
Rte_Call_SwitchInput_DIN1_Get_SwitchInValue(&SwitchValue)
This function reads the status of DIN0 and stores it in SwitchValue.
Is Switch 1 closed?
FALSE
TRUE
Implicit write process
Rte_IWrite(&swc04_datastruct_expl_out)
This function writes the value of swc04_datastruct_expl_out into ImplWriteBuffer.
Copy process from ImplWriteBuffer to Rte-Buffer
ActivateTask(Task_In)
This function sets the event for Task_In.
TerminateTask()
TerminateTask()
This function terminates the calling task. The calling task is transferred from the State "RUN" state into state "SUSPENDED".
osCheckInterruptEnabled()
This function enables all interrupts.
osSchedulePrio()
This function identfy the task with the highest priority in state "READY" and it stores the task id of the identified task into a global variable
osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".

Figure 6-11: Program flow chart for the implicit communication of the writing task in SingleCoreV2_0



osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".
Task_Impl_In()
Task_Impl_In()
Copy process from Rte-Buffer to ImplReadBuffer
RunnableIn()
RunnableIn()
Implicit read process
Rte_IRead(&swc03_datastruct_expl_in)
This function reads the content of the ImpReadBuffer and saves it to swc03_datastruct_expl_In.
TerminateTask()
TerminateTask()
This function terminates the calling task. The calling task is transferred from the State "RUN" state into state "SUSPENDED".
osCheckInterruptEnabled()
This function enables all interrupts.
osSchedulePrio()
This function identfy the task with the highest priority in state "READY" and it stores the task id of the identified task into a global variable
osDispatcher()
This function saves the context of the actual running task and calls the high priority task in state "READY".

Figure 6-12: Program flow chart for the implicit communication of the reading task in SingleCoreV2_0

7. Evaluation of experiments

This section is intended to evaluate both implementations of the designed experiments. Once the defined experiments had been implemented, the correct execution of both applications, the communication time and memory consumption can be conducted. To verify the correct execution of both implementations a scheduling analysis was performed. This scheduling analyses includes the question, if the defined time constraints in section 4 on page 36 are fulfilled and whether switch 0 and switch 1 have an impact on the communication of SWCEvent03 and SWCEvent04 as expected. The second part of this section verifies the time that is spent to execute the intra-task and inter-task communication, which results provide the ability to form a statement regarding to the performance of both communication types. Furthermore, the memory consumption of both communication types were analysed and evaluated to conclude the advantages and disadvantages of both communication types.

7.1. Scheduling analysis

The following sub section verifies the implemented experiments SingleCore_V1_0 and SingleCore_V2_0 regarding to their scheduling process.

7.1.1. Scheduling analysis methodology

For the scheduling analysis of both implementations, the real-time analyser winIDEA from ISystems, see section 5.3.4 on page 50, has been used to trace the calling sequence of the defined tasks.
The function execution is recorded for the first 25 ms of the application, for all possible switch combinations. However, the scheduling process has been analysed for only the possible status combinations of switch 0 (SW0) and switch 1 (SW1). As described in section 4 on page 36 should only SW0 and SW1 have an impact on the scheduling of the write processes.

These were

- both, switch 0 and switch 1, open (Vx_0_0000),
- both closed (Vx_0_1111),
- switch 0 closed (Vx_0_1011) and
- switch 1 closed (Vx_0_0100).

The missing number x represents the number related to the implementation, where 1 stands for SingleCoreV1_0 and 2 stands for SingleCoreV2_0.

The execution time of each function is recorded in a text format and summarized in Excel tables. The data collection of Vx_0_1111 is illustrated in section 11.6 on page 137 for SingleCoreV1_0 and in section 11.7 on page 149 for SingleCoreV2_0. This summary includes only the necessary communication functions, such as tasks, runnables, read and write functions and functions. Based on this data; charts were generated to illustrate the execution sequence of the application.

7.1.2. Verified requirements

The following is a list of verified requirements for the scheduling analysis of both implementations, indicated with an ID for each requirement.

1. The write-process should be triggered periodically. It is required that the write-process from

$$\left.\begin{array}{l}\text{SwcEvent03 to SwcEvent04}\\ \text{SwcEvent04 to SwcEvent03}\\ \text{SwcIoHwPeriod05 to SwcIoHwPeriod06}\\ \text{SwcIoHwPeriod06 to SwcIoHwPeriod05}\end{array}\right\}\text{ is invoked each millisecond.}$$

$$\left.\begin{array}{l}\text{SwcPeriod01 to SwcPeriod02}\\ \text{SwcPeriod02 to SwcPeriod01}\end{array}\right\}\text{ is invoked each 5 millisecond.}$$

2. The runnable, which is responsible for the read-process of each communication, should be invoked as soon as new data is available.

3. Switch 0 and switch 1 should have an influence on the calling sequence of write- and read-process of the communication between SWCEvent03 and SWCEvent04. It is expected that the communication between SWCEvent3 and SWCEvent4 only occur when the correspondent switch is open.

7.1.2.1. Requirements number 1 test description and results

Due to the fact that write- and read-processes are included in one extended task for each communication in SingleCoreV1_0, the invocation time of the task is equal to the invocation time of the writing process[5]. Therefore, the verification of the calling sequence of the writing processes is taken on a task level.

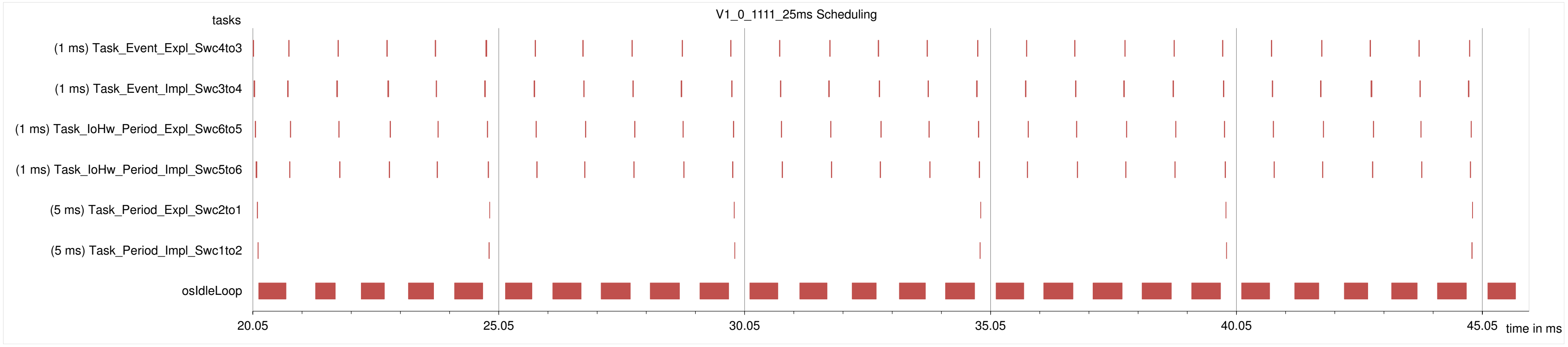


Figure 7-1: Task calling sequence for the first 25 ms of SingleCoreV1_0 for the case when each switch is closed

Figure 7-1 shows the task calling sequence per millisecond for the first 25 ms for the case when each switch is closed (V1_0_1111) of SingleCoreV1_0. As it can be seen the first task starts around the 20th millisecond after the initialisation process[6] of SingleCoreV1_0. The figure shows that the tasks and therefore the write-processes, are called as expected; Task_Event_Expl_Swc3to4, Task_Event_Impl_Swc3to4, Task_IoHw_Period_Expl_Swc6to5 and Task_IoHw_Period_Impl_Swc5to6 were called each millisecond and Task_Period_Expl_Swc2to1 and Task_Period_Impl_Swc1to2 every fifth millisecond for the case when each switch is closed. This time constraint for the periodic calling sequence is also met for the cases when all switches are open, for the case when switch 1 is the only open switch and for the case when switch 1 is the only closed switch. The measurements of these cases can be seen in, Figure 7-2 and Figure 7-3 and Figure 7-4 on the following page.

[5] A detailed explanation of the implementation of extended task can be found in section 6.3 on page 54.
[6] More information about the initialisation process of SingleCore_V1_0 is presented in chapter 6.3 on page 54.

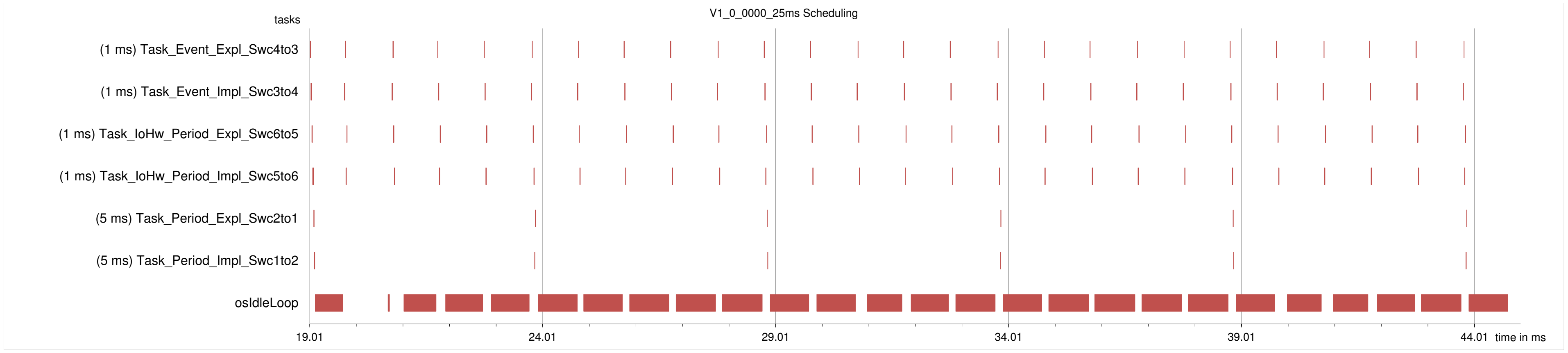


Figure 7-2: Task calling sequence for the first 25 ms of SingleCoreV1_0 for the case when each switch is open

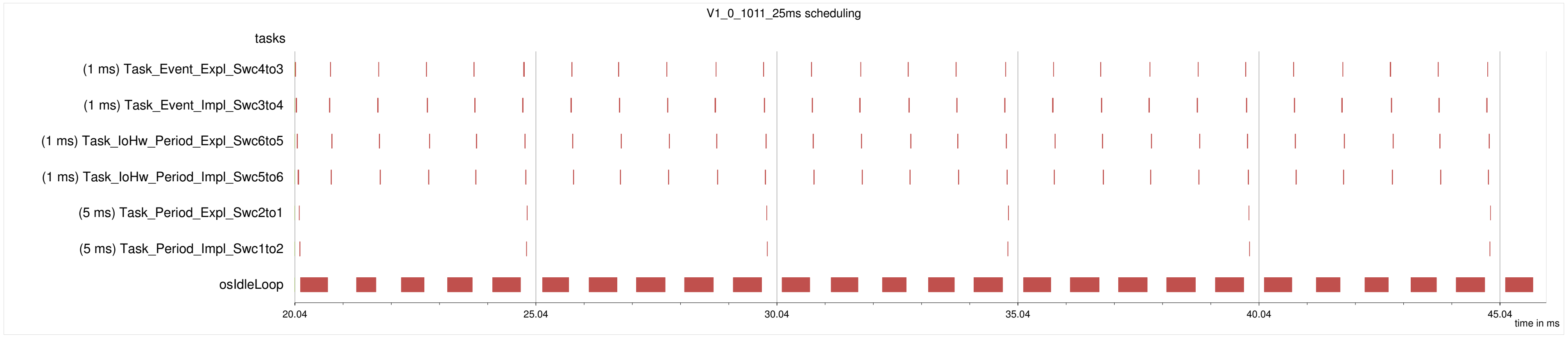


Figure 7-3: Task calling sequence for the first 25 ms of SingleCoreV1_0 for the case when only switch 1 is open

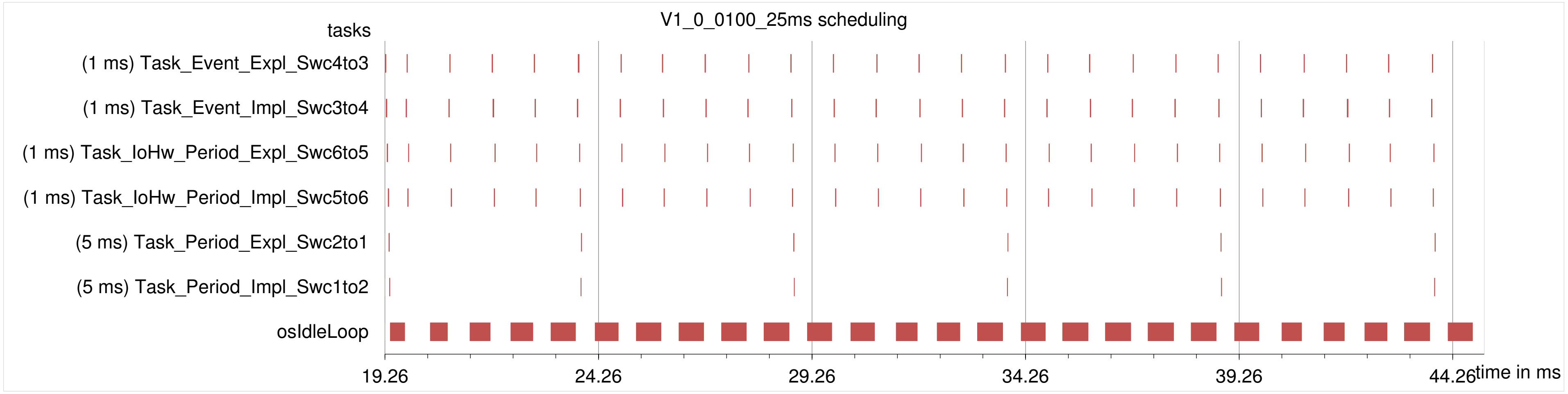


Figure 7-4: Task calling sequence for the first 25 ms of SingleCoreV1_0 for the case when only switch 1 is close

The requirement for SingleCoreV1_0 is also valid for the second implementation SingleCoreV2_0. Contrary to the first implementation, the write- and read-processes in SingleCore_V2_0 are mapped to two tasks. Hence, the following observation of the invocation of the writing process is carried out on the task-level, with a focus on the tasks ending with OUT that include the write process. Figure 7-5 illustrates the task calling sequence per millisecond for the first 25 ms for the case when each switch is closed (V2_0_1111).

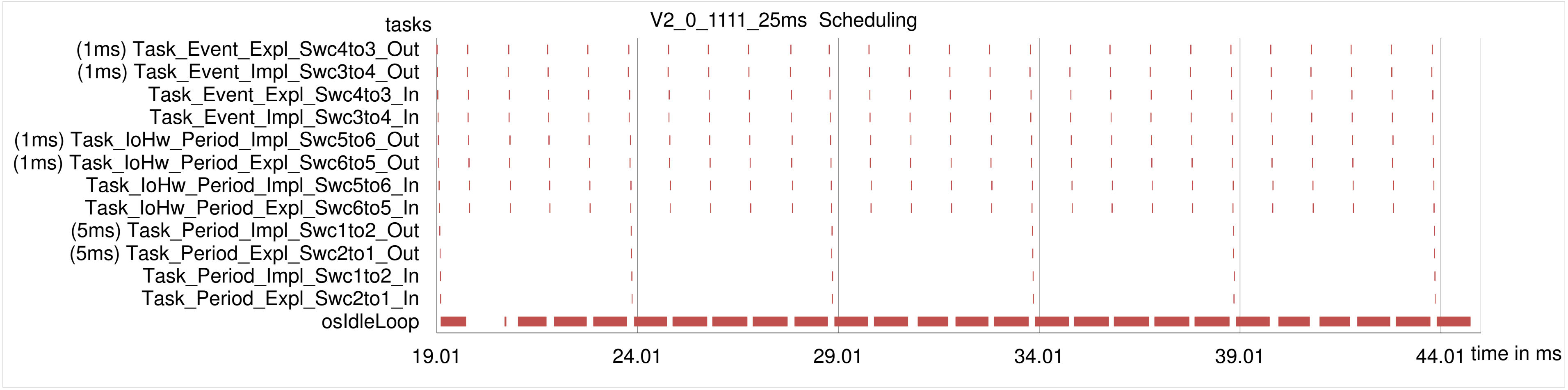


Figure 7-5: Task calling sequence for the first 25 ms of SingleCoreV2_0 for the case when each switch is closed

This figure proves that each writing task is called within their time constraints:

$$\left.\begin{array}{l}\text{Task\_Event\_Expl\_Swc4to3\_Out}\\ \text{Task\_Event\_Impl\_Swc3to4\_Out}\\ \text{Task\_IoHw\_Period\_Impl\_Swc5to6\_Out}\\ \text{Task\_IoHw\_Period\_Impl\_Swc6to5\_Out}\end{array}\right\}\ \text{invoked each millisecond}$$

$$\left.\begin{array}{l}\text{Task\_Period\_Impl\_Swc1to2\_Out}\\ \text{Task\_Period\_Impl\_Swc2to1\_Out}\end{array}\right\}\ \text{invoked each 5 millisecond}$$

These time constrains are also fulfilled when all switches are open (V2_0_0000), when switch 1 is the only open switch (V2_0_1011) and when switch 1 is the only closed switch (V2_0_0100). These measurements are detailed in Figure 7-6 on page 77, Figure 7-7 and Figure 7-8 on page 78 respectively. Each of these figures represent the calling sequence of the write and read task with one millisecond. In addition the figures show that the task scheduling process begins after the initialisation process of the system, and therefore within the 19th and 20th millisecond of the software run time. All writing tasks are called within the first millisecond of the scheduling process. Moreover, the writing tasks that have to be called each fifth millisecond are actually called in every fife millisecond from the last time of their execution. Also, each writing task that needs to be executed every one millisecond is performed within this time constraint. Therefore, the implementation SingleCore_V2_0 satisfies the requirement that each write process is triggered within their periodical call sequence.

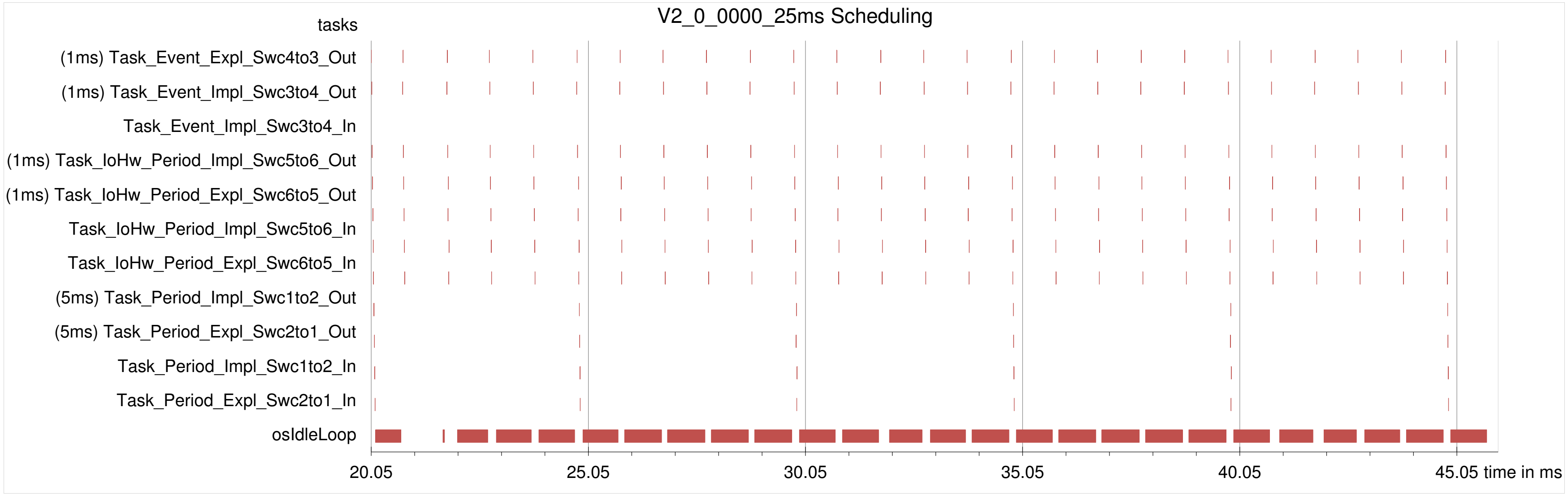


Figure 7-6: Task calling sequence for the first 25 ms of SingleCoreV2_0 for the case when all switches are open

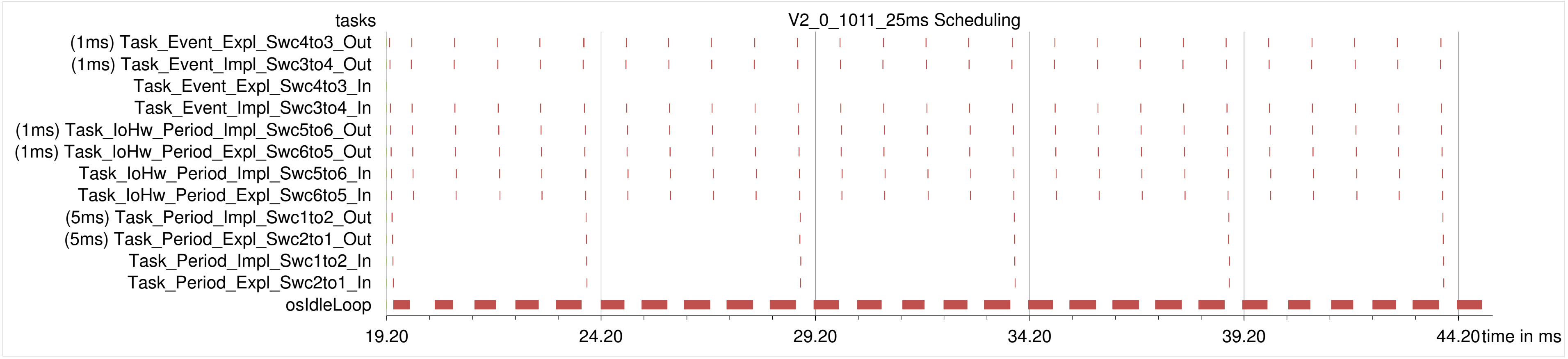


Figure 7-7: Task calling sequence for the first 25 ms of SingleCoreV2_0 for the case when switch 1 is the only open switch

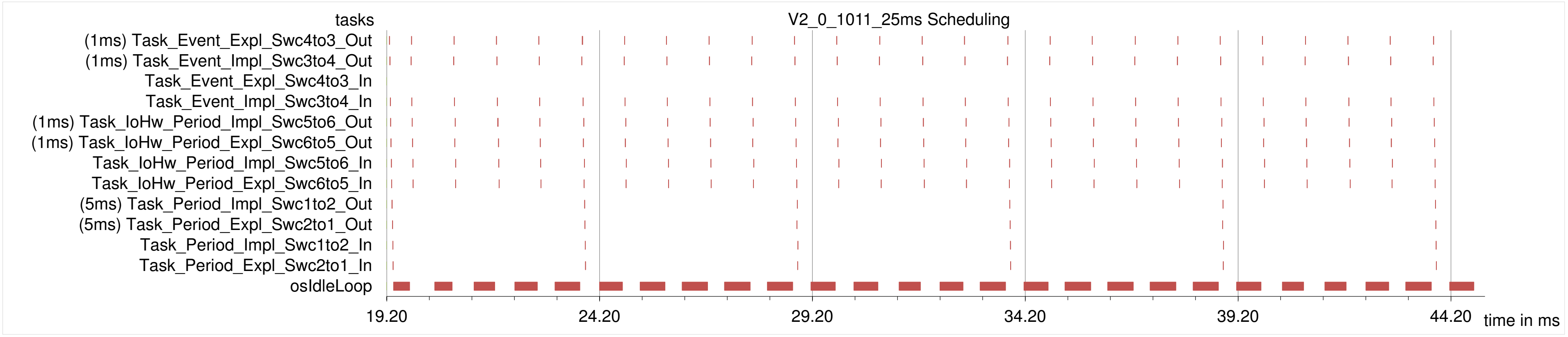


Figure 7-8: Task calling sequence for the first 25 ms of SingleCoreV2_0 for the case when switch 1 is the only closed switch

In addition to the correct calling pattern of the writing process in each case, it can be observed that in all four cases of task scheduling (V2_0_0000, V2_0_1111, V2_0_1011, V2_0_0100), the first reading task was executed after two successive write tasks. This calling pattern results from the implementation of scheduler of the operating system, which is used in both implementations.

In order to understand the scheduling mechanism from Vector, it is necessary to examine how the tasks are implemented in the source code as described in section 6.3 on page 58.

The scheduling mechanism from Vector is based on three queues: QTaskActivation, QTail and QHead. The main queue, QTaskActivation[number of priorities] [queue position], is implemented as a two dimensional array, where the identify number of tasks in status “READY” are placed in the order of their activation (first-in-first-out FIFO).

To simply illustrate the QTaskActivation queue, it can be represented as a two dimensional matrix. The rows represent priority queues, and the columns represent the position in the priority queue. Based on the information in Table 6-2 on page 61 the first row has no possible task identifier, because no task with a hardware priority of zero is available. The second row could be filled with the task with the task identifier (2), (3), (4) and (5) and so on. The QTaskActivation queue is filled with a ready task from the tail, which is defined through the QTail queue. The tasks are called from the head of the QTaskActivation queue, which is defined through the QHead queue.

A detailed description how those three queues are used is described in the following section. In case of the SingleCoreV2_0 implementation, which has six priorities, the main queue QTaskActivation is defined into 6 rows and 6 columns as seen in Figure 7-9 .

	Position	0	1	2	3	4	5	
Hardware priority	0	0	0					
	1	0	0	0	0	0	0	(2), (3), (4), (5)
	2	0	0	0	0	0	0	(6), (7), (8), (9)
	3	0	0	0	0	0	0	(10), (11), (12), (13)
	4	0	0					(0)
	5	0	0					

Figure 7-9: QTaskActivation queue

During the RAM initialisation, in the initialisation process of the application (in function *OsStartOs* see section 11.5.1 on page 132) all queues are initialised with zero. The queue will be then modified through the same function in the initialisation process. All tasks that autostart are activated, they will be first set to the state “READY” and then added into the QTaskActivation queue through the function osSysActivateTask(TaskIndex). Every other task, for which autostart mechanism is not activated, will be added into the QActivationqueue as soon as the corresponding alarm to each task occurs. The task alarms are set during the execution of the function Rte_Start , see section 11.5.1 on page 132.
In case of SingleCore_V2_0 is the autostart mechanism from task SchM_Sync and SchM_Async activated. These two tasks are added first into the QTaskActivation queue.

Two arrays are used to determine the start and end of each priority queue inside the QTaskActivation queue. QTail[number of priority] signifies the end of the QTaskActivation queue for each priority and is used to add task identifiers to the QTaskActivation queue. QHead[number of priority] signifies the start of each priority queue and is used to get the task in state "READY" with the highest priority to be executed next. Both queues are initialised with zero at the beginning.
As soon as a task has to be add into the QTaskActivation queue its state will be transformed into the state "READY" and the QTail queue will be parsed for the position in which its task index will be insert into the QTaskActivation queue.

For example, when task SchM_Sync with the task index of 1 and the hardware priority of 5 has to be added into the QTaskActivation queue, then the QTail queue will be searched for the right position in QTaskActivation queue. The priority of SchM_Sync not only defines in which element of QTail queue the right position for the QTaskActivation queue is contained, it also defines in which row of QTaskActivation queue it will be added. In this case SchM_Sync with the priority of 5 will be added into the QTaskActivation queue in the fifth row in the first column ("zero") as seen in Figure 7-10 on page 81, because the QTail queue fifth element contains the number zero. Before SchM_Sync is inserted into the queue QTaskActivation at the defined position, the value at the fifth element of QTail will be incremented.

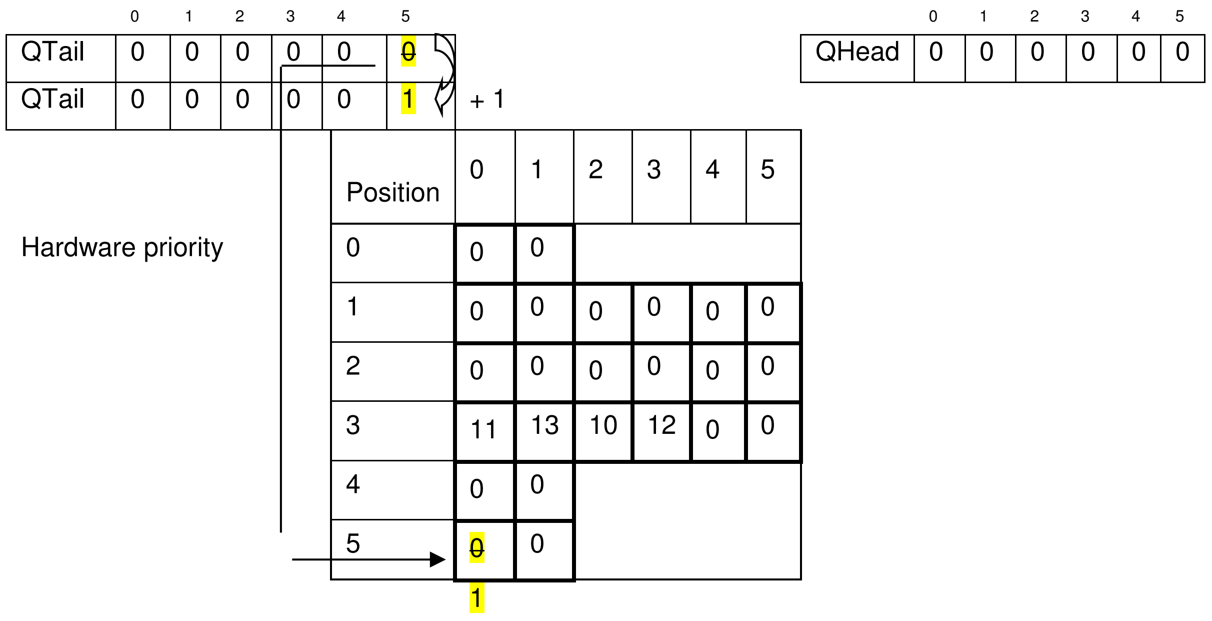


Figure 7-10: Insert process of task SchM_Sync to the QTaskActivation queue

Afterwards, the variable (QHighPrio) is modified. QHighPrio defines the highest priority of all tasks which are in state "READY" and it is initialised with zero. After each insert into the QTaskActivation queue through the QTail queue, the content of QHighPrio is bitwise OR with the priority mask of the task, which is added in the QTaskActivation. The priority mask of each task can be seen in Table 6-2 on page 61 for SingleCoreV1_0 and in Table 6-3 on page 67 for SingleCoreV2_0. The result of this operation is then stored in QHighPrio. With this method, only the highest priority will be stored in

QHighPrio. Therefore, the amount of leading zeros of each priority mask signifies the corresponding hardware priority of each priority mask.

For instance, Figure 7-11 illustrates the modification of QHighPrio after adding task SchM_Sync into the QTaskActivation queue. The priority mask of SchM_Sync has the value of 0x040000000. This priority mask is ORed OR with the QHighPrio and the result is stored back into QHighPrio. In this example, the new value of QHighPrio is 0x04000000 in hexadecimal, which represents 0000 0100 0000 0000 0000 0000 0000 0000 in binary. The new QHighPrio has 5 leading zeros, therefore priority 5 is the highest priority of all tasks in state “READY”.

QHighPrio	0x	0	0	0	0	0	0	0	0
Priority mask of SchM_Sync	0x	0	4	0	0	0	0	0	0
QHighPrio	0x	0	4	0	0	0	0	0	0

0x	= hexadecimal
b	= binary

Figure 7-11: QHighPrio ored with priotiy mask of task SchM_Sync

If it is required to call the task that is in state READY with the highest priority, firstly, the highest priority of all tasks in state “READY” will be identified through the content of QHighPrio.

For example,
Figure 7-12 shows the calling process if the priority mask, which is stored in QHighPrio, has three leading zeros. This means that three is the highest priority in this example. Therefore, the third element of the QHead queue will be considered to identify which task with the priority of three has to be called next.

The zero of the third element of QHead queue signifies the start position of the QActivationTask queue. The next task that will be executed is placed in the third row and column zero of QActivationTask. In this example, the task Task_Period_Expl_Swc2to1_Out with the task index 11 will be called next. Before the OS system (Dispatcher) sets this task to the state “RUN”, the third element of QHead queue is incremented to point to the next task as the start of the priority queue.

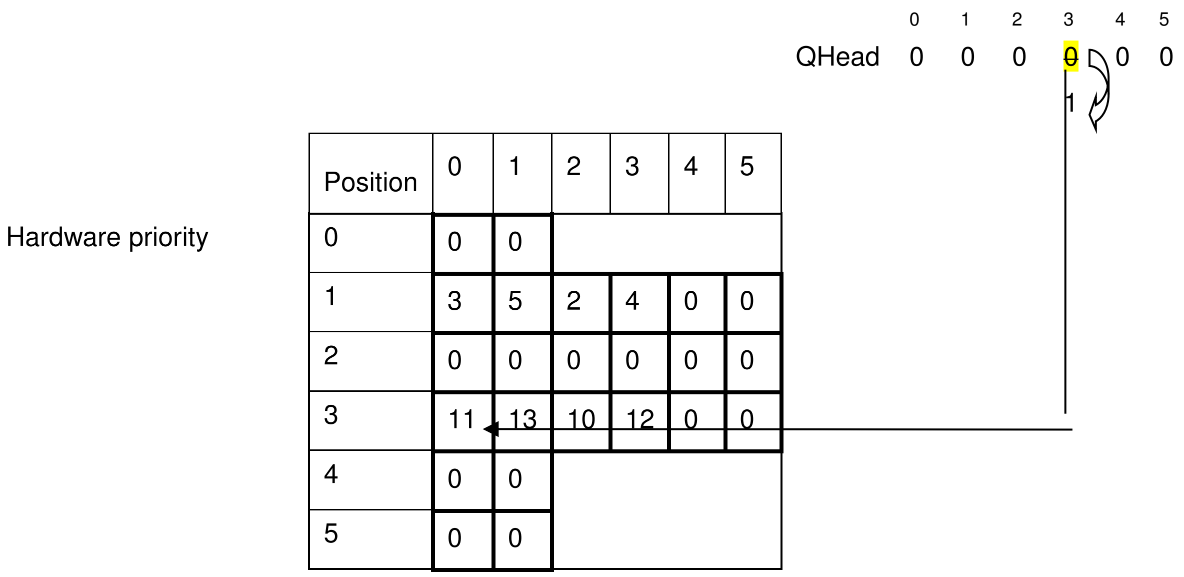


Figure 7-12: Dequeue process of task Task_Period_Expl_Swc2to1_Out out of QTaskActivation queue

To avoid adding a task index in a non-existing column, each element of QTail queue, as well as in QHead queue, has a maximum activation as seen in Figure 7-13 and Figure 7-14. After each modification, the content of the modified queue (QTail or QHead queue) will be checked if the maximum activation for each element is reached. If this is the case, the element with the maximum value will be reset to zero.

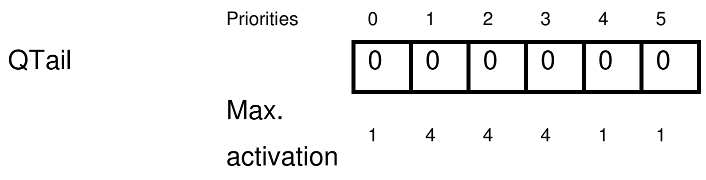


Figure 7-13: QTail queue with its maximum activation

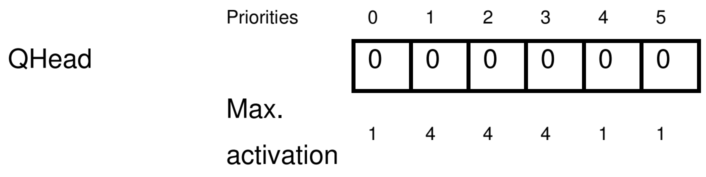


Figure 7-14: QTail queue with its maximum activation

To avoid the condition where the tail and head queue overlap each other, the element with the same number as the highest priority QHighPrio of both queues are compared with each other before a new task is executed. If both elements contain the same value, both pointer (head and tail) are pointing to the same column and row in the queue QActivationQeue, which means that this row of the QActivationQueue is empty and both queues have to be reset to zero.

With this implementation of the scheduling, the AUTOSAR requirement that tasks, which have the same priority will be executed in their activation order, see section 4 on page 36 can be easily fulfilled. In case of SingleCoreV2_0 have for example the tasks:

- Task_Event_Expl_Swc4to3_Out (task index = 3)
- Task_Event_Expl_Swc4to3_In (task index = 2)
- Task_Event_Impl_Swc3to4_Out (task index = 5) and
- Task_Event_Impl_Swc3to4_In (task index = 4)

the same hardware priority of 1 as illustrated in Table 6-3 on page 67.

Due to the fact that both writing tasks are firstly inserted into the queue as seen in Figure 7-12 on page 83, the triggered read tasks of both writing tasks are only able to be added after both writing tasks.

7.1.2.2. Requirements 2 and 3 test description and results

The verification to test if the read processes were triggered after new data is available was first observed on the task level of each implementation. Due to the fact that the write- and read-processes are included in their runnables of both implementations, the verification of the read-process for SingleCoreV1_0 and SingleCoreV2_0 had to be conducted at runnable and read function level instead of on task level.

The following figures in this subsection illustrate the calling sequence of runnables and communication functions. The main purpose of these figures is to provide an overview of the function calling sequence. For this reason, the squares in these figures are not representing the real execution duration of each function.

In addition, the implicit communication could not be captured in the program trace, since this communication type is generated as macros, as described in section 6.3. on page 58. For this reason, it was necessary to debug line-by-line through the first millisecond of the application with the debugger of WinIDEA, see section 5.3.4.1.2 on page 51. This method made it possible to investigate whether the implicit communication functions were executed. If data was written to and read out of the memory (RteBuffer), then the execution of the implicit write- and read-process were successful. The execution of the implicit write- and read-process are represented as yellow diamonds, and the corresponding implicit runnable are surrounded with a yellow rectangle in the following figures.

7.1.2.2.1. Verification of requirement 2 and 3 on SingleCoreV1_0

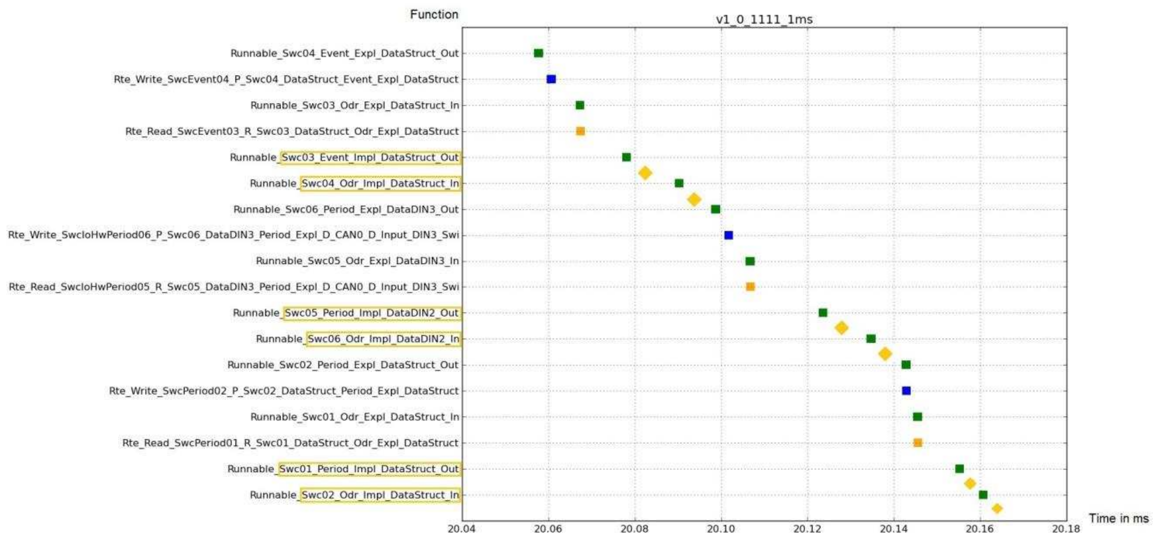


Figure 7-15: Runnable, write and read level scheduling for the case when each switch is closed from SingleCoreV1_0

Figure 7-15 shows the runnable, write and read functions when each switch was closed per millisecond for the first millisecond of the application SingleCoreV1_0. It can be observed that after the execution of each Runnable_.._Out (green squares), the execution of the Runnable_In with the read process is followed. The displayed execution sequence represents the correct scheduling of the communication process of each software component, where after each writing process the corresponding read process is followed. Moreover, each communication between the software components occur as expected when all switches are closed.

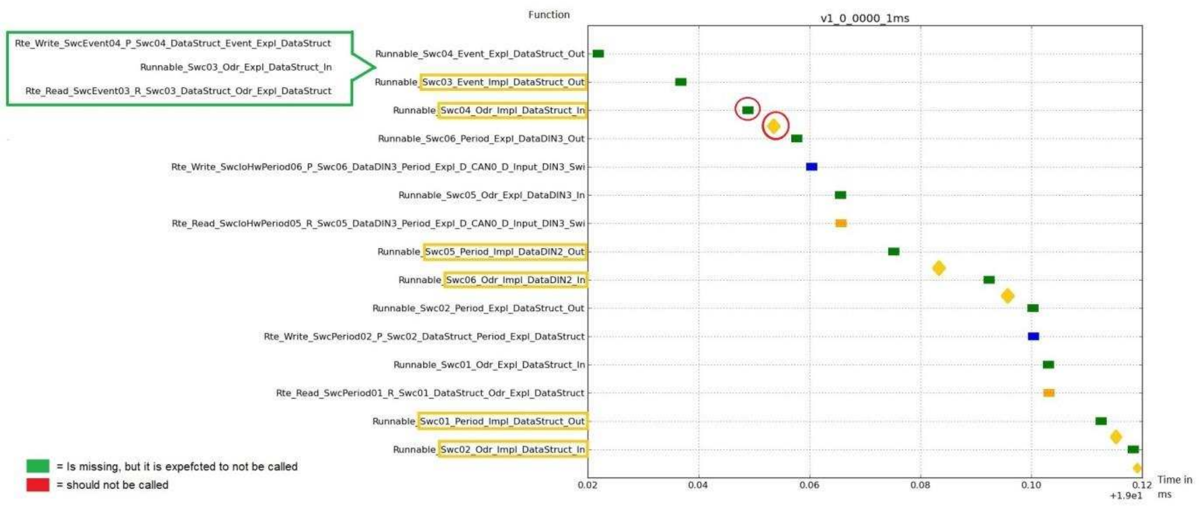


Figure 7-16: Runnable, write and read level scheduling for the case when each switch is open from SingleCoreV1_0

Figure 7-16 represents the case when each switch is open. In this case, it is required that the communication between SWCEvent03 and SWCEvent04 does not occur, since the corresponding switches (switch 0 and switch 1) are open. By comparing the read process in Figure 7-16 with the correct scheduling provided in Figure 7-15 on page 85**,** it can be observed that the write- and read-process of the explicit communication between SWCEvent04 and SWCEvent03:

- Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct,
- Runnable_Swc03_Odr_Expl_DataStruct_In and
- Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct

did not occur as expected. The implicit read process in Figure 7-16 with the Runnable_Swc04_Odr_Impl_DataStruct_In occurred, even if the write process had not been executed since switch 0 had not been closed. It also can be observed that the implicit write- and read-process between SWCIoHwPeriod05 and SWCIoHwPeriod06 occurred. This communication is expected, because the communication between these software components was design to exchange the status of switch 2. That means that the communication is independent of the status of switch 2.

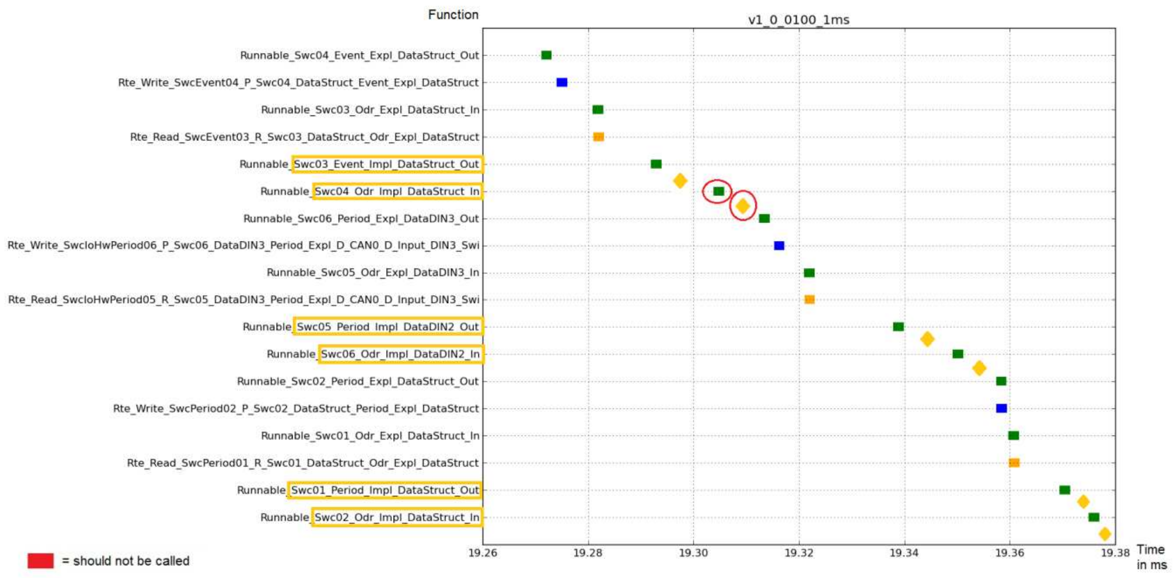


Figure 7-17: Runnable, write and read level scheduling for the case when only switch 1 is closed from SingleCoreV1_0

This incorrect execution of the implicit read process is also illustrated in the cases where switch 1 is the only closed switch as seen in Figure 7-17. It can be observed that although switch 0 is open the implicit reading process still occurs. However, in this case; the explicit communication between SWCEvent04 and SWCEvent03 occurred, since switch 1 is closed, which is represented through the following functions:

- Runnable_Swc04_Event_Expl_DataStruct_Out
- Rte_Wrtie_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct
- Runnable_Swc03_Odr_Expl_DataStruct_In and
- Rte_Read_SwcEvent03_R_Swc03_R_Swc03_DataStruct_Odr_Expl_DataStruct.

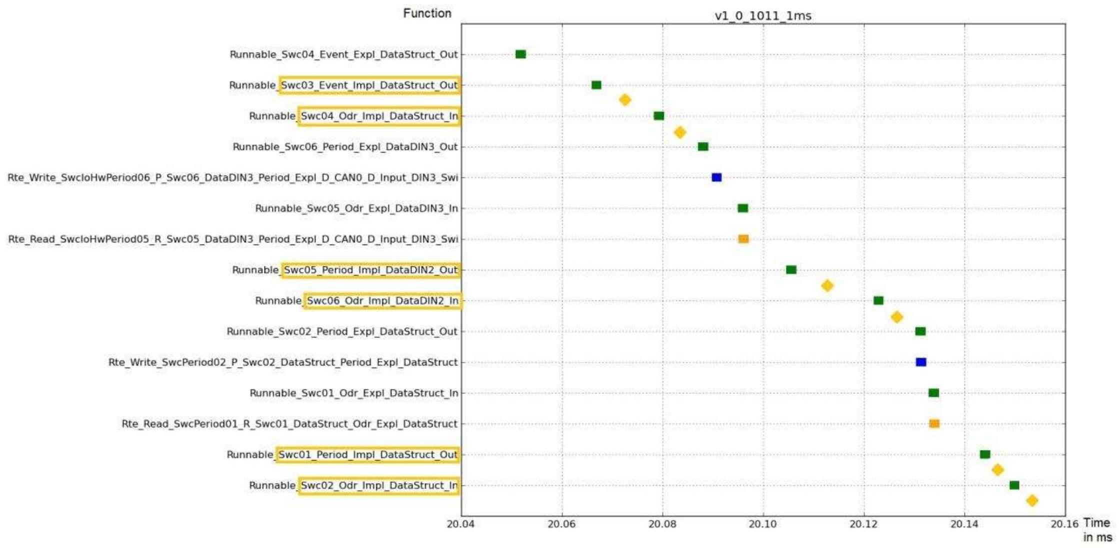


Figure 7-18: Runnable, write and read level scheduling for the case when only switch 1 is open from SingleCoreV1_0

In the case where only switch 1 is open as illustrated in
Figure 7-18, it is expected that the implicit communication between SWCEvent03 and SWCEvent04 occurred. Therefore, in this case, there is no incorrect scheduling to observe.

In all of these four cases, the status of switch 1 has an influence on the calling sequence of the read and write process. But the status of switch 0 only has a correct impact on the implicit write process itself but not on the read process. The read process of the implicit read in SWCEvent04 is called without any impact of the status of switch 0. This incorrect execution of the implicit read process is due to the generated SetEvent outside the implicit write function as seen in section 6.3.1.1 on page 61.

Therefore, the requirement that runnables, which include the read process, are called when new data is available is fulfilled for each explicit communication, but not for the implicit communication. Also, this scheduling analysis of SingleCore_V1_0 illustrates that only switch 1 has an influence on the calling sequence of the explicit write and read process between SWCEvent04 and SWCEvent03. Switch 0 has only an impact on the implicit write process between SWCEvent03 and SWCEvent04, but not an impact on the implicit read process. However, the scheduling of the communication between the SWCPeriod01 and SWCPeriod02 such as between SWCIoHwPeriod05 and SWC0IoHwPeriod06 remained constant as expected.

7.1.2.2.2. Verification of requirement 2 and 3 on SingleCoreV2_0

The same conclusion from SingleCore_V1_0 is observed in the runnable and communication process scheduling of SingleCoreV2_0, where the explicit read process is called only when new data is available and the implicit read process is called independent of the execution of the implicit write-process.

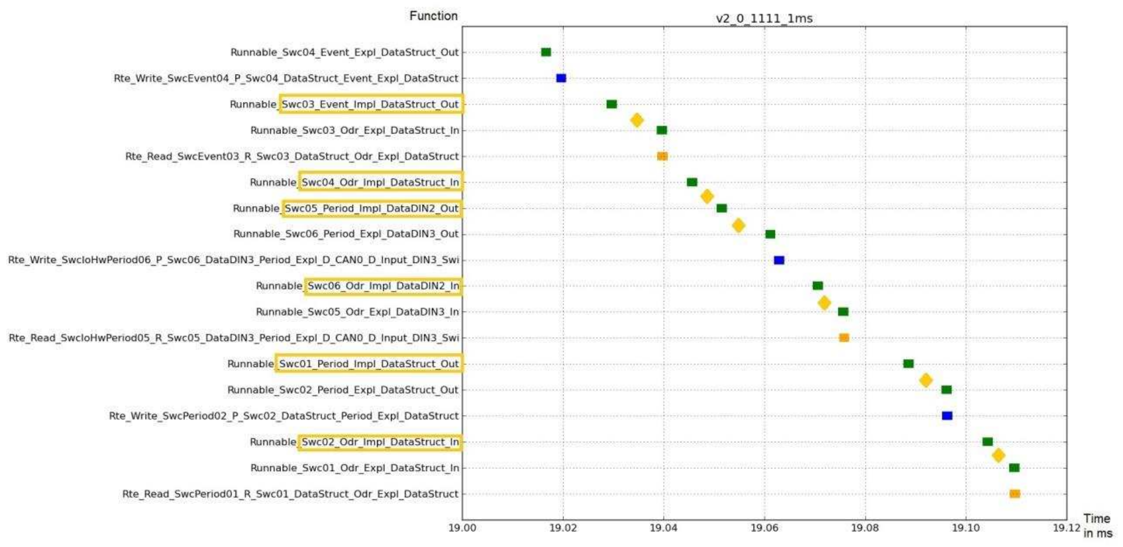


Figure 7-19: Runnable, write and read level scheduling for the case when all switches are closed from SingleCoreV2_0

It is expected when all switches are closed that all writing processes as well the reading processes occur. Figure 7-19 shows the correct write and read execution on runnable and communication level from SingleCoreV2_0 when each switch is closed.

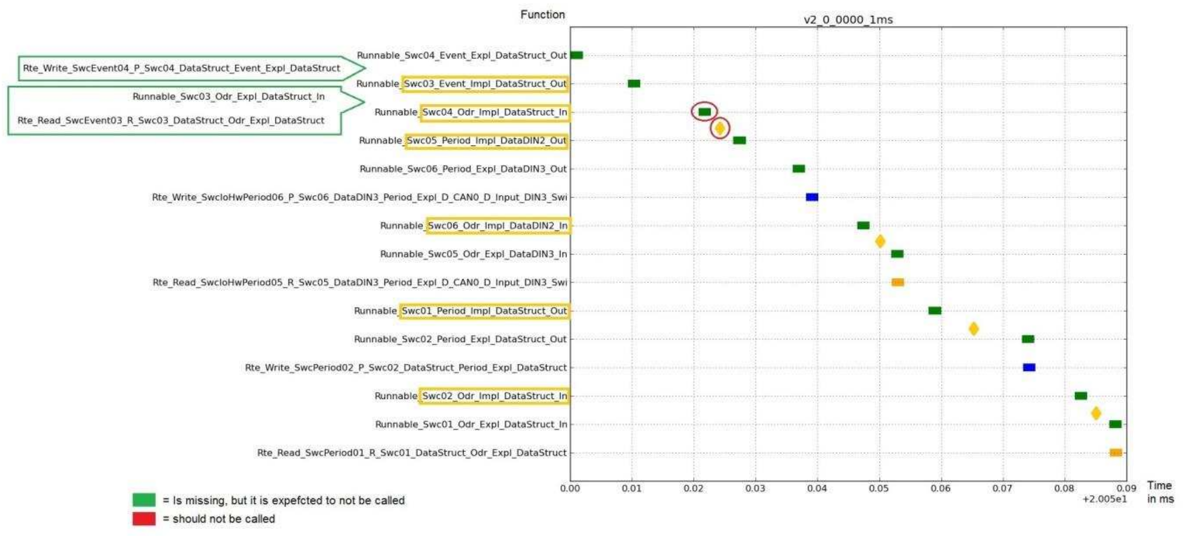


Figure 7-20: Runnable, write and read level scheduling for the case when all switches are open from SingleCoreV2_0

Figure 7-20 on page 88 shows that the read process in Runnable_Swc04_Odr_Impl_DataStruct_In for the case when each switch is open is called independent of the status of switch 0.

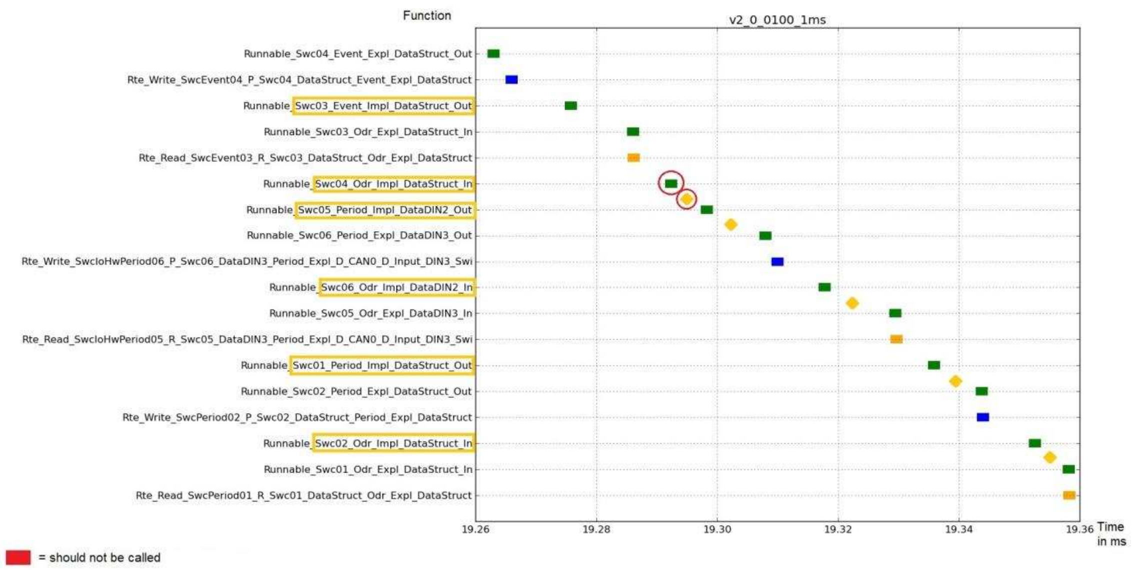


Figure 7-21: Runnable, write and read level scheduling for the case when switch 1 is the only close switch from SingleCoreV2_0

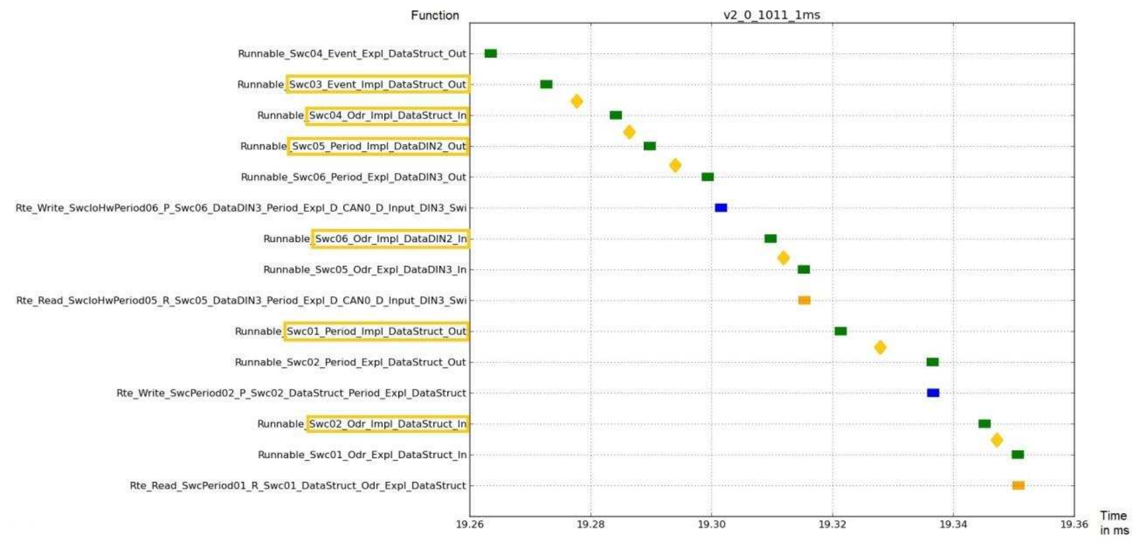


Figure 7-22: Runnable, write and read level scheduling for the case when switch 1 is the only open switch from SingleCoreV2_0

This failed behaviour was not just observed for the case when none switches are closed, it also occurrs for the cases where switch 1 is the only opened switch as seen in Figure 7-21. It also occurs when switch 1 is the only close switch. The incorrect read that is observed in each implicit communication can be drawn back to the incorrect placement of the notification function ActivateTask outside the implicit write function as shown in section 6.3.1.2 on page 67. Therefore, the status of switch 1 has the only influence on the calling sequence of the explicit write- and read-process, where switch 0 has just an influence on the implicit write process. Therefore, requirement number 2 is not fulfilled, because the implicit read process is still triggered when the write process did not occur.

7.1.3. Validation of the scheduling process

	SingleCoreV1_0				SingleCoreV2_0			
	1111	0000	0100	1011	1111	0000	0100	1011
Does the communication between SWC03 and SWC04 and the communication between SWC06 and SWC05 occurs every millisecond and the communication between SWC01 and SWC02 every fifth milliseconds?	✓	✓	✓	✓	✓	✓	✓	✓
Are the Runnables, which include the read process, only triggered when new data is available/was written?	✓	$^1/_2$ Only explicit communication	$^1/_2$ Only explicit communication	✓	✓	$^1/_2$ Only explicit communication	$^1/_2$ Only explicit communicatio n	✓
Does switch 0 and switch 1 has an influence on the write and read process?	✓	$^1/_2$ Only on the write and explicit read process	$^1/_2$ Only on the write and explicit read process	✓	✓	$^1/_2$ Only on the write and explicit read process	$^1/_2$ Only on the write and explicit read process	✓

Table 7-1: Results the scheduling validation of both implementations

In conclusion, Table 7-1 on page 90 shows that the write process of both implementation are called in their time constraints. However, runnables only invoke when new data is available for the cases each switch is closed or when switch 1 is the only opened switch. Switch 0, which is in the second case closed, triggers the implicit communication between SWCEvent03 and SWCEvent04 as expected. Nevertheless, if switch 0 is open, for instance for the cases 0100 and 0000 in Table 7-1 on page 90 , the implicit read process still takes place.

This unexpected execution of the read process is caused from the incorrect placement of the notification function SetEvent (in SingleCoreV1_0) and ActivateTask (in SingleCoreV2_0). The whole software source code is generated through the configured AUTOSAR development tools, see section 5.3.2 on page 49. The incorrect call of the notification function is therefore caused through the implementation of the AUTOSAR generator, which translate the entered configurations into source code. Therefore, switch 1 that triggers the explicit communication has an influence on the communication process of both communications. Whereby, switch 0 only has an impact on the implicit write processes, but not on the implicit read processes.

It might be assumed that with this unnecessary read process old data could be read. A closer look at the implicit writing and reading processes in section 6.3 on page 58 shows that the synchronisation process (copy process) takes place before the read process was notified. Afterwards the content of this RteBuffer is then copied in the implicit read buffer. This synchronisation process of the implicit read buffer and the RteBuffer guarantee that only the newest value is read. If non write processes occur, then the default value will be read. This default value is set in the configuration phase, see section 11.5.1 on page 132 with a value that would not cause any damage.

7.2. Performance analysis

This sub section explains the validation of both implemented communications, intra-task (SIngleCoreV1_0) and inter-task (SingleCoreV2_0) communication, regarding to their execution time. Firstly the methodology is explained with which the performance analyse was conducted. The results of this performance validation are then linked to the literature review.

7.2.1. Performance measurement methodology

The analysis of the communication execution time in both implementations is based on the same data that has been used for the scheduling analysis. An example of such data is given in section 11.6 on page 137 and in section 11.7 on page 149. This data has been gained from tracing all executed functions for the first 25 ms of the application with the real-time analyser winIDEA from ISystems, see section 5.3.4.1.1 on page 50. In contrast to the scheduling analyses, the performance analysis is focused on the communication functions, and the functions, which notify the read process to be executed (notification), rather than the calling pattern of the task functions. This communication consists of three parts, write process, read process and notification. A detailed overview of the functions that form this communication and their descriptions can be seen in section 6.3 on page 58. The following performance analysis begins with the description of the performance measurement methodology for the explicit and implicit communication.

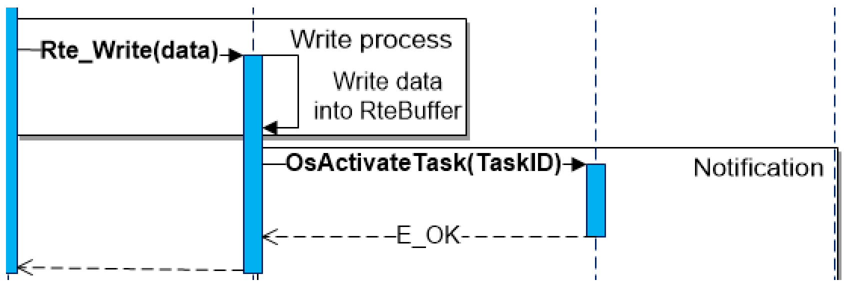


Figure 7-23: Performance measurement methodology of explicit write of SingleCoreV2_0

Figure 7-23 illustrates the execution time of the explicit write process that comprises the time that is spent in the Rte_Write function minus the time that was needed to execute the ActivateTask function by SingleCoreV2_0 and the SetEvent function for the SingleCoreV1_0.

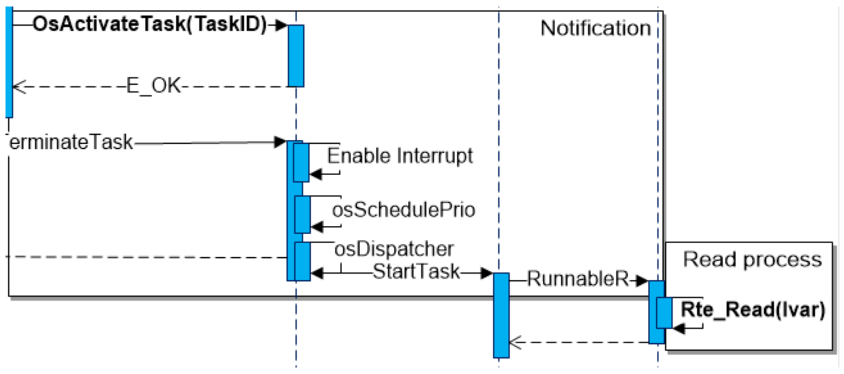


Figure 7-24: Performance measurement methodology of the explicit notification process of SingleCoreV2_0

The notification process of the explicit communication was measured from the beginning of the execution of ActivateTask function to the entry point of the Rte_Read function as demonstrated in Figure 7-24. The measurement of the notification process follows from the measurement of the read process, which determines the execution time of the Rte_Read function.

As the implicit write and read functions were generated as macros, it is not possible to apply the same performance measurement methodology as used for the explicit communication.

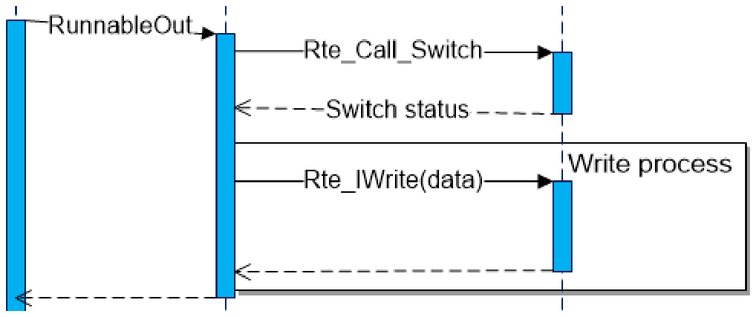


Figure 7-25: Performance measurement methodology of the implicit write process of SingleCoreV2_0

Therefore, the implicit write process execution time was inferred through the execution time of the RunnableOut function minus the execution time for the Rte_Call_Switch function as seen in Figure 7-25 .

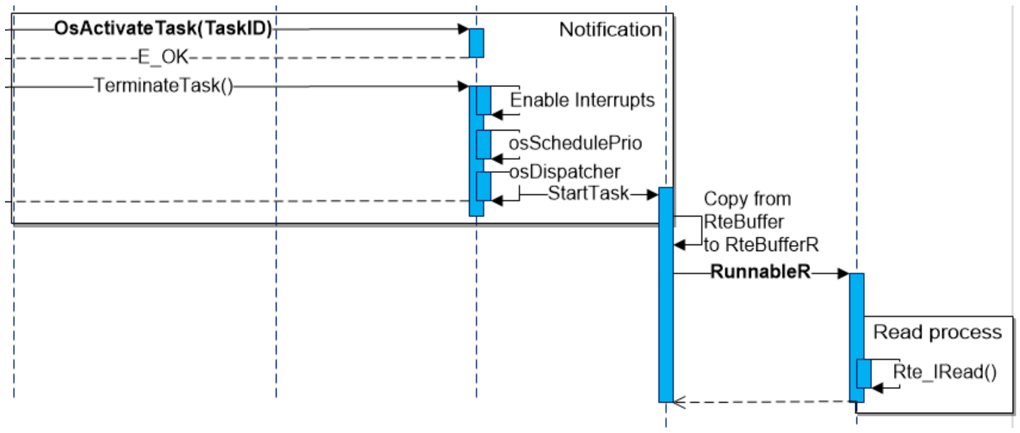
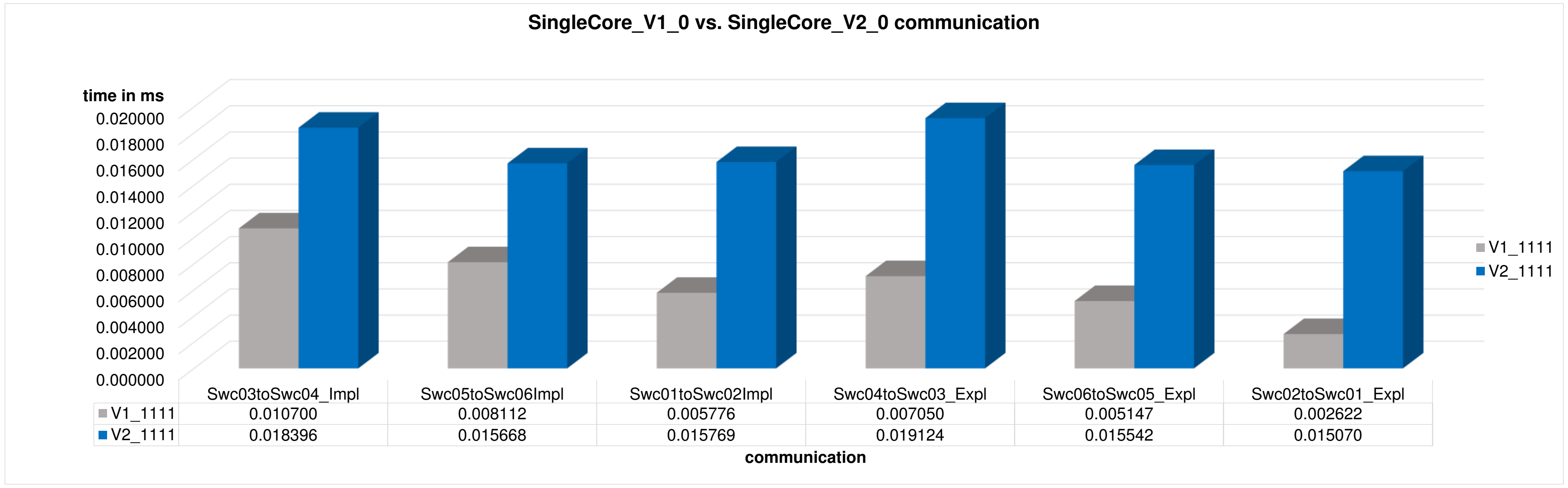


Figure 7-26: Performance measurement methodology of the implicit notification process of SingleCoreV2_0

The measurement of the execution time of the implicit notification process begins with the entry point of the ActivateTask function, but in contrast to the explicit notification process it ends with the entry point of the read-process. Whereas, the read-process is defined as the execution time of RunnableR, because the inline function RteIRead is the only instruction of the function RunnableR.

This performance analysis represents the worst case scenario when all switches are closed (1111) for both implementations. For comparison, the average of the measured communication execution time was taken by running the Python code, see section 11.1 on page 118, was implemented by Josh Higgins, and the results are demonstrated in the following charts.

7.2.1. Performance analysis results



	Swc03toSwc04_Impl	Swc05toSwc06Impl	Swc01toSwc02Impl	Swc04toSwc03_Expl	Swc06toSwc05_Expl	Swc02toSwc01_Expl
■ V1_1111	0.010700	0.008112	0.005776	0.007050	0.005147	0.002622
■ V2_1111	0.018396	0.015668	0.015769	0.019124	0.015542	0.015070

Figure 7-27: Intra-task communication versus inter-task communication graphical overview

Figure 7-27 demonstrates the time that is spent for the intra-task communication of all software components in SingleCoreV1_0 (V1_1111) and the inter-task communication of all software components in SingleCoreV2_0 in milliseconds. The graphical illustration of the time measurements already illustrate that the inter-task communication of SingleCoreV2_0 (V2_0_1111) takes more time to execute the required communication between the software components.

Table 7-2 provides a detailed overview of the exact differences of both implemented communication types, where the communication between each software component is summarised in microseconds.

Communication	Communication Time (µs)	Normalized
Intra-task (SingleCoreV1_0)	39.407	1
Inter-task (SingleCoreV2_0)	99.569	$\frac{99.569}{39.407} = 2.527$

Table 7-2: Intra-task versus inter-task communication time

To be able to compare the measurements of this thesis with the master thesis of (Moghaddam, 2013), the same procedure for the calculation of the percentage difference between both implementations were used. The execution time of intra-task communication was declared as the normal value with which the execution time of the inter-task communication is compared. As can be seen in the last column of Table 7-2 by setting the inter-task communication over intra-core communication time, the inter-task communication takes 153 % more time than for the intra-task communication.

A closer look at the execution time of the write, read and notification process could clarify the reason for this large difference between both communication types.

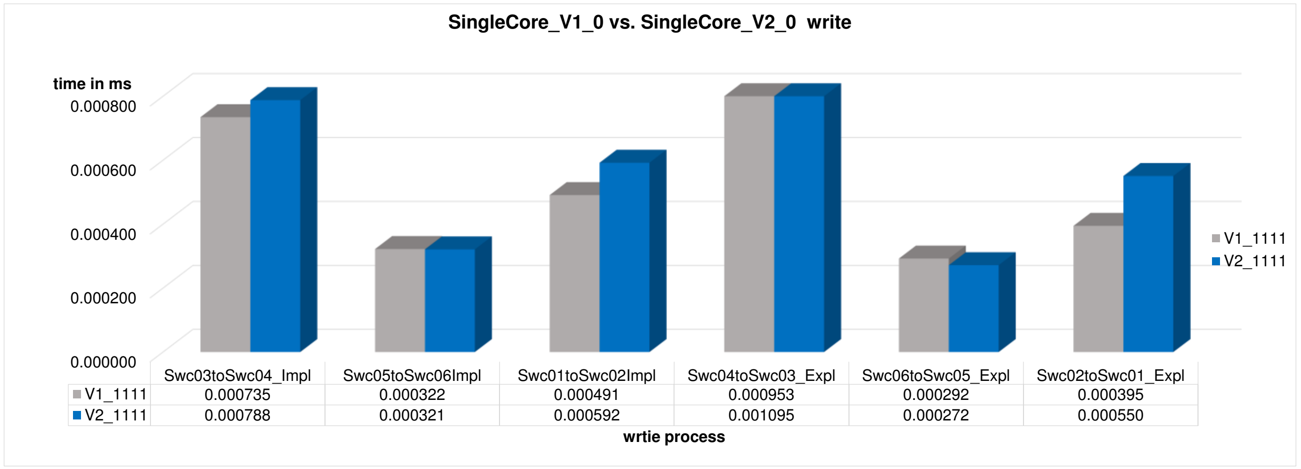


Figure 7-28: Intra-task write process versus inter-task write process

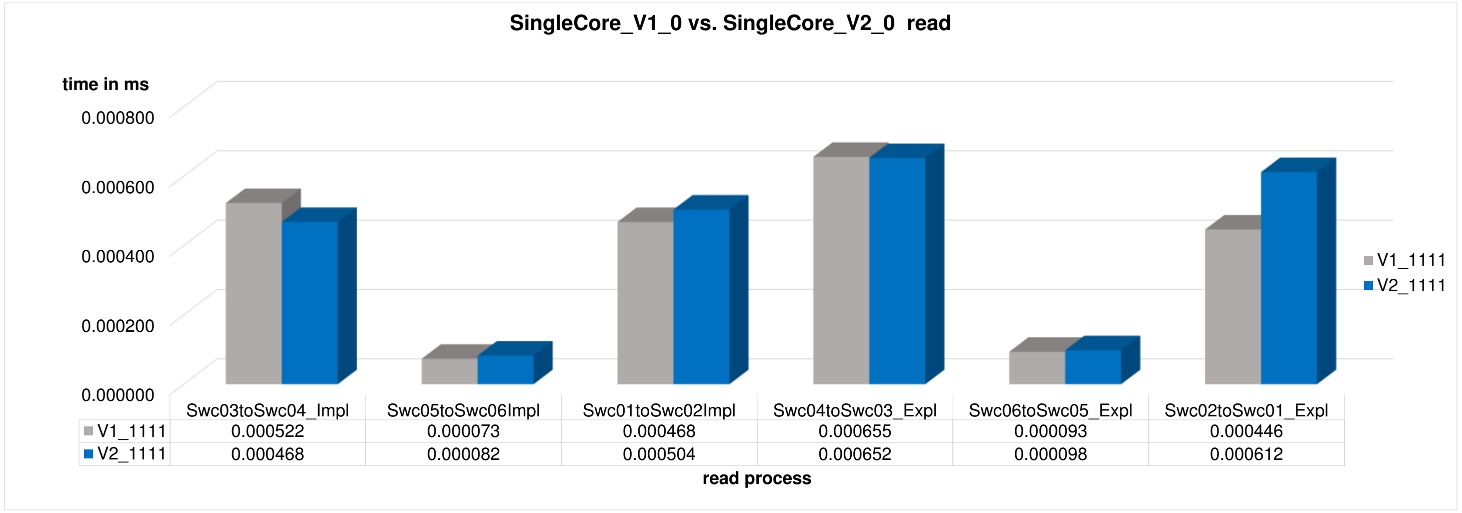
Figure 7-28 illustrates that the write-process of the SingleCoreV1_0 takes less time than the execution time for the write-process in SingleCoreV1_0 with the exception of the implicit communication between SWCIoHwPeriod05 and SWCIoHwPeriod06. Table 7-3 supports this statement by summarising the executed time for each implementation in the second column and calculate the difference between the two communication types.

Write process	**Communication Time (µs)**	**Difference (µs)**
Intra-task (SingleCoreV1_0)	3.188	3.618 - 3.188 = 0.430
Inter-task (SingleCoreV2_0)	3.618	

Table 7-3: Time that was spent to execute the write process of the intra-task versus inter-task communication

Both illustrations demonstrate that the write-process is not the main reason for the 153 % longer communication of inter-task communication versus intra-task communication. Moreover, this result shows that the write-process of intra-task communication takes 0.430 µs less than the write process of inter-task communication.

A closer look at the read process in Figure 7-29 also illustrate that the read-process of SingleCoreV2_0 could not cause the great difference of 60.126 µs between both communications.



	Swc03toSwc04_Impl	Swc05toSwc06Impl	Swc01toSwc02Impl	Swc04toSwc03_Expl	Swc06toSwc05_Expl	Swc02toSwc01_Expl
■ V1_1111	0.000522	0.000073	0.000468	0.000655	0.000093	0.000446
■ V2_1111	0.000468	0.000082	0.000504	0.000652	0.000098	0.000612

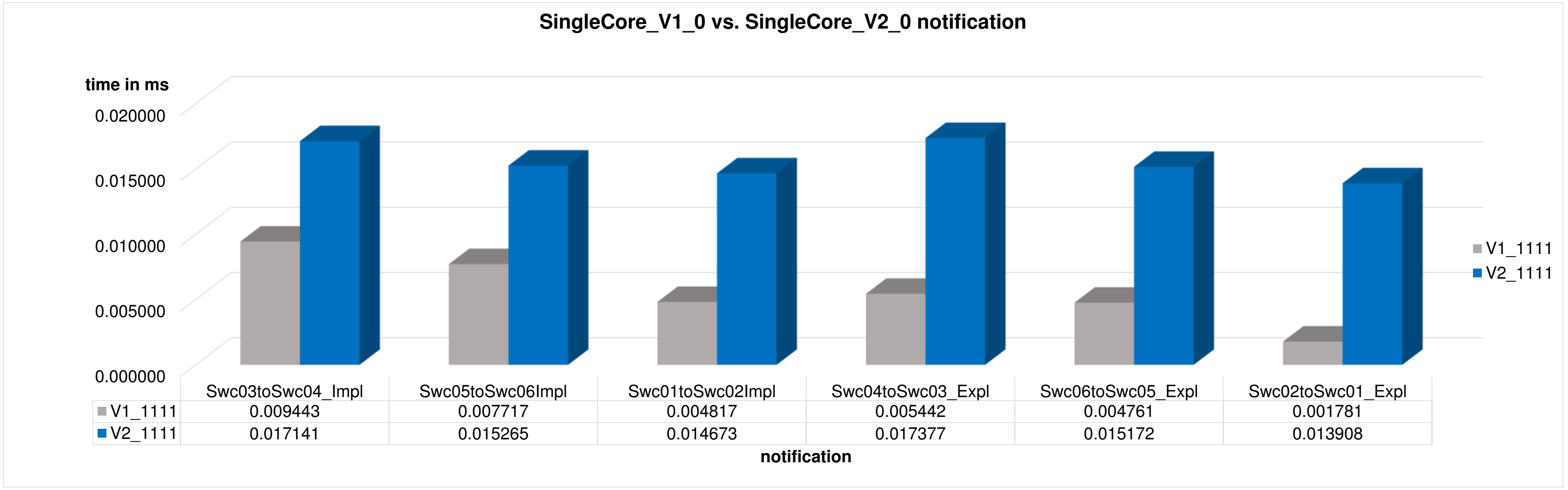
Figure 7-29: Intra-task read-process versus inter-task read process

Table 7-4 shows that the read process of SingleCoreV1_0 differs 0.159 µs from the read-process in SingleCoreV2_0. The difference of the write process (0.430 µs) added to the difference of the read process forms only a small fraction of the whole communication.

Read process	**Communication Time (µs)**	**Difference (µs)**
Intra-task (SingleCoreV1_0)	2.257	2.416 – 2.257= 0.159
Inter-task (SingleCoreV2_0)	2.416	

Table 7-4: Time that was spent to execute the read process of the intra-task versus inter-task communication

Finally Figure 7-30 shows that the notification process of SingleCoreV2_0 is the significant reason for the expensive inter-task communication.



	Swc03toSwc04_Impl	Swc05toSwc06Impl	Swc01toSwc02Impl	Swc04toSwc03_Expl	Swc06toSwc05_Expl	Swc02toSwc01_Expl
■ V1_1111	0.009443	0.007717	0.004817	0.005442	0.004761	0.001781
■ V2_1111	0.017141	0.015265	0.014673	0.017377	0.015172	0.013908

Figure 7-30: Intra-task notification versus inter-task notification

Table 7-5 provides the exact numbers for the notification process of both implementations. It shows that the notification process of SingleCoreV2_0 takes 60.162 µs more time than the notification process of SingleCoreV1_0.

Notification process	**Communication Time (µs)**	**Difference (µs)**
Intra-task (SingleCoreV1_0)	33.961	93.536 – 33.961= 60.162
Inter-task (SingleCoreV2_0)	93.536	

Table 7-5: Time that was spent to execute the read process of the intra-task versus inter-task communication

7.2.2. Verification of the performance analysis

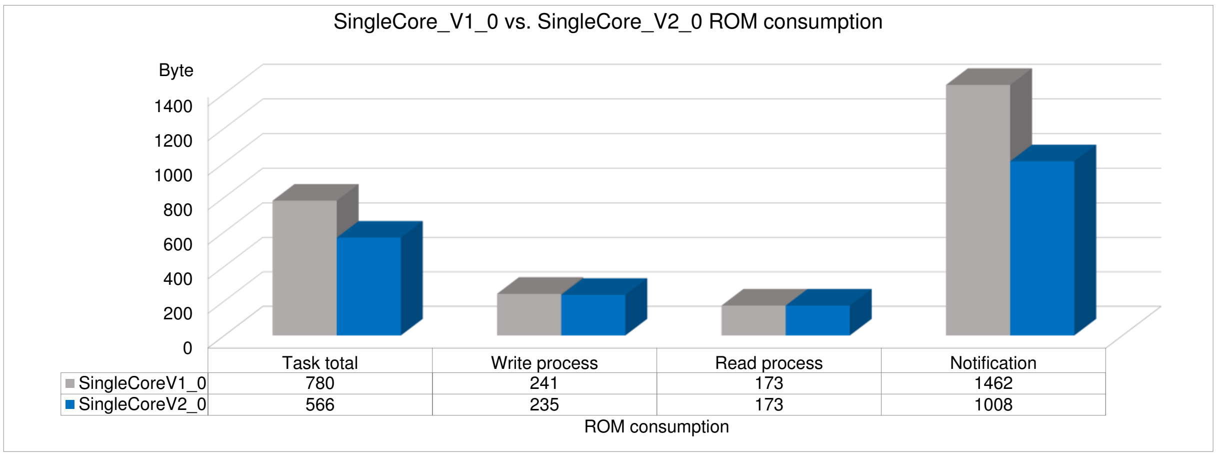
But these results also illustrate that the missing GetEvent function in each implicit communication of SingleCoreV1_0, see section 6.3.1.1 on page 61, has a smaller impact on the notification process than the incorrect scheduling of the write and read tasks of SingleCoreV2_0. The missing GetEvent function in each implicit communication of SingleCoreV1_0 leads to a longer notification process compared to the notification process of the explicit communication in SingleCoreV1_0. This increase in the execution time of the notification process in SingleCoreV1_0 is not as great as the increase of execution time of the notification process in SingleCoreV2_0. The incorrect scheduling of SingleCoreV2_0 can be seen in Figure 7-5 on page 76, where two sequential write process are executed until the first read process occurs. This misbehaviour and the calculation of the notification process lead to that the execution time of the second write function has be included into the calculation of the notification process of the first write-read communication. This disorder of the write process in SingleCoreV2_0 causes 153 % higher inter-task communication time than the intra-task communication.

The measured 153% difference between intra-/inter-task communication differs from a previous study conducted by Moghaddam, 2013, which measured 56% difference between intra-/inter-core communication. This disparity of almost 100% can be assumed to be due to the scheduling of the SingleCoreV2_0 component, and improving the scheduling mechanism here could reduce the difference between the results of these investigations. A sensible approach would be to improve the scheduling mechanism to ensure that the read process is only invoked after its corresponding write process, to eliminate or reduce the not corresponding write operation, which were observed previously in the performance analysis of SingleCoreV2_0.

7.3. Memory consumption

The memory consumption of both implementations was detected through the .map file that was generated from the linker of the WindRiver compiler, see section 5.3.3 on page 49.
The memory consumption of an application is one of the criteria to decide if a product will get to production. If an application requires more memory space than expected, the costs for the production will increase simultaneously. This section therefore analyse the need of memory of both implementations. Firstly, the amount of ROM is verified that is needed to store the static instructions of the application. The second part of this sub section focuses on the amount of RAM memory, where variables are stored, which content is modified during runtime.

7.3.1. ROM consumption



	Task total	Write process	Read process	Notification
■ SingleCoreV1_0	780	241	173	1462
■ SingleCoreV2_0	566	235	173	1008

Figure 7-31: ROM consumption of SingleCoreV1_0 vs. SingleCoreV2_0

Figure 7-31 illustrates the ROM memory consumption in byte that were gained through the .map file. It can be observed that on the one hand the read functions of both implementations are taking the same amount of ROM space for their execution. On the other hand Figure 7-31 shows that the communication of SingleCoreV1_0 takes 674 Bytes more ROM space for the tasks, write process and notification process than the communication in SingleCoreV2_0.

Communication type	ROM (Byte)	Difference (Byte)
Intra-task (SingleCoreV1_0)	2656	2656 – 1982= 674
Inter-task (SingleCoreV2_0)	1982	

Table 7-6: ROM consumption of both communications types

To identify the cause of the different amount of ROM that is needed to execute the functions, the execution coverage of WinIDEA was used as described in section 5.3.4.1.3 on page 51 and the tasks, write and notification level were analysed regarding to their memory consumption.

7.3.1.1. Task level and notification level

Source code 7-1 and Source code 7-2 on page 102 shows an exemplary ROM consumption of the implemented explicit tasks of both implementation. The comparison of the ROM consumption of both implementations shows that the difference of 214 Bytes on task level is caused through the following aspects:

- The write and read process is generated in one extended task in SingleCoreV1_0. This type of task leads to a greater number of executed instruction caused firstly through the ability to suspend the task to wait for an event and to continue it if the event occurs. The mechanism to wait for an event is realised with the first three functions in line 1 to 3 of Source code 7-1 and one if- statement to distinguish the occurred event, which are not needed in the basic task of SingleCoreV2_0 as seen in Source code 7-2 on page 102.

- Secondary, the notification process of SingleCoreV1_0 includes the functions: Schedule, GetEvent and ClearEvent, where in SingleCoreV2_0 only the execution of TerminateTask is needed, see Source code 7-2 on page 102. The greater amount of functions that has to be executed to complete the notification process in SingleCoreV1_0 leads to the higher amount of ROM (454 more bytes) that is needed compared to SingleCoreV2_0.

Whereby the additional 8 bytes that are added on the ROM consumption of SingleCoreV2_0 through the additional task function, as seen in Source code 7-2 on page 102, is not significant compared to the amount of instructions that are needed for the notification and wait mechanism.

		ROM [Byte]
0:	*Task_Event_Expl_Swc4to3func*	
1:	*{*	8
2:	*(void)WaitEvent(Rte_Ev_Run_SwcEvent04_Runnable_Swc04_Event_Expl_DataStruct_Out);*	6
3:	*(void)GetEvent(Task_Event_Expl_Swc4to3, &ev);*	10
4:	*(void)ClearEvent(ev & (Rte_Ev_Run_SwcEvent04_Runnable_Swc04_Event_Expl_DataStruct_Out));*	10
5:	*if ((ev & Rte_Ev_Run_SwcEvent04_Runnable_Swc04_Event_Expl_DataStruct_Out) != (EventMaskType)0)*	10
6:	*Runnable_Swc04_Event_Expl_DataStruct_Out();*	4
7:	*(void)Schedule();*	4
8:	*(void)GetEvent(Task_Event_Expl_Swc4to3, &evRun);*	10
9:	*if ((evRun & Rte_Ev_Run_SwcEvent03_Runnable_Swc03_Odr_Expl_DataStruct_In) != (EventMaskType)0)*	8
10:	*Runnable_Swc03_Odr_Expl_DataStruct_In();*	4
11:	*(void)ClearEvent(evRun & Rte_Ev_Run_SwcEvent03_Runnable_Swc03_Odr_Expl_DataStruct_In);*	10
12:	*}*	2
13:	*}*	8
	Total:	94

Source code 7-1: Source code of the Task_Event_Expl_Swc4to3 function of SingleCoreV1_0 with its ROM memory consumption

		ROM [Byte]
0:	*Task_Event_Expl_Swc4to3_Outfunc*	
1:	*{*	8
2:	*Runnable_Swc04_Event_Expl_DataStruct_Out();*	4
3:	*(void)TerminateTask();*	4
4:	*}*	8
	Total:	24
5:	*Task_Event_Expl_Swc4to3_Infunc*	
6:	*{*	
7:	*Runnable_Swc03_Odr_Expl_DataStruct_In();*	4
8:	*(void)TerminateTask();*	4
9:	*}*	8
	Total:	24
	Total:	48

Source code 7-2: Source code of the write and read task of the communication between SWCEvent04 and SWCEvent03 of SingleCoreV2_0 with its ROM memory consumption

7.3.1.2. Write process level

To understand the cause of the 6 bytes difference at the write-process level in Figure 7-31 on page 100, a closer look at the executed write functions is needed.

Write function	**V1_0 Size in Byte**	**V2_0 Size in Byte**	**Difference**
Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_...	**80**	**78**	**2**
Rte_IWrite_Runnable_Swc03_Event_Impl_DataStruct_Out_...	22	22	0
Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_...	**36**	**34**	**2**
Rte_IWrite_Runnable_Swc05_Period_Impl_DataDIN2_Out_...	01	01	0
Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_...	**80**	**78**	**2**
Rte_IWriteRef_Runnable_Swc01_Period_Impl_DataStruct_...	22	22	0
Write process	**241**	**235**	**6**

Table 7-7: Overview of the write-process and its ROM consumption for both implementations

Table 7-7 provides such detailed overview of the write-process and its ROM consumption for both implementations. It shows that the explicit write function in SingleCoreV1_0 needs more ROM memory to be executed compared to the amount of memory that is needed for SingleCoreV2_0. Whereby, Source code 7-3 and Source code 7-4 on following page illustrate that the need of more ROM memory is caused from the generated notification function in line 3. SingleCoreV1_0 calls the SetEvent function to start the notification process, while in SIngleCoreV2_0 the function ActivateTask, is used. The implementation of the function SetEvent (422 bytes) takes more memory space than the implementation of ActivateTask (408 bytes). The higher ROM consumption of the function ActivateTask can be traced/drawn back to the fact that by fetching the this task more ROM memory is needed.

		ROM size in byte
0:	*Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct*	
1:	*{*	8
2:	*Rte_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct = *(data);*	54
3:	*(void)SetEvent(Task_Event_Expl_Swc4to3, Rte_Ev_Run_SwcEvent03_Runnable_Swc03_Odr_Expl_DataStruct_In);*	8
4:	*return ret;*	2
5:	*}*	8
	Total:	80

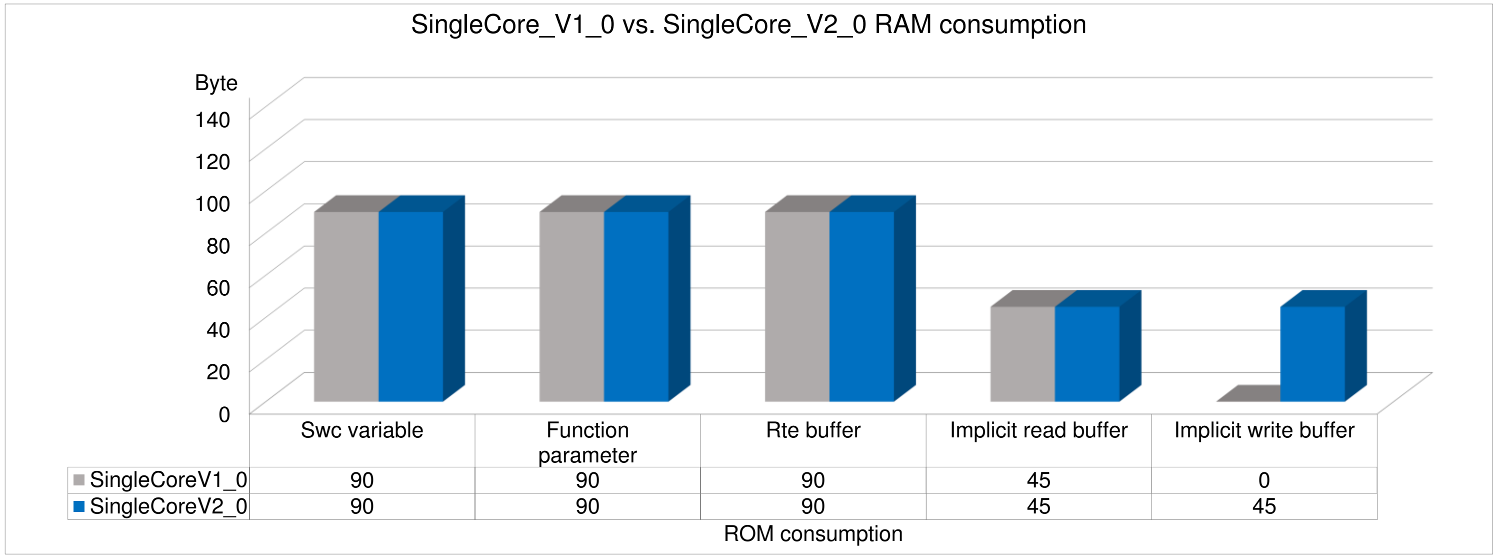
Source code 7-3: Source code of the Rte_Write of SWCIoHwPeriod06 in SingleCoreV1_0 with its ROM memory consumption

		ROM size in byte
0:	*Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct*	
1:	*{*	8
2:	*Rte_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct = *(data);*	54
3:	*(void)ActivateTask(Task_Event_Expl_Swc4to3_In);*	6
4:	*return ret;*	2
5:	*}*	8
	Total:	78

Source code 7-4: Source code of the Rte_Write of SWCIoHwPeriod06 in SingleCoreV2_0 with its ROM memory consumption

7.3.2. RAM consumption

The RAM consumption of both implementations differs in only one aspect. Figure 7-32 shows that the RAM consumption has an additional write buffer compared to SingleCoreV1_0.



	Swc variable	Function parameter	Rte buffer	Implicit read buffer	Implicit write buffer
■ SingleCoreV1_0	90	90	90	45	0
■ SingleCoreV2_0	90	90	90	45	45

Figure 7-32: RAM consumption of SingleCoreV1_0 vs. SingleCoreV2_0

7.3.3. Verification of the analyses of the memory consumption

The analysis of the memory consumption indicates that the intra-task communication in SingleCoreV1_0 takes 674 Bytes more ROM space than the inter-task communication of SingleCoreV2_0. This large amount of ROM difference between both implementations can be traced back to the implementation of both communication types. In SingleCoreV1_0 is the write and read process implemented in one extended task. Whereas, in SingleCoreV2_0 the write and read process were generated in two separated basic tasks. Not only has the extend task of SingleCoreV1_0 a greater amount of instructions, which is caused through the fact that it includes both communication processes, it also has more instructions because of its ability to transfer into the state “WAIT”. The ability of an extended task to wait for an event to get executed again lead to a longer (more instructions) notification process compared to the notification process of inter-task communication in SingleCoreV2_0. As a result, the implemented intra-task communication of SingleCoreV1_0 takes 674 more bytes of ROM than the inter-core communication in SingleCoreV2_0.

However, a closer look at the RAM consumption shows that in contrast to the previous ROM consumption the intra-task communication in SingleCoreV1_0 takes 45 bytes less of RAM than the inter-task communication of SingleCoreV2_0. This result can be drawn back to the implementation of the implicit inter-task communication of SingleCoreV2_0. This communication is generated under the approach of “Copy strategy” as described in section 3.3 on page 29. This approach includes that each implicit write and read-process is provided with its identical copy of the RteBuffer. Figure 7-34 on page 107 demonstrates this implementation. A comparison of the implementation of inter-task communication in Figure 7-34 with the intra-task communication of SingleCoreV1_0 in Figure 7-33 shows that both communication types are implemented with the same approach. The only difference between both implementations is that in intra-task communication a struct variable (ImpBuffer) with two members, member Out for the write and member IN for the read process, is used as the identical copy of the RteBuffer. While in inter-task communication, due to two separated tasks, two separate variables are used, which cause the additional 45 more Bytes in RAM.

The previous conducted studies, see section 3.5 on page 33, do not provide any information of the memory consumption. Therefore, with the analysis of the memory consumption a new aspect is introduce to analyse communication between tasks.

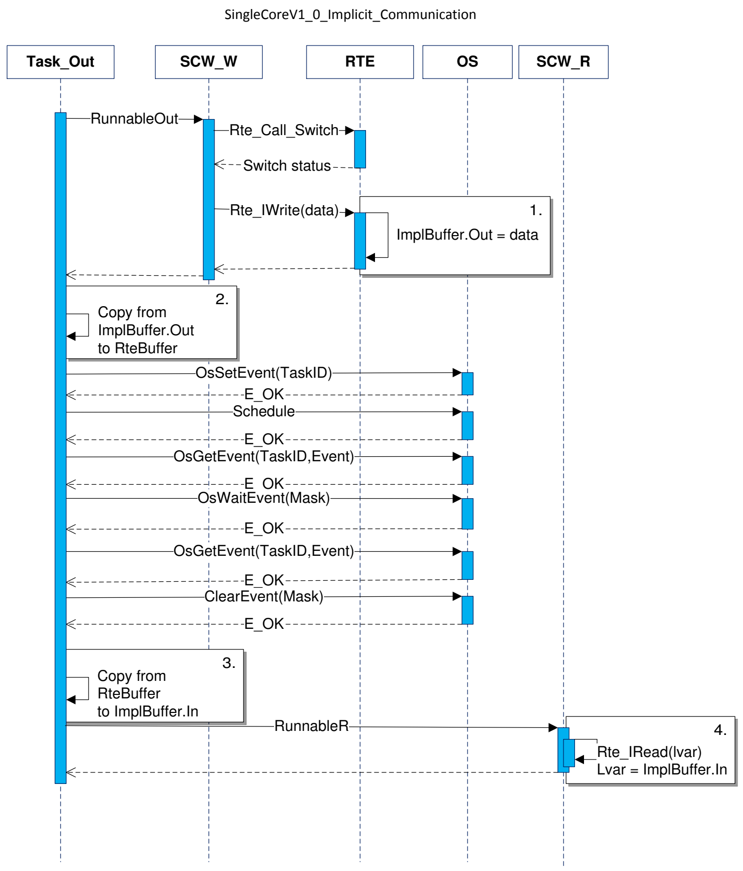


Figure 7-33: Sequence diagram of SingleCoreV1_0 implicit communication with focus on the copy process

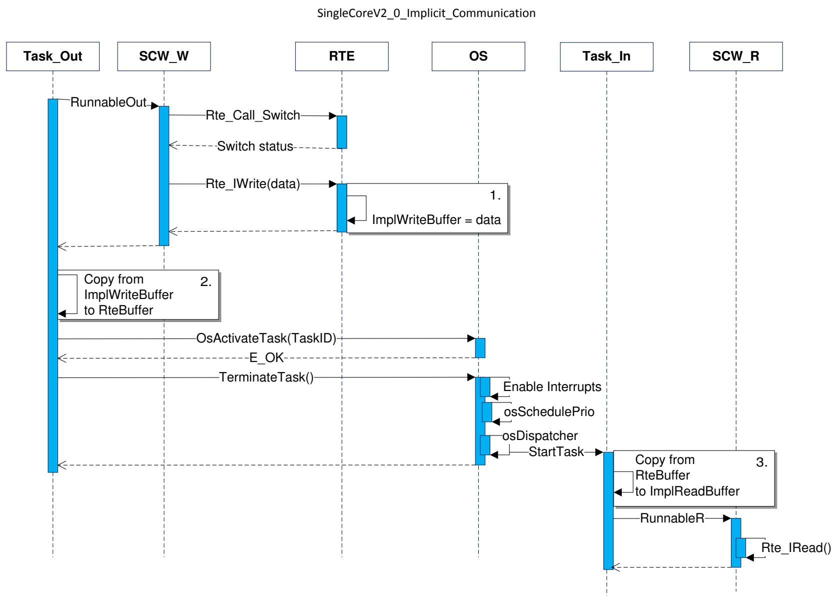


Figure 7-34: Sequence diagram of SingleCoreV2_0 implicit communication with focus on the copy process

8. Summary and conclusion

This section provides a summary of this work by addressing each objective and how these were archived.

1. Objective: To be able to conclude possible speed up gain of a heterogonous multi-core ECU, it is necessary first to investigate the current solution for inter-core communication.

The literature review in section 3 on page 27, presented conducted studies in this field and highlighted the possible bottlenecks of inter-core communication such as synchronization strategy, resource management and notification. Moreover, possible mechanism to tackle these challenges were presented and analysed. The investigation of current solution for inter-core communication also pointed out that it would be beneficial to put the theory of analysed mechanism into practice in the real use case study in automotive industry using a heterogeneous multi-core microcontroller.

2. Objective: The implementation of the mechanism to improve the inter-core communication requires appropriated hardware and software.

Therefore, the requirements for an appropriated system and the most suitable hardware were presented in section 5 on page 42. The investigation of possible hard- and software concluded that the hardware supplied already provided multi-core ECUs, but the software only supported AUTOSAR 3.2, supporting the single-core software architecture. The universal ECU VC121-12 from Vector featuring a heterogeneous dual- core microcontroller was used in this project. This ECU provided the opportunity to investigate the current state of intra-core communication regarding to synchronization strategy, resource management and notification.

3. Objective: For the investigation of the intra-core communication suitable experiments for intra-core and inter-core communication should be design. In addition, these experiments should represent software that is used in automotive industry and includes two different communication technics (explicit and implicit) defined in AUTOSAR standard.

These experiments were designed based on the finding from the review of existing systems, and information on ECU software in the automotive industry. These resources and the design of the experiment were presented section 4.

4. Objective: Before any possible improvements could be conducted the designed experiments has to be implemented by using the AUTOSAR methodology.

Therefore, the implementation and the detailed procedure of the hardware setup as well as the software tool sequence are provided in section 6. In addition, the output of the implementation is described and visualized.

5. Objective: The evaluation of the implemented experiments is mandatory to achieve improvements regarding to performance and memory consumption.

The implemented experiments were evaluated in section 7. The results of the evaluation are as follows:
<u>Scheduling</u>

- The implicit read process of SingleCoreV1_0 is executed regardless of whether new data was written by the writing process. This incorrect scheduling of the implicit reading process is due to that the notification process (SetEvent function) is placed outside the implicit write function. Therefore, the implicit reading process in SingleCoreV1_0 is invoked regardless of whether the writing process was executed or not.

- The evaluation of the scheduling process of SingleCoreV2_0 also showed an unexpected behaviour. Two write processes are called sequentially before one read process is triggered. An analyses of the scheduling mechanism resulted in that this behaviour is caused due to the fact that both writing tasks that have the same priority have to be called in the order of their activation, (AUTOSAR, 2014a p. 82 - 83).

<u>Performance analyses</u>
The results of the conducted scheduling analysis have also an impact on the performance of both communication types, intra-task communication and inter-task communication. The results of the performance analysis are as follows:

- In addition to the missing SetEvent function in the implicit write process of SingleCoreV1_0, the performance analysis highlighted that also each implicit communication is missing the second GetEvent function after the writing runnable was executed, as show in Figure 6-8 on page**.** Such an implementation lead to the implicit notification process structure so that is consists of the functions: WaitEvent, GetEvent, CleaEvent. That means the implicit notification process includes two more functions (WaitEvent and the first GetEvent) then the explicit notification process.

- Nevertheless, the execution time of the notification process in SingleCoreV1_0 is not as great as the execution time of the notification process in SingleCoreV2_0. The incorrect scheduling of SingleCoreV2_0, which was identifies in the scheduling analysis, caused the execution time of the second write function to be included in the calculation of the notification process of the first write-read communication. Finally, the disorder of the write process in SingleCoreV2_0 causes 153 % higher inter-task communication time than the intra-task communication of SingleCoreV1_0.

Memory consumption

The analysis of the memory consumption indicates that the implementation of SingleCoreV1_0, has a higher ROM consumption compared to the implementation of SingleCoreV2_0. The write and the read process are implemented in one extended task in SingleCoreV1_0, whereby the write and read process of SingleCoreV2_0 are implemented in two separate basic tasks. The extended task has the ability to wait for an event and to continue its execution as soon as the corresponded event occurs. Therefore, the extended task needs more instructions to perform this function. Furthermore, the number of instructions that are needed to execute the notification process of the intra-task communication in SingleCoreV1_0 are higher compared to SingleCoreV2_0. Therefore, the intra-task communication of SingleCoreV1_0 takes 674 Bytes more ROM space than the inter-task communication of SingleCoreV2_0.

In contrast to the ROM consumption, the intra-task communication in SingleCoreV1_0 takes 45 Bytes less of RAM than the inter-task communication of SingleCoreV2_0. This result is due to the fact that the implicit communication of the inter-task communication in SingleCoreV2_0 is implemented with an implicit Buffer.

6. Objective: Propose possible improvements in multi-core software design, especially for intra and inter-core communication, in order to improve execution time and memory consumption.

Possible improvements are presented in the following section 9 Future work.

It can be concluded that initial aim and objectives for this project are achieved. However, it is clear that further investigation is necessary to fully profile intra-core and inter-core communication software for multi-core processors applications in automobile industry, using future releases of AUTOSAR software.

9. Future work

The conducted work in this thesis is proving that the implementation of intra-core communication still offer scope for improvements especially regarding to the upcoming challenge inter-core communication.

For instance, it would be essential to subtract the execution time of the second write process from the execution time of the notification of SingleCoreV2_0. This would require to re-write the python code in section 11.1 in a way that it identifies the entry and exit point of the second write process automatically. A manual procedure would consume to much time and would lead to faults.

In addition, it would be beneficial to implement the designed experiments of this work on the updated version of the VC121-12 ECU software to AUTOSAR 4.2.1. The availability of the BSW software for AUTOSAR 4.2.1 was announced on 17th of February 2015 by Vector, (Vector, 2015). This means that the VC121-12 ECU software for AUTOSAR 4.2.1 will be released in the near future. The implementation of the designed experiments would enable comparison of the provided solution of Vector for inter-core communication with their current solution for intra-core communication.

10. References

Alekseev, Prof. Dr. (2012). *Script out of the module operating system*.

Amalthea. (2016). AMALTHEA An Open Platform Project for Embedded Multicore Systems. Retrieved 09/03/16, from http://www.amalthea-project.org/

Arm. Mictor 38. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0499d/BEHHIEGG.html

Atmel. (2008). 8-bit Microcontroller with 32K/64K/128K Bytes of ISP Flash and CAN Controller. 428. http://www.atmel.com/images/doc7679.pdf

Atmel. (2010). STK600 User Guide. 42. https://webcache.googleusercontent.com/search?q=cache:BXYYwmj6aDQJ:https://eewiki.net/download/attachments/27295747/AVRStudio%2520-%2520STK600%2520User%2520Guide%255B1%255D.pdf%3Fversion%3D1%26modificationDate%3D1382578574333%26api%3Dv2+&cd=1&hl=de&ct=clnk&gl=de

Atmel. (2016). STK600-TQFP64. Retrieved 13/03/16, from http://www.atmel.com/tools/STK600-TQFP64.aspx

AUTOSAR. (2003). AUTOSAR Development Partnership. Retrieved 02/05/15, 2015, from http://www.autosar.org/

AUTOSAR. (2008a). AUTOSAR Methodology. 38.

AUTOSAR. (2008b). Specification of BSW Scheduler. http://www.autosar.org/fileadmin/files/releases/3-1/software-architecture/system-services/standard/AUTOSAR_SWS_BSW_Scheduler.pdf

AUTOSAR. (2011). Specification of RTE. 621. http://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/rte/standard/AUTOSAR_SWS_RTE.pdf

AUTOSAR. (2012a). AUTOSAR Glossary. 87. http://www.autosar.org/fileadmin/files/releases/3-2/main/auxiliary/AUTOSAR_Glossary.pdf

AUTOSAR. (2012b). AUTOSAR Release 4.0. http://www.autosar.org/specifications/release-40/

AUTOSAR. (2012c). Specification of Operating System 230. http://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf

AUTOSAR. (2014a). Specification of Operating System. 144. http://www.autosar.org/fileadmin/files/releases/3-2/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf

AUTOSAR. (2014b). Specification of Timing Extensions. http://www.autosar.org/fileadmin/files/releases/4-2/methodology-and-templates/templates/standard/AUTOSAR_TPS_TimingExtensions.pdf

Ciulla, Vincent T. (2000). Electronic control unit (ECU). *Autorepair.about.com, 1*, 1-1.

Devika K., Syama R. (2013). An Overview of AUTOSAR Multicore Operating System Implementation. *International Journal of Innovative Research in Science, Engineering and Technology*.

Dijkstra, Edsger W. (1968). A constructive approach to the problem of program correctness. *BIT Numerical Mathematics, 8*(3), 174-186.

Dijkstra, Edsger W. (2001). Solution of a problem in concurrent programming control *Pioneers and Their Contributions to Software Engineering* (pp. 289-294): Springer.

Feljan, Juraj, & Carlson, Jan. (2013). *The impact of intra-core and inter-core task communication on architectural analysis of multicore embedded systems.* Paper presented at the The Eighth International Conference on Software Engineering Advances (ICSEA).

Fons, F, & Fons, M. (2012). FPGA-based automotive ECU design addresses AUTOSAR and ISO 26262 standards. *Xcell journal, 78*, 20.

Francisco, & Fons, Mariano. (2012). FPGA-based automotive ECU design addresses AUTOSAR and ISO 26262 standards. *EETimes, 1*, 13-13.

Freescale Semiconductor, Inc. (2009). Embedded Multicore: An Introduction. http://www.funkschau.de/fileadmin/media/whitepaper/files/111_embmcrm.pdf

Freescale Semiconductor, Inc. (2014). Automotive MCUs and MPUs. (27/04/15). http://cache.nxp.com/files/microcontrollers/doc/roadmap/BRAUTOPRDCTMAP.pdf

Gai, Paolo, Lipari, Giuseppe, & Natale, Marco Di. (2001). *Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip.* Paper presented at the Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE.

Gallmeister, Bill. (1995). *POSIX. 4 Programmers Guide: Programming for the real world*: " O'Reilly Media, Inc.".

GmbH, Robert Bosch. (2008). Mobility Solutions. 10/07/08, from http://www.bosch-presse.de/presseforum/details.htm?txtID=3645&locale=en

GmbH, Vector Informatik. (2011a). *TechnicalReference_Mircrosar_Os (Unpublished)*. Vector Informatik GmbH.

GmbH, Vector Informatik. (2011b). *User Manual VC121-12 (Unpublished)*. Vector Informatik GmbH.

Grave, Rudolf. (Embedded Multi-Core Conference, 17/06/15). *Software integration challenge multi-core - experience from real world projects*, Munich.

Höttger, Robert, Krawczyk, Lukas, & Igel, Burkhard. (2015). Model-based automotive partitioning and mapping for embedded multicore systems. *International Journal of Computer, Control, Quantum and Information Engineering, 9*(1), 268-274.

Instruments, National. (2009). ECU Designing and Testing using National Instruments Products. http://www.ni.com/white-paper/3312/en/

Intel. (2014). Intel Core2 Duo Processor E6700. Intel Web side.

ISO. (2004). ISO/IEC 14776-115:2004. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38217

ISO. (2013). ISO 11898-6. http://www.iso.org/iso/catalogue_detail.htm?csnumber=59165

iSystem.). winIDEA. Retrieved 06/03/15, from http://www.isystem.com/products/software/winidea

ISystem.). *winIDEA help section*. (9.12.270).

iSystem. (2009). Collect Trace Information on MCUs without Trace Port. 6. http://www.isystem.com/files/downloads/Articles/Collect%20Trace%20Information%20on%20MCUs%20without%20Trace%20Ports.pdf

iSYSTEM. (2012). winIDEA's help section. Retrieved 13/02/15, from http://www.isystem.com/downloads/winIDEA/help/index.html?iC5000OnchipAnalyzer.html

ITEA. (2016). ITEA 3. from https://itea3.org/

Jena, Santosh Kumar, & Srinivas, MB. (2012). *On the suitability of multi-core processing for embedded automotive systems.* Paper presented at the Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on.

JOSE, SAN. (2007). Freescale Opens Licensing of Power Architecture(TM) e200 Core Family Through IPextreme. http://www.design-reuse.com/news/15544/freescale-opens-licensing-power-architecture-e200-core-through-ipextreme.html

Knutsson, Sven. (2010). *PowerPoint-version av Autosar - Automotive Open System Architecture*. Götenburg University.

Lakshmanan, Karthik Singaram, Bhatia, Gaurav, & Rajkumar, Ragunathan. (2011). Autosar extensions for predictable task synchronization in multi-core ECUs: SAE Technical Paper.

Moghaddam, Abdollah Safaei. (2013). *Performance Evaluation and Modeling of a Multicore AUTOSAR system.* (Master of Science Thesis), Chalmers Unicersity of Technology. Retrieved from http://publications.lib.chalmers.se/records/fulltext/193956/193956.pdf

Noergaard, Tammy. (2005). *Embedded systems architecture: a comprehensive guide for engineers and programmers*. Amsterdam;London;: Newnes.

OSEK/VDX. (2005). OSEK/VDX. *Operating System*, 86. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf

Rajkumar, Ragunathan. (1990). *Real-time synchronization protocols for shared memory multiprocessors.* Paper presented at the Distributed Computing Systems, 1990. Proceedings., 10th International Conference on.

Rath, Dominic. (2008). Open On-Chip Debugger: Spen.

Reinl, Roland. (2014). *VC Startup Code*. Vector.

Ribbens, William B. (2013). *Understanding automotive electronics: an engineering perspective* (Vol. 7th.;7th;7;7th;). Amsterdam: Butterworth-Heinemann.

Rouse, Margaret. (2007). Definition debugging. Retrieved 13/03/16, from http://searchsoftwarequality.techtarget.com/definition/debugging

Rudolf Grave , Stefan Krämer (2014). Software optimal auf mehrere Kerne verteilen. *Elektronik automotive*.

Salah, Mohamed. (2016). Webinar: Migrating to AUTOSAR 4.2: Mentor Graphics.

Scott, Tony. (2004). "Keynote Talk". CeBIT America Conference.

Semiconductor, Freescale. (2008). e200z0 Power Architectur Core Reference Manual. 288. http://cache.freescale.com/files/32bit/doc/ref_manual/e200z0RM.pdf

Semiconductor, NXP. (2014). SBC Gen2 with CAN High Speed and LIN Interface. http://cache.nxp.com/files/analog/doc/data_sheet/MC33903-MC33904-MC33905.pdf?fpsp=1&WT_TYPE=Data%20Sheets&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation&fileExt=.pdf

Semiconductors, NXP. (2007). MPC5510EVB User Manual. 78. http://cache.freescale.com/files/dsp/doc/ref_manual/MPC5510EVBUM.pdf

Semiconductors, NXP. (2014). MPC5510 Microcontroller Family Reference Manual. 1018. http://cache.nxp.com/files/32bit/doc/ref_manual/MPC5510RM.pdf

Senthilkumar, K., & Ramadoss, Ramesh. (2011). *Designing multicore ECU architecture in vehicle networks using AUTOSAR*.

Sha, Lui and Rajkumar, Ragunathan and Lehoczky, John P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *Computers, IEEE Transactions on, 9*, 1175--1185.

Siebert, Fridtjof. (2010). Multicore Systems–Challenges for the Real-Time Software Developer. 8.

Silverio Miyashiro, Magda A, Ferreira, Mauricio GV, & Sant'Anna, Nilson. (2015). *CMMI-DEV process areas modeled on a process for critical embedded systems development.* Paper presented at the Science and Information Conference (SAI), 2015.

Soffel, Volker. (2004). Developing Embedded Systems—A Tools Introduction. *embedded, 2*, 2-2.

Standard, I. S. O. (1993). ISO 11898, 1993. *Road vehicles--interchange of digital information--Controller Area Network (CAN) for high-speed communication, 100*, 100-100.

StMicroelectronics. (2013a). Getting started tutorial for SPC564Bxx and SPC56ECxx family.

StMicroelectronics. (2013b). SPC564Bxx, SPC56ECxx 32-bit MCU family built on the embedded Power Architecture.

StMicroelectronics. (2016). STM8AF62. Retrieved 14/03/16, from http://www.st.com/web/en/catalog/mmc/SC1244/SS1214/LN6

Vector. (2007). *MICROSAR BSW Scheduler*.

Vector. (2010). ELearning CAN Controller Area Network. http://elearning.vector.com/index.php?seite=vl_can_introduction_en&root=378422&wbt_ls_kapitel_id=489560&wbt_ls_seite_id=489561&d=yes

Vector. (2011). *User Manual VC121-12*.

Vector. (2012). *Startup Vector AUTOSAR Solution*.

Vector. (2014a). ECU Development VC121-12. 1. https://vector.com/vi_downloadcenter_en.html#

Vector. (2014b). VC121-12 Factsheet. http://vector.com/portal/medien/cmc/factsheets/VC121-12_FactSheet_EN.pdf

Vector. (2015). Vector News. Retrieved 23/03/16, from https://vector.com/vi_news_en.html#!vi_news_detail_iframe_en,,,1355155,detail.html

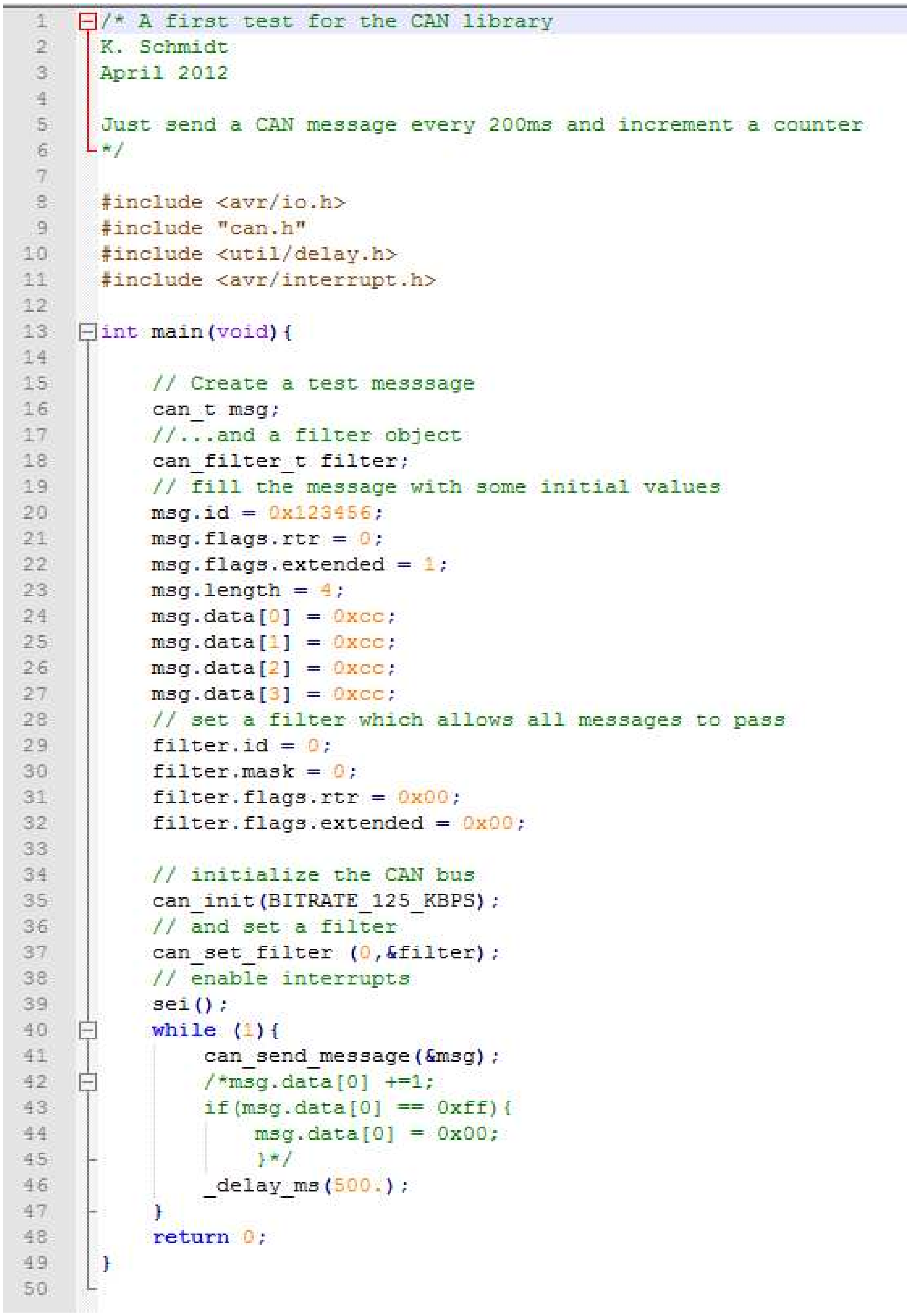
Wei, Ting-Ying, Qiu, Zhi-liang, Young, Chung-ping, & Chang, Da-Wei. (2011). *Development of heterogeneous multi-core embedded platform for automotive applications.* Paper presented at the 2011 International Conference on Circuits, System and Simulation (IPCSIT).

WindRiver. (2010). *User's Guid WindRiver Diab Compiler 5.8*.

Zeng, Haibo, & Natale, Marco Di. (2011). *Mechanisms for guaranteeing data consistency and flow preservation in AUTOSAR software on multi-core platforms.* Paper presented at the Industrial Embedded Systems (SIES), 2011 6th IEEE International Symposium on.

11. Appendix

11.1. STK600 source code

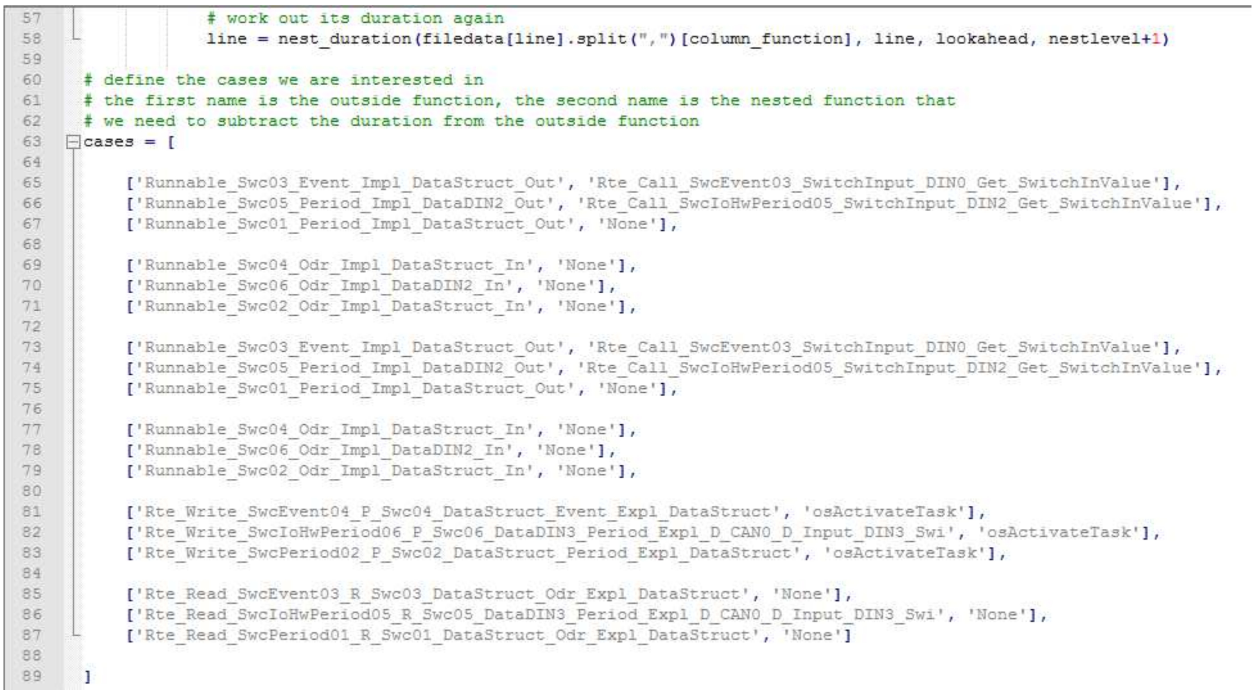


```c
/* A first test for the CAN library
K. Schmidt
April 2012

Just send a CAN message every 200ms and increment a counter
*/

#include <avr/io.h>
#include "can.h"
#include <util/delay.h>
#include <avr/interrupt.h>

int main(void){

    // Create a test messsage
    can_t msg;
    //...and a filter object
    can_filter_t filter;
    // fill the message with some initial values
    msg.id = 0x123456;
    msg.flags.rtr = 0;
    msg.flags.extended = 1;
    msg.length = 4;
    msg.data[0] = 0xcc;
    msg.data[1] = 0xcc;
    msg.data[2] = 0xcc;
    msg.data[3] = 0xcc;
    // set a filter which allows all messages to pass
    filter.id = 0;
    filter.mask = 0;
    filter.flags.rtr = 0x00;
    filter.flags.extended = 0x00;

    // initialize the CAN bus
    can_init(BITRATE_125_KBPS);
    // and set a filter
    can_set_filter (0,&filter);
    // enable interrupts
    sei();
    while (1){
        can_send_message(&msg);
        /*msg.data[0] +=1;
        if(msg.data[0] == 0xff){
            msg.data[0] = 0x00;
            }*/
        _delay_ms(500.);
    }
    return 0;
}
```

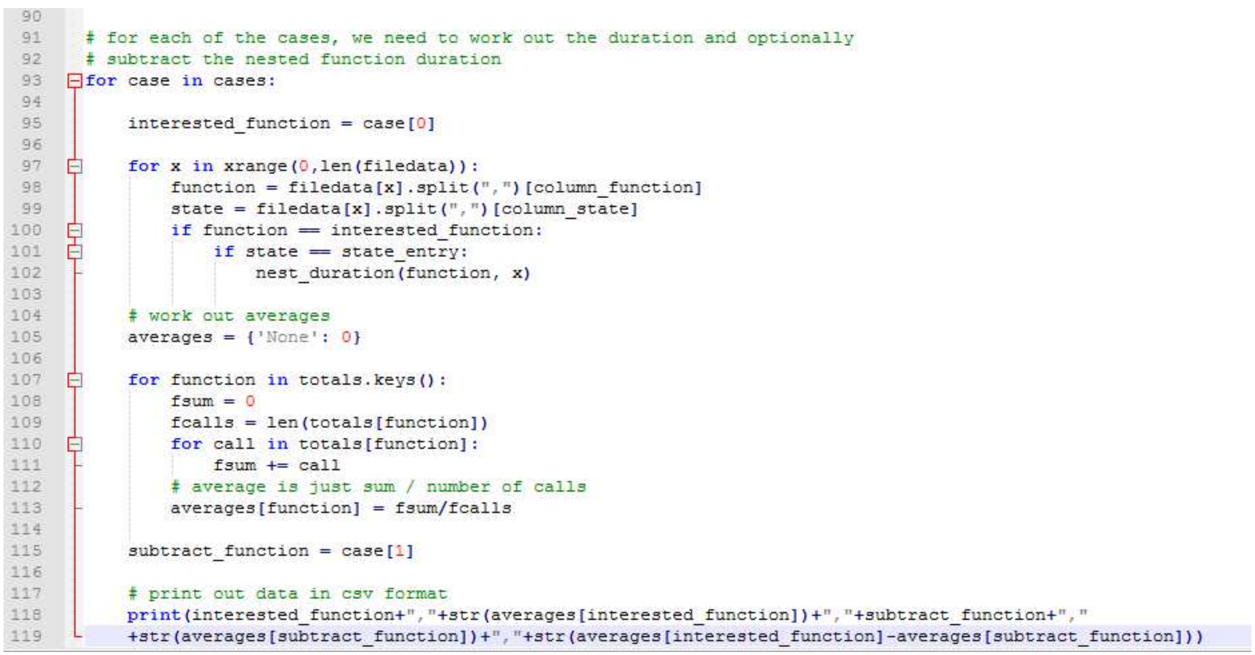
Source code 11-1: STK600 CAN messages

11.1. Python source code

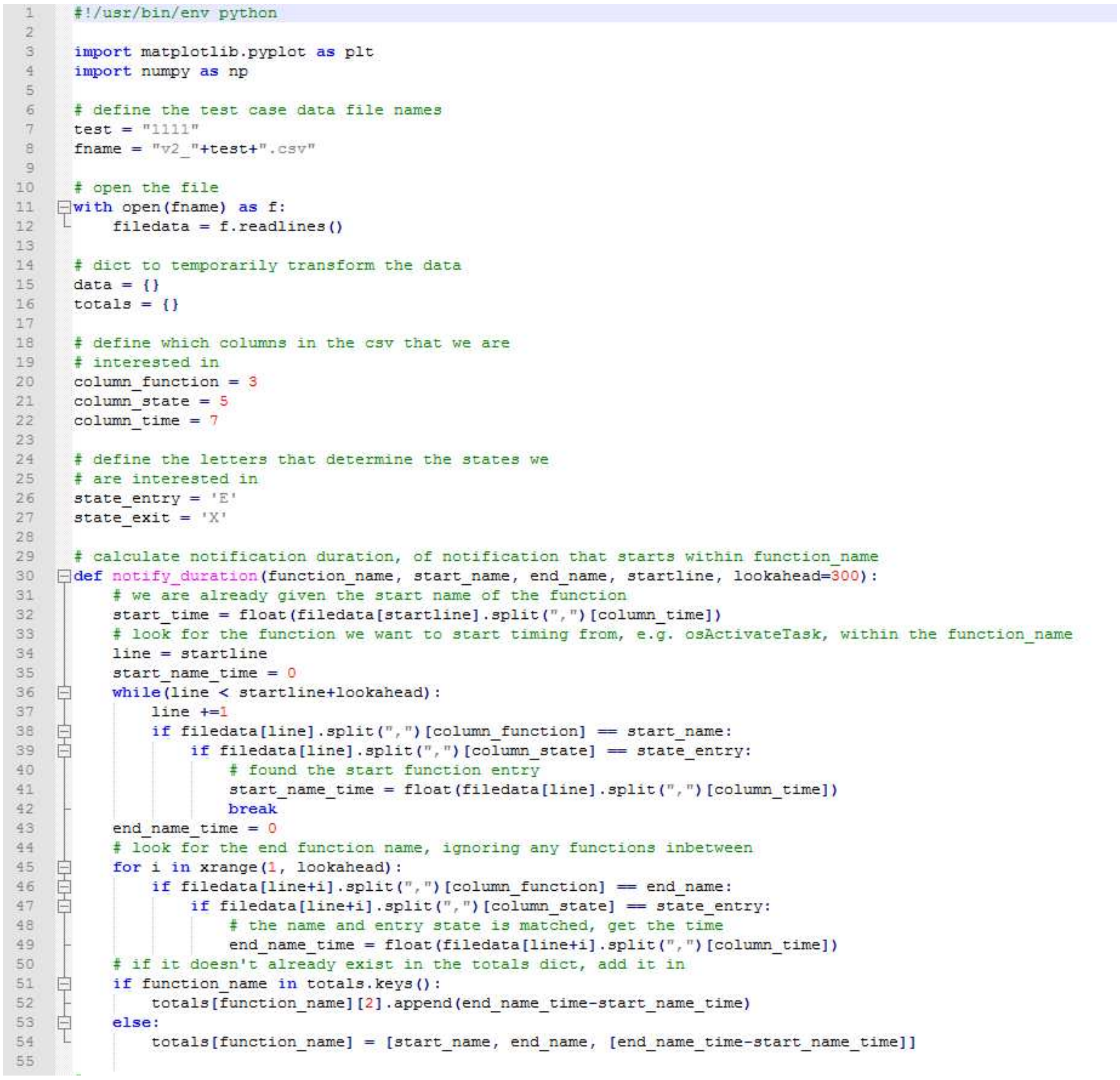
```python
#!/usr/bin/env python

import matplotlib.pyplot as plt
import numpy as np

# define the test case data file names
test = "1111"
fname = "v2_"+test+".csv"

# open the file
with open(fname) as f:
    filedata = f.readlines()

# dict to temporarily transform the data
data = {}
totals = {}

# define which columns in the csv that we are
# interested in
column_function = 3
column_state = 5
column_time = 7

# define the letters that determine the states we
# are interested in
state_entry = 'E'
state_exit = 'X'

# this function is called recursively to populate the data dict
# whenever we find a function entry in the data csv
def nest_duration(function_name, startline, lookahead=200, nestlevel=0):
    start_time = float(filedata[startline].split(",")[column_time])
    # look for the exit
    line = startline
    # lookahead up to 200 lines by default for the exit for this function
    while(line < startline+lookahead):
        line +=1
        if filedata[line].split(",")[column_function] == function_name:
            # found same function
            state = filedata[line].split(",")[column_state]
            if state == state_exit:
                # found the function's exit, calculate the duration
                stop_time = float(filedata[line].split(",")[column_time])
                duration = stop_time - start_time
                # if doesn't exist already, add to totals
                if function_name in totals.keys():
                    totals[function_name].append(duration);
                else:
                    totals[function_name] = [duration];
                # return the line it was on
                return line
            else:
                # same but no exit
                pass
        else:
            # found function inside this function
```



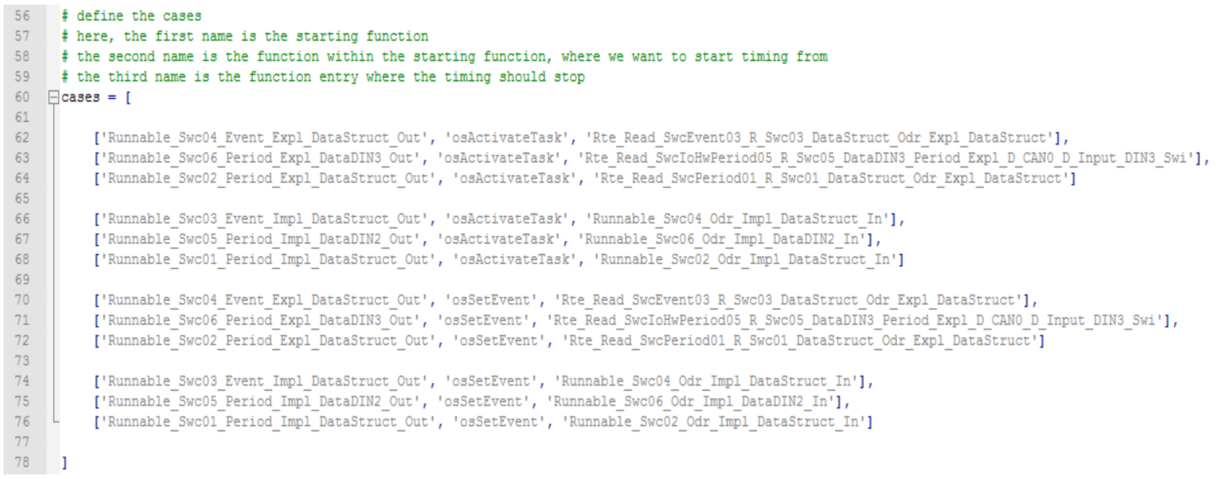
```
            # work out its duration again
            line = nest_duration(filedata[line].split(",")[column_function], line, lookahead, nestlevel+1)

# define the cases we are interested in
# the first name is the outside function, the second name is the nested function that
# we need to subtract the duration from the outside function
cases = [

    ['Runnable_Swc03_Event_Impl_DataStruct_Out', 'Rte_Call_SwcEvent03_SwitchInput_DIN0_Get_SwitchInValue'],
    ['Runnable_Swc05_Period_Impl_DataDIN2_Out', 'Rte_Call_SwcIoHwPeriod05_SwitchInput_DIN2_Get_SwitchInValue'],
    ['Runnable_Swc01_Period_Impl_DataStruct_Out', 'None'],

    ['Runnable_Swc04_Odr_Impl_DataStruct_In', 'None'],
    ['Runnable_Swc06_Odr_Impl_DataDIN2_In', 'None'],
    ['Runnable_Swc02_Odr_Impl_DataStruct_In', 'None'],

    ['Runnable_Swc03_Event_Impl_DataStruct_Out', 'Rte_Call_SwcEvent03_SwitchInput_DIN0_Get_SwitchInValue'],
    ['Runnable_Swc05_Period_Impl_DataDIN2_Out', 'Rte_Call_SwcIoHwPeriod05_SwitchInput_DIN2_Get_SwitchInValue'],
    ['Runnable_Swc01_Period_Impl_DataStruct_Out', 'None'],

    ['Runnable_Swc04_Odr_Impl_DataStruct_In', 'None'],
    ['Runnable_Swc06_Odr_Impl_DataDIN2_In', 'None'],
    ['Runnable_Swc02_Odr_Impl_DataStruct_In', 'None'],

    ['Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct', 'osActivateTask'],
    ['Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi', 'osActivateTask'],
    ['Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct', 'osActivateTask'],

    ['Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct', 'None'],
    ['Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi', 'None'],
    ['Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct', 'None']

]
```



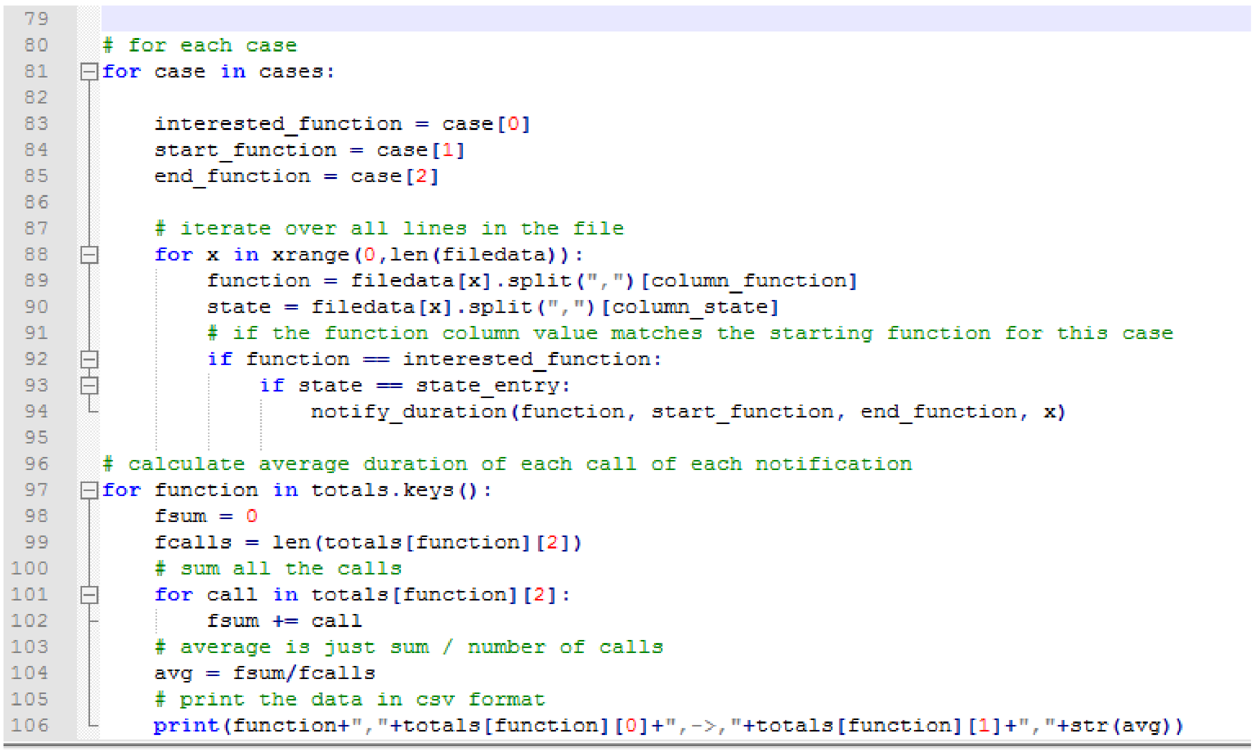
```

# for each of the cases, we need to work out the duration and optionally
# subtract the nested function duration
for case in cases:

    interested_function = case[0]

    for x in xrange(0,len(filedata)):
        function = filedata[x].split(",")[column_function]
        state = filedata[x].split(",")[column_state]
        if function == interested_function:
            if state == state_entry:
                nest_duration(function, x)

    # work out averages
    averages = {'None': 0}

    for function in totals.keys():
        fsum = 0
        fcalls = len(totals[function])
        for call in totals[function]:
            fsum += call
        # average is just sum / number of calls
        averages[function] = fsum/fcalls

    subtract_function = case[1]

    # print out data in csv format
    print(interested_function+","+str(averages[interested_function])+","+subtract_function+","
    +str(averages[subtract_function])+","+str(averages[interested_function]-averages[subtract_function]))
```



```python
#!/usr/bin/env python

import matplotlib.pyplot as plt
import numpy as np

# define the test case data file names
test = "1111"
fname = "v2_"+test+".csv"

# open the file
with open(fname) as f:
    filedata = f.readlines()

# dict to temporarily transform the data
data = {}
totals = {}

# define which columns in the csv that we are
# interested in
column_function = 3
column_state = 5
column_time = 7

# define the letters that determine the states we
# are interested in
state_entry = 'E'
state_exit = 'X'

# calculate notification duration, of notification that starts within function_name
def notify_duration(function_name, start_name, end_name, startline, lookahead=300):
    # we are already given the start name of the function
    start_time = float(filedata[startline].split(",")[column_time])
    # look for the function we want to start timing from, e.g. osActivateTask, within the function_name
    line = startline
    start_name_time = 0
    while(line < startline+lookahead):
        line +=1
        if filedata[line].split(",")[column_function] == start_name:
            if filedata[line].split(",")[column_state] == state_entry:
                # found the start function entry
                start_name_time = float(filedata[line].split(",")[column_time])
                break
    end_name_time = 0
    # look for the end function name, ignoring any functions inbetween
    for i in xrange(1, lookahead):
        if filedata[line+i].split(",")[column_function] == end_name:
            if filedata[line+i].split(",")[column_state] == state_entry:
                # the name and entry state is matched, get the time
                end_name_time = float(filedata[line+i].split(",")[column_time])
    # if it doesn't already exist in the totals dict, add it in
    if function_name in totals.keys():
        totals[function_name][2].append(end_name_time-start_name_time)
    else:
        totals[function_name] = [start_name, end_name, [end_name_time-start_name_time]]

```



```python
# define the cases
# here, the first name is the starting function
# the second name is the function within the starting function, where we want to start timing from
# the third name is the function entry where the timing should stop
cases = [

    ['Runnable_Swc04_Event_Expl_DataStruct_Out', 'osActivateTask', 'Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct'],
    ['Runnable_Swc06_Period_Expl_DataDIN3_Out', 'osActivateTask', 'Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi'],
    ['Runnable_Swc02_Period_Expl_DataStruct_Out', 'osActivateTask', 'Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct']

    ['Runnable_Swc03_Event_Impl_DataStruct_Out', 'osActivateTask', 'Runnable_Swc04_Odr_Impl_DataStruct_In'],
    ['Runnable_Swc05_Period_Impl_DataDIN2_Out', 'osActivateTask', 'Runnable_Swc06_Odr_Impl_DataDIN2_In'],
    ['Runnable_Swc01_Period_Impl_DataStruct_Out', 'osActivateTask', 'Runnable_Swc02_Odr_Impl_DataStruct_In']

    ['Runnable_Swc04_Event_Expl_DataStruct_Out', 'osSetEvent', 'Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct'],
    ['Runnable_Swc06_Period_Expl_DataDIN3_Out', 'osSetEvent', 'Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi'],
    ['Runnable_Swc02_Period_Expl_DataStruct_Out', 'osSetEvent', 'Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct']

    ['Runnable_Swc03_Event_Impl_DataStruct_Out', 'osSetEvent', 'Runnable_Swc04_Odr_Impl_DataStruct_In'],
    ['Runnable_Swc05_Period_Impl_DataDIN2_Out', 'osSetEvent', 'Runnable_Swc06_Odr_Impl_DataDIN2_In'],
    ['Runnable_Swc01_Period_Impl_DataStruct_Out', 'osSetEvent', 'Runnable_Swc02_Odr_Impl_DataStruct_In']

]
```



```python

# for each case
for case in cases:

    interested_function = case[0]
    start_function = case[1]
    end_function = case[2]

    # iterate over all lines in the file
    for x in xrange(0,len(filedata)):
        function = filedata[x].split(",")[column_function]
        state = filedata[x].split(",")[column_state]
        # if the function column value matches the starting function for this case
        if function == interested_function:
            if state == state_entry:
                notify_duration(function, start_function, end_function, x)

# calculate average duration of each call of each notification
for function in totals.keys():
    fsum = 0
    fcalls = len(totals[function][2])
    # sum all the calls
    for call in totals[function][2]:
        fsum += call
    # average is just sum / number of calls
    avg = fsum/fcalls
    # print the data in csv format
    print(function+","+totals[function][0]+",->,"+totals[function][1]+","+str(avg))
```

11.2. **User manual for DaVinciDeveloper**

11.2.1. Project Assistant

- Start Davinci Developer, Make sure that no workspace is open.
- File => Project Assistant => New ECU Project
- If appropriate use old setting by pressing Use button

1. **General Settings**

- Choose *New ECU configuration*
- Enter *Project Name, Author, Version and Description*

2. **Project Folder**

- Enter project root path

3. **Software Integration Package (SIP)**

- Choose *Manual Path:* *C:\Projects\DemoApplication\MainController\MICROSAR\Generators\Components*
- Keep default setting, e.g
  - *Plattform:* Mpc55xx_Flexcan2,
  - *Compiler:* Diab,
  - *Derivative:* Mpc5500_Spc56ec64

4. **Output Paths**

- Keep default configurations and *single file* ticked

5. **Tools**

- Keep default paths,
  - *DaVinci Configurator Pro:* C:\Program Files (x86)\Vector DaVinci Configurator 4.2 SP2\DaVinciCFG.exe
  - *DaVinci Developer: C:\Program Files (x86)\Vector DaVinci Developer 3.9\Bin\DaVinciDEV.exe*
  - *GENy:* C:\Program Files (x86)\Vector GENy 1.4\GENy.exe

6. **DaVinci Developer Workspace**
    - Create new DaVInci Developer Workspace
    - Workspace Format: DCF
7. **Input Files**

- Tick *Single ECU*
- Include all files from C:\Projects\DemoApplication\MainController\DemoApplication\Config\System include all files:
- CAN0.dbc
- CAN1.dbc
- CAN2.dbc
- CAN3.dbc
- CAN4.dbc
- CAN5.dbc
- LIN0.ldf
- LIN1.ldf

FlexRay_300_NoCan.xml

VC121_SystemDescr_VC121.arxml

- Activate *VC121* tick box in ECU Instances

11.2.2. Configure SWCs and RTE

11.2.2.1. Create SWCs

- Right click to Application Component Types

Name	Name of the SWC
Composition/ Atomic	Choose composition if it is a composition of SWCs, see herefore AUTOSAR specification, Choose Atomic for atomic SWC
Type	Application
Supports multiple instantiation	Enables an instantiation of a component for more than once with the same
Symbol	The RTE generator uses symbols instead of names.

- OK

- Double-click on Software Design in the ECU Projects view to open the Software Design view for the ECU

- To use the defined application component type drag and drop the component from the libary to the software design view.

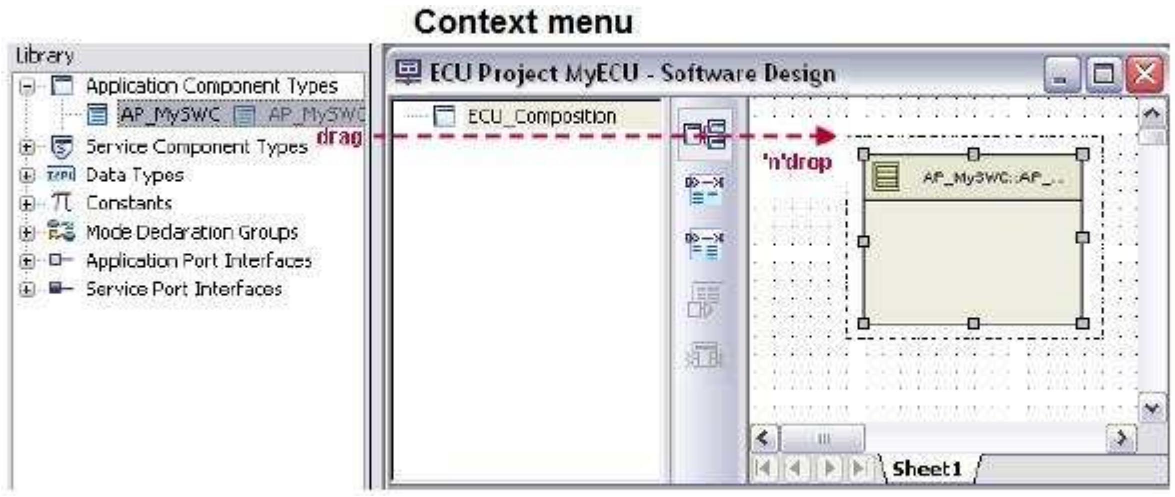


Figure 11-1: Drag and Drop of software components

- A application component type becomes automatically an application component prototype when it is used. In the library, the softwarecomponents are types and the ports are interfaces. As soon as they are used, they will become:
    - Port Interface => Port Prototype
    - Component Type => Component Prototype

- To change their name open the property window (with right-click) form the context menu

11.2.2.2. Create Ports

11.2.2.2.1. Define Data Types

- Define data types by right-click on Data Type in the library and select out of the list the predefined data types or create a new one.

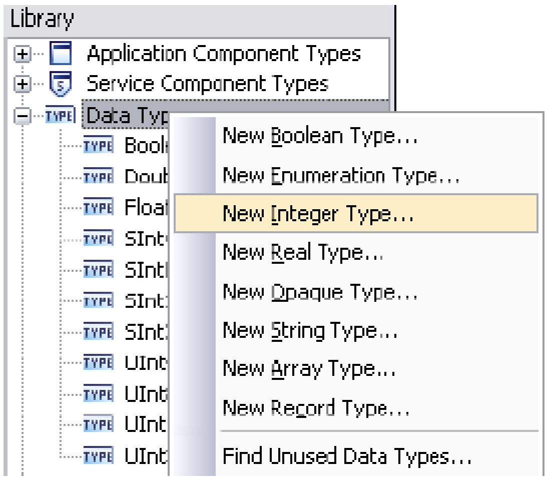


Figure 11-2: Right-click on data type

Boolean	Boolean has the value space required to support the mathematical concept of binary-valued logic: (true, false).
Enumeration	Enumerated types is a user defined data type by using identifier to create variables.
Real type	Real type represent an approximation of a real number
Opaque type	Opaque type is a type which is exposed in APIs via a pointer but never concretely defined. Its value can only be manipulated by calling subroutines that have access to the missing information.
String type	String type is a one-dimensional array of characters which is terminated be a null character '\0'.
Array type	Array type stores a fixed-size sequential collection of elements of the same type.
Record Type	Record type also called struct is a collection of items of different types.

- Right-click on *Data Types* in libary window => select new *Record Type* add all necessary Record elements

11.2.2.2.2. Define Application Port Interfaces (Ports)

- Right-Click on *Application Port Interface* in the library window and select *New S/R Port Interface*

- Enter a name and choose tab Data Element Prototype
- Enter the name of the Data element and select the type

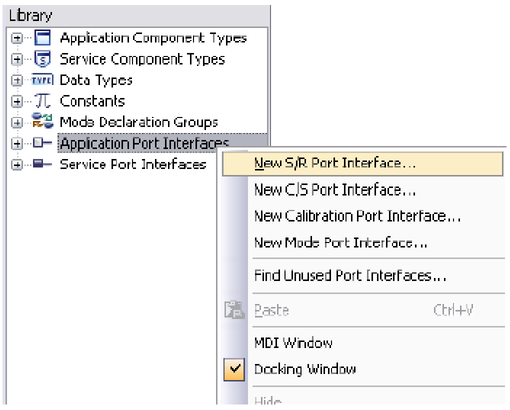

Library
Application Component Types
Service Component Types
Data Types
Constants
Mode Declaration Groups
Application Port Interfaces
Service Port Interfaces
New S/R Port Interface...
New C/S Port Interface...
New Calibration Port Interface...
New Mode Port Interface...
Find Unused Port Interfaces...
Paste Ctrl+V
MDI Window
Docking Window
Hide

Figure 11-3: Insert S/R Port Interface

- Select an application port interface from the library and place it onto the application component via drag'n'drop.
- Double- click on the added port and select whether it is a sender or receiver port

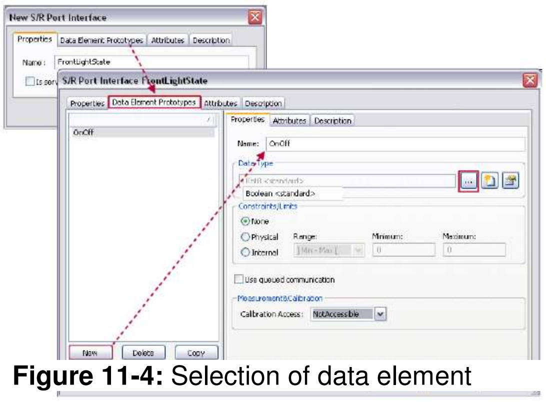


Figure 11-4: Selection of data element

Init value

- Each port should have an init value (constant), use therefore the library to define constants
- Right-click on Constants in the library, select the data type and enter a name

Refer Init value to the SWC

- Double-click on the application port in the software design view and select the tab *Communication Spec*
- Click […] and choose constant

Reception handle

- To react on incoming information the Rx Filter can be enabled
  Click on the Receiver Port and select the option Rx Filter:

  - Always (Every reception will be handled)
  - MasekdNewDiffersMaskedOld (Only those receptions will be handled in which the new value differs from the previous one)

Connector

- Use the Botton **Draw Connector** to connect ports manually.
- Ports have to be compatible (matching data elements)

11.2.2.3. Define runnables

Order.	Name	Description	Checkbox
1.	Symbol, Name	What the runnable is called in the template file	
2.	Trigger	When is the runnable executed	
3.	Port access	What data can the runnable access	
4.	Mapping	In which task context does the runnable work	

Table 11-1: Check- list for defining a runnable

- Double-click on SWC in the library
- Click on the icon [fx]
- Click [New] and select [Runnable]
- Enter Name and Symbol for the new runnable on the Properties tab
  - **Can be invoked concurrently** runnable can be executed while it is already running, it can be safely executed concurrently. There are a few criteria for the implementation code:
    - No static (or global) non-constant data
    - No return of address to static non constant data
    - Only work on data provided by caller
    - No modification of own code at runtime
    - No call of non-re-entrant runnables

Not suggested because it will be not mapped to a task.

- **Triggers for runnables**
  - Select the trigger tab and click on *New* and choose **periodical** for a periodical timing event.
  - Select **on data reception** the runnable will be activated as soon as data is received at the appropriate port.
  - Select **on operation invocation**, see 11.2.2.4 Communication with external signals , p. 128
- **Connect Port with runnable**

  Select the tab **Port Access**, you can choose between the following options:

Direct write [dataSendPoint]	Explicit writing process
Buffered write [dataWriteAccess]	Implicit writing process
Direct read [dataReceivePointByArgument	Explicit reading process
Buffered read [implicit read, dataReadAccess]	Implicit reading process

11.2.2.4. Communication with external signals

Import Service Coponent

- Import the XML file of the service component, therefore:
    - File->Import XML File… -> Click [Add File…] -> Select File
    - For the purpose of *VCx_Components* folder from the DemoApplication C:\Projects\MultiCore_V1_0\VCx_Components\IoHwAb\Description
    - Unlock the service component: right Click on the service component-> Object Locking -> Unlock Object

- Create an application interface with the right data type *DT_CAN0_D_Input_DIN0_Swi*
- Drag and drop this interface onto your software component to create a port
- Create a deligation port, therefore:
    - Right-Click on your software component by activating the ECU_Composition interface view
    - Select *Complete Ports …* make sure *Delegation Ports only* is activated
    - Rename the added delegation port to be able to differ from other ports and select *Select signal*

- Add a *Service port* to the software component
    - Select the software component in the *Software Design* view and select the *Port Prototype List*
    - Click **New** -> Choose a *Service port interface* (VC_IoHwAb_SwitchInputValue)
    - Select the tab *Port API Option* and *Enable API usage by address*

- Connect the created service port of the software component with a server port
    - Select *Service Mapping* out of the ECU Project view
    - Select the *Application Component View*
    - Right-Click on the *Application Component Port* and select *Map Service Port …*

- Now you are able to add the *Invoke Operation* as an *Port Access* into your runnable
    - Make sure it is asynncron and triggered periodically

- Connect the delegation port with a network signal via *Data mapping*
    - Select *Data mapping* out of the ECU Project window
    - Select the *Data element view mode*
    - Right-Click on the Data Element, which will be connected to a network signal

11.2.2.5. Tasks and Task Mapping

A runnable has to be mapped to a task if it is not re-entrant but could be called re-entrant during system operation.

- Open *task mapping view*
- Right-click in the left view below Task and select **Add Task** , type the name of the task and its priority
- The **Activation** defines how many task activation are stored in parallel – 1 is recommended. (a task type could be Auto, Basic or Extended)
- Select if the task is **Full-preemptive** or **Non-preemptive** and confirm with **OK**
- Assign your runnables to the task via drag'n'drop

11.2.2.6. Code Generation with DaVinci Developer

- Check first your configuration for error and missing information,
  Right-click on the root of your ECU Project view and select *Check*
- Repeat the last configuration and select *Open generator list*
- Activate the *MICROSAR RTE generator* and the *Component Template Generator*
- Repeat first selection (Right-click on root ECU project) and select *Generate Code* The Developer works off the generator list and generates:
    - The configuration files for the BSW
    - The operating system
    - The RTE
    - The component templates files without deleting your software parts within, if already exist

11.3. **Automotive microcontroller**

Application	**Microcontrollers**
Engine Control and Management Interfaces	MPC5674F, MPC5673F, MPC563xM, MPC5644A, MPC5643A, MPC5642A, MPC5746M, MPC5777M
Hybrid and Electric Auxillaries	MPC5674F, MPC5673F, MPC563xM, S12G, MPC5644A, MPC5643A, MPC5642A, MPC5744P, MPC5746G, MPC5747C, MPC5747G, MPC5748C, MPC5748G, MPC5746M
Watchdog	S08QD4, S08SG, S08AW, S08SC4, S08RN
High Temperature	MPC5744P,S08SG
Body Control Module and Gateway	MPC5668x, MPC560xB, MPC560xD, MPC564xB/C, MPC5746G, MPC5747C, MPC5747G, MPC5748C, MPC5748G
HVAC, Lighting, Seats, Window Lift, Doors	MPC560xB, MPC560xD, S08D, S08AW, S08EL, S08SG, S08SL, S08MP16, S08SC4, S08RN
Body Motor Control	S08MP16, S08RN
Infotainment	all i.MX, SVFxxxR, MAC57D5xx
Telematics	i.MX251, i.MX281, i.MX53, i.MX351, i.MX 6S1, i.MX 6U1, MAC57D5xx
Instrument Cluster	MPC560xS, i.MX534, i.MX 6S1, i.MX 6U1, SVFxxxR, S12H, S12XH, S12XHY, S12ZVFP, S12ZVH, S12ZVHY, MAC57D5xx
Braking Systems	MPC564xL, MPC560xP, MPC5744P, S12XE, S12XS
Electronic Power Steering	MPC564xL, MPC560xP, MPC5744P S12G
Semi-Active Suspension	MPC564xL, MPC5744P
Airbag	MPC560xP, MPC5744P, S12XF, S12XE, S12XS, S08SG
Electronic Stability Control	MPC564xL, MPC560xP, MPC5744P
Lane Departure	i.MX534, MPC567xK, i.MX 6S4, i.MX 6U4, i.MX 6D4, i.MX 6Q4, MPC577xK
Advanced Cruise Control	MPC564xL, MPC567xK, MPC5744P, SCP2201, SCP2207, MPC577xK
Precrash, Blindspot Detection, Backup Warning	MPC564xL, MPC567xK, MPC5604E, MPC5744P, MPC577xK, S08RN
Multi-function Display	MAC57D5xx

Table 11-2: This overview is a section out of (Freescale Semiconductor, 2014)

11.4. **VC121-12 ECU component diagram**

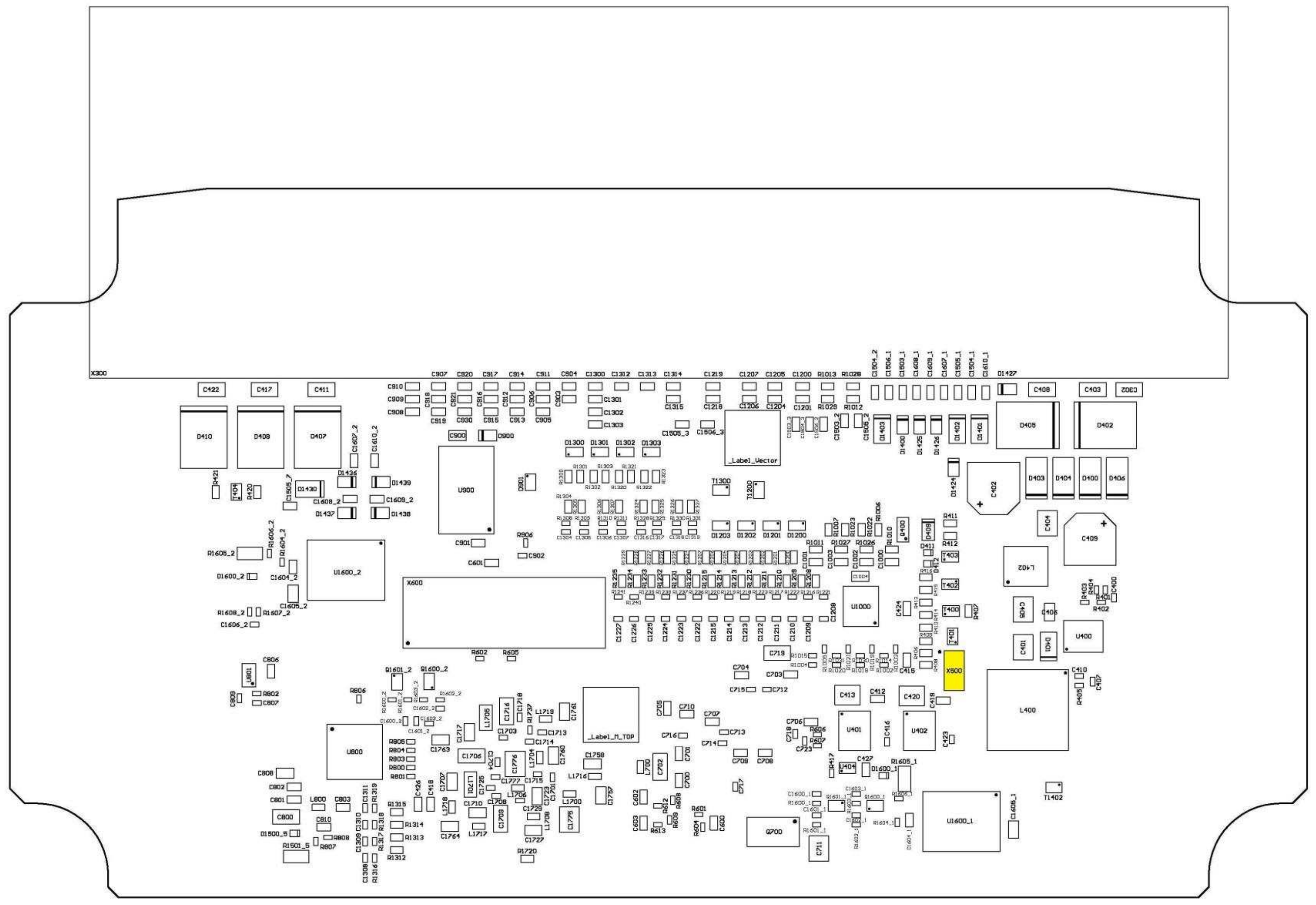
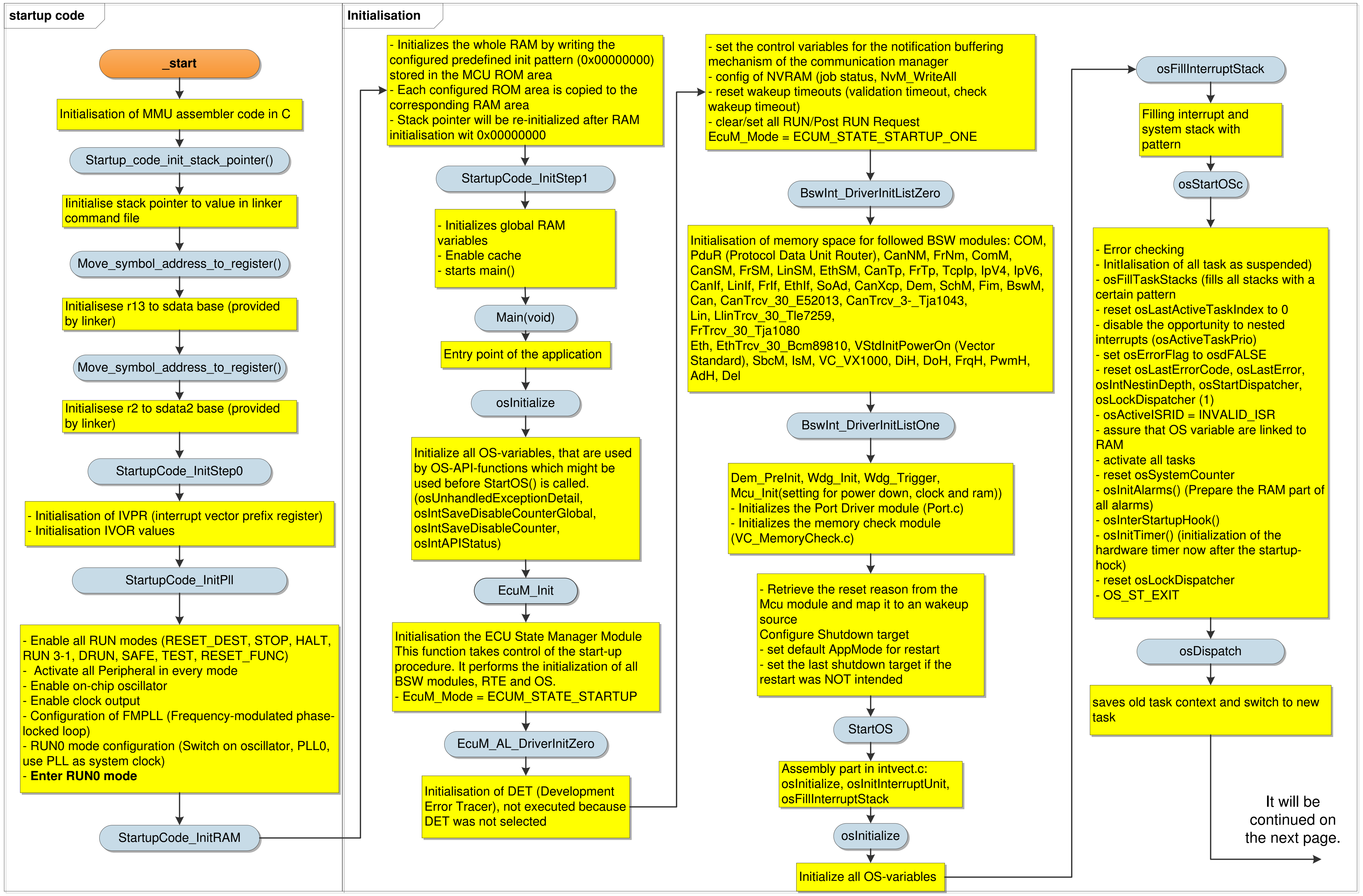


Figure 11-5: VC121-12 ECU component diagram

11.5. System program flow chart

11.5.1. Start up and initialisation of VC121-12



startup code
Initialisation
_start
Initialisation of MMU assembler code in C
Startup_code_init_stack_pointer()
Iinitialise stack pointer to value in linker command file
Move_symbol_address_to_register()
Initialisese r13 to sdata base (provided by linker)
Move_symbol_address_to_register()
Initialisese r2 to sdata2 base (provided by linker)
StartupCode_InitStep0
- Initialisation of IVPR (interrupt vector prefix register) - Initialisation IVOR values
StartupCode_InitPll
- Enable all RUN modes (RESET_DEST, STOP, HALT, RUN 3-1, DRUN, SAFE, TEST, RESET_FUNC) - Activate all Peripheral in every mode - Enable on-chip oscillator - Enable clock output - Configuration of FMPLL (Frequency-modulated phase-locked loop) - RUN0 mode configuration (Switch on oscillator, PLL0, use PLL as system clock) - **Enter RUN0 mode**
StartupCode_InitRAM
- Initializes the whole RAM by writing the configured predefined init pattern (0x00000000) stored in the MCU ROM area - Each configured ROM area is copied to the corresponding RAM area - Stack pointer will be re-initialized after RAM initialisation wit 0x00000000
StartupCode_InitStep1
- Initializes global RAM variables - Enable cache - starts main()
Main(void)
Entry point of the application
osInitialize
Initialize all OS-variables, that are used by OS-API-functions which might be used before StartOS() is called. (osUnhandledExceptionDetail, osIntSaveDisableCounterGlobal, osIntSaveDisableCounter, osIntAPIStatus)
EcuM_Init
Initialisation the ECU State Manager Module This function takes control of the start-up procedure. It performs the initialization of all BSW modules, RTE and OS. - EcuM_Mode = ECUM_STATE_STARTUP
EcuM_AL_DriverInitZero
Initialisation of DET (Development Error Tracer), not executed because DET was not selected
- set the control variables for the notification buffering mechanism of the communication manager - config of NVRAM (job status, NvM_WriteAll - reset wakeup timeouts (validation timeout, check wakeup timeout) - clear/set all RUN/Post RUN Request EcuM_Mode = ECUM_STATE_STARTUP_ONE
BswInt_DriverInitListZero
Initialisation of memory space for followed BSW modules: COM, PduR (Protocol Data Unit Router), CanNM, FrNm, ComM, CanSM, FrSM, LinSM, EthSM, CanTp, FrTp, TcpIp, IpV4, IpV6, CanIf, LinIf, FrIf, EthIf, SoAd, CanXcp, Dem, SchM, Fim, BswM, Can, CanTrcv_30_E52013, CanTrcv_3-_Tja1043, Lin, LlinTrcv_30_Tle7259, FrTrcv_30_Tja1080 Eth, EthTrcv_30_Bcm89810, VStdInitPowerOn (Vector Standard), SbcM, IsM, VC_VX1000, DiH, DoH, FrqH, PwmH, AdH, Del
BswInt_DriverInitListOne
Dem_PreInit, Wdg_Init, Wdg_Trigger, Mcu_Init(setting for power down, clock and ram)) - Initializes the Port Driver module (Port.c) - Initializes the memory check module (VC_MemoryCheck.c)
- Retrieve the reset reason from the Mcu module and map it to an wakeup source Configure Shutdown target - set default AppMode for restart - set the last shutdown target if the restart was NOT intended
StartOS
Assembly part in intvect.c: osInitialize, osInitInterruptUnit, osFillInterruptStack
osInitialize
Initialize all OS-variables
osFillInterruptStack
Filling interrupt and system stack with pattern
osStartOSc
- Error checking - Initlalisation of all task as suspended) - osFillTaskStacks (fills all stacks with a certain pattern - reset osLastActiveTaskIndex to 0 - disable the opportunity to nested interrupts (osActiveTaskPrio) - set osErrorFlag to osdFALSE - reset osLastErrorCode, osLastError, osIntNestinDepth, osStartDispatcher, osLockDispatcher (1) - osActiveISRID = INVALID_ISR - assure that OS variable are linked to RAM - activate all tasks - reset osSystemCounter - osInitAlarms() (Prepare the RAM part of all alarms) - osInterStartupHook() - osInitTimer() (initialization of the hardware timer now after the startup-hock) - reset osLockDispatcher - OS_ST_EXIT
osDispatch
saves old task context and switch to new task
It will be continued on the next page.

Figure 11-6: Program flow chart for the start up and initialisation of VC121-12

11.5.2. SchM_AsyncTask part 1

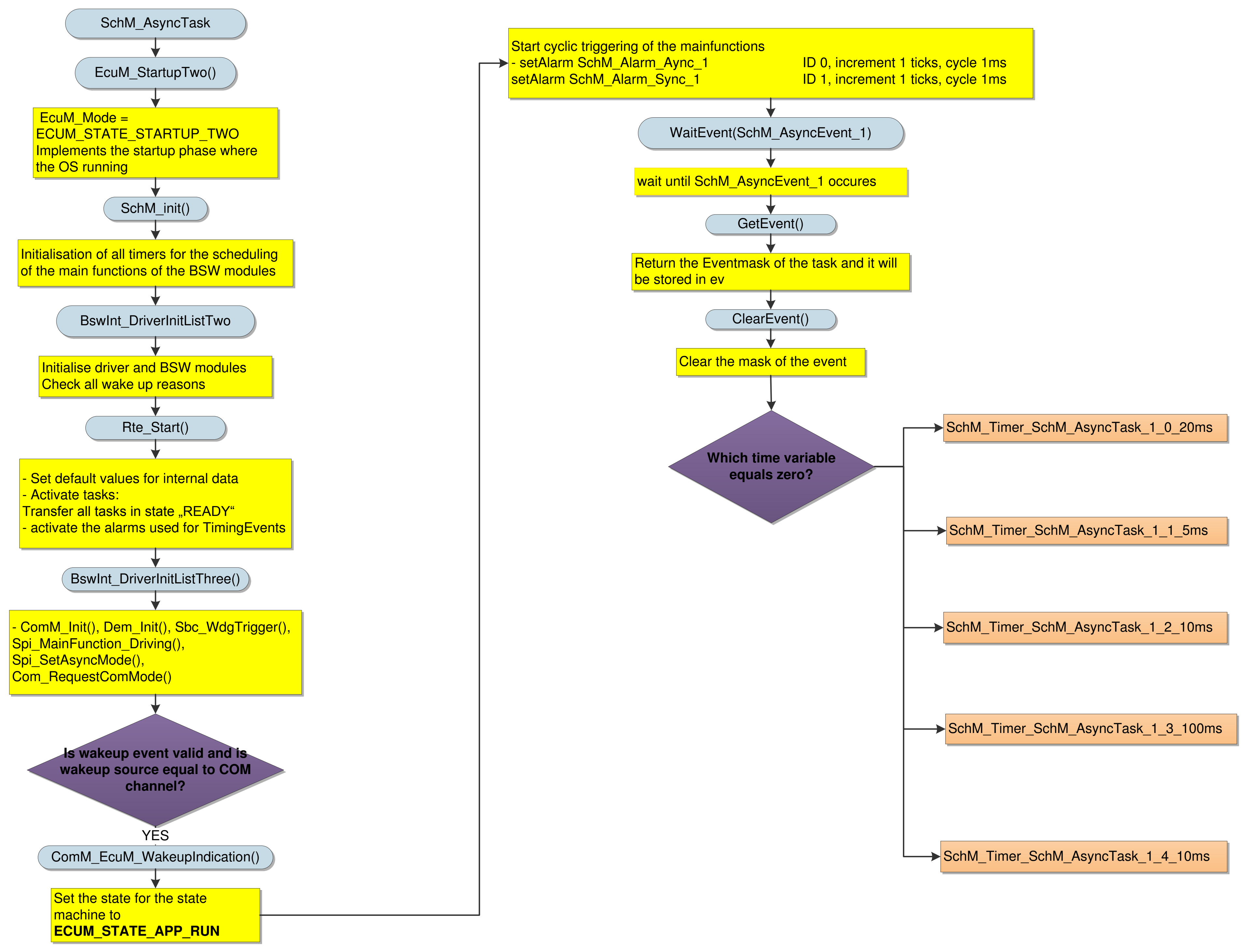

SchM_AsyncTask
EcuM_StartupTwo()
EcuM_Mode = ECUM_STATE_STARTUP_TWO Implements the startup phase where the OS running
SchM_init()
Initialisation of all timers for the scheduling of the main functions of the BSW modules
BswInt_DriverInitListTwo
Initialise driver and BSW modules Check all wake up reasons
Rte_Start()
- Set default values for internal data
- Activate tasks: Transfer all tasks in state „READY“
- activate the alarms used for TimingEvents
BswInt_DriverInitListThree()
- ComM_Init(), Dem_Init(), Sbc_WdgTrigger(), Spi_MainFunction_Driving(), Spi_SetAsyncMode(), Com_RequestComMode()
Is wakeup event valid and is wakeup source equal to COM channel?
YES
ComM_EcuM_WakeupIndication()
Set the state for the state machine to ECUM_STATE_APP_RUN
Start cyclic triggering of the mainfunctions
- setAlarm SchM_Alarm_Aync_1 ID 0, increment 1 ticks, cycle 1ms
setAlarm SchM_Alarm_Sync_1 ID 1, increment 1 ticks, cycle 1ms
WaitEvent(SchM_AsyncEvent_1)
wait until SchM_AsyncEvent_1 occures
GetEvent()
Return the Eventmask of the task and it will be stored in ev
ClearEvent()
Clear the mask of the event
Which time variable equals zero?
SchM_Timer_SchM_AsyncTask_1_0_20ms
SchM_Timer_SchM_AsyncTask_1_1_5ms
SchM_Timer_SchM_AsyncTask_1_2_10ms
SchM_Timer_SchM_AsyncTask_1_3_100ms
SchM_Timer_SchM_AsyncTask_1_4_10ms

Figure 11-7: Program flow chart for the SchM_AsyncTask part 1

11.5.3. SchM_AsyncTask part 2

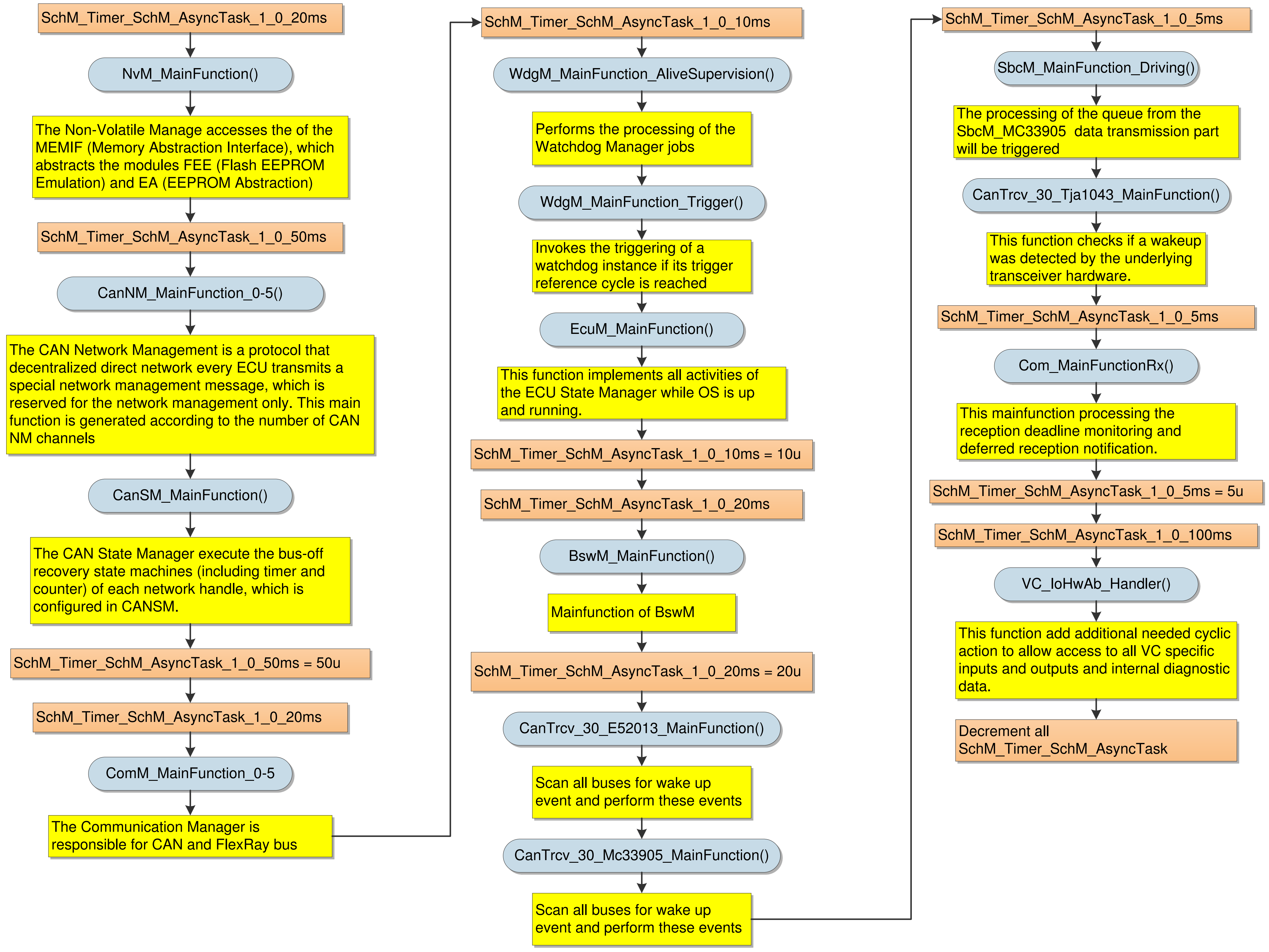

SchM_Timer_SchM_AsyncTask_1_0_20ms
NvM_MainFunction()
The Non-Volatile Manage accesses the of the MEMIF (Memory Abstraction Interface), which abstracts the modules FEE (Flash EEPROM Emulation) and EA (EEPROM Abstraction)
SchM_Timer_SchM_AsyncTask_1_0_50ms
CanNM_MainFunction_0-5()
The CAN Network Management is a protocol that decentralized direct network every ECU transmits a special network management message, which is reserved for the network management only. This main function is generated according to the number of CAN NM channels
CanSM_MainFunction()
The CAN State Manager execute the bus-off recovery state machines (including timer and counter) of each network handle, which is configured in CANSM.
SchM_Timer_SchM_AsyncTask_1_0_50ms = 50u
SchM_Timer_SchM_AsyncTask_1_0_20ms
ComM_MainFunction_0-5
The Communication Manager is responsible for CAN and FlexRay bus
SchM_Timer_SchM_AsyncTask_1_0_10ms
WdgM_MainFunction_AliveSupervision()
Performs the processing of the Watchdog Manager jobs
WdgM_MainFunction_Trigger()
Invokes the triggering of a watchdog instance if its trigger reference cycle is reached
EcuM_MainFunction()
This function implements all activities of the ECU State Manager while OS is up and running.
SchM_Timer_SchM_AsyncTask_1_0_10ms = 10u
SchM_Timer_SchM_AsyncTask_1_0_20ms
BswM_MainFunction()
Mainfunction of BswM
SchM_Timer_SchM_AsyncTask_1_0_20ms = 20u
CanTrcv_30_E52013_MainFunction()
Scan all buses for wake up event and perform these events
CanTrcv_30_Mc33905_MainFunction()
Scan all buses for wake up event and perform these events
SchM_Timer_SchM_AsyncTask_1_0_5ms
SbcM_MainFunction_Driving()
The processing of the queue from the SbcM_MC33905 data transmission part will be triggered
CanTrcv_30_Tja1043_MainFunction()
This function checks if a wakeup was detected by the underlying transceiver hardware.
SchM_Timer_SchM_AsyncTask_1_0_5ms
Com_MainFunctionRx()
This mainfunction processing the reception deadline monitoring and deferred reception notification.
SchM_Timer_SchM_AsyncTask_1_0_5ms = 5u
SchM_Timer_SchM_AsyncTask_1_0_100ms
VC_IoHwAb_Handler()
This function add additional needed cyclic action to allow access to all VC specific inputs and outputs and internal diagnostic data.
Decrement all SchM_Timer_SchM_AsyncTask

Figure 11-8: Program flow chart for the SchM_AsyncTask part 2

11.5.4. SchM_AsyncTask part 3

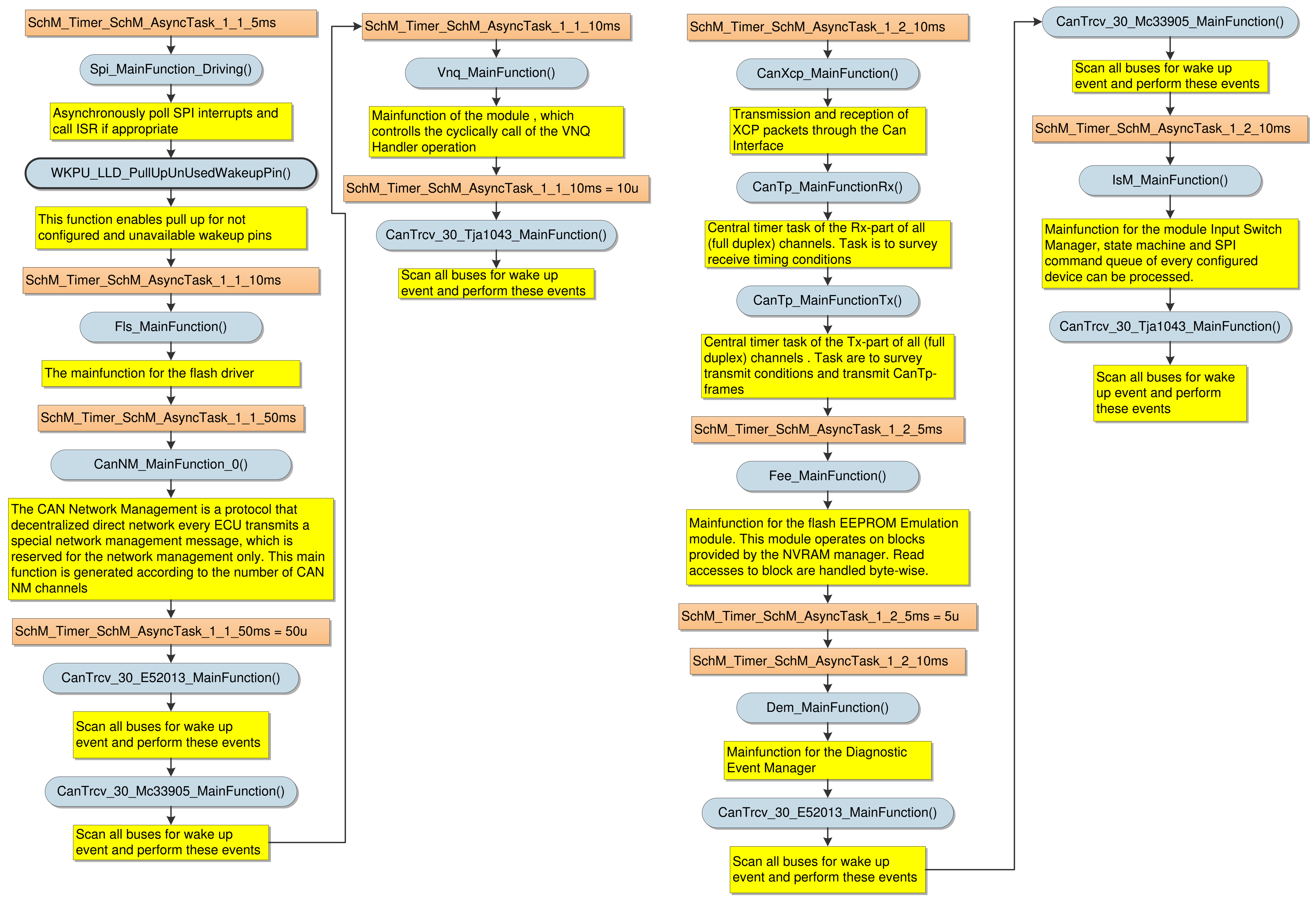

SchM_Timer_SchM_AsyncTask_1_1_5ms
Spi_MainFunction_Driving()
Asynchronously poll SPI interrupts and call ISR if appropriate
WKPU_LLD_PullUpUnUsedWakeupPin()
This function enables pull up for not configured and unavailable wakeup pins
SchM_Timer_SchM_AsyncTask_1_1_10ms
Fls_MainFunction()
The mainfunction for the flash driver
SchM_Timer_SchM_AsyncTask_1_1_50ms
CanNM_MainFunction_0()
The CAN Network Management is a protocol that decentralized direct network every ECU transmits a special network management message, which is reserved for the network management only. This main function is generated according to the number of CAN NM channels
SchM_Timer_SchM_AsyncTask_1_1_50ms = 50u
CanTrcv_30_E52013_MainFunction()
Scan all buses for wake up event and perform these events
CanTrcv_30_Mc33905_MainFunction()
Scan all buses for wake up event and perform these events
SchM_Timer_SchM_AsyncTask_1_1_10ms
Vnq_MainFunction()
Mainfunction of the module , which controlls the cyclically call of the VNQ Handler operation
SchM_Timer_SchM_AsyncTask_1_1_10ms = 10u
CanTrcv_30_Tja1043_MainFunction()
Scan all buses for wake up event and perform these events
SchM_Timer_SchM_AsyncTask_1_2_10ms
CanXcp_MainFunction()
Transmission and reception of XCP packets through the Can Interface
CanTp_MainFunctionRx()
Central timer task of the Rx-part of all (full duplex) channels. Task is to survey receive timing conditions
CanTp_MainFunctionTx()
Central timer task of the Tx-part of all (full duplex) channels . Task are to survey transmit conditions and transmit CanTp-frames
SchM_Timer_SchM_AsyncTask_1_2_5ms
Fee_MainFunction()
Mainfunction for the flash EEPROM Emulation module. This module operates on blocks provided by the NVRAM manager. Read accesses to block are handled byte-wise.
SchM_Timer_SchM_AsyncTask_1_2_5ms = 5u
SchM_Timer_SchM_AsyncTask_1_2_10ms
Dem_MainFunction()
Mainfunction for the Diagnostic Event Manager
CanTrcv_30_E52013_MainFunction()
Scan all buses for wake up event and perform these events
CanTrcv_30_Mc33905_MainFunction()
Scan all buses for wake up event and perform these events
SchM_Timer_SchM_AsyncTask_1_2_10ms
IsM_MainFunction()
Mainfunction for the module Input Switch Manager, state machine and SPI command queue of every configured device can be processed.
CanTrcv_30_Tja1043_MainFunction()
Scan all buses for wake up event and perform these events

Figure 11-9: Program flow chart for the SchM_AsyncTask part 3

11.5.5. SchM_AsyncTask part 4

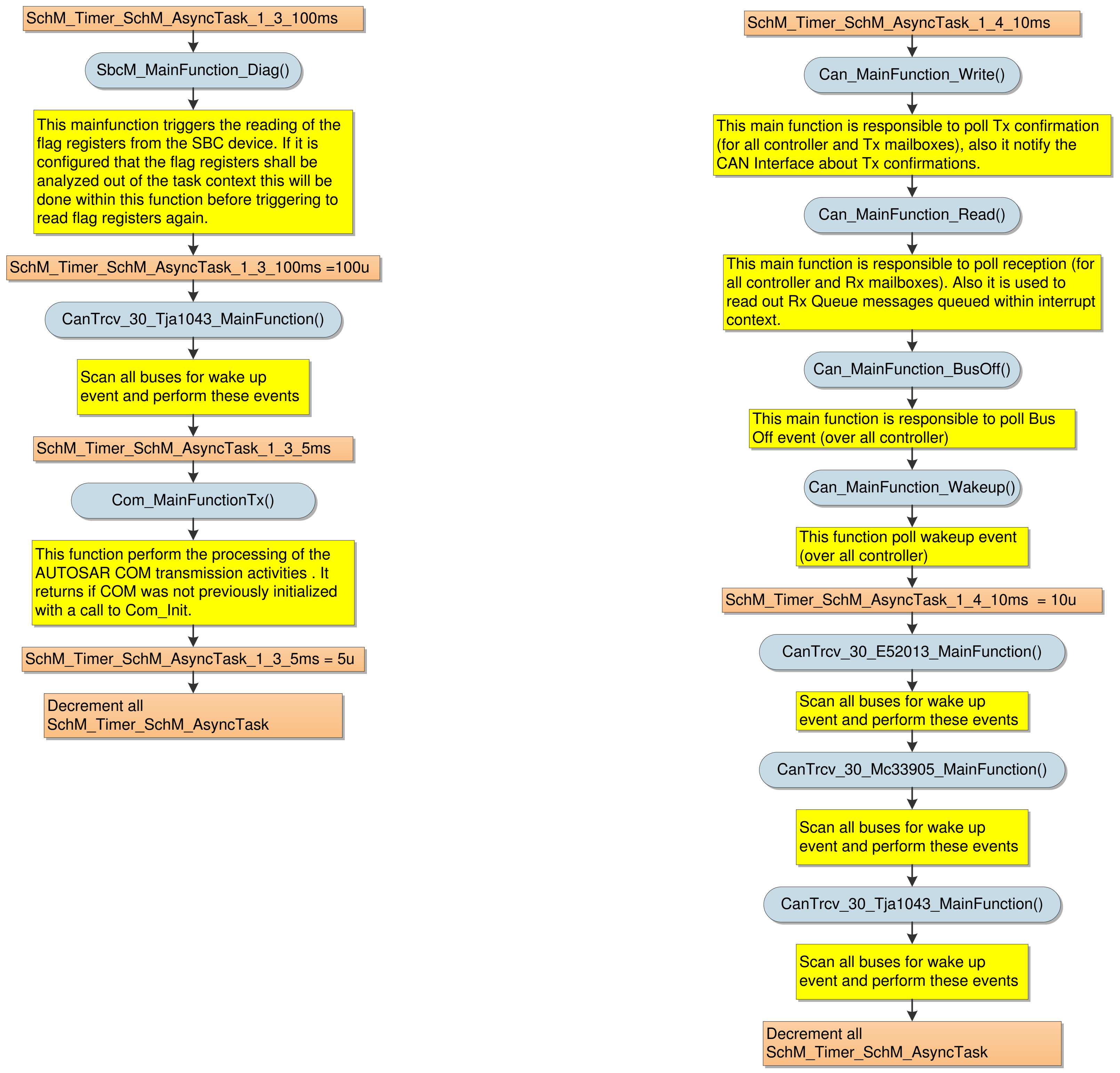

SchM_Timer_SchM_AsyncTask_1_3_100ms
SbcM_MainFunction_Diag()
This mainfunction triggers the reading of the flag registers from the SBC device. If it is configured that the flag registers shall be analyzed out of the task context this will be done within this function before triggering to read flag registers again.
SchM_Timer_SchM_AsyncTask_1_3_100ms =100u
CanTrcv_30_Tja1043_MainFunction()
Scan all buses for wake up event and perform these events
SchM_Timer_SchM_AsyncTask_1_3_5ms
Com_MainFunctionTx()
This function perform the processing of the AUTOSAR COM transmission activities . It returns if COM was not previously initialized with a call to Com_Init.
SchM_Timer_SchM_AsyncTask_1_3_5ms = 5u
Decrement all SchM_Timer_SchM_AsyncTask
SchM_Timer_SchM_AsyncTask_1_4_10ms
Can_MainFunction_Write()
This main function is responsible to poll Tx confirmation (for all controller and Tx mailboxes), also it notify the CAN Interface about Tx confirmations.
Can_MainFunction_Read()
This main function is responsible to poll reception (for all controller and Rx mailboxes). Also it is used to read out Rx Queue messages queued within interrupt context.
Can_MainFunction_BusOff()
This main function is responsible to poll Bus Off event (over all controller)
Can_MainFunction_Wakeup()
This function poll wakeup event (over all controller)
SchM_Timer_SchM_AsyncTask_1_4_10ms = 10u
CanTrcv_30_E52013_MainFunction()
Scan all buses for wake up event and perform these events
CanTrcv_30_Mc33905_MainFunction()
Scan all buses for wake up event and perform these events
CanTrcv_30_Tja1043_MainFunction()
Scan all buses for wake up event and perform these events
Decrement all SchM_Timer_SchM_AsyncTask

Figure 11-10: Program flow chart for the SchM_AsyncTask part 3

11.6. **Performance analyses of SingleCoreV1_0 V1_0_1111**

11.6.1. V1_0_1111 write-process 0 to 6 ms

	Name	Entry in mse	End in ms	Duration in m	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
0	Task_Event_Expl_Swc4to3func	20.052158	20.068304	0.016146	Runnable_Swc04_Event_Expl_DataStruct_Out	20.056712	20.062561	0.005849	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	20.059637	20.062447	0.001115
	Task_Event_Impl_Swc3to4func	20.073101	20.089822	0.016721	Runnable_Swc03_Event_Impl_DataStruct_Out	20.077049	20.080439	0.003390	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	20.094292	20.107327	0.013035	Runnable_Swc06_Period_Expl_DataDIN3_Out	20.097737	20.102599	0.004862	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	20.100767	20.102478	0.000345
	Task_IoHw_Period_Impl_Swc5to6func	20.111681	20.133818	0.022137	Runnable_Swc05_Period_Impl_DataDIN2_Out	20.122561	20.125529	0.002968	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1func	20.138524	20.146281	0.007757	Runnable_Swc02_Period_Expl_DataStruct_Out	20.141817	20.144074	0.002257	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	20.141908	20.143974	0.000727
	Task_Period_Impl_Swc1to2func	20.150864	20.160083	0.009219	Runnable_Swc01_Period_Impl_DataStruct_Out	20.154175	20.154579	0.000404	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	20.168599	20.724529	0.555930								
1	Task_Event_Impl_Swc3to4func	20.754995	20.771530	0.016535	Runnable_Swc03_Event_Impl_DataStruct_Out	20.757521	20.760684	0.003163	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	20.776423	20.788622	0.012199	Runnable_Swc04_Event_Expl_DataStruct_Out	20.778557	20.783276	0.004719	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	20.781156	20.783181	0.000812
	Task_IoHw_Period_Impl_Swc5to6func	20.793531	20.805218	0.011687	Runnable_Swc05_Period_Impl_DataDIN2_Out	20.795173	20.797537	0.002364	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	20.809775	20.820547	0.010772	Runnable_Swc06_Period_Expl_DataDIN3_Out	20.811801	20.816040	0.004239	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	20.814185	20.815923	0.000324
	osIdleLoop	21.323354	21.724636	0.401282								
2	Task_Event_Impl_Swc3to4func	21.755307	21.772689	0.017382	Runnable_Swc03_Event_Impl_DataStruct_Out	21.758294	21.761715	0.003421	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	21.777546	21.790417	0.012871	Runnable_Swc04_Event_Expl_DataStruct_Out	21.779849	21.785448	0.005599	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	21.782758	21.785338	0.000979
	Task_IoHw_Period_Expl_Swc6to5func	21.795592	21.806535	0.010943	Runnable_Swc06_Period_Expl_DataDIN3_Out	21.797638	21.802212	0.004574	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	21.800227	21.802096	0.000345
	Task_IoHw_Period_Impl_Swc5to6func	21.811403	21.823774	0.012371	Runnable_Swc05_Period_Impl_DataDIN2_Out	21.813264	21.815714	0.002450	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	22.254081	22.724710	0.470629								
3	Task_Event_Expl_Swc4to3func	22.770773	22.785583	0.014810	Runnable_Swc04_Event_Expl_DataStruct_Out	22.773440	22.779227	0.005787	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	22.776371	22.779111	0.001101
	Task_Event_Impl_Swc3to4func	22.790952	22.813777	0.022825	Runnable_Swc03_Event_Impl_DataStruct_Out	22.792666	22.796117	0.003451	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Impl_Swc5to6func	22.819412	22.833007	0.013595	Runnable_Swc05_Period_Impl_DataDIN2_Out	22.821046	22.823920	0.002874	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	22.837634	22.849385	0.011751	Runnable_Swc06_Period_Expl_DataDIN3_Out	22.839357	22.844245	0.004888	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	22.842095	22.844130	0.000369
	osIdleLoop	23.211772	23.724801	0.513029								
4	Task_Event_Expl_Swc4to3func	23.754452	23.768900	0.014448	Runnable_Swc04_Event_Expl_DataStruct_Out	23.756884	23.762462	0.005578	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	23.759790	23.762347	0.001001
	Task_Event_Impl_Swc3to4func	23.774312	23.788663	0.014351	Runnable_Swc03_Event_Impl_DataStruct_Out	23.776701	23.779424	0.002723	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Impl_Swc5to6func	23.793412	23.806235	0.012823	Runnable_Swc05_Period_Impl_DataDIN2_Out	23.795578	23.798049	0.002471	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	23.810910	23.821486	0.010576	Runnable_Swc06_Period_Expl_DataDIN3_Out	23.812869	23.817023	0.004154	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	23.815145	23.816914	0.000338
	osIdleLoop	24.150054	24.724892	0.574838								
5	Task_Event_Impl_Swc3to4func	24.763335	24.780280	0.016945	Runnable_Swc03_Event_Impl_DataStruct_Out	24.766058	24.769368	0.003310	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	24.785082	24.808711	0.023629	Runnable_Swc04_Event_Expl_DataStruct_Out	24.787042	24.802228	0.015186	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	24.799548	24.802111	0.000974
	Task_IoHw_Period_Expl_Swc6to5func	24.814277	24.827021	0.012744	Runnable_Swc06_Period_Expl_DataDIN3_Out	24.816581	24.821407	0.004826	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	24.819579	24.821294	0.000321
	Task_IoHw_Period_Impl_Swc5to6func	24.831870	24.844173	0.012303	Runnable_Swc05_Period_Impl_DataDIN2_Out	24.833637	24.836017	0.002380	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Period_Impl_Swc1to2func	24.848787	24.857269	0.008482	Runnable_Swc01_Period_Impl_DataStruct_Out	24.850393	24.851006	0.000613	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1func	24.862588	24.868449	0.005861	Runnable_Swc02_Period_Expl_DataStruct_Out	24.864263	24.866228	0.001965	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	24.864353	24.866136	0.000725
	osIdleLoop	25.184918	25.725011	0.540093								
6	Task_Event_Impl_Swc3to4func	25.764524	25.781963	0.017439	Runnable_Swc03_Event_Impl_DataStruct_Out	25.767467	25.770679	0.003212	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	25.786893	25.799658	0.012765	Runnable_Swc04_Event_Expl_DataStruct_Out	25.788931	25.794267	0.005336	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	25.791840	25.794156	0.000979
	Task_IoHw_Period_Expl_Swc6to5func	25.804350	25.815961	0.011611	Runnable_Swc06_Period_Expl_DataDIN3_Out	25.806446	25.811136	0.004690	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	25.809481	25.811014	0.000300
	Task_IoHw_Period_Impl_Swc5to6func	25.820808	25.832743	0.011935	Runnable_Swc05_Period_Impl_DataDIN2_Out	25.822774	25.824994	0.002220	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	26.148072	26.725092	0.577020								

11.6.2. V1_0_1111 notification process 0 to 6 ms

	SetEvent Entry in ms	End in ms	Duration in ms	OsSchedule Entry in ms	End in ms	Duration in ms	osGetEvent Entry in ms	End in ms	Duration in ms	OsWait Entry in ms	End in ms	Duration in ms	OsClearEvent Entry in ms	End in ms	Duration in ms	Notification sum in ms
0	20.060628	20.062323	0.001695	20.062587	20.065444	0.002857	20.065523	20.066254	0.000731	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005283
	20.081240	20.082634	0.001394	20.082668	20.085275	0.002607	20.087006	20.087615	0.000609	20.085403	20.086940	0.001537	20.087660	20.088668	0.001008	0.007155
	20.100993	20.102359	0.001366	20.102625	20.104817	0.002192	20.104896	20.105625	0.000729	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004287
	20.125633	20.127155	0.001522	20.127184	20.129533	0.002349	20.131470	20.132150	0.000680	20.129680	20.131397	0.001717	20.132221	20.133540	0.001319	0.007587
	20.142511	20.143850	0.001339	0.000000	0.000000	0.000000	20.144108	20.144421	0.000313	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001652
	20.155175	20.156263	0.001088	0.000000	0.000000	0.000000	20.157810	20.158142	0.000332	20.156370	20.157775	0.001405	20.158186	20.159201	0.001015	0.003840
1	20.761475	20.762799	0.001324	20.762834	20.766068	0.003234	20.768287	20.768900	0.000613	20.766262	20.768221	0.001959	20.768955	20.770099	0.001144	0.008274
	20.781846	20.783059	0.001213	20.783288	20.785942	0.002654	20.786016	20.786695	0.000679	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004546
	20.797634	20.798947	0.001313	20.798977	20.801305	0.002328	20.803086	20.803692	0.000606	20.801444	20.803021	0.001577	20.803751	20.804932	0.001181	0.007005
	20.814398	20.815812	0.001414	20.816062	20.818380	0.002318	20.818457	20.819169	0.000712	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004444
2	21.762641	21.764271	0.001630	21.764307	21.767441	0.003134	21.769641	21.770247	0.000606	21.767635	21.769576	0.001941	21.770307	21.771522	0.001215	0.008526
	21.783613	21.785214	0.001601	21.785469	21.787580	0.002111	21.787657	21.788370	0.000713	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004425
	21.800461	21.801985	0.001524	21.802234	21.804432	0.002198	21.804501	21.805140	0.000639	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004361
	21.815830	21.817345	0.001515	21.817364	21.819610	0.002246	21.821351	21.821965	0.000614	21.819726	21.821285	0.001559	21.822027	21.823257	0.001230	0.007164
3	22.777348	22.778987	0.001639	22.779254	22.782370	0.003116	22.782445	22.783137	0.000692	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005447
	22.796798	22.798161	0.001363	22.798187	22.801058	0.002871	22.810724	22.811340	0.000616	22.801191	22.810658	0.009467	22.811406	22.812694	0.001288	0.015605
	22.824068	22.825786	0.001718	22.825817	22.828976	0.003159	22.830817	22.831505	0.000688	22.829110	22.830743	0.001633	22.838196	22.839233	0.001037	0.008235
	22.842348	22.844014	0.001666	22.844272	22.847041	0.002769	22.847113	22.847770	0.000657	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005092
4	23.760674	23.762230	0.001556	23.762488	23.765561	0.003073	23.765636	23.766328	0.000692	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005321
	23.780010	23.781162	0.001152	23.781188	23.783895	0.002707	23.785688	23.786247	0.000559	23.784035	23.785629	0.001594	23.786307	23.787504	0.001197	0.007209
	23.798160	23.799600	0.001440	23.799629	23.802046	0.002417	23.803777	23.804389	0.000612	23.802163	23.803712	0.001549	23.804468	23.805891	0.001423	0.007441
	23.815365	23.816796	0.001431	23.817038	23.819116	0.002078	23.819194	23.819907	0.000713	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004222
5	24.770239	24.771892	0.001653	24.771921	24.774850	0.002929	24.777062	24.777751	0.000689	24.775047	24.776988	0.001941	24.777823	24.779158	0.001335	0.008547
	24.800398	24.801987	0.001589	24.802256	24.805391	0.003135	24.805476	24.806255	0.000779	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005503
	24.819784	24.821178	0.001394	24.821433	24.824198	0.002765	24.824275	24.825002	0.000727	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004886
	24.836151	24.837719	0.001568	24.837748	24.840455	0.002707	24.842076	24.842611	0.000535	24.840572	24.842019	0.001447	24.842673	24.843888	0.001215	0.007472
	24.851581	24.852792	0.001211	0.000000	0.000000	0.000000	24.854637	24.854969	0.000332	24.852908	24.854602	0.001694	24.855014	24.856033	0.001019	0.004256
	24.864956	24.866014	0.001058	0.000000	0.000000	0.000000	24.866259	24.866550	0.000291	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001349
6	25.771615	25.773307	0.001692	25.773342	25.776608	0.003266	25.778942	25.779551	0.000609	25.776821	25.778876	0.002055	25.779596	25.780607	0.001011	0.008633
	25.792695	25.794032	0.001337	25.794290	25.796714	0.002424	25.796785	25.797443	0.000658	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004419
	25.809663	25.810896	0.001233	25.811163	25.813565	0.002402	25.813626	25.814189	0.000563	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004198
	25.825106	25.826555	0.001449	25.826583	25.828992	0.002409	25.830731	25.831326	0.000595	25.829110	25.830666	0.001556	25.831379	25.832477	0.001098	0.007107

11.6.3. V1_0_1111 read-process 0 to 6 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
0	Runnable_Swc03_Odr_Expl_DataStruct_In	20.066346	20.067232	0.000886	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	20.066456	20.067129	0.000673	0.001788	0.007071
	Runnable_Swc04_Odr_Impl_DataStruct_In	20.089273	20.089771	0.000498	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007155
	Runnable_Swc05_Odr_Expl_DataDIN3_In	20.105707	20.105993	0.000286	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	20.105789	20.105884	0.000095	0.000440	0.004727
	Runnable_Swc06_Odr_Impl_DataDIN2_In	20.133704	20.133763	0.000059	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007587
	Runnable_Swc01_Odr_Expl_DataStruct_In	20.144485	20.145143	0.000658	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	20.144562	20.145038	0.000476	0.001203	0.002855
	Runnable_Swc02_Odr_Impl_DataStruct_In	20.159661	20.160041	0.000380	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.003840
1	Runnable_Swc04_Odr_Impl_DataStruct_In	20.770859	20.771480	0.000621	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008274
	Runnable_Swc03_Odr_Expl_DataStruct_In	20.786763	20.787452	0.000689	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	20.786844	20.787347	0.000503	0.001315	0.005861
	Runnable_Swc06_Odr_Impl_DataDIN2_In	20.805110	20.805170	0.000060	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007005
	Runnable_Swc05_Odr_Expl_DataDIN3_In	20.819241	20.819503	0.000262	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	20.819314	20.819402	0.000088	0.000412	0.004856
2	Runnable_Swc04_Odr_Impl_DataStruct_In	21.772127	21.772629	0.000502	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008526
	Runnable_Swc03_Odr_Expl_DataStruct_In	21.788453	21.789260	0.000807	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	21.788551	21.789156	0.000605	0.001584	0.006009
	Runnable_Swc05_Odr_Expl_DataDIN3_In	21.805207	21.805459	0.000252	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	21.805273	21.805356	0.000083	0.000428	0.004789
	Runnable_Swc06_Odr_Impl_DataDIN2_In	21.823618	21.823725	0.000107	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007164
3	Runnable_Swc03_Odr_Expl_DataStruct_In	22.783235	22.784167	0.000932	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	22.783352	22.784063	0.000711	0.001812	0.007259
	Runnable_Swc04_Odr_Impl_DataStruct_In	22.813239	22.813698	0.000459	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.015605
	Runnable_Swc06_Odr_Impl_DataDIN2_In	22.832879	22.832967	0.000088	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008235
	Runnable_Swc05_Odr_Expl_DataDIN3_In	22.847849	22.848127	0.000278	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	22.847927	22.848020	0.000093	0.000462	0.005554
4	Runnable_Swc03_Odr_Expl_DataStruct_In	23.766426	23.767361	0.000935	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	23.766543	23.767254	0.000711	0.001712	0.007033
	Runnable_Swc04_Odr_Impl_DataStruct_In	23.788109	23.788609	0.000500	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007209
	Runnable_Swc06_Odr_Impl_DataDIN2_In	23.806116	23.806188	0.000072	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007441
	Runnable_Swc05_Odr_Expl_DataDIN3_In	23.819992	23.820281	0.000289	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	23.820077	23.820174	0.000097	0.000435	0.004657
5	Runnable_Swc04_Odr_Impl_DataStruct_In	24.779738	24.780220	0.000482	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008547
	Runnable_Swc03_Odr_Expl_DataStruct_In	24.806353	24.807285	0.000932	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	24.806470	24.807181	0.000711	0.001685	0.007188
	Runnable_Swc05_Odr_Expl_DataDIN3_In	24.825150	24.825559	0.000409	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	24.825299	24.825447	0.000148	0.000469	0.005355
	Runnable_Swc06_Odr_Impl_DataDIN2_In	24.844072	24.844131	0.000059	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007472
	Runnable_Swc02_Odr_Impl_DataStruct_In	24.856688	24.857224	0.000536	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.004256
	Runnable_Swc01_Odr_Expl_DataStruct_In	24.866619	24.867324	0.000705	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	24.866703	24.867220	0.000517	0.001242	0.002591
6	Runnable_Swc04_Odr_Impl_DataStruct_In	25.781322	25.781909	0.000587	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008633
	Runnable_Swc03_Odr_Expl_DataStruct_In	25.797527	25.798347	0.000820	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	25.797626	25.798238	0.000612	0.001591	0.006010
	Runnable_Swc05_Odr_Expl_DataDIN3_In	25.814274	25.814566	0.000292	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	25.814359	25.814456	0.000097	0.000397	0.004595
	Runnable_Swc06_Odr_Impl_DataDIN2_In	25.832648	25.832704	0.000056	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007107

11.6.4. V1_0_1111 write-process 7 to 12 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
7	Task_Event_Expl_Swc4to3func	26.755061	26.769437	0.014376	Runnable_Swc04_Event_Expl_DataStruct_Out	26.757685	26.763292	0.005607	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	26.760596	26.763174	0.000988
	Task_Event_Impl_Swc3to4func	26.774613	26.788788	0.014175	Runnable_Swc03_Event_Impl_DataStruct_Out	26.776366	26.779045	0.002679	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Impl_Swc5to6func	26.793898	26.805826	0.011928	Runnable_Swc05_Period_Impl_DataDIN2_Out	26.795613	26.798179	0.002566	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	26.810416	26.821187	0.010771	Runnable_Swc06_Period_Expl_DataDIN3_Out	26.812174	26.816610	0.004436	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	26.814798	26.816505	0.000323
	osIdleLoop	27.130627	27.725165	0.594538								
8	Task_Event_Expl_Swc4to3func	27.755099	27.769087	0.013988	Runnable_Swc04_Event_Expl_DataStruct_Out	27.757549	27.762863	0.005314	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	27.760127	27.762747	0.001007
	Task_Event_Impl_Swc3to4func	27.774377	27.788785	0.014408	Runnable_Swc03_Event_Impl_DataStruct_Out	27.776604	27.779624	0.003020	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Impl_Swc5to6func	27.793648	27.806171	0.012523	Runnable_Swc05_Period_Impl_DataDIN2_Out	27.795586	27.798156	0.002570	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	27.810845	27.821528	0.010683	Runnable_Swc06_Period_Expl_DataDIN3_Out	27.812642	27.817060	0.004418	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	27.815365	27.816950	0.000291
	osIdleLoop	28.129354	28.725258	0.595904								
9	Task_Event_Impl_Swc3to4func	28.754757	28.771330	0.016573	Runnable_Swc03_Event_Impl_DataStruct_Out	28.757385	28.760784	0.003399	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	28.776351	28.788474	0.012123	Runnable_Swc04_Event_Expl_DataStruct_Out	28.778514	28.783130	0.004616	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	28.781004	28.783027	0.000821
	Task_IoHw_Period_Expl_Swc6to5func	28.793266	28.804472	0.011206	Runnable_Swc06_Period_Expl_DataDIN3_Out	28.795263	28.799840	0.004577	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	28.797990	28.799739	0.000323
	Task_IoHw_Period_Impl_Swc5to6func	28.809250	28.821058	0.011808	Runnable_Swc05_Period_Impl_DataDIN2_Out	28.810950	28.813421	0.002471	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	29.139099	29.725398	0.586299								
10	Task_Event_Expl_Swc4to3func	29.762608	29.776343	0.013735	Runnable_Swc04_Event_Expl_DataStruct_Out	29.765049	29.770699	0.005650	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	29.767956	29.770584	0.001017
	Task_Event_Impl_Swc3to4func	29.781672	29.796701	0.015029	Runnable_Swc03_Event_Impl_DataStruct_Out	29.783849	29.786973	0.003124	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Impl_Swc5to6func	29.801486	29.814493	0.013007	Runnable_Swc05_Period_Impl_DataDIN2_Out	29.803144	29.805998	0.002854	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	29.819047	29.830052	0.011005	Runnable_Swc06_Period_Expl_DataDIN3_Out	29.820999	29.825643	0.004644	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	29.823803	29.825532	0.000331
	Task_Period_Expl_Swc2to1func	29.834741	29.840449	0.005708	Runnable_Swc02_Period_Expl_DataStruct_Out	29.836457	29.838364	0.001907	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	29.836541	29.838272	0.000685
	Task_Period_Impl_Swc1to2func	29.845107	29.852548	0.007441	Runnable_Swc01_Period_Impl_DataStruct_Out	29.846713	29.847124	0.000411	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	30.155472	30.725447	0.569975								
11	Task_Event_Expl_Swc4to3func	30.756139	30.769723	0.013584	Runnable_Swc04_Event_Expl_DataStruct_Out	30.758549	30.764299	0.005750	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	30.761474	30.764190	0.001065
	Task_Event_Impl_Swc3to4func	30.774946	30.788906	0.013960	Runnable_Swc03_Event_Impl_DataStruct_Out	30.776979	30.779807	0.002828	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	30.793920	30.804966	0.011046	Runnable_Swc06_Period_Expl_DataDIN3_Out	30.795963	30.800434	0.004471	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	30.798784	30.800321	0.000316
	Task_IoHw_Period_Impl_Swc5to6func	30.809663	30.821962	0.012299	Runnable_Swc05_Period_Impl_DataDIN2_Out	30.811631	30.814046	0.002415	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	31.169399	31.725557	0.556158								
12	Task_Event_Impl_Swc3to4func	31.755700	31.773027	0.017327	Runnable_Swc03_Event_Impl_DataStruct_Out	31.758440	31.761859	0.003419	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	31.778040	31.790938	0.012898	Runnable_Swc04_Event_Expl_DataStruct_Out	31.780149	31.785275	0.005126	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	31.783030	31.785165	0.000870
	Task_IoHw_Period_Expl_Swc6to5func	31.795992	31.806901	0.010909	Runnable_Swc06_Period_Expl_DataDIN3_Out	31.797874	31.802110	0.004236	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	31.800264	31.801996	0.000327
	Task_IoHw_Period_Impl_Swc5to6func	31.811592	31.823479	0.011887	Runnable_Swc05_Period_Impl_DataDIN2_Out	31.813437	31.815630	0.002193	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	32.235327	32.725629	0.490302								

11.6.5. V1_0_1111 notification process 7 to 12 ms

	SetEvent Entry in ms	End in ms	Duration in ms	OsSchedule Entry in ms	End in ms	Duration in ms	osGetEvent Entry in ms	End in ms	Duration in ms	OsWait Entry in ms	End in ms	Duration in ms	OsClearEvent Entry in ms	End in ms	Duration in ms	Notification sum in ms
7	26.761460	26.763050	0.001590	26.763321	26.766433	0.003112	26.766508	26.767200	0.000692	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005394
	26.779736	26.780983	0.001247	26.781003	26.783788	0.002785	26.785614	26.786220	0.000606	26.783935	26.785548	0.001613	26.786283	26.787514	0.001231	0.007482
	26.798256	26.799347	0.001091	26.799377	26.801706	0.002329	26.803524	26.804141	0.000617	26.801853	26.803458	0.001605	26.804212	26.805533	0.001321	0.006963
	26.815003	26.816387	0.001384	26.816621	26.818743	0.002122	26.818821	26.819532	0.000711	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004217
8	27.761010	27.762623	0.001613	27.762890	27.765798	0.002908	27.765875	27.766592	0.000717	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005238
	27.780210	27.781443	0.001233	27.781467	27.783832	0.002365	27.785740	27.786337	0.000597	27.783973	27.785675	0.001702	27.786393	27.787533	0.001140	0.007037
	27.798259	27.799500	0.001241	27.799529	27.801760	0.002231	27.803584	27.804357	0.000773	27.801906	27.803500	0.001594	27.804422	27.805682	0.001260	0.007099
	27.815538	27.816832	0.001294	27.817075	27.819147	0.002072	27.819209	27.819776	0.000567	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.003933
9	28.761570	28.762956	0.001386	28.762985	28.766109	0.003124	28.768151	28.768764	0.000613	28.766278	28.768085	0.001807	28.768820	28.769961	0.001141	0.008071
	28.781703	28.782905	0.001202	28.783151	28.785552	0.002401	28.785626	28.786315	0.000689	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004292
	28.798197	28.799623	0.001426	28.799854	28.802025	0.002171	28.802103	28.802813	0.000710	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004307
	28.813527	28.814828	0.001301	28.814857	28.817087	0.002230	28.818900	28.819598	0.000698	28.817233	28.818825	0.001592	28.819647	28.820718	0.001071	0.006892
10	29.768848	29.770459	0.001611	29.770726	29.773416	0.002690	29.773494	29.794313	0.020819	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.025120
	29.787709	29.789074	0.001365	29.789111	29.791699	0.002588	29.793633	29.812659	0.019026	29.791863	29.793560	0.001697	29.812730	29.814052	0.001322	0.025998
	29.806130	29.807655	0.001525	29.807682	29.810093	0.002411	29.811884	29.828441	0.016557	29.810215	29.811800	0.001585	29.828831	29.829992	0.001161	0.023239
	29.824016	29.825414	0.001398	29.825658	29.827732	0.002074	29.827801	29.838683	0.010882	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.014354
	29.837104	29.838150	0.001046	0.000000	0.000000	0.000000	29.838395	29.850642	0.012247	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.013293
	29.847705	29.848761	0.001056	0.000000	0.000000	0.000000	29.850294	29.850642	0.000348	29.848858	29.850257	0.001399	29.850686	29.851682	0.000996	0.003799
11	30.762418	30.764069	0.001651	30.764327	30.766833	0.002506	30.766907	30.767592	0.000685	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004842
	30.780422	30.781684	0.001262	30.781705	30.784009	0.002304	30.785951	30.786564	0.000613	30.784172	30.785885	0.001713	30.786620	30.787759	0.001139	0.007031
	30.798984	30.800205	0.001221	30.800460	30.802723	0.002263	30.802793	30.803444	0.000651	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004135
	30.814142	30.815532	0.001390	30.815547	30.817742	0.002195	30.819658	30.820437	0.000779	30.817886	30.819573	0.001687	30.820494	30.821678	0.001184	0.007235
12	31.762730	31.764317	0.001587	31.764352	31.767590	0.003238	31.769921	31.770517	0.000596	31.767803	31.769857	0.002054	31.770569	31.771670	0.001101	0.008576
	31.783776	31.785041	0.001265	31.785296	31.788099	0.002803	31.788178	31.788908	0.000730	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004798
	31.800473	31.801878	0.001405	31.802131	31.804278	0.002147	31.804349	31.805012	0.000663	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004215
	31.815736	31.817141	0.001405	31.817164	31.819496	0.002332	31.821307	31.821989	0.000682	31.819640	31.821233	0.001593	31.822039	31.823111	0.001072	0.007084

11.6.6. V1_0_1111 read-process 7 to 12 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
7	Runnable_Swc03_Odr_Expl_DataStruct_In	26.767292	26.768181	0.000889	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	26.767402	26.768074	0.000672	0.001660	0.007054
	Runnable_Swc04_Odr_Impl_DataStruct_In	26.788174	26.788722	0.000548	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007482
	Runnable_Swc06_Odr_Impl_DataDIN2_In	26.805717	26.805779	0.000062	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006963
	Runnable_Swc05_Odr_Expl_DataDIN3_In	26.819605	26.819874	0.000269	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	26.819678	26.819765	0.000087	0.000410	0.004627
8	Runnable_Swc03_Odr_Expl_DataStruct_In	27.766689	27.767623	0.000934	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	27.766807	27.767518	0.000711	0.001718	0.006956
	Runnable_Swc04_Odr_Impl_DataStruct_In	27.788193	27.788734	0.000541	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007037
	Runnable_Swc06_Odr_Impl_DataDIN2_In	27.806023	27.806124	0.000101	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007099
	Runnable_Swc05_Odr_Expl_DataDIN3_In	27.819839	27.820095	0.000256	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	27.819903	27.819984	0.000081	0.000372	0.004305
9	Runnable_Swc04_Odr_Impl_DataStruct_In	28.770676	28.771266	0.000590	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008071
	Runnable_Swc03_Odr_Expl_DataStruct_In	28.786392	28.787165	0.000773	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	28.786485	28.787056	0.000571	0.001392	0.005684
	Runnable_Swc05_Odr_Expl_DataDIN3_In	28.802879	28.803138	0.000259	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	28.802946	28.803029	0.000083	0.000406	0.004713
	Runnable_Swc06_Odr_Impl_DataDIN2_In	28.820943	28.821014	0.000071	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006892
10	Runnable_Swc03_Odr_Expl_DataStruct_In	29.774275	29.774992	0.000717	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	29.774360	29.774884	0.000524	0.001541	0.026661
	Runnable_Swc04_Odr_Impl_DataStruct_In	29.796191	29.796632	0.000441	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.025998
	Runnable_Swc06_Odr_Impl_DataDIN2_In	29.814352	29.814443	0.000091	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.023239
	Runnable_Swc05_Odr_Expl_DataDIN3_In	29.828514	29.828773	0.000259	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	29.828587	29.828672	0.000085	0.000416	0.014770
	Runnable_Swc01_Odr_Expl_DataStruct_In	29.838736	29.839306	0.000570	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	29.838800	29.839202	0.000402	0.001087	0.014380
	Runnable_Swc02_Odr_Impl_DataStruct_In	29.852132	29.852505	0.000373	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.003799
11	Runnable_Swc03_Odr_Expl_DataStruct_In	30.767689	30.768623	0.000934	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	30.767807	30.768520	0.000713	0.001778	0.006620
	Runnable_Swc04_Odr_Impl_DataStruct_In	30.788364	30.788860	0.000496	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007031
	Runnable_Swc05_Odr_Expl_DataDIN3_In	30.803529	30.803815	0.000286	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	30.803613	30.803711	0.000098	0.000414	0.004549
	Runnable_Swc06_Odr_Impl_DataDIN2_In	30.821855	30.821915	0.000060	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007235
12	Runnable_Swc04_Odr_Impl_DataStruct_In	31.772385	31.772972	0.000587	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008576
	Runnable_Swc03_Odr_Expl_DataStruct_In	31.788996	31.789850	0.000854	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	31.789102	31.789747	0.000645	0.001515	0.006313
	Runnable_Swc05_Odr_Expl_DataDIN3_In	31.805118	31.805441	0.000323	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	31.805224	31.805336	0.000112	0.000439	0.004654
	Runnable_Swc06_Odr_Impl_DataDIN2_In	31.823364	31.823440	0.000076	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007084

11.6.7. V1_0_1111 write-process 13 to 18 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
13	Task_Event_Expl_Swc4to3func	32.762963	32.777763	0.014800	Runnable_Swc04_Event_Expl_DataStruct_Out	32.765703	32.771554	0.005851	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	32.768636	32.771438	0.001111
	Task_Event_Impl_Swc3to4func	32.783026	32.797699	0.014673	Runnable_Swc03_Event_Impl_DataStruct_Out	32.784885	32.788079	0.003194	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Impl_Swc5to6func	32.802648	32.815430	0.012782	Runnable_Swc05_Period_Impl_DataDIN2_Out	32.804582	32.807068	0.002486	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	32.820234	32.830380	0.010146	Runnable_Swc06_Period_Expl_DataDIN3_Out	32.822115	32.825925	0.003810	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	32.824127	32.825805	0.000352
	osIdleLoop	33.195118	33.725729	0.530611								
14	Task_Event_Expl_Swc4to3func	33.755875	33.770427	0.014552	Runnable_Swc04_Event_Expl_DataStruct_Out	33.758412	33.764245	0.005833	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	33.761336	33.764129	0.001111
	Task_Event_Impl_Swc3to4func	33.775651	33.790275	0.014624	Runnable_Swc03_Event_Impl_DataStruct_Out	33.777449	33.780793	0.003344	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	33.795257	33.806263	0.011006	Runnable_Swc06_Period_Expl_DataDIN3_Out	33.797208	33.801832	0.004624	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	33.800116	33.801723	0.000291
	Task_IoHw_Period_Impl_Swc5to6func	33.810907	33.822967	0.012060	Runnable_Swc05_Period_Impl_DataDIN2_Out	33.812717	33.815232	0.002515	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	34.133981	34.725810	0.591829								
15	Task_Event_Impl_Swc3to4func	34.763399	34.779489	0.016090	Runnable_Swc03_Event_Impl_DataStruct_Out	34.765931	34.769348	0.003417	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	34.784495	34.796522	0.012027	Runnable_Swc04_Event_Expl_DataStruct_Out	34.786748	34.791350	0.004602	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	34.789328	34.791238	0.000705
	Task_IoHw_Period_Expl_Swc6to5func	34.801712	34.812476	0.010764	Runnable_Swc06_Period_Expl_DataDIN3_Out	34.803762	34.808026	0.004264	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	34.806215	34.807905	0.000310
	Task_IoHw_Period_Impl_Swc5to6func	34.817062	34.829179	0.012117	Runnable_Swc05_Period_Impl_DataDIN2_Out	34.818880	34.821586	0.002706	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Period_Impl_Swc1to2func	34.834012	34.842376	0.008364	Runnable_Swc01_Period_Impl_DataStruct_Out	34.835684	34.836295	0.000611	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1func	34.847360	34.852842	0.005482	Runnable_Swc02_Period_Expl_DataStruct_Out	34.848975	34.850746	0.001771	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	34.849051	34.850647	0.000630
	osIdleLoop	35.161945	35.725903	0.563958								
16	Task_Event_Impl_Swc3to4func	35.756495	35.772764	0.016269	Runnable_Swc03_Event_Impl_DataStruct_Out	35.759037	35.762241	0.003204	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	35.777619	35.789784	0.012165	Runnable_Swc04_Event_Expl_DataStruct_Out	35.779725	35.784604	0.004879	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	35.782490	35.784493	0.000845
	Task_IoHw_Period_Impl_Swc5to6func	35.794977	35.807057	0.012080	Runnable_Swc05_Period_Impl_DataDIN2_Out	35.797118	35.799471	0.002353	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	35.811490	35.822007	0.010517	Runnable_Swc06_Period_Expl_DataDIN3_Out	35.813350	35.817566	0.004216	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	35.815772	35.817459	0.000339
	osIdleLoop	36.130363	36.725992	0.595629								
17	Task_Event_Expl_Swc4to3func	36.756185	36.770535	0.014350	Runnable_Swc04_Event_Expl_DataStruct_Out	36.758749	36.764434	0.005685	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	36.761671	36.764320	0.001049
	Task_Event_Impl_Swc3to4func	36.775694	36.790897	0.015203	Runnable_Swc03_Event_Impl_DataStruct_Out	36.777749	36.780966	0.003217	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	36.795575	36.807500	0.011925	Runnable_Swc06_Period_Expl_DataDIN3_Out	36.797828	36.802761	0.004933	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	36.800853	36.802650	0.000350
	Task_IoHw_Period_Impl_Swc5to6func	36.812182	36.824108	0.011926	Runnable_Swc05_Period_Impl_DataDIN2_Out	36.814009	36.816385	0.002376	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	37.133990	37.726083	0.592093								
18	Task_Event_Impl_Swc3to4func	37.756008	37.772982	0.016974	Runnable_Swc03_Event_Impl_DataStruct_Out	37.758576	37.761851	0.003275	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	37.777863	37.789671	0.011808	Runnable_Swc04_Event_Expl_DataStruct_Out	37.779957	37.784440	0.004483	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	37.782410	37.784329	0.000757
	Task_IoHw_Period_Impl_Swc5to6func	37.795057	37.807773	0.012716	Runnable_Swc05_Period_Impl_DataDIN2_Out	37.797216	37.799557	0.002341	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	37.812412	37.822750	0.010338	Runnable_Swc06_Period_Expl_DataDIN3_Out	37.813999	37.818326	0.004327	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	37.816507	37.818205	0.000330
	osIdleLoop	38.130345	38.726174	0.595829								

11.6.8. V1_0_1111 notification process 13 to 18 ms

	SetEvent Entry in ms	End in ms	Duration in ms	OsSchedule Entry in ms	End in ms	Duration in ms	osGetEvent Entry in ms	End in ms	Duration in ms	OsWait Entry in ms	End in ms	Duration in ms	OsClearEvent Entry in ms	End in ms	Duration in ms	Notification sum in ms
13	32.769623	32.771314	0.001691	32.771581	32.774697	0.003116	32.774772	32.775463	0.000691	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005498
	32.788685	32.789940	0.001255	32.789972	32.792814	0.002842	32.794597	32.795212	0.000615	32.792957	32.794531	0.001574	32.795276	32.796541	0.001265	0.007551
	32.807181	32.808660	0.001479	32.808675	32.811170	0.002495	32.813116	32.813737	0.000621	32.811320	32.813049	0.001729	32.813794	32.814970	0.001176	0.007500
	32.824361	32.825687	0.001326	32.825950	32.828141	0.002191	32.828209	32.828844	0.000635	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004152
14	33.762323	33.764005	0.001682	33.764272	33.767360	0.003088	33.767434	33.768118	0.000684	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005454
	33.781589	33.783031	0.001442	33.783063	33.785641	0.002578	33.787424	33.788036	0.000612	33.785784	33.787358	0.001574	33.788091	33.789230	0.001139	0.007345
	33.800296	33.801612	0.001316	33.801847	33.803923	0.002076	33.803992	33.804635	0.000643	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004035
	33.815350	33.816677	0.001327	33.816700	33.819025	0.002325	33.820671	33.821356	0.000685	33.819125	33.820597	0.001472	33.821415	33.822593	0.001178	0.006987
15	34.770144	34.771537	0.001393	34.771566	34.774377	0.002811	34.776497	34.777110	0.000613	34.774571	34.776431	0.001860	34.777168	34.778322	0.001154	0.007831
	34.789909	34.791114	0.001205	34.791373	34.793497	0.002124	34.793572	34.794263	0.000691	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004020
	34.806407	34.807787	0.001380	34.808052	34.810232	0.002180	34.810304	34.810962	0.000658	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004218
	34.821703	34.823145	0.001442	34.823164	34.825418	0.002254	34.827138	34.827663	0.000525	34.825532	34.827082	0.001550	34.827717	34.828845	0.001128	0.006899
	34.836826	34.837937	0.001111	0.000000	0.000000	0.000000	34.839828	34.840160	0.000332	34.838054	34.839793	0.001739	34.840205	34.841231	0.001026	0.004208
	34.849557	34.850523	0.000966	0.000000	0.000000	0.000000	34.850778	34.851074	0.000296	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001262
16	35.763112	35.764699	0.001587	35.764734	35.767789	0.003055	35.769669	35.770280	0.000611	35.767942	35.769603	0.001661	35.770332	35.771434	0.001102	0.008016
	35.783218	35.784376	0.001158	35.784627	35.786779	0.002152	35.786853	35.787537	0.000684	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.003994
	35.799554	35.800800	0.001246	35.800819	35.803078	0.002259	35.804906	35.805520	0.000614	35.803222	35.804840	0.001618	35.805582	35.806786	0.001204	0.006941
	35.815993	35.817341	0.001348	35.817577	35.819716	0.002139	35.819795	35.820511	0.000716	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004203
17	36.762596	36.764196	0.001600	36.764459	36.767515	0.003056	36.767589	36.768274	0.000685	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005341
	36.781752	36.783088	0.001336	36.783123	36.785968	0.002845	36.787743	36.788368	0.000625	36.786109	36.787676	0.001567	36.788435	36.789723	0.001288	0.007661
	36.801085	36.802532	0.001447	36.802777	36.804989	0.002212	36.805066	36.805779	0.000713	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004372
	36.816493	36.817856	0.001363	36.817885	36.820116	0.002231	36.822080	36.822760	0.000680	36.820269	36.822006	0.001737	36.822805	36.823823	0.001018	0.007029
18	37.762782	37.764433	0.001651	37.764467	37.767478	0.003011	37.769703	37.770301	0.000598	37.767673	37.769639	0.001966	37.770362	37.771553	0.001191	0.008417
	37.783043	37.784205	0.001162	37.784462	37.786779	0.002317	37.786853	37.787534	0.000681	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004160
	37.799660	37.801064	0.001404	37.801092	37.803505	0.002413	37.805387	37.806002	0.000615	37.803647	37.805321	0.001674	37.806063	37.807300	0.001237	0.007343
	37.816726	37.818094	0.001368	37.818352	37.820544	0.002192	37.820623	37.821347	0.000724	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004284

11.6.9. V1_0_1111 read-process 13 to 18 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
13	Runnable_Swc03_Odr_Expl_DataStruct_In	32.775555	32.776446	0.000891	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	32.775665	32.776338	0.000673	0.001784	0.007282
	Runnable_Swc04_Odr_Impl_DataStruct_In	32.797146	32.797645	0.000499	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007551
	Runnable_Swc06_Odr_Impl_DataDIN2_In	32.815270	32.815366	0.000096	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007500
	Runnable_Swc05_Odr_Expl_DataDIN3_In	32.828929	32.829215	0.000286	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	32.829013	32.829111	0.000098	0.000450	0.004602
14	Runnable_Swc03_Odr_Expl_DataStruct_In	33.768210	33.769101	0.000891	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	33.768320	33.768993	0.000673	0.001784	0.007238
	Runnable_Swc04_Odr_Impl_DataStruct_In	33.789760	33.790205	0.000445	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007345
	Runnable_Swc05_Odr_Expl_DataDIN3_In	33.804719	33.805008	0.000289	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	33.804804	33.804902	0.000098	0.000389	0.004424
	Runnable_Swc06_Odr_Impl_DataDIN2_In	33.822846	33.822923	0.000077	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006987
15	Runnable_Swc04_Odr_Impl_DataStruct_In	34.778927	34.779429	0.000502	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007831
	Runnable_Swc03_Odr_Expl_DataStruct_In	34.794352	34.795210	0.000858	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	34.794458	34.795109	0.000651	0.001356	0.005376
	Runnable_Swc05_Odr_Expl_DataDIN3_In	34.811050	34.811343	0.000293	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	34.811138	34.811238	0.000100	0.000410	0.004628
	Runnable_Swc06_Odr_Impl_DataDIN2_In	34.829063	34.829133	0.000070	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006899
	Runnable_Swc02_Odr_Impl_DataStruct_In	34.841836	34.842332	0.000496	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.004208
	Runnable_Swc01_Odr_Expl_DataStruct_In	34.851127	34.851698	0.000571	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	34.851191	34.851593	0.000402	0.001032	0.002294
16	Runnable_Swc04_Odr_Impl_DataStruct_In	35.772139	35.772716	0.000577	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008016
	Runnable_Swc03_Odr_Expl_DataStruct_In	35.787635	35.788572	0.000937	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	35.787752	35.788465	0.000713	0.001558	0.005552
	Runnable_Swc06_Odr_Impl_DataDIN2_In	35.806957	35.807014	0.000057	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006941
	Runnable_Swc05_Odr_Expl_DataDIN3_In	35.820569	35.820806	0.000237	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	35.820626	35.820702	0.000076	0.000415	0.004618
17	Runnable_Swc03_Odr_Expl_DataStruct_In	36.768371	36.769307	0.000936	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	36.768488	36.769202	0.000714	0.001763	0.007104
	Runnable_Swc04_Odr_Impl_DataStruct_In	36.790338	36.790844	0.000506	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007661
	Runnable_Swc05_Odr_Expl_DataDIN3_In	36.805861	36.806147	0.000286	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	36.805943	36.806038	0.000095	0.000445	0.004817
	Runnable_Swc06_Odr_Impl_DataDIN2_In	36.824001	36.824061	0.000060	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007029
18	Runnable_Swc04_Odr_Impl_DataStruct_In	37.772313	37.772934	0.000621	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008417
	Runnable_Swc03_Odr_Expl_DataStruct_In	37.787617	37.788436	0.000819	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	37.787717	37.788329	0.000612	0.001369	0.005529
	Runnable_Swc06_Odr_Impl_DataDIN2_In	37.807634	37.807731	0.000097	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007343
	Runnable_Swc05_Odr_Expl_DataDIN3_In	37.821401	37.821632	0.000231	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	37.821456	37.821529	0.000073	0.000403	0.004687

11.6.10. V1_0_1111 write-process 19 to 25 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
19	Task_Event_Impl_Swc3to4func	38.756250	38.773112	0.016862	Runnable_Swc03_Event_Impl_DataStruct_Out	38.758881	38.762041	0.003160	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	38.778034	38.789887	0.011853	Runnable_Swc04_Event_Expl_DataStruct_Out	38.780191	38.784843	0.004652	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	38.782952	38.784738	0.000681
	Task_IoHw_Period_Impl_Swc5to6func	38.794845	38.806908	0.012063	Runnable_Swc05_Period_Impl_DataDIN2_Out	38.796756	38.799145	0.002389	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	38.811694	38.822567	0.010873	Runnable_Swc06_Period_Expl_DataDIN3_Out	38.813726	38.818113	0.004387	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	38.816203	38.818003	0.000343
	osIdleLoop	39.140009	39.726308	0.586299								
20	Task_Event_Expl_Swc4to3func	39.764209	39.777939	0.013730	Runnable_Swc04_Event_Expl_DataStruct_Out	39.766421	39.771994	0.005573	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	39.769334	39.771874	0.000998
	Task_Event_Impl_Swc3to4func	39.783051	39.796847	0.013796	Runnable_Swc03_Event_Impl_DataStruct_Out	39.785363	39.787680	0.002317	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	39.801976	39.813088	0.011112	Runnable_Swc06_Period_Expl_DataDIN3_Out	39.804210	39.808616	0.004406	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	39.806939	39.808505	0.000317
	Task_IoHw_Period_Impl_Swc5to6func	39.817631	39.829524	0.011893	Runnable_Swc05_Period_Impl_DataDIN2_Out	39.819463	39.821823	0.002360	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1func	39.834285	39.840112	0.005827	Runnable_Swc02_Period_Expl_DataStruct_Out	39.835860	39.837900	0.002040	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	39.835948	39.837802	0.000709
	Task_Period_Impl_Swc1to2func	39.845087	39.852753	0.007666	Runnable_Swc01_Period_Impl_DataStruct_Out	39.846656	39.847050	0.000394	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	40.155236	40.726356	0.571120								
21	Task_Event_Expl_Swc4to3func	40.756832	40.771073	0.014241	Runnable_Swc04_Event_Expl_DataStruct_Out	40.759528	40.764980	0.005452	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	40.762416	40.764865	0.000946
	Task_Event_Impl_Swc3to4func	40.776372	40.791099	0.014727	Runnable_Swc03_Event_Impl_DataStruct_Out	40.778471	40.781327	0.002856	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	40.796036	40.807129	0.011093	Runnable_Swc06_Period_Expl_DataDIN3_Out	40.798142	40.802624	0.004482	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	40.800754	40.802514	0.000352
	Task_IoHw_Period_Impl_Swc5to6func	40.811911	40.824102	0.012191	Runnable_Swc05_Period_Impl_DataDIN2_Out	40.813824	40.816409	0.002585	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	41.234945	41.726475	0.491530								
22	Task_Event_Impl_Swc3to4func	41.757046	41.774936	0.017890	Runnable_Swc03_Event_Impl_DataStruct_Out	41.760240	41.763632	0.003392	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	41.779687	41.792937	0.013250	Runnable_Swc04_Event_Expl_DataStruct_Out	41.781886	41.787286	0.005400	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	41.784795	41.787174	0.000984
	Task_IoHw_Period_Impl_Swc5to6func	41.797936	41.809826	0.011890	Runnable_Swc05_Period_Impl_DataDIN2_Out	41.799790	41.802518	0.002728	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	41.814472	41.824830	0.010358	Runnable_Swc06_Period_Expl_DataDIN3_Out	41.816190	41.820308	0.004118	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	41.818451	41.820187	0.000352
	osIdleLoop	42.242609	42.720237	0.477628								
23	Task_Event_Expl_Swc4to3func	42.764288	42.779125	0.014837	Runnable_Swc04_Event_Expl_DataStruct_Out	42.767076	42.772927	0.005851	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	42.770009	42.772818	0.001109
	Task_Event_Impl_Swc3to4func	42.784304	42.807522	0.023218	Runnable_Swc03_Event_Impl_DataStruct_Out	42.786049	42.789275	0.003226	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Impl_Swc5to6func	42.813091	42.826569	0.013478	Runnable_Swc05_Period_Impl_DataDIN2_Out	42.815083	42.817828	0.002745	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	42.831091	42.842252	0.011161	Runnable_Swc06_Period_Expl_DataDIN3_Out	42.832972	42.837694	0.004722	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	42.835566	42.837585	0.000368
	osIdleLoop	43.205981	43.726629	0.520648								
24	Task_Event_Expl_Swc4to3func	43.757018	43.771831	0.014813	Runnable_Swc04_Event_Expl_DataStruct_Out	43.759858	43.765534	0.005676	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	43.762771	43.765420	0.001042
	Task_Event_Impl_Swc3to4func	43.777201	43.792394	0.015193	Runnable_Swc03_Event_Impl_DataStruct_Out	43.779403	43.782746	0.003343	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	43.797584	43.808708	0.011124	Runnable_Swc06_Period_Expl_DataDIN3_Out	43.799747	43.804432	0.004685	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	43.802484	43.804323	0.000342
	Task_IoHw_Period_Impl_Swc5to6func	43.813462	43.825212	0.011750	Runnable_Swc05_Period_Impl_DataDIN2_Out	43.815253	43.817565	0.002312	*Macros no functions*	0.000000	0.000000	0.000000
	osIdleLoop	44.136563	44.726738	0.590175								
25	Task_Event_Impl_Swc3to4func	44.763745	44.780785	0.017040	Runnable_Swc03_Event_Impl_DataStruct_Out	44.766394	44.769724	0.003330	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3func	44.785869	44.798166	0.012297	Runnable_Swc04_Event_Expl_DataStruct_Out	44.788158	44.792695	0.004537	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	44.790737	44.792584	0.000706
	Task_IoHw_Period_Impl_Swc5to6func	44.803159	44.815360	0.012201	Runnable_Swc05_Period_Impl_DataDIN2_Out	44.805212	44.807706	0.002494	*Macros no functions*	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5func	44.820027	44.830315	0.010288	Runnable_Swc06_Period_Expl_DataDIN3_Out	44.821876	44.825956	0.004080	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	44.824449	44.825850	0.000267
	Task_Period_Impl_Swc1to2func	44.834977	44.843551	0.008574	Runnable_Swc01_Period_Impl_DataStruct_Out	44.836648	44.837160	0.000512	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1func	44.848323	44.853840	0.005517	Runnable_Swc02_Period_Expl_DataStruct_Out	44.849938	44.851801	0.001863	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	44.850013	44.851709	0.000624
	osIdleLoop	45.161690	45.726810	0.565120								

11.6.11. V1_0_1111 notification process 19 to 25 ms

	SetEvent Entry in ms	End in ms	Duration in ms	OsSchedule Entry in ms	End in ms	Duration in ms	osGetEvent Entry in ms	End in ms	Duration in ms	OsWait Entry in ms	End in ms	Duration in ms	OsClearEvent Entry in ms	End in ms	Duration in ms	Notification sum in ms
19	38.762917	38.764507	0.001590	38.764542	38.767563	0.003021	38.769905	38.770574	0.000669	38.767778	38.769833	0.002055	38.770634	38.771842	0.001208	0.008543
	38.783509	38.784614	0.001105	38.784860	38.787151	0.002291	38.787225	38.787907	0.000682	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004078
	38.799237	38.800492	0.001255	38.800511	38.802785	0.002274	38.804594	38.805190	0.000596	38.802915	38.804530	0.001615	38.805243	38.806359	0.001116	0.006856
	38.816430	38.817887	0.001457	38.818136	38.820469	0.002333	38.820540	38.821192	0.000652	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004442
20	39.770208	39.771750	0.001542	39.772024	39.774953	0.002929	39.775032	39.775762	0.000730	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005201
	39.788305	39.789697	0.001392	39.789730	39.792238	0.002508	39.793924	39.794537	0.000613	39.792357	39.793858	0.001501	39.794594	39.795767	0.001173	0.007187
	39.807138	39.808387	0.001249	39.808633	39.810677	0.002044	39.810746	39.811386	0.000640	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.003933
	39.821942	39.823417	0.001475	39.823435	39.825677	0.002242	39.827511	39.828035	0.000524	39.825813	39.827455	0.001642	39.828087	39.829191	0.001104	0.006987
	39.836533	39.837678	0.001145	0.000000	0.000000	0.000000	39.837930	39.838213	0.000283	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001428
	39.847581	39.848757	0.001176	0.000000	0.000000	0.000000	39.850401	39.850733	0.000332	39.848883	39.850366	0.001483	39.850778	39.851802	0.001024	0.004015
21	40.763238	40.764741	0.001503	40.765007	40.768079	0.003072	40.768153	40.768837	0.000684	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005259
	40.782013	40.783296	0.001283	40.783320	40.786250	0.002930	40.788025	40.788647	0.000622	40.786391	40.787958	0.001567	40.788709	40.789941	0.001232	0.007634
	40.800988	40.802396	0.001408	40.802640	40.804847	0.002207	40.804908	40.805471	0.000563	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004178
	40.816503	40.817774	0.001271	40.817802	40.820098	0.002296	40.822062	40.822742	0.000680	40.820251	40.821988	0.001737	40.822786	40.823805	0.001019	0.007003
22	41.764508	41.766115	0.001607	41.766149	41.769362	0.003213	41.771698	41.772320	0.000622	41.769576	41.771631	0.002055	41.772382	41.773579	0.001197	0.008694
	41.785655	41.787050	0.001395	41.787309	41.790224	0.002915	41.790298	41.790980	0.000682	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004992
	41.802610	41.803866	0.001256	41.803887	41.806186	0.002299	41.807921	41.808452	0.000531	41.806322	41.807864	0.001542	41.808500	41.809567	0.001067	0.006695
	41.818692	41.820076	0.001384	41.820334	41.822604	0.002270	41.822674	41.823314	0.000640	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004294
23	42.770996	42.772696	0.001700	42.772954	42.776079	0.003125	42.776153	42.776836	0.000683	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005508
	42.790076	42.791528	0.001452	42.791566	42.794464	0.002898	42.804387	42.805059	0.000672	42.794685	42.804314	0.009629	42.805128	42.806448	0.001320	0.015971
	42.817925	42.819419	0.001494	42.819447	42.822215	0.002768	42.824143	42.824832	0.000689	42.822362	42.824070	0.001708	42.824903	42.826227	0.001324	0.007983
	42.835818	42.837469	0.001651	42.837716	42.839990	0.002274	42.840069	42.840795	0.000726	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004651
24	43.763696	43.765303	0.001607	43.765559	43.768615	0.003056	43.768690	43.769383	0.000693	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.005356
	43.783482	43.784762	0.001280	43.784798	43.787436	0.002638	43.789377	43.790047	0.000670	43.787601	43.789305	0.001704	43.790109	43.791339	0.001230	0.007522
	43.802708	43.804205	0.001497	43.804447	43.806523	0.002076	43.806592	43.807234	0.000642	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004215
	43.817665	43.818886	0.001221	43.818909	43.821234	0.002325	43.822881	43.823577	0.000696	43.821334	43.822806	0.001472	43.823646	43.824949	0.001303	0.007017
25	44.770660	44.772353	0.001693	44.772388	44.775405	0.003017	44.777624	44.778237	0.000613	44.775598	44.777558	0.001960	44.778293	44.779434	0.001141	0.008424
	44.791318	44.792459	0.001141	44.792718	44.795117	0.002399	44.795196	44.795928	0.000732	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004272
	44.807784	44.809036	0.001252	44.809064	44.811478	0.002414	44.813278	44.813891	0.000613	44.811624	44.813212	0.001588	44.813948	44.815087	0.001139	0.007006
	44.824598	44.825732	0.001134	44.825967	44.828095	0.002128	44.828165	44.828805	0.000640	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.003902
	44.837736	44.838967	0.001231	0.000000	0.000000	0.000000	44.840740	44.841087	0.000347	44.839097	44.840703	0.001606	44.841133	44.842144	0.001011	0.004195
	44.850515	44.851587	0.001072	0.000000	0.000000	0.000000	44.851834	44.852136	0.000302	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001374

11.6.12. V1_0_1111 read-process 19 to 25 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
19	Runnable_Swc04_Odr_Impl_DataStruct_In	38.772502	38.773048	0.000546	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008543
	Runnable_Swc03_Odr_Expl_DataStruct_In	38.787990	38.788805	0.000815	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	38.788090	38.788702	0.000612	0.001293	0.005371
	Runnable_Swc06_Odr_Impl_DataDIN2_In	38.806748	38.806861	0.000113	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006856
	Runnable_Swc05_Odr_Expl_DataDIN3_In	38.821247	38.821478	0.000231	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	38.821301	38.821374	0.000073	0.000416	0.004858
20	Runnable_Swc03_Odr_Expl_DataStruct_In	39.775845	39.776663	0.000818	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	39.775945	39.776556	0.000611	0.001609	0.006810
	Runnable_Swc04_Odr_Impl_DataStruct_In	39.796317	39.796777	0.000460	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007187
	Runnable_Swc05_Odr_Expl_DataDIN3_In	39.811498	39.811837	0.000339	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	39.811610	39.811729	0.000119	0.000436	0.004369
	Runnable_Swc06_Odr_Impl_DataDIN2_In	39.829416	39.829485	0.000069	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006987
	Runnable_Swc01_Odr_Expl_DataStruct_In	39.838283	39.838988	0.000705	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	39.838367	39.838884	0.000517	0.001226	0.002654
	Runnable_Swc02_Odr_Impl_DataStruct_In	39.852297	39.852706	0.000409	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.004015
21	Runnable_Swc03_Odr_Expl_DataStruct_In	40.768935	40.769870	0.000935	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	40.769052	40.769765	0.000713	0.001659	0.006918
	Runnable_Swc04_Odr_Impl_DataStruct_In	40.790546	40.791045	0.000499	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007634
	Runnable_Swc05_Odr_Expl_DataDIN3_In	40.805556	40.805845	0.000289	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	40.805641	40.805738	0.000097	0.000449	0.004627
	Runnable_Swc06_Odr_Impl_DataDIN2_In	40.823982	40.824045	0.000063	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007003
22	Runnable_Swc04_Odr_Impl_DataStruct_In	41.774294	41.774881	0.000587	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008694
	Runnable_Swc03_Odr_Expl_DataStruct_In	41.791063	41.791877	0.000814	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	41.791163	41.791774	0.000611	0.001595	0.006587
	Runnable_Swc06_Odr_Impl_DataDIN2_In	41.809724	41.809779	0.000055	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.006695
	Runnable_Swc05_Odr_Expl_DataDIN3_In	41.823387	41.823652	0.000265	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	41.823459	41.823547	0.000088	0.000440	0.004734
23	Runnable_Swc03_Odr_Expl_DataStruct_In	42.776929	42.777828	0.000899	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	42.777040	42.777720	0.000680	0.001789	0.007297
	Runnable_Swc04_Odr_Impl_DataStruct_In	42.806993	42.807450	0.000457	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.015971
	Runnable_Swc06_Odr_Impl_DataDIN2_In	42.826452	42.826524	0.000072	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007983
	Runnable_Swc05_Odr_Expl_DataDIN3_In	42.840861	42.841105	0.000244	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	42.840928	42.841009	0.000081	0.000449	0.005100
24	Runnable_Swc03_Odr_Expl_DataStruct_In	43.769480	43.770412	0.000932	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	43.769598	43.770309	0.000711	0.001753	0.007109
	Runnable_Swc04_Odr_Impl_DataStruct_In	43.791874	43.792323	0.000449	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007522
	Runnable_Swc05_Odr_Expl_DataDIN3_In	43.807316	43.807596	0.000280	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	43.807398	43.807493	0.000095	0.000437	0.004652
	Runnable_Swc06_Odr_Impl_DataDIN2_In	43.825113	43.825169	0.000056	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007017
25	Runnable_Swc04_Odr_Impl_DataStruct_In	44.780134	44.780714	0.000580	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.008424
	Runnable_Swc03_Odr_Expl_DataStruct_In	44.796025	44.796953	0.000928	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	44.796141	44.796847	0.000706	0.001412	0.005684
	Runnable_Swc06_Odr_Impl_DataDIN2_In	44.815264	44.815322	0.000058	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.007006
	Runnable_Swc05_Odr_Expl_DataDIN3_In	44.828878	44.829143	0.000265	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	44.828950	44.829038	0.000088	0.000355	0.004257
	Runnable_Swc02_Odr_Impl_DataStruct_In	44.842894	44.843506	0.000612	*Macros no functions*	0.000000	0.000000	0.000000	0.000000	0.004195
	Runnable_Swc01_Odr_Expl_DataStruct_In	44.852184	44.852706	0.000522	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	44.852240	44.852602	0.000362	0.000986	0.002360
							Communication with notification su			0.815256
							Communication without notification			0.059585
							OsIdleLoop sum:			13.897184

11.7. Performance analyses of SingleCoreV2_0 V1_0_1111

11.7.1. V2_0_1111 write-process 0 to 3 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
0	Task_Event_Expl_Swc4to3_Outfunc	19.015534	19.022523	0.006989	Runnable_Swc04_Event_Expl_DataStruct_Out	19.015671	19.022497	0.006826	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	19.018663	19.022384	0.001136
	Task_Event_Impl_Swc3to4_Outfunc	19.028638	19.033610	0.004972	Runnable_Swc03_Event_Impl_DataStruct_Out	19.028719	19.030820	0.002101	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	19.038633	19.039480	0.000847								
	Task_Event_Impl_Swc3to4_Infunc	19.044113	19.045184	0.001071								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	19.050461	19.055198	0.004737	Runnable_Swc05_Period_Impl_DataDIN2_Out	19.050564	19.052780	0.002216	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	19.060152	19.064431	0.004279	Runnable_Swc06_Period_Expl_DataDIN3_Out	19.060218	19.064420	0.004202	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Peri	19.061961	19.064314	0.000314
	Task_IoHw_Period_Impl_Swc5to6_Infunc	19.069477	19.069743	0.000266								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	19.074525	19.075149	0.000624								
	Task_Period_Impl_Swc1to2_Outfunc	19.087443	19.090264	0.002821	Runnable_Swc01_Period_Impl_DataStruct_Out	19.087654	19.087672	0.000018	Macros no functions	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1_Outfunc	19.095108	19.098158	0.003050	Runnable_Swc02_Period_Expl_DataStruct_Out	19.095180	19.098142	0.002962	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	19.095287	19.098038	0.000900
	Task_Period_Impl_Swc1to2_Infunc	19.102803	19.103788	0.000985								
	Task_Period_Expl_Swc2to1_Infunc	19.108581	19.109536	0.000955								
	osIdleLoop	19.119381	19.735301	0.615920								
1	Task_Event_Impl_Swc3to4_Outfunc	19.769866	19.775971	0.006105	Runnable_Swc03_Event_Impl_DataStruct_Out	19.769986	19.772676	0.002690	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	19.781873	19.787027	0.005154	Runnable_Swc04_Event_Expl_DataStruct_Out	19.781948	19.787013	0.005065	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	19.783960	19.786911	0.000809
	Task_Event_Impl_Swc3to4_Infunc	19.791931	19.792997	0.001066								
	Task_Event_Expl_Swc4to3_Infunc	19.798015	19.798861	0.000846								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	19.803829	19.808317	0.004488	Runnable_Swc05_Period_Impl_DataDIN2_Out	19.803909	19.806033	0.002124	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	19.813423	19.817578	0.004155	Runnable_Swc06_Period_Expl_DataDIN3_Out	19.813482	19.817566	0.004084	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	19.815357	19.817459	0.000273
	Task_IoHw_Period_Impl_Swc5to6_Infunc	19.822722	19.823078	0.000356								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	19.828156	19.828679	0.000523								
	osIdleLoop	20.710336	20.735411	0.025075								
2	Task_Event_Impl_Swc3to4_Outfunc	20.789103	20.795517	0.006414	Runnable_Swc03_Event_Impl_DataStruct_Out	20.789226	20.791890	0.002664	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	20.801592	20.806882	0.005290	Runnable_Swc04_Event_Expl_DataStruct_Out	20.801671	20.806863	0.005192	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	20.803589	20.806756	0.000990
	Task_Event_Impl_Swc3to4_Infunc	20.812004	20.813075	0.001071								
	Task_Event_Expl_Swc4to3_Infunc	20.818015	20.818866	0.000851								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	20.824087	20.828671	0.004584	Runnable_Swc06_Period_Expl_DataDIN3_Out	20.824185	20.828658	0.004473	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	20.826458	20.828550	0.000286
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	20.833428	20.837889	0.004461	Runnable_Swc05_Period_Impl_DataDIN2_Out	20.833508	20.835687	0.002179	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	20.843162	20.843781	0.000619								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	20.848723	20.849003	0.000280								
	osIdleLoop	21.043445	21.735494	0.692049								
3	Task_Event_Expl_Swc4to3_Outfunc	21.768452	21.775629	0.007177	Runnable_Swc04_Event_Expl_DataStruct_Out	21.768589	21.775605	0.007016	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	21.771600	21.775493	0.001240
	Task_Event_Impl_Swc3to4_Outfunc	21.781619	21.786762	0.005143	Runnable_Swc03_Event_Impl_DataStruct_Out	21.781698	21.783947	0.002249	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	21.791879	21.792738	0.000859								
	Task_Event_Impl_Swc3to4_Infunc	21.797776	21.798741	0.000965								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	21.804243	21.808894	0.004651	Runnable_Swc05_Period_Impl_DataDIN2_Out	21.804346	21.806563	0.002217	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	21.813457	21.818047	0.004590	Runnable_Swc06_Period_Expl_DataDIN3_Out	21.813543	21.818032	0.004489	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	21.815634	21.817923	0.000291
	Task_IoHw_Period_Impl_Swc5to6_Infunc	21.823060	21.823517	0.000457								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	21.828488	21.829080	0.000592								
	osIdleLoop	21.941562	22.735567	0.794005								

11.7.2. V2_0_1111 notification process 0 to 3 ms

	osActivate Task Entry in ms	End in ms	Duration in ms	osCheckInterruptEnabled Entry in ms	End in ms	Duration in ms	osSchedule Prio Entry in ms	End in ms	Duration in ms	osDispatecher Entry in ms	End in ms	Duration	Notification Duration in ms (without ActivateTask)		Duration notification
0	19.019674	19.022259	0.002585											Task_Event_Expl_Swc4to3_Outfunc	
	19.031566	19.033591	0.002025	19.023028	19.023325	0.000297	19.024524	19.025367	0.000843	19.026473	19.028427	0.001954	0.003094	Task_Event_Impl_Swc3to4_Outfunc	
				19.033987	19.034261	0.000274	19.035310	19.036099	0.000789	19.036904	19.038427	0.001523	0.002586	Task_Event_Expl_Swc4to3_Infunc	0.013237
				19.039736	19.039972	0.000236	19.040899	19.041625	0.000726	19.042505	19.043945	0.001440	0.002402	Task_Event_Impl_Swc3to4_Infunc	0.007860
	19.052912	19.055176	0.002264	19.045669	19.046036	0.000367	19.047113	19.047836	0.000723	19.048678	19.050277	0.001599	0.002689	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	19.062157	19.064196	0.002039	19.055643	19.055926	0.000283	19.056838	19.057532	0.000694	19.058409	19.059945	0.001536	0.002513	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				19.064660	19.064890	0.000230	19.065995	19.066832	0.000837	19.067713	19.069263	0.001550	0.002617	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011673
				19.070086	19.070354	0.000268	19.071342	19.072172	0.000830	19.072925	19.074299	0.001374	0.002472	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007394
	19.088331	19.090245	0.001914	19.081841	19.082151	0.000310	19.083619	19.084509	0.000890	19.085420	19.087199	0.001779	0.002979	Task_Period_Impl_Swc1to2_Outfunc	
	19.095983	19.097834	0.001851	19.090634	19.090879	0.000245	19.091997	19.092812	0.000815	19.093606	19.094931	0.001325	0.002385	Task_Period_Expl_Swc2to1_Outfunc	
				19.098468	19.098709	0.000241	19.099599	19.100285	0.000686	19.101219	19.102631	0.001412	0.002339	Task_Period_Impl_Swc1to2_Infunc	0.009688
				19.104124	19.104472	0.000348	19.105460	19.106353	0.000893	19.107159	19.108404	0.001245	0.002486	Task_Period_Expl_Swc2to1_Infunc	0.007661
1	19.773487	19.775944	0.002457											Task_Event_Impl_Swc3to4_Outfunc	
	19.784652	19.786794	0.002142	19.776503	19.776807	0.000304	19.778013	19.778879	0.000866	19.779894	19.781695	0.001801	0.002971	Task_Event_Expl_Swc4to3_Outfunc	
				19.787303	19.787536	0.000233	19.788427	19.789106	0.000679	19.790042	19.791727	0.001685	0.002597	Task_Event_Impl_Swc3to4_Infunc	0.013179
				19.793428	19.793819	0.000391	19.794779	19.795608	0.000829	19.796387	19.797836	0.001449	0.002669	Task_Event_Expl_Swc4to3_Infunc	0.008474
	19.806156	19.808294	0.002138	19.799110	19.799326	0.000216	19.800426	19.801254	0.000828	19.802063	19.803609	0.001546	0.002590	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	19.815512	19.817341	0.001829	19.808761	19.809036	0.000275	19.810029	19.810826	0.000797	19.811639	19.813213	0.001574	0.002646	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				19.817813	19.818044	0.000231	19.819046	19.819046	0.000000	19.820802	19.822499	0.001697	0.001928	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.010867
				19.823503	19.823891	0.000388	19.824835	19.825562	0.000727	19.826463	19.827968	0.001505	0.002620	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.006733
2	20.792895	20.795490	0.002595											Task_Event_Impl_Swc3to4_Outfunc	
	20.804390	20.806567	0.002177	20.796056	20.796351	0.000295	20.797505	20.798398	0.000893	20.799437	20.801372	0.001935	0.003123	Task_Event_Expl_Swc4to3_Outfunc	
				20.807252	20.807499	0.000247	20.808386	20.809123	0.000737	20.810142	20.811799	0.001657	0.002641	Task_Event_Impl_Swc3to4_Infunc	0.013649
				20.813560	20.813945	0.000385	20.814873	20.815599	0.000726	20.816378	20.817831	0.001453	0.002564	Task_Event_Expl_Swc4to3_Infunc	0.008453
	20.826626	20.828432	0.001806	20.819264	20.819627	0.000363	20.820674	20.821399	0.000725	20.822235	20.823872	0.001637	0.002725	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	20.835804	20.837867	0.002063	20.828934	20.829163	0.000229	20.830188	20.830881	0.000693	20.831695	20.833245	0.001550	0.002472	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				20.838334	20.838681	0.000347	20.839795	20.840624	0.000829	20.841492	20.842972	0.001480	0.002656	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011395
				20.844138	20.844499	0.000361	20.845456	20.846181	0.000725	20.847002	20.848540	0.001538	0.002624	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007962
3	21.772716	21.775369	0.002653											Task_Event_Expl_Swc4to3_Outfunc	
	21.784703	21.786744	0.002041	21.776107	21.776388	0.000281	21.777707	21.778571	0.000864	21.779576	21.781409	0.001833	0.002978	Task_Event_Impl_Swc3to4_Outfunc	
				21.787112	21.787352	0.000240	21.788336	21.789021	0.000685	21.789953	21.791695	0.001742	0.002667	Task_Event_Expl_Swc4to3_Infunc	0.013441
				21.793122	21.793388	0.000266	21.794460	21.795188	0.000728	21.796072	21.797590	0.001518	0.002512	Task_Event_Impl_Swc3to4_Infunc	0.008079
	21.806697	21.808879	0.002182	21.799159	21.799442	0.000283	21.800620	21.801353	0.000733	21.802241	21.804027	0.001786	0.002802	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	21.815807	21.817805	0.001998	21.809204	21.809453	0.000249	21.810321	21.810981	0.000660	21.811776	21.813259	0.001483	0.002392	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				21.818344	21.818572	0.000228	21.819551	21.820244	0.000693	21.821183	21.822840	0.001657	0.002578	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011742
				21.823874	21.824153	0.000279	21.825183	21.825979	0.000796	21.826834	21.828263	0.001429	0.002504	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007537

11.7.3. V2_0_1111 read-process 0 to 3 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
0									0.001712	0.014949
									0.000000	0.007860
	Runnable_Swc03_Odr_Expl_DataStruct_In	19.038696	19.039467	0.000771	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	19.038789	19.039365	0.000576		
	Runnable_Swc04_Odr_Impl_DataStruct_In	19.044652	19.045160	0.000508	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.011673
									0.000481	0.007875
	Runnable_Swc06_Odr_Impl_DataDIN2_In	19.069639	19.069726	0.000087	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	19.074662	19.075120	0.000458	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	19.074835	19.075002	0.000167		
									0.000000	0.009688
									0.001559	0.009220
	Runnable_Swc02_Odr_Impl_DataStruct_In	19.103304	19.103771	0.000467	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc01_Odr_Expl_DataStruct_In	19.108653	19.109522	0.000869	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	19.108761	19.109420	0.000659		
1									0.000000	0.013179
									0.001385	0.009859
	Runnable_Swc04_Odr_Impl_DataStruct_In	19.792470	19.792976	0.000506	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	19.798078	19.798848	0.000770	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	19.798171	19.798747	0.000576		
									0.000000	0.010867
									0.000419	0.007152
	Runnable_Swc06_Odr_Impl_DataDIN2_In	19.822942	19.823057	0.000115	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	19.828273	19.828666	0.000393	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	19.828419	19.828565	0.000146		
2									0.000000	0.013649
									0.001559	0.010012
	Runnable_Swc04_Odr_Impl_DataStruct_In	20.812543	20.813051	0.000508	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	20.818077	20.818847	0.000770	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	20.818169	20.818738	0.000569		
									0.000458	0.011853
									0.000000	0.007962
	Runnable_Swc05_Odr_Expl_DataDIN3_In	20.843305	20.843763	0.000458	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	20.843484	20.843656	0.000172		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	20.848891	20.848983	0.000092	*Macros no functions*	0.000000	0.000000	0.000000		
3									0.001816	0.015257
									0.000000	0.008079
	Runnable_Swc03_Odr_Expl_DataStruct_In	21.791941	21.792719	0.000778	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	21.792035	21.792611	0.000576		
	Runnable_Swc04_Odr_Impl_DataStruct_In	21.798262	21.798720	0.000458	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.011742
									0.000458	0.007995
	Runnable_Swc06_Odr_Impl_DataDIN2_In	21.823355	21.823499	0.000144	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	21.828626	21.829067	0.000441	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	21.828798	21.828965	0.000167		

11.7.4. V2_0_1111 write-process 4 to 7 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
4	Task_Event_Expl_Swc4to3_Outfunc	22.769752	22.776619	0.006867	Runnable_Swc04_Event_Expl_DataStruct_Out	22.769889	22.776595	0.006706	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	22.772872	22.776484	0.001117
	Task_Event_Impl_Swc3to4_Outfunc	22.782137	22.786867	0.004730	Runnable_Swc03_Event_Impl_DataStruct_Out	22.782216	22.784335	0.002119	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	22.792270	22.793143	0.000873								
	Task_Event_Impl_Swc3to4_Infunc	22.798148	22.799117	0.000969								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	22.804142	22.808698	0.004556	Runnable_Swc05_Period_Impl_DataDIN2_Out	22.804238	22.806477	0.002239	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	22.813954	22.818156	0.004202	Runnable_Swc06_Period_Expl_DataDIN3_Out	22.814024	22.818142	0.004118	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	22.815938	22.818032	0.000261
	Task_IoHw_Period_Impl_Swc5to6_Infunc	22.823268	22.823626	0.000358								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	22.828601	22.829119	0.000518								
	osIdleLoop	22.916945	23.735676	0.818731								
5	Task_Event_Impl_Swc3to4_Outfunc	23.778772	23.784922	0.006150	Runnable_Swc03_Event_Impl_DataStruct_Out	23.778881	23.781397	0.002516	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	23.790848	23.796275	0.005427	Runnable_Swc04_Event_Expl_DataStruct_Out	23.790933	23.796260	0.005327	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Ex	23.792871	23.796156	0.000954
	Task_Event_Impl_Swc3to4_Infunc	23.801103	23.802108	0.001005								
	Task_Event_Expl_Swc4to3_Infunc	23.813261	23.814135	0.000874								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	23.820110	23.825894	0.005784	Runnable_Swc06_Period_Expl_DataDIN3_Out	23.820224	23.825879	0.005655	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	23.822878	23.825769	0.000395
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	23.831865	23.836355	0.004490	Runnable_Swc05_Period_Impl_DataDIN2_Out	23.831947	23.834168		Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	23.841346	23.841856	0.000510								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	23.846968	23.847232	0.000264								0.000000
	Task_Period_Impl_Swc1to2_Outfunc	23.852227	23.854946	0.002719	Runnable_Swc01_Period_Impl_DataStruct_Out	23.852481	23.852499	0.000018	Macros no functions	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1_Outfunc	23.859635	23.862587	0.002952	Runnable_Swc02_Period_Expl_DataStruct_Out	23.859706	23.862566	0.002860	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	23.859812	23.862463	0.000810
	Task_Period_Impl_Swc1to2_Infunc	23.867467	23.868429	0.000962								
	Task_Period_Expl_Swc2to1_Infunc	23.873156	23.874212	0.001056								
	osIdleLoop	23.935244	24.735757	0.800513								
6	Task_Event_Impl_Swc3to4_Outfunc	24.769471	24.775798	0.006327	Runnable_Swc03_Event_Impl_DataStruct_Out	24.769577	24.772109	0.002532	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	24.781745	24.786946	0.005201	Runnable_Swc04_Event_Expl_DataStruct_Out	24.781819	24.786932	0.005113	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	24.783766	24.786829	0.000851
	Task_Event_Impl_Swc3to4_Infunc	24.791994	24.793006	0.001012								
	Task_Event_Expl_Swc4to3_Infunc	24.797896	24.798707	0.000811								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	24.803706	24.808420	0.004714	Runnable_Swc06_Period_Expl_DataDIN3_Out	24.803804	24.808410	0.004606	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	24.806063	24.808305	0.000282
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	24.813403	24.817751	0.004348	Runnable_Swc05_Period_Impl_DataDIN2_Out	24.813489	24.815656	0.002167	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	24.822372	24.822984	0.000612								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	24.827724	24.828000	0.000276								
	osIdleLoop	24.891845	25.735839	0.843994								
7	Task_Event_Expl_Swc4to3_Outfunc	25.768947	25.775778	0.006831	Runnable_Swc04_Event_Expl_DataStruct_Out	25.769066	25.775757	0.006691	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	25.771864	25.775647	0.001192
	Task_Event_Impl_Swc3to4_Outfunc	25.781193	25.786200	0.005007	Runnable_Swc03_Event_Impl_DataStruct_Out	25.781274	25.783474	0.002200	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	25.791606	25.792472	0.000866								
	Task_Event_Impl_Swc3to4_Infunc	25.797850	25.798934	0.001084								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	25.804498	25.809132	0.004634	Runnable_Swc05_Period_Impl_DataDIN2_Out	25.804600	25.806816	0.002216	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	25.814607	25.818837	0.004230	Runnable_Swc06_Period_Expl_DataDIN3_Out	25.814675	25.818823	0.004148	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	25.816473	25.818714	0.000295
	Task_IoHw_Period_Impl_Swc5to6_Infunc	25.823849	25.824106	0.000257								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	25.828926	25.829544	0.000618								
	osIdleLoop	25.881818	26.735946	0.854128								

11.7.5. V2_0_1111 notification process 4 to 7 ms

	osActivate Task Entry in ms	End in ms	Duration in ms	osCheckInterruptEnabled Entry in ms	End in ms	Duration in ms	osSchedule Prio Entry in ms	End in ms	Duration in ms	osDispatecher Entry in ms	End in ms	Duration	Notification Duration in ms (without ActivateTask)		Duration notification
4	22.773835	22.776330	0.002495											Task_Event_Expl_Swc4to3_Outfunc	
	22.784961	22.786851	0.001890	22.777083	22.777379	0.000296	22.778533	22.779260	0.000727	22.780223	22.781954	0.001731	0.002754	Task_Event_Impl_Swc3to4_Outfunc	
				22.787177	22.787425	0.000248	22.788557	22.789379	0.000822	22.790348	22.792095	0.001747	0.002817	Task_Event_Expl_Swc4to3_Infunc	0.012796
				22.793486	22.793754	0.000268	22.794742	22.795580	0.000838	22.796462	22.797945	0.001483	0.002589	Task_Event_Impl_Swc3to4_Infunc	0.008169
	22.806591	22.808676	0.002085	22.799561	22.799826	0.000265	22.800900	22.801726	0.000826	22.802505	22.803954	0.001449	0.002540	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	22.816081	22.817914	0.001833	22.809143	22.809490	0.000347	22.810499	22.811315	0.000816	22.812195	22.813745	0.001550	0.002713	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				22.818453	22.818672	0.000219	22.819764	22.820580	0.000816	22.821488	22.823045	0.001557	0.002592	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011592
				22.823983	22.824255	0.000272	22.825342	22.826190	0.000848	22.826959	22.828381	0.001422	0.002542	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007325
5	23.782328	23.784897	0.002569											Task_Event_Impl_Swc3to4_Outfunc	
	23.793672	23.796003	0.002331	23.785420	23.785716	0.000296	23.786915	23.787753	0.000838	23.788763	23.790636	0.001873	0.003007	Task_Event_Expl_Swc4to3_Outfunc	
				23.796571	23.796799	0.000228	23.797733	23.798539	0.000806	23.799478	23.800927	0.001449	0.002483	Task_Event_Impl_Swc3to4_Infunc	0.013486
				23.802552	23.802809	0.000257	23.803781	23.804609	0.000828	23.805370	23.806736	0.001366	0.002451	Task_Event_Expl_Swc4to3_Infunc	0.008270
	23.823154	23.825650	0.002496	23.814666	23.814963	0.000297	23.816148	23.816999	0.000851	23.817934	23.819872	0.001938	0.003086	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	23.834285	23.836337	0.002052	23.826204	23.826434	0.000230	23.827812	23.828671	0.000859	23.829705	23.831636	0.001931	0.003020	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				23.836732	23.836999	0.000267	23.837941	23.838725	0.000784	23.839622	23.841149	0.001527	0.002578	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.012584
			0.000000	23.842145	23.842506	0.000361	23.843578	23.844324	0.000746	23.845211	23.846754	0.001543	0.002650	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007790
	23.853149	23.854932	0.001783	23.847562	23.847899	0.000337	23.848916	23.849635	0.000719	23.850441	23.851972	0.001531	0.002587	Task_Period_Impl_Swc1to2_Outfunc	
	23.860500	23.862341	0.001841	23.855229	23.855463	0.000234	23.856374	23.857043	0.000669	23.857929	23.859427	0.001498	0.002401	Task_Period_Expl_Swc2to1_Outfunc	
				23.863012	23.863281	0.000269	23.864172	23.864867	0.000695	23.865811	23.867286	0.001475	0.002439	Task_Period_Impl_Swc1to2_Infunc	0.009575
				23.868725	23.868972	0.000247	23.869951	23.870788	0.000837	23.871620	23.872968	0.001348	0.002432	Task_Period_Expl_Swc2to1_Infunc	0.007674
6	24.773122	24.775772	0.002650											Task_Event_Impl_Swc3to4_Outfunc	
	24.784493	24.786705	0.002212	24.776330	24.776642	0.000312	24.777900	24.778843	0.000943	24.779804	24.781536	0.001732	0.002987	Task_Event_Expl_Swc4to3_Outfunc	
				24.787229	24.787463	0.000234	24.788533	24.789367	0.000834	24.790318	24.791822	0.001504	0.002572	Task_Event_Impl_Swc3to4_Infunc	0.013410
				24.793437	24.793735	0.000298	24.794647	24.795363	0.000716	24.796184	24.797722	0.001538	0.002552	Task_Event_Expl_Swc4to3_Infunc	0.008348
	24.806227	24.808187	0.001960	24.799145	24.799535	0.000390	24.800635	24.801463	0.000828	24.802204	24.803518	0.001314	0.002532	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	24.815756	24.817734	0.001978	24.808629	24.808853	0.000224	24.809921	24.810725	0.000804	24.811634	24.813190	0.001556	0.002584	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				24.818081	24.818335	0.000254	24.819210	24.819872	0.000662	24.820656	24.822145	0.001489	0.002405	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011297
				24.823280	24.823599	0.000319	24.824572	24.825399	0.000827	24.826164	24.827540	0.001376	0.002522	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007517
7	25.772932	25.775523	0.002591											Task_Event_Expl_Swc4to3_Outfunc	
	25.784225	25.786177	0.001952	25.776195	25.776461	0.000266	25.777616	25.778342	0.000726	25.779283	25.780981	0.001698	0.002690	Task_Event_Impl_Swc3to4_Outfunc	
				25.786658	25.786936	0.000278	25.787944	25.788742	0.000798	25.789700	25.791422	0.001722	0.002798	Task_Event_Expl_Swc4to3_Infunc	0.013086
				25.793018	25.793415	0.000397	25.794450	25.795172	0.000722	25.796031	25.797677	0.001646	0.002765	Task_Event_Impl_Swc3to4_Infunc	0.008381
	25.806946	25.809111	0.002165	25.799554	25.799961	0.000407	25.801147	25.801872	0.000725	25.802714	25.804313	0.001599	0.002731	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	25.816650	25.818596	0.001946	25.809550	25.809879	0.000329	25.810982	25.811751	0.000769	25.812705	25.814431	0.001726	0.002824	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				25.819120	25.819361	0.000241	25.820582	25.821298	0.000716	25.822190	25.823668	0.001478	0.002435	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011654
				25.824443	25.824744	0.000301	25.825709	25.826444	0.000735	25.827240	25.828731	0.001491		Task_IoHw_Period_Expl_Swc6to5_Infunc	0.004638

11.7.6. V2_0_1111 read-process 4 to 7 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
4									0.001707	0.014503
									0.000000	0.008169
	Runnable_Swc03_Odr_Expl_DataStruct_In	22.792334	22.793126	0.000792	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	22.792430	22.793020	0.000590		
	Runnable_Swc04_Odr_Impl_DataStruct_In	22.798635	22.799094	0.000459	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.011592
									0.000405	0.007730
	Runnable_Swc06_Odr_Impl_DataDIN2_In	22.823494	22.823608	0.000114	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	22.828715	22.829105	0.000390	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	22.828858	22.829002	0.000144		
5									0.000000	0.013486
									0.001530	0.009800
	Runnable_Swc04_Odr_Impl_DataStruct_In	23.801609	23.802085	0.000476	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	23.813323	23.814108	0.000785	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_E	23.813417	23.813993	0.000576		
									0.000537	0.013121
									0.000000	0.007790
	Runnable_Swc05_Odr_Expl_DataDIN3_In	23.841461	23.841841	0.000380	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	23.841603	23.841745	0.000142		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	23.847130	23.847216	0.000086	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.009575
									0.001545	0.009219
	Runnable_Swc02_Odr_Impl_DataStruct_In	23.867958	23.868414	0.000456	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc01_Odr_Expl_DataStruct_In	23.873237	23.874197	0.000960	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	23.873358	23.874093	0.000735		
6									0.000000	0.013410
									0.001390	0.009738
	Runnable_Swc04_Odr_Impl_DataStruct_In	24.792505	24.792985	0.000480	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	24.797954	24.798685	0.000731	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	24.798042	24.798581	0.000539		
									0.000454	0.011751
									0.000000	0.007517
	Runnable_Swc05_Odr_Expl_DataDIN3_In	24.822514	24.822969	0.000455	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	24.822693	24.822865	0.000172		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	24.827898	24.827986	0.000088	*Macros no functions*	0.000000	0.000000	0.000000		
7									0.001766	0.014852
									0.000000	0.008381
	Runnable_Swc03_Odr_Expl_DataStruct_In	25.791668	25.792445	0.000777	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	25.791762	25.792336	0.000574		
	Runnable_Swc04_Odr_Impl_DataStruct_In	25.798389	25.798903	0.000514	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.011654
									0.000467	0.005105
	Runnable_Swc06_Odr_Impl_DataDIN2_In	25.824005	25.824089	0.000084	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	25.829069	25.829526	0.000457	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	25.829248	25.829420	0.000172		

11.7.7. V2_0_1111 write-process 8 to 11 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
8	Task_Event_Expl_Swc4to3_Outfunc	26.768756	26.775173	0.006417	Runnable_Swc04_Event_Expl_DataStruct_Out	26.768875	26.775150	0.006275	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	26.771631	26.775038	0.001008
	Task_Event_Impl_Swc3to4_Outfunc	26.781127	26.785830	0.004703	Runnable_Swc03_Event_Impl_DataStruct_Out	26.781200	26.783110	0.001910	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	26.791161	26.792033	0.000872								
	Task_Event_Impl_Swc3to4_Infunc	26.796977	26.798040	0.001063								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	26.803080	26.807223	0.004143	Runnable_Swc05_Period_Impl_DataDIN2_Out	26.803149	26.805022	0.001873	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	26.812342	26.816670	0.004328	Runnable_Swc06_Period_Expl_DataDIN3_Out	26.812402	26.816657	0.004255	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	26.814337	26.816550	0.000294
	Task_IoHw_Period_Impl_Swc5to6_Infunc	26.821649	26.822006	0.000357								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	26.827074	26.827600	0.000526								
	osIdleLoop	26.888500	27.736029	0.847529								
9	Task_Event_Impl_Swc3to4_Outfunc	27.812897	27.818974	0.006077	Runnable_Swc03_Event_Impl_DataStruct_Out	27.812999	27.815425	0.002426	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	27.825195	27.830481	0.005286	Runnable_Swc04_Event_Expl_DataStruct_Out	27.825287	27.830468	0.005181	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	27.827519	27.830365	0.000943
	Task_Event_Impl_Swc3to4_Infunc	27.835468	27.836537	0.001069								
	Task_Event_Expl_Swc4to3_Infunc	27.841506	27.842354	0.000848								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	27.847439	27.851978	0.004539	Runnable_Swc06_Period_Expl_DataDIN3_Out	27.847525	27.851966	0.004441	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	27.849757	27.851859	0.000273
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	27.857192	27.861140	0.003948	Runnable_Swc05_Period_Impl_DataDIN2_Out	27.857272	27.859306	0.002034	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	27.865964	27.866482	0.000518								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	27.871522	27.871878	0.000356								
	osIdleLoop	27.929063	28.736103	0.807040								
10	Task_Event_Expl_Swc4to3_Outfunc	28.778443	28.785453	0.007010	Runnable_Swc04_Event_Expl_DataStruct_Out	28.778581	28.785431	0.006850	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	28.781584	28.785320	0.001206
	Task_Event_Impl_Swc3to4_Outfunc	28.791357	28.796123	0.004766	Runnable_Swc03_Event_Impl_DataStruct_Out	28.791439	28.793544	0.002105	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	28.801443	28.802302	0.000859								
	Task_Event_Impl_Swc3to4_Infunc	28.807285	28.808296	0.001011								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	28.813516	28.818135	0.004619	Runnable_Swc05_Period_Impl_DataDIN2_Out	28.813617	28.815912	0.002295	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	28.823210	28.827920	0.004710	Runnable_Swc06_Period_Expl_DataDIN3_Out	28.823287	28.827905	0.004618	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Peri	28.825407	28.827796	0.000316
	Task_IoHw_Period_Impl_Swc5to6_Infunc	28.832695	28.833043	0.000348								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	28.837919	28.838438	0.000519								
	Task_Period_Expl_Swc2to1_Outfunc	28.843401	28.846691	0.003290	Runnable_Swc02_Period_Expl_DataStruct_Out	28.843482	28.846668	0.003186	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	28.843602	28.846543	0.000950
	Task_Period_Impl_Swc1to2_Outfunc	28.851863	28.854612	0.002749	Runnable_Swc01_Period_Impl_DataStruct_Out	28.852081	28.852099	0.000018	Macros no functions	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1_Infunc	28.859302	28.860376	0.001074								
	Task_Period_Impl_Swc1to2_Infunc	28.865338	28.866232	0.000894								
	osIdleLoop	28.920153	29.736192	0.816039								
11	Task_Event_Expl_Swc4to3_Outfunc	29.769389	29.775310	0.005921	Runnable_Swc04_Event_Expl_DataStruct_Out	29.769491	29.775286	0.005795	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	29.771883	29.775174	0.000924
	Task_Event_Impl_Swc3to4_Outfunc	29.781429	29.786456	0.005027	Runnable_Swc03_Event_Impl_DataStruct_Out	29.781510	29.783647	0.002137	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	29.791606	29.792485	0.000879				0.000000				
	Task_Event_Impl_Swc3to4_Infunc	29.797439	29.798395	0.000956				0.000000				
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	29.803148	29.807979	0.004831	Runnable_Swc06_Period_Expl_DataDIN3_Out	29.803233	29.807962	0.004729	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	29.805441	29.807850	0.000323
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	29.813125	29.816859	0.003734	Runnable_Swc05_Period_Impl_DataDIN2_Out	29.813190	29.815035	0.001845	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	29.821829	29.822356	0.000527								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	29.827023	29.827294	0.000271								
	osIdleLoop	29.907772	30.736283	0.828511								

11.7.8. V2_0_1111 notification process 8 to 11 ms

	osActivate Task Entry in ms	End in ms	Duration in ms	osCheckInterruptEnabled Entry in ms	End in ms	Duration in ms	osSchedule Prio Entry in ms	End in ms	Duration in ms	osDispatecher Entry in ms	End in ms	Duration	Notification Duration in ms (without ActivateTask)		Duration notification
8	26.772515	26.774914	0.002399											Task_Event_Expl_Swc4to3_Outfunc	
	26.783848	26.785810	0.001962	26.775630	26.775898	0.000268	26.777112	26.777934	0.000822	26.778984	26.780918	0.001934	0.003024	Task_Event_Impl_Swc3to4_Outfunc	
				26.786227	26.786490	0.000263	26.787381	26.788069	0.000688	26.789068	26.790954	0.001886	0.002837	Task_Event_Expl_Swc4to3_Infunc	0.012963
				26.792457	26.792846	0.000389	26.793780	26.794499	0.000719	26.795283	26.796772	0.001489	0.002597	Task_Event_Impl_Swc3to4_Infunc	0.008268
	26.805135	26.807202	0.002067	26.798539	26.798927	0.000388	26.800004	26.800827	0.000823	26.801558	26.802872	0.001314	0.002525	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	26.814513	26.816432	0.001919	26.807641	26.807963	0.000322	26.808921	26.809671	0.000750	26.810593	26.812163	0.001570	0.002642	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				26.816919	26.817127	0.000208	26.818233	26.819069	0.000836	26.819954	26.821459	0.001505	0.002549	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011586
				26.822437	26.822835	0.000398	26.823858	26.824707	0.000849	26.825540	26.826881	0.001341	0.002588	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007413
9	27.816374	27.818946	0.002572											Task_Event_Impl_Swc3to4_Outfunc	
	27.828338	27.830241	0.001903	27.819532	27.819843	0.000311	27.820998	27.821762	0.000764	27.822880	27.824995	0.002115	0.003190	Task_Event_Expl_Swc4to3_Outfunc	
				27.830750	27.830972	0.000222	27.831869	27.832550	0.000681	27.833569	27.835245	0.001676	0.002579	Task_Event_Impl_Swc3to4_Infunc	0.013627
				27.837001	27.837372	0.000371	27.838269	27.838997	0.000728	27.839869	27.841322	0.001453	0.002552	Task_Event_Expl_Swc4to3_Infunc	0.008103
	27.849912	27.851741	0.001829	27.842718	27.843054	0.000336	27.844121	27.844836	0.000715	27.845650	27.847213	0.001563	0.002614	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	27.859385	27.861124	0.001739	27.852213	27.852436	0.000223	27.853736	27.854590	0.000854	27.855405	27.856981	0.001576	0.002653	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				27.861449	27.861690	0.000241	27.862648	27.863351	0.000703	27.864236	27.865772	0.001536	0.002480	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.010910
				27.866765	27.867027	0.000262	27.868023	27.868860	0.000837	27.869750	27.871299	0.001549	0.002648	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007385
10	28.782666	28.785196	0.002530											Task_Event_Expl_Swc4to3_Outfunc	
	28.794185	28.796107	0.001922	28.785897	28.786179	0.000282	28.787446	28.788280	0.000834	28.789292	28.791145	0.001853	0.002969	Task_Event_Impl_Swc3to4_Outfunc	
				28.796453	28.796708	0.000292	28.797732	28.798426	0.000688	28.799347	28.801231	0.001574	0.002554	Task_Event_Expl_Swc4to3_Infunc	0.012819
				28.802598	28.802851	0.000266	28.803894	28.804626	0.000829	28.805448	28.807063	0.001359	0.002454	Task_Event_Impl_Swc3to4_Infunc	0.007789
	28.816019	28.818113	0.002094	28.808721	28.808998	0.000351	28.810168	28.810890	0.000797	28.811728	28.813322	0.001477	0.002625	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	28.825605	28.827678	0.002073	28.818579	28.818862	0.000243	28.819871	28.820670	0.000855	28.821549	28.823022	0.001547	0.002645	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				28.828216	28.828444	0.000296	28.829490	28.830190	0.000761	28.830999	28.832504	0.001577	0.002634	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.012083
				28.833386	28.833664	0.000379	28.834615	28.835453	0.000731	28.836312	28.837727	0.001670	0.002780	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007835
	28.844387	28.846378	0.001991	28.838735	28.839081	0.000282	28.840119	28.840845	0.000916	28.841671	28.843213	0.001775	0.002973	Task_Period_Expl_Swc2to1_Outfunc	
	28.852758	28.854596	0.001838	28.847149	28.847416	0.000240	28.848452	28.849134	0.000744	28.850067	28.851645	0.001498	0.002482	Task_Period_Impl_Swc1to2_Outfunc	
				28.854915	28.855145	0.000228	28.856041	28.856715	0.000829	28.857590	28.859086	0.001444	0.002501	Task_Period_Expl_Swc2to1_Infunc	0.009723
				28.860686	28.861027	0.000228	28.861955	28.862681	0.000829	28.863517	28.865136	0.001444	0.002501	Task_Period_Impl_Swc1to2_Infunc	0.007914
11	29.772654	29.775021	0.002367											Task_Event_Expl_Swc4to3_Outfunc	
	29.784398	29.786432	0.002034	29.775774	29.776051	0.000277	29.777422	29.778287	0.000865	29.779314	29.781249	0.001935	0.003077	Task_Event_Impl_Swc3to4_Outfunc	
				29.786927	29.787206	0.000279	29.788181	29.788863	0.000682	29.789729	29.791427	0.001698	0.002659	Task_Event_Expl_Swc4to3_Infunc	0.013130
				29.792882	29.793127	0.000245	29.794114	29.794953	0.000839	29.795814	29.797236	0.001422	0.002506	Task_Event_Impl_Swc3to4_Infunc	0.008078
	29.805646	29.807732	0.002086	29.798719	29.798963	0.000244	29.800010	29.800727	0.000717	29.801515	29.802963	0.001448	0.002409	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	29.815121	29.816843	0.001722	29.808323	29.808563	0.000240	29.809759	29.810589	0.000830	29.811469	29.812945	0.001476	0.002546	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				29.817182	29.817417	0.000235	29.818411	29.819207	0.000796	29.820109	29.821640	0.001531	0.002562	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.010928
				29.822645	29.822870	0.000225	29.823973	29.824817	0.000844	29.825544	29.826840	0.001296	0.002365	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007176

11.7.9. V2_0_1111 read-process 8 to 11 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
									0.001596	0.014559
									0.000000	0.008268
	Runnable_Swc03_Odr_Expl_DataStruct_In	26.791225	26.792012	0.000787	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	26.791321	26.791909	0.000588		
	Runnable_Swc04_Odr_Impl_DataStruct_In	26.797511	26.798015	0.000504	*Macros no functions*	0.000000	0.000000	0.000000		
8									0.000000	0.011586
									0.000437	0.007850
	Runnable_Swc06_Odr_Impl_DataDIN2_In	26.821869	26.821985	0.000116	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	26.827188	26.827582	0.000394	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	26.827331	26.827474	0.000143		
									0.000000	0.013627
									0.001517	0.009620
	Runnable_Swc04_Odr_Impl_DataStruct_In	27.836007	27.836514	0.000507	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	27.841568	27.842336	0.000768	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	27.841662	27.842236	0.000574		
9									0.000416	0.011326
									0.000000	0.007385
	Runnable_Swc05_Odr_Expl_DataDIN3_In	27.866079	27.866468	0.000389	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	27.866222	27.866365	0.000143		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	27.871742	27.871857	0.000115	Macros no functions	0.000000	0.000000	0.000000		
									0.001789	0.014608
									0.000000	0.007789
	Runnable_Swc03_Odr_Expl_DataStruct_In	28.801506	28.802287	0.000781	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	28.801601	28.802184	0.000583		
	Runnable_Swc04_Odr_Impl_DataStruct_In	28.807796	28.808275	0.000479	Macros no functions	0.000000	0.000000	0.000000		
									0.000000	0.012083
10									0.000316	0.008151
	Runnable_Swc06_Odr_Impl_DataDIN2_In	28.832914	28.833026	0.000112	Macros no functions	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	28.838033	28.838423	0.000390	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	28.838176	28.838320	0.000144		
									0.001699	0.011422
									0.000000	0.007914
	Runnable_Swc01_Odr_Expl_DataStruct_In	28.859384	28.860360	0.000976	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	28.859507	28.860256	0.000749		
	Runnable_Swc02_Odr_Impl_DataStruct_In	28.865792	28.866216	0.000424	Macros no functions	0.000000	0.000000	0.000000		
									0.001514	0.014644
									0.000000	0.008078
	Runnable_Swc03_Odr_Expl_DataStruct_In	29.791670	29.792465	0.000795	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	29.791766	29.792356	0.000590		
11	Runnable_Swc04_Odr_Impl_DataStruct_In	29.797926	29.798379	0.000453	Macros no functions	0.000000	0.000000	0.000000		
									0.000470	0.011398
									0.000000	0.007176
	Runnable_Swc05_Odr_Expl_DataDIN3_In	29.821949	29.822341	0.000392	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	29.822098	29.822245	0.000147		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	29.827191	29.827279	0.000088	Macros no functions	0.000000	0.000000	0.000000		

11.7.10. V2_0_1111 write-process 12 to 15 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
	Task_Event_Impl_Swc3to4_Outfunc	30.769695	30.776067	0.006372	Runnable_Swc03_Event_Impl_DataStruct_Out	30.769825	30.772561	0.002736	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	30.782418	30.787887	0.005469	Runnable_Swc04_Event_Expl_DataStruct_Out	30.782491	30.787871	0.005380	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	30.784488	30.787765	0.001024
	Task_Event_Impl_Swc3to4_Infunc	30.792995	30.794062	0.001067								
	Task_Event_Expl_Swc4to3_Infunc	30.799096	30.799907	0.000811								
12	Task_IoHw_Period_Expl_Swc6to5_Outfunc	30.804801	30.809529	0.004728	Runnable_Swc06_Period_Expl_DataDIN3_Out	30.804879	30.809514	0.004635	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	30.807032	30.809412	0.000321
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	30.814455	30.818862	0.004407	Runnable_Swc05_Period_Impl_DataDIN2_Out	30.814532	30.816640	0.002108	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	30.824084	30.824626	0.000542								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	30.829804	30.830153	0.000349								
	osIdleLoop	30.994400	31.736402	0.742002								
	Task_Event_Expl_Swc4to3_Outfunc	31.769388	31.776566	0.007178	Runnable_Swc04_Event_Expl_DataStruct_Out	31.769526	31.776542	0.007016	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	31.772536	31.776436	0.001238
	Task_Event_Impl_Swc3to4_Outfunc	31.782710	31.787866	0.005156	Runnable_Swc03_Event_Impl_DataStruct_Out	31.782789	31.785038	0.002249	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	31.793042	31.793897	0.000855								
	Task_Event_Impl_Swc3to4_Infunc	31.799248	31.800215	0.000967								
13	Task_IoHw_Period_Impl_Swc5to6_Outfunc	31.805095	31.809587	0.004492	Runnable_Swc05_Period_Impl_DataDIN2_Out	31.805186	31.807412	0.002226	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	31.814662	31.819033	0.004371	Runnable_Swc06_Period_Expl_DataDIN3_Out	31.814732	31.819021	0.004289	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	31.816746	31.818914	0.000294
	Task_IoHw_Period_Impl_Swc5to6_Infunc	31.824067	31.824323	0.000256								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	31.829275	31.829809	0.000534								
	osIdleLoop	31.934700	32.736474	0.801774								
	Task_Event_Expl_Swc4to3_Outfunc	32.770705	32.777628	0.006923	Runnable_Swc04_Event_Expl_DataStruct_Out	32.770838	32.777605	0.006767	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	32.773763	32.777493	0.001135
	Task_Event_Impl_Swc3to4_Outfunc	32.783530	32.788571	0.005041	Runnable_Swc03_Event_Impl_DataStruct_Out	32.783616	32.785747	0.002131	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	32.794061	32.794927	0.000866								
	Task_Event_Impl_Swc3to4_Infunc	32.800194	32.801148	0.000954								
14	Task_IoHw_Period_Expl_Swc6to5_Outfunc	32.806505	32.811154	0.004649	Runnable_Swc06_Period_Expl_DataDIN3_Out	32.806600	32.811141	0.004541	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	32.808927	32.811039	0.000258
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	32.816220	32.820591	0.004371	Runnable_Swc05_Period_Impl_DataDIN2_Out	32.816301	32.818469	0.002168	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	32.825899	32.826517	0.000618								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	32.831695	32.832052	0.000357								
	osIdleLoop	32.896572	33.736565	0.839993								
	Task_Event_Impl_Swc3to4_Outfunc	33.777548	33.783650	0.006102	Runnable_Swc03_Event_Impl_DataStruct_Out	33.777668	33.780340	0.002672	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	33.789202	33.794364	0.005162	Runnable_Swc04_Event_Expl_DataStruct_Out	33.789283	33.794350	0.005067	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Ex	33.791491	33.794247	0.000885
	Task_Event_Impl_Swc3to4_Infunc	33.799367	33.800353	0.000986								
	Task_Event_Expl_Swc4to3_Infunc	33.805188	33.806038	0.000850								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	33.811273	33.815663	0.004390	Runnable_Swc06_Period_Expl_DataDIN3_Out	33.811348	33.815649	0.004301	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	33.813344	33.815541	0.000280
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	33.820972	33.825188	0.004216	Runnable_Swc05_Period_Impl_DataDIN2_Out	33.821042	33.822995	0.001953	Macros no functions	0.000000	0.000000	0.000000
15	Task_IoHw_Period_Expl_Swc6to5_Infunc	33.830564	33.831082	0.000518								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	33.836031	33.836294	0.000263								
	Task_Period_Impl_Swc1to2_Outfunc	33.841474	33.844082	0.002608	Runnable_Swc01_Period_Impl_DataStruct_Out	33.841736	33.841752	0.000016	Macros no functions	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1_Outfunc	33.848872	33.851821	0.002949	Runnable_Swc02_Period_Expl_DataStruct_Out	33.848942	33.851806	0.002864	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	33.849048	33.851702	0.000812
	Task_Period_Impl_Swc1to2_Infunc	33.856439	33.857402	0.000963								
	Task_Period_Expl_Swc2to1_Infunc	33.862175	33.863256	0.001081								
	osIdleLoop	33.911226	34.736656	0.825430								

11.7.11. V2_0_1111 notification process 12 to 15 ms

	osActivate Task Entry in ms	End in ms	Duration in ms	osCheckInterruptEnabled Entry in ms	End in ms	Duration in ms	osSchedule Prio Entry in ms	End in ms	Duration in ms	osDispatecher Entry in ms	End in ms	Duration	Notification Duration in ms (without ActivateTask)		Duration notification
12	30.773506	30.776042	0.002536											Task_Event_Impl_Swc3to4_Outfunc	
	30.785337	30.787590	0.002253	30.776558	30.776861	0.000303	30.778223	30.779189	0.000966	30.780255	30.782209	0.001954	0.003223	Task_Event_Expl_Swc4to3_Outfunc	
				30.788217	30.788479	0.000262	30.789594	30.790403	0.000809	30.791249	30.792781	0.001532	0.002603	Task_Event_Impl_Swc3to4_Infunc	0.013831
				30.794507	30.794881	0.000374	30.795787	30.796506	0.000719	30.797381	30.798890	0.001509	0.002602	Task_Event_Expl_Swc4to3_Infunc	0.008525
	30.807237	30.809296	0.002059	30.800345	30.800699	0.000354	30.801716	30.802436	0.000720	30.803195	30.804590	0.001395	0.002469	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	30.816752	30.818840	0.002088	30.809825	30.810054	0.000229	30.811079	30.811780	0.000701	30.812693	30.814268	0.001575	0.002505	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				30.819307	30.819590	0.000283	30.820590	30.821397	0.000807	30.822286	30.823863	0.001577	0.002667	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011638
				30.824983	30.825343	0.000360	30.826364	30.827097	0.000733	30.828005	30.829581	0.001576	0.002669	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007966
13	31.773652	31.776314	0.002662											Task_Event_Expl_Swc4to3_Outfunc	
	31.785798	31.787846	0.002048	31.777051	31.777333	0.000282	31.778651	31.779516	0.000865	31.780553	31.782499	0.001946	0.003093	Task_Event_Impl_Swc3to4_Outfunc	
				31.788256	31.788515	0.000259	31.789490	31.790176	0.000686	31.791095	31.792831	0.001736	0.002681	Task_Event_Expl_Swc4to3_Infunc	0.013592
				31.794328	31.794726	0.000398	31.795681	31.796516	0.000835	31.797435	31.799045	0.001610	0.002843	Task_Event_Impl_Swc3to4_Infunc	0.008427
	31.807519	31.809566	0.002047	31.800646	31.800926	0.000280	31.801957	31.802672	0.000715	31.803458	31.804904	0.001446	0.002441	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	31.816922	31.818796	0.001874	31.810004	31.810262	0.000258	31.811227	31.811916	0.000689	31.812841	31.814449	0.001608	0.002555	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				31.819283	31.819490	0.000207	31.820567	31.821380	0.000813	31.822291	31.823854	0.001563	0.002583	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011556
				31.824646	31.824881	0.000235	31.826016	31.826881	0.000865	31.827639	31.829054	0.001415	0.002515	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007228
14	32.774774	32.777369	0.002595											Task_Event_Expl_Swc4to3_Outfunc	
	32.786493	32.788549	0.002056	32.778092	32.778388	0.000296	32.779586	32.780415	0.000829	32.781425	32.783318	0.001893	0.003018	Task_Event_Impl_Swc3to4_Outfunc	
				32.789016	32.789299	0.000283	32.790241	32.791025	0.000784	32.792022	32.793854	0.001832	0.002899	Task_Event_Expl_Swc4to3_Infunc	0.013553
				32.795385	32.795689	0.000304	32.796830	32.797681	0.000851	32.798474	32.799990	0.001516	0.002671	Task_Event_Impl_Swc3to4_Infunc	0.008492
	32.809069	32.810923	0.001854	32.801546	32.801806	0.000260	32.803004	32.803827	0.000823	32.804678	32.806313	0.001635	0.002718	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	32.818589	32.820568	0.001979	32.811430	32.811652	0.000222	32.812799	32.813507	0.000708	32.814435	32.816040	0.001605	0.002535	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				32.821049	32.821399	0.000350	32.822378	32.823071	0.000693	32.824015	32.825672	0.001657	0.002700	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011460
				32.826874	32.827242	0.000368	32.828314	32.829060	0.000746	32.829950	32.831499	0.001549	0.002663	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007960
15	33.781169	33.783625	0.002456											Task_Event_Impl_Swc3to4_Outfunc	
	33.792252	33.794123	0.001871	33.784155	33.784442	0.000287	33.785529	33.786242	0.000713	33.787226	33.789018	0.001792	0.002792	Task_Event_Expl_Swc4to3_Outfunc	
				33.794647	33.794881	0.000292	33.795824	33.796607	0.000688	33.797539	33.799163	0.001574	0.002554	Task_Event_Impl_Swc3to4_Infunc	0.012964
				33.800710	33.800991	0.000266	33.801942	33.802772	0.000829	33.803551	33.805004	0.001359	0.002454	Task_Event_Expl_Swc4to3_Infunc	0.007865
	33.813506	33.815423	0.001917	33.806422	33.806690	0.000351	33.807790	33.808618	0.000797	33.809460	33.811077	0.001477	0.002625	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	33.823111	33.825171	0.002060	33.815932	33.816154	0.000243	33.817439	33.818297	0.000855	33.819200	33.820763	0.001547	0.002645	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				33.825532	33.825772	0.000296	33.826946	33.827797	0.000761	33.828714	33.830345	0.001577	0.002634	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011412
				33.831365	33.831627	0.000379	33.832609	33.833344	0.000731	33.834254	33.835818	0.001670	0.002780	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007992
	33.842337	33.844068	0.001731	33.836691	33.837054	0.000282	33.838101	33.838826	0.000916	33.839646	33.841218	0.001775	0.002973	Task_Period_Impl_Swc1to2_Outfunc	
	33.849736	33.851578	0.001842	33.844365	33.844590	0.000240	33.845600	33.846409	0.000744	33.847192	33.848663	0.001498	0.002482	Task_Period_Expl_Swc2to1_Outfunc	
				33.852131	33.852354	0.000228	33.853260	33.853941	0.000829	33.854870	33.856259	0.001444	0.002501	Task_Period_Impl_Swc1to2_Infunc	0.009663
				33.857698	33.857944	0.000228	33.858914	33.859815	0.000829	33.860633	33.861963	0.001444	0.002501	Task_Period_Expl_Swc2to1_Infunc	0.007807

11.7.12. V2_0_1111 read-process 12 to 15 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
12									0.000000	0.013831
									0.001563	0.010088
	Runnable_Swc04_Odr_Impl_DataStruct_In	30.793534	30.794040	0.000506	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	30.799154	30.799885	0.000731	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	30.799242	30.799781	0.000539		
									0.000470	0.012108
				0.000000					0.000000	0.007966
	Runnable_Swc05_Odr_Expl_DataDIN3_In	30.824203	30.824608	0.000405	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	30.824353	30.824502	0.000149		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	30.830023	30.830136	0.000113	*Macros no functions*	0.000000	0.000000	0.000000		
13									0.001807	0.015399
									0.000000	0.008427
	Runnable_Swc03_Odr_Expl_DataStruct_In	31.793104	31.793876	0.000772	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	31.793196	31.793765	0.000569		
	Runnable_Swc04_Odr_Impl_DataStruct_In	31.799735	31.800194	0.000459	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.011556
				0.000000					0.000440	0.007668
	Runnable_Swc06_Odr_Impl_DataDIN2_In	31.824223	31.824307	0.000084	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	31.829394	31.829790	0.000396	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	31.829544	31.829690	0.000146		
14									0.001711	0.015264
									0.000000	0.008492
	Runnable_Swc03_Odr_Expl_DataStruct_In	32.794123	32.794904	0.000781	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	32.794217	32.794793	0.000576		
	Runnable_Swc04_Odr_Impl_DataStruct_In	32.800676	32.801128	0.000452	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000431	0.011891
									0.000000	0.007960
	Runnable_Swc05_Odr_Expl_DataDIN3_In	32.826042	32.826499	0.000457	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	32.826220	32.826393	0.000173		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	32.831921	32.832035	0.000114	*Macros no functions*	0.000000	0.000000	0.000000		
15									0.000000	0.012964
									0.001459	0.009324
	Runnable_Swc04_Odr_Impl_DataStruct_In	33.799868	33.800336	0.000468	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	33.805250	33.806019	0.000769	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_E	33.805344	33.805918	0.000574		
									0.000423	0.011835
									0.000000	0.007992
	Runnable_Swc05_Odr_Expl_DataDIN3_In	33.830679	33.831068	0.000389	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	33.830822	33.830965	0.000143		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	33.836187	33.836274	0.000087	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.009663
									0.001568	0.009375
	Runnable_Swc02_Odr_Impl_DataStruct_In	33.856931	33.857387	0.000456	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc01_Odr_Expl_DataStruct_In	33.862258	33.863242	0.000984	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	33.862382	33.863138	0.000756		
					Macros no functions					

11.7.13. V2_0_1111 write-process 16 to 19 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
16	Task_Event_Impl_Swc3to4_Outfunc	34.770837	34.776759	0.005922	Runnable_Swc03_Event_Impl_DataStruct_Out	34.770953	34.773531	0.002578	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	34.782426	34.787929	0.005503	Runnable_Swc04_Event_Expl_DataStruct_Out	34.782497	34.787910	0.005413	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	34.784501	34.787802	0.000993
	Task_Event_Impl_Swc3to4_Infunc	34.792785	34.793747	0.000962								
	Task_Event_Expl_Swc4to3_Infunc	34.798670	34.799547	0.000877								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	34.804745	34.808944	0.004199	Runnable_Swc05_Period_Impl_DataDIN2_Out	34.804819	34.806730	0.001911	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	34.814307	34.818703	0.004396	Runnable_Swc06_Period_Expl_DataDIN3_Out	34.814374	34.818688	0.004314	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	34.816375	34.818585	0.000283
	Task_IoHw_Period_Impl_Swc5to6_Infunc	34.824067	34.824323	0.000256								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	34.829055	34.829582	0.000527								
	osIdleLoop	34.890154	35.736729	0.846575								
17	Task_Event_Expl_Swc4to3_Outfunc	35.769438	35.775911	0.006473	Runnable_Swc04_Event_Expl_DataStruct_Out	35.769557	35.775887	0.006330	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	35.772321	35.775774	0.001025
	Task_Event_Impl_Swc3to4_Outfunc	35.781480	35.786262	0.004782	Runnable_Swc03_Event_Impl_DataStruct_Out	35.781548	35.783680	0.002132	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	35.791325	35.792201	0.000876								
	Task_Event_Impl_Swc3to4_Infunc	35.797241	35.798312	0.001071								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	35.803583	35.808164	0.004581	Runnable_Swc06_Period_Expl_DataDIN3_Out	35.803661	35.808145	0.004484	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	35.805755	35.808039	0.000347
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	35.813645	35.817579	0.003934	Runnable_Swc05_Period_Impl_DataDIN2_Out	35.813720	35.815795	0.002075	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	35.822728	35.823246	0.000518								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	35.828505	35.828777	0.000272								
	osIdleLoop	35.877589	36.736838	0.859249								
18	Task_Event_Impl_Swc3to4_Outfunc	36.769310	36.775231	0.005921	Runnable_Swc03_Event_Impl_DataStruct_Out	36.769423	36.771987	0.002564	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	36.780880	36.785955	0.005075	Runnable_Swc04_Event_Expl_DataStruct_Out	36.780947	36.785936	0.004989	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	36.783092	36.785829	0.000842
	Task_Event_Impl_Swc3to4_Infunc	36.791149	36.792168	0.001019								
	Task_Event_Expl_Swc4to3_Infunc	36.797433	36.798288	0.000855								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	36.803189	36.807342	0.004153	Runnable_Swc05_Period_Impl_DataDIN2_Out	36.803259	36.805221	0.001962	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	36.812254	36.816475	0.004221	Runnable_Swc06_Period_Expl_DataDIN3_Out	36.812324	36.816464	0.004140	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	36.814296	36.816359	0.000277
	Task_IoHw_Period_Impl_Swc5to6_Infunc	36.821513	36.821771	0.000258								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	36.826564	36.827091	0.000527								
	osIdleLoop	36.886291	37.736919	0.850628								
19	Task_Event_Impl_Swc3to4_Outfunc	37.770032	37.775723	0.005691	Runnable_Swc03_Event_Impl_DataStruct_Out	37.770125	37.772458	0.002333	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	37.781642	37.787165	0.005523	Runnable_Swc04_Event_Expl_DataStruct_Out	37.781701	37.787151	0.005450	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	37.783726	37.787054	0.000980
	Task_Event_Impl_Swc3to4_Infunc	37.791940	37.792990	0.001050								
	Task_Event_Expl_Swc4to3_Infunc	37.798187	37.799000	0.000813								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	37.804075	37.808543	0.004468	Runnable_Swc05_Period_Impl_DataDIN2_Out	37.804158	37.806286	0.002128	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	37.813483	37.817720	0.004237	Runnable_Swc06_Period_Expl_DataDIN3_Out	37.813562	37.817705	0.004143	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	37.815453	37.817603	0.000278
	Task_IoHw_Period_Impl_Swc5to6_Infunc	37.823042	37.823420	0.000378								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	37.828279	37.828873	0.000594								
	osIdleLoop	37.885536	38.737010	0.851474								

11.7.14. V2_0_1111 notification process 16 to 19 ms

	osActivate Task Entry in ms	End in ms	Duration in ms	osCheckInterruptEnabled Entry in ms	End in ms	Duration in ms	osSchedule Prio Entry in ms	End in ms	Duration in ms	osDispatecher Entry in ms	End in ms	Duration	Notification Duration in ms (without ActivateTask)		Duration notification
16	34.774375	34.776734	0.002359											Task_Event_Impl_Swc3to4_Outfunc	
	34.785319	34.787627	0.002308	34.777264	34.777536	0.000272	34.778612	34.779316	0.000704	34.780331	34.782218	0.001887	0.002863	Task_Event_Expl_Swc4to3_Outfunc	
				34.788313	34.788561	0.000248	34.789515	34.790180	0.000665	34.791084	34.792604	0.001520	0.002433	Task_Event_Impl_Swc3to4_Infunc	0.013158
				34.794131	34.794490	0.000359	34.795433	34.796162	0.000729	34.797042	34.798490	0.001448	0.002536	Task_Event_Expl_Swc4to3_Infunc	0.008239
	34.806838	34.808917	0.002079	34.799931	34.800190	0.000259	34.801291	34.802127	0.000836	34.802946	34.804536	0.001590	0.002685	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	34.816542	34.818469	0.001927	34.809476	34.809826	0.000350	34.810826	34.811609	0.000783	34.812455	34.814100	0.001645	0.002778	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				34.819013	34.819262	0.000249	34.820450	34.821288	0.000838	34.822217	34.823854	0.001637	0.002724	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011977
				34.824646	34.824881	0.000235	34.825875	34.826609	0.000734	34.827387	34.828836	0.001449	0.002418	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007325
17	35.773201	35.775629	0.002428											Task_Event_Expl_Swc4to3_Outfunc	
	35.784321	35.786244	0.001923	35.776389	35.776688	0.000299	35.777953	35.778788	0.000835	35.779695	35.781272	0.001577	0.002711	Task_Event_Impl_Swc3to4_Outfunc	
				35.786612	35.786870	0.000258	35.787912	35.788681	0.000769	35.789526	35.791154	0.001628	0.002655	Task_Event_Expl_Swc4to3_Infunc	0.012576
				35.792585	35.792834	0.000249	35.793914	35.794651	0.000737	35.795536	35.797054	0.001518	0.002504	Task_Event_Impl_Swc3to4_Infunc	0.007958
	35.805921	35.807858	0.001937	35.798804	35.799088	0.000284	35.800282	35.801017	0.000735	35.801834	35.803395	0.001561	0.002580	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	35.815864	35.817562	0.001698	35.808534	35.808806	0.000272	35.810086	35.810925	0.000839	35.811853	35.813463	0.001610	0.002721	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				35.817915	35.818170	0.000255	35.819406	35.820125	0.000719	35.821018	35.822531	0.001513	0.002487	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011079
				35.823529	35.823790	0.000261	35.824872	35.825607	0.000735	35.826572	35.828290	0.001718	0.002714	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007417
18	36.772858	36.775206	0.002348											Task_Event_Impl_Swc3to4_Outfunc	
	36.783810	36.785705	0.001895	36.775729	36.776025	0.000296	36.777224	36.777944	0.000720	36.778933	36.780709	0.001776	0.002792	Task_Event_Expl_Swc4to3_Outfunc	
				36.786325	36.786579	0.000254	36.787718	36.788525	0.000807	36.789422	36.790945	0.001523	0.002584	Task_Event_Impl_Swc3to4_Infunc	0.012799
				36.792673	36.792991	0.000318	36.793963	36.794790	0.000827	36.795616	36.797227	0.001611	0.002756	Task_Event_Expl_Swc4to3_Infunc	0.008254
	36.805330	36.807321	0.001991	36.798632	36.798909	0.000277	36.800011	36.800836	0.000825	36.801593	36.802981	0.001388	0.002490	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	36.814455	36.816241	0.001786	36.807766	36.808018	0.000252	36.808976	36.809654	0.000678	36.810475	36.812045	0.001570	0.002500	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				36.816684	36.816908	0.000224	36.817969	36.818797	0.000828	36.819696	36.821299	0.001603	0.002655	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011367
				36.822121	36.822391	0.000270	36.823393	36.824126	0.000733	36.824896	36.826345	0.001449	0.002452	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007151
19	37.773288	37.775697	0.002409											Task_Event_Impl_Swc3to4_Outfunc	
	37.784584	37.786932	0.002348	37.776228	37.776515	0.000287	37.777669	37.778397	0.000728	37.779453	37.781431	0.001978	0.002993	Task_Event_Expl_Swc4to3_Outfunc	
				37.787462	37.787708	0.000246	37.788590	37.789279	0.000689	37.790184	37.791759	0.001575	0.002510	Task_Event_Impl_Swc3to4_Infunc	0.013435
				37.793354	37.793626	0.000272	37.794605	37.795452	0.000847	37.796371	37.797981	0.001610	0.002729	Task_Event_Expl_Swc4to3_Infunc	0.008637
	37.806401	37.808526	0.002125	37.799458	37.799762	0.000304	37.800783	37.801490	0.000707	37.802325	37.803899	0.001574	0.002585	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	37.815615	37.817487	0.001872	37.808886	37.809126	0.000240	37.810053	37.810734	0.000681	37.811659	37.813268	0.001609	0.002530	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				37.818016	37.818244	0.000228	37.819528	37.820388	0.000860	37.821281	37.822818	0.001537	0.002625	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011517
				37.823811	37.824099	0.000288	37.825078	37.825816	0.000738	37.826677	37.828077	0.001400	0.002426	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007301

11.7.15. V2_0_1111 read-process 16 to 19 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
16									0.000000	0.013158
									0.001583	0.009822
	Runnable_Swc04_Odr_Impl_DataStruct_In	34.793272	34.793728	0.000456	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	34.798734	34.799528	0.000794	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	34.798830	34.799420	0.000590		
									0.000000	0.011977
									0.000427	0.007752
	Runnable_Swc06_Odr_Impl_DataDIN2_In	34.824223	34.824307	0.000084	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	34.829170	34.829563	0.000393	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	34.829312	34.829456	0.000144		
17									0.001614	0.014190
									0.000000	0.007958
	Runnable_Swc03_Odr_Expl_DataStruct_In	35.791389	35.792182	0.000793	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	35.791485	35.792074	0.000589		
	Runnable_Swc04_Odr_Impl_DataStruct_In	35.797779	35.798288	0.000509	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000491	0.011570
									0.000000	0.007417
	Runnable_Swc05_Odr_Expl_DataDIN3_In	35.822842	35.823232	0.000390	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	35.822985	35.823129	0.000144		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	35.828673	35.828761	0.000088	*Macros no functions*	0.000000	0.000000	0.000000		
18									0.000000	0.012799
									0.001418	0.009672
	Runnable_Swc04_Odr_Impl_DataStruct_In	36.791659	36.792143	0.000484	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	36.797496	36.798271	0.000775	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	36.797589	36.798165	0.000576		
									0.000000	0.011367
									0.000420	0.007571
	Runnable_Swc06_Odr_Impl_DataDIN2_In	36.821669	36.821753	0.000084	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	36.826679	36.827073	0.000394	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	36.826822	36.826965	0.000143		
19									0.000000	0.013435
									0.001519	0.010156
	Runnable_Swc04_Odr_Impl_DataStruct_In	37.792475	37.792972	0.000497	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	37.798245	37.798977	0.000732	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	37.798333	37.798872	0.000539		
									0.000000	0.011517
									0.000445	0.007746
	Runnable_Swc06_Odr_Impl_DataDIN2_In	37.823280	37.823401	0.000121	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	37.828417	37.828859	0.000442	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	37.828589	37.828756	0.000167		

11.7.16. V2_0_1111 write-process 20 to 23 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
20	Task_Event_Expl_Swc4to3_Outfunc	38.779210	38.785498	0.006288	Runnable_Swc04_Event_Expl_DataStruct_Out	38.779326	38.785472	0.006146	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	38.782050	38.785356	0.000987
	Task_Event_Impl_Swc3to4_Outfunc	38.791301	38.795873	0.004572	Runnable_Swc03_Event_Impl_DataStruct_Out	38.791378	38.793462	0.002084	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	38.800833	38.801683	0.000850								
	Task_Event_Impl_Swc3to4_Infunc	38.806441	38.807512	0.001071								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	38.812621	38.817295	0.004674	Runnable_Swc06_Period_Expl_DataDIN3_Out	38.812709	38.817279	0.004570	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	38.814876	38.817169	0.000352
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	38.822436	38.826557	0.004121	Runnable_Swc05_Period_Impl_DataDIN2_Out	38.822510	38.824504	0.001994	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	38.831593	38.832128	0.000535								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	38.837386	38.837651	0.000265								
	Task_Period_Expl_Swc2to1_Outfunc	38.842490	38.845530	0.003040	Runnable_Swc02_Period_Expl_DataStruct_Out	38.842560	38.845515	0.002955	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	38.842666	38.845411	0.000901
	Task_Period_Impl_Swc1to2_Outfunc	38.850399	38.852949	0.002550	Runnable_Swc01_Period_Impl_DataStruct_Out	38.850654	38.850669	0.000015	Macros no functions	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1_Infunc	38.857659	38.858855	0.001196								
	Task_Period_Impl_Swc1to2_Infunc	38.863684	38.864605	0.000921								
	osIdleLoop	38.918663	39.737083	0.818420								
21	Task_Event_Expl_Swc4to3_Outfunc	39.770629	39.777278	0.006649	Runnable_Swc04_Event_Expl_DataStruct_Out	39.770748	39.777257	0.006509	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	39.773506	39.777147	0.001098
	Task_Event_Impl_Swc3to4_Outfunc	39.782736	39.787330	0.004594	Runnable_Swc03_Event_Impl_DataStruct_Out	39.782811	39.784853	0.002042	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	39.792306	39.793168	0.000862								
	Task_Event_Impl_Swc3to4_Infunc	39.798185	39.799138	0.000953								
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	39.804092	39.808609	0.004517	Runnable_Swc06_Period_Expl_DataDIN3_Out	39.804171	39.808591	0.004420	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	39.806284	39.808478	0.000286
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	39.813856	39.818125	0.004269	Runnable_Swc05_Period_Impl_DataDIN2_Out	39.813936	39.816136	0.002200	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	39.823257	39.823665	0.000408								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	39.828731	39.828994	0.000263								
	osIdleLoop	39.974972	40.737192	0.762220								
22	Task_Event_Impl_Swc3to4_Outfunc	40.771059	40.777476	0.006417	Runnable_Swc03_Event_Impl_DataStruct_Out	40.771188	40.773916	0.002728	*Macros no functions*	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	40.783471	40.789025	0.005554	Runnable_Swc04_Event_Expl_DataStruct_Out	40.783539	40.789008	0.005469	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	40.785800	40.788902	0.000991
	Task_Event_Impl_Swc3to4_Infunc	40.793785	40.794788	0.001003								
	Task_Event_Expl_Swc4to3_Infunc	40.799609	40.800592	0.000983								0.000000
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	40.805493	40.809961	0.004468	Runnable_Swc05_Period_Impl_DataDIN2_Out	40.805576	40.807695	0.002119	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	40.814627	40.818966	0.004339	Runnable_Swc06_Period_Expl_DataDIN3_Out	40.814703	40.818955	0.004252	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	40.816534	40.818850	0.000313
	Task_IoHw_Period_Impl_Swc5to6_Infunc	40.824159	40.824517	0.000358								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	40.829519	40.830035	0.000516								
	osIdleLoop	40.998327	41.737294	0.738967								
23	Task_Event_Expl_Swc4to3_Outfunc	41.770661	41.777856	0.007195	Runnable_Swc04_Event_Expl_DataStruct_Out	41.770800	41.777832	0.007032	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	41.773810	41.777727	0.001247
	Task_Event_Impl_Swc3to4_Outfunc	41.783901	41.788776	0.004875	Runnable_Swc03_Event_Impl_DataStruct_Out	41.783979	41.786235	0.002256	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	41.794133	41.794991	0.000858								
	Task_Event_Impl_Swc3to4_Infunc	41.800194	41.801147	0.000953								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	41.806585	41.810866	0.004281	Runnable_Swc05_Period_Impl_DataDIN2_Out	41.806670	41.808667	0.001997	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	41.815635	41.819875	0.004240	Runnable_Swc06_Period_Expl_DataDIN3_Out	41.815706	41.819860	0.004154	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	41.817567	41.819758	0.000274
	Task_IoHw_Period_Impl_Swc5to6_Infunc	41.824940	41.825199	0.000259								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	41.830002	41.830408	0.000406								
	osIdleLoop	41.935127	42.737349	0.802222								

11.7.17. V2_0_1111 notification process 20 to 23 ms

	osActivate Task Entry in ms	End in ms	Duration in ms	osCheckInterruptEnabled Entry in ms	End in ms	Duration in ms	osSchedule Prio Entry in ms	End in ms	Duration in ms	osDispatecher Entry in ms	End in ms	Duration	Notification Duration in ms (without ActivateTask)		Duration notification
20	38.782891	38.785210	0.002319											Task_Event_Expl_Swc4to3_Outfunc	
	38.794089	38.795850	0.001761	38.786030	38.786324	0.000294	38.787530	38.788353	0.000823	38.789345	38.791090	0.001745	0.002862	Task_Event_Impl_Swc3to4_Outfunc	
				38.796330	38.796606	0.000292	38.797678	38.798488	0.000688	38.799311	38.800649	0.001574	0.002554	Task_Event_Expl_Swc4to3_Infunc	0.012307
				38.802067	38.802335	0.000266	38.803314	38.804151	0.000829	38.804958	38.806254	0.001359	0.002454	Task_Event_Impl_Swc3to4_Infunc	0.007619
	38.815044	38.816985	0.001941	38.807996	38.808263	0.000351	38.809325	38.810053	0.000797	38.810871	38.812431	0.001477	0.002625	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	38.824616	38.826537	0.001921	38.817619	38.817863	0.000243	38.818985	38.819724	0.000855	38.820630	38.822227	0.001547	0.002645	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				38.826947	38.827281	0.000296	38.828224	38.829007	0.000761	38.829909	38.831409	0.001577	0.002634	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011341
				38.832411	38.832670	0.000379	38.833788	38.834644	0.000731	38.835562	38.837172	0.001670	0.002780	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007870
	38.843355	38.845199	0.001844	38.837988	38.838218	0.000282	38.839295	38.840072	0.000916	38.840841	38.842277	0.001775	0.002973	Task_Period_Expl_Swc2to1_Outfunc	
	38.851227	38.852933	0.001706	38.845840	38.846072	0.000240	38.847014	38.847812	0.000744	38.848734	38.850145	0.001498	0.002482	Task_Period_Impl_Swc1to2_Outfunc	
				38.853258	38.853499	0.000228	38.854487	38.855296	0.000829	38.856113	38.857454	0.001444	0.002501	Task_Period_Expl_Swc2to1_Infunc	0.009377
				38.859225	38.859572	0.000228	38.860469	38.861190	0.000829	38.861995	38.863504	0.001444	0.002501	Task_Period_Impl_Swc1to2_Infunc	0.007904
21	39.774451	39.776994	0.002543											Task_Event_Expl_Swc4to3_Outfunc	
	39.785484	39.787315	0.001831	39.777695	39.777979	0.000284	39.779236	39.780079	0.000843	39.780963	39.782527	0.001564	0.002691	Task_Event_Impl_Swc3to4_Outfunc	
				39.787640	39.787897	0.000257	39.788917	39.789699	0.000782	39.790539	39.792131	0.001592	0.002631	Task_Event_Expl_Swc4to3_Infunc	0.012459
				39.793491	39.793725	0.000234	39.794842	39.795697	0.000855	39.796553	39.797981	0.001428	0.002517	Task_Event_Impl_Swc3to4_Infunc	0.007841
	39.806451	39.808359	0.001908	39.799522	39.799772	0.000250	39.800818	39.801536	0.000718	39.802345	39.803881	0.001536	0.002504	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	39.816225	39.818108	0.001883	39.808980	39.809236	0.000256	39.810379	39.811190	0.000811	39.812021	39.813645	0.001624	0.002691	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				39.818468	39.818708	0.000240	39.819851	39.820597	0.000746	39.821505	39.823077	0.001572	0.002558	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011426
				39.823954	39.824190	0.000236	39.825200	39.825944	0.000744	39.826892	39.828540	0.001648	0.002628	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007477
22	40.774861	40.777451	0.002590											Task_Event_Impl_Swc3to4_Outfunc	
	40.786667	40.788778	0.002111	40.777968	40.778261	0.000293	40.779461	40.780234	0.000773	40.781283	40.783263	0.001980	0.003046	Task_Event_Expl_Swc4to3_Outfunc	
				40.789368	40.789626	0.000258	40.790476	40.791148	0.000672	40.792011	40.793581	0.001570	0.002500	Task_Event_Impl_Swc3to4_Infunc	0.013690
			0.000000	40.795219	40.795590	0.000371	40.796481	40.797190	0.000709	40.797981	40.799431	0.001450	0.002530	Task_Event_Expl_Swc4to3_Infunc	0.008144
	40.807812	40.809944	0.002132	40.800976	40.801244	0.000268	40.802291	40.803009	0.000718	40.803799	40.805281	0.001482	0.002468	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	40.816729	40.818732	0.002003	40.810305	40.810545	0.000240	40.811503	40.812181	0.000678	40.812969	40.814445	0.001476	0.002394	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				40.819175	40.819399	0.000224	40.820564	40.821405	0.000841	40.822312	40.823936	0.001624	0.002689	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011554
				40.824874	40.825109	0.000235	40.826193	40.827051	0.000858	40.827891	40.829299	0.001408	0.002501	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007551
23	41.774935	41.777605	0.002670											Task_Event_Expl_Swc4to3_Outfunc	
	41.786875	41.788760	0.001885	41.778327	41.778624	0.000297	41.779942	41.780806	0.000864	41.781834	41.783722	0.001888	0.003049	Task_Event_Impl_Swc3to4_Outfunc	
				41.789086	41.789342	0.000256	41.790466	41.791294	0.000828	41.792301	41.793949	0.001648	0.002732	Task_Event_Expl_Swc4to3_Infunc	0.013326
				41.795449	41.795742	0.000293	41.796775	41.797499	0.000724	41.798351	41.800013	0.001662	0.002679	Task_Event_Impl_Swc3to4_Infunc	0.008154
	41.808779	41.810851	0.002072	41.801625	41.801924	0.000299	41.803101	41.803835	0.000734	41.804688	41.806372	0.001684	0.002717	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	41.817724	41.819641	0.001917	41.811169	41.811417	0.000248	41.812329	41.813118	0.000789	41.813911	41.815427	0.001516	0.002553	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				41.820171	41.820399	0.000228	41.821409	41.822107	0.000698	41.823056	41.824727	0.001671	0.002597	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011462
				41.825563	41.825799	0.000236	41.826800	41.827536	0.000736	41.828315	41.829790	0.001475	0.002447	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007220

11.7.18. V2_0_1111 read-process 20 to 23 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
20									0.001556	0.013863
									0.000000	0.007619
	Runnable_Swc03_Odr_Expl_DataStruct_In	38.800895	38.801664	0.000769	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	38.800987	38.801556	0.000569		
	Runnable_Swc04_Odr_Impl_DataStruct_In	38.806979	38.807487	0.000508	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000501	0.011842
									0.000000	0.007870
	Runnable_Swc05_Odr_Expl_DataDIN3_In	38.831712	38.832114	0.000402	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	38.831862	38.832011	0.000149		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	38.837548	38.837635	0.000087	*Macros no functions*	0.000000	0.000000	0.000000		
									0.001740	0.011117
									0.000000	0.007904
	Runnable_Swc01_Odr_Expl_DataStruct_In	38.857751	38.858836	0.001085	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	38.857890	38.858729	0.000839		
	Runnable_Swc02_Odr_Impl_DataStruct_In	38.864152	38.864589	0.000437	*Macros no functions*	0.000000	0.000000	0.000000		
21									0.001681	0.014140
									0.000000	0.007841
	Runnable_Swc03_Odr_Expl_DataStruct_In	39.792369	39.799119	0.006750	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	39.792464	39.793047	0.000583		
	Runnable_Swc04_Odr_Impl_DataStruct_In	39.798667	39.799119	0.000452	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000396	0.011822
									0.000000	0.007477
	Runnable_Swc05_Odr_Expl_DataDIN3_In	39.823340	39.823650	0.000310	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	39.823444	39.823554	0.000110		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	39.828887	39.828974	0.000087	*Macros no functions*	0.000000	0.000000	0.000000		
22									0.000000	0.013690
									0.001668	0.009812
	Runnable_Swc04_Odr_Impl_DataStruct_In	40.794291	40.794766	0.000475	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	40.799683	40.800573	0.000890	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	40.799795	40.800472	0.000677		
									0.000000	0.011554
									0.000457	0.008008
	Runnable_Swc06_Odr_Impl_DataDIN2_In	40.824385	40.824499	0.000114	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	40.829633	40.830022	0.000389	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	40.829776	40.829920	0.000144		
23									0.001816	0.015142
									0.000000	0.008154
	Runnable_Swc03_Odr_Expl_DataStruct_In	41.794195	41.794968	0.000773	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	41.794287	41.794856	0.000569		
	Runnable_Swc04_Odr_Impl_DataStruct_In	41.800671	41.801123	0.000452	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.011462
									0.000384	0.007604
	Runnable_Swc06_Odr_Impl_DataDIN2_In	41.825096	41.825181	0.000085	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	41.830085	41.830394	0.000309	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	41.830189	41.830299	0.000110		

11.7.19. V2_0_1111 write-process 24 to 25 ms

	Name	Entry in msec	End in ms	Duration in ms	Runnable Out	Entry in ms	End in ms	Duration in ms	Write function	Entry in ms	End in ms	Duration in ms
	Task_Event_Expl_Swc4to3_Outfunc	42.771070	42.778041	0.006971	Runnable_Swc04_Event_Expl_DataStruct_Out	42.771208	42.778011	0.006803	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Expl_DataStruct	42.774199	42.777899	0.001161
	Task_Event_Impl_Swc3to4_Outfunc	42.783993	42.788629	0.004636	Runnable_Swc03_Event_Impl_DataStruct_Out	42.784079	42.786189	0.002110	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Infunc	42.793670	42.794517	0.000847								
	Task_Event_Impl_Swc3to4_Infunc	42.799431	42.800509	0.001078								
24	Task_IoHw_Period_Expl_Swc6to5_Outfunc	42.805878	42.810334	0.004456	Runnable_Swc06_Period_Expl_DataDIN3_Out	42.805973	42.810321	0.004348	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	42.808169	42.810214	0.000265
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	42.815337	42.819485	0.004148	Runnable_Swc05_Period_Impl_DataDIN2_Out	42.815413	42.817331	0.001918	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Infunc	42.824419	42.824938	0.000519								
	Task_IoHw_Period_Impl_Swc5to6_Infunc	42.829759	42.830023	0.000264								
	osIdleLoop	42.894045	43.737456	0.843411								
	Task_Event_Impl_Swc3to4_Outfunc	43.779999	43.785825	0.005826	Runnable_Swc03_Event_Impl_DataStruct_Out	43.780108	43.782595	0.002487	Macros no functions	0.000000	0.000000	0.000000
	Task_Event_Expl_Swc4to3_Outfunc	43.791842	43.797244	0.005402	Runnable_Swc04_Event_Expl_DataStruct_Out	43.791905	43.797226	0.005321	Rte_Write_SwcEvent04_P_Swc04_DataStruct_Event_Ex	43.793980	43.797120	0.000928
	Task_Event_Impl_Swc3to4_Infunc	43.802485	43.803508	0.001023								
	Task_Event_Expl_Swc4to3_Infunc	43.808341	43.809136	0.000795								
	Task_IoHw_Period_Impl_Swc5to6_Outfunc	43.814114	43.818525	0.004411	Runnable_Swc05_Period_Impl_DataDIN2_Out	43.814210	43.816349	0.002139	Macros no functions	0.000000	0.000000	0.000000
	Task_IoHw_Period_Expl_Swc6to5_Outfunc	43.823300	43.827729	0.004429	Runnable_Swc06_Period_Expl_DataDIN3_Out	43.823376	43.827714	0.004338	Rte_Write_SwcIoHwPeriod06_P_Swc06_DataDIN3_Peri	43.825341	43.827605	0.000323
25	Task_IoHw_Period_Impl_Swc5to6_Infunc	43.832913	43.833270	0.000357								
	Task_IoHw_Period_Expl_Swc6to5_Infunc	43.838135	43.838746	0.000611								
	Task_Period_Impl_Swc1to2_Outfunc	43.843952	43.846528	0.002576	Runnable_Swc01_Period_Impl_DataStruct_Out	43.844163	43.844179	0.000016	Macros no functions	0.000000	0.000000	0.000000
	Task_Period_Expl_Swc2to1_Outfunc	43.851402	43.854587	0.003185	Runnable_Swc02_Period_Expl_DataStruct_Out	43.851483	43.854566	0.003083	Rte_Write_SwcPeriod02_P_Swc02_DataStruct_Period_Expl_DataStruct	43.851606	43.854456	0.000914
	Task_Period_Impl_Swc1to2_Infunc	43.859358	43.860351	0.000993								
	Task_Period_Expl_Swc2to1_Infunc	43.865302	43.866363	0.001061								
	osIdleLoop	43.914072	44.737563	0.823491								

11.7.20. V2_0_1111 notification process 24 to 25 ms

	osActivate Task Entry in ms	End in ms	Duration in ms	osCheckInterruptEnabled Entry in ms	End in ms	Duration in ms	osSchedule Prio Entry in ms	End in ms	Duration in ms	osDispatecher Entry in ms	End in ms	Duration	Notification Duration in ms (without ActivateTask)		Duration notification
24	42.775210	42.777749	0.002539											Task_Event_Expl_Swc4to3_Outfunc	
	42.786806	42.788610	0.001804	42.778633	42.778961	0.000328	42.780226	42.781085	0.000859	42.782120	42.783813	0.001693	0.002880	Task_Event_Impl_Swc3to4_Outfunc	
				42.789020	42.789272	0.000252	42.790169	42.790844	0.000675	42.791804	42.793486	0.001682	0.002609	Task_Event_Expl_Swc4to3_Infunc	0.012664
				42.794787	42.795025	0.000238	42.796090	42.796825	0.000735	42.797727	42.799245	0.001518	0.002491	Task_Event_Impl_Swc3to4_Infunc	0.007751
	42.808316	42.810096	0.001780	42.801061	42.801379	0.000318	42.802609	42.803436	0.000827	42.804226	42.805686	0.001460	0.002605	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
	42.817444	42.819470	0.002026	42.810590	42.810826	0.000236	42.811984	42.812732	0.000748	42.813623	42.815163	0.001540	0.002524	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
				42.819795	42.820044	0.000249	42.821053	42.821854	0.000801	42.822675	42.824222	0.001547	0.002597	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.011049
				42.825235	42.825572	0.000337	42.826500	42.827226	0.000726	42.828036	42.829572	0.001536	0.002599	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.007741
25	43.783433	43.785799	0.002366											Task_Event_Impl_Swc3to4_Outfunc	
	43.794733	43.796945	0.002212	43.786357	43.786670	0.000313	43.787935	43.788770	0.000835	43.789777	43.791636	0.001859	0.003007	Task_Event_Expl_Swc4to3_Outfunc	
				43.797601	43.797851	0.000292	43.798886	43.799662	0.000688	43.800529	43.802268	0.001574	0.002554	Task_Event_Impl_Swc3to4_Infunc	0.013329
				43.803952	43.804309	0.000266	43.805214	43.805927	0.000829	43.806712	43.808159	0.001359	0.002454	Task_Event_Expl_Swc4to3_Infunc	0.008243
	43.816461	43.818508	0.002047	43.809412	43.809627	0.000351	43.810727	43.811563	0.000797	43.812363	43.813899	0.001477	0.002625	Task_IoHw_Period_Impl_Swc5to6_Outfunc	
	43.825546	43.827487	0.001941	43.818876	43.819109	0.000243	43.820066	43.820745	0.000855	43.821566	43.823113	0.001547	0.002645	Task_IoHw_Period_Expl_Swc6to5_Outfunc	
				43.828025	43.828245	0.000296	43.829314	43.830025	0.000761	43.830979	43.832690	0.001577	0.002634	Task_IoHw_Period_Impl_Swc5to6_Infunc	0.011755
				43.833701	43.834099	0.000379	43.835108	43.835845	0.000731	43.836597	43.837940	0.001670	0.002780	Task_IoHw_Period_Expl_Swc6to5_Infunc	0.007712
	43.844764	43.846514	0.001750	43.839029	43.839252	0.000282	43.840537	43.841381	0.000916	43.842207	43.843740	0.001775	0.002973	Task_Period_Impl_Swc1to2_Outfunc	
	43.852403	43.854339	0.001936	43.846811	43.847052	0.000240	43.848036	43.848724	0.000744	43.849614	43.851181	0.001498	0.002482	Task_Period_Expl_Swc2to1_Outfunc	
				43.855012	43.855263	0.000228	43.856169	43.856843	0.000829	43.857726	43.859186	0.001444	0.002501	Task_Period_Impl_Swc1to2_Infunc	0.009918
				43.860681	43.861035	0.000228	43.861977	43.862769	0.000829	43.863644	43.865113	0.001444	0.002501	Task_Period_Expl_Swc2to1_Infunc	0.007931

11.7.21. V2_0_1111 read-process 24 to 25 ms

	Runnable In	Entry in ms	End in ms	Duration in ms	Read function	Entry in ms	End in ms	Duration in ms	Communication (write + read only) sum in ms	Communication (w,r,n)
									0.001737	0.014401
									0.000000	0.007751
	Runnable_Swc03_Odr_Expl_DataStruct_In	42.793732	42.794504	0.000772	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_Expl_DataStruct	42.793826	42.794402	0.000576		
	Runnable_Swc04_Odr_Impl_DataStruct_In	42.799970	42.800482	0.000512	*Macros no functions*	0.000000	0.000000	0.000000		
24									0.000409	0.011458
									0.000000	0.007741
	Runnable_Swc05_Odr_Expl_DataDIN3_In	42.824533	42.824923	0.000390	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	42.824676	42.824820	0.000144		
	Runnable_Swc06_Odr_Impl_DataDIN2_In	42.829921	42.830007	0.000086	*Macros no functions*	0.000000	0.000000	0.000000		
									0.000000	0.013329
									0.001463	0.009706
	Runnable_Swc04_Odr_Impl_DataStruct_In	43.803001	43.803485	0.000484	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc03_Odr_Expl_DataStruct_In	43.808399	43.809122	0.000723	Rte_Read_SwcEvent03_R_Swc03_DataStruct_Odr_E	43.808485	43.809020	0.000535		
									0.000000	0.011755
									0.000495	0.008207
25	Runnable_Swc06_Odr_Impl_DataDIN2_In	43.833133	43.833248	0.000115	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc05_Odr_Expl_DataDIN3_In	43.838278	43.838732	0.000454	Rte_Read_SwcIoHwPeriod05_R_Swc05_DataDIN3_Period_Expl_D_CAN0_D_Input_DIN3_Swi	43.838457	43.838629	0.000172		
									0.000000	0.009918
									0.001656	0.009587
	Runnable_Swc02_Odr_Impl_DataStruct_In	43.859864	43.860334	0.000470	*Macros no functions*	0.000000	0.000000	0.000000		
	Runnable_Swc01_Odr_Expl_DataStruct_In	43.865383	43.866349	0.000966	Rte_Read_SwcPeriod01_R_Swc01_DataStruct_Odr_Expl_DataStruct	43.865505	43.866247	0.000742		