Document Title	Specification of ECU State Manager with fixed state machine
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	444

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.1

Document Change History

Date	Release	Changed by	Change Description
2017-12-08	4.3.1	AUTOSAR Release Management	• Removal of unused artifacts of SWS_DIODriver • Marked the specification as obsolete
2016-11-30	4.3.0	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2015-07-31	4.2.2	AUTOSAR Release Management	• Adaptations related to renaming of DET • Table for "EcuM_SleepModeType" added • Missing modules in Table2 "Driver Initialization Details" added • Requirement regarding "state of wakeup sources belonging to previous sleep modes" added
2014-10-31	4.2.1	AUTOSAR Release Management	• Incorporation of MultiCore concept • Defined initialization order for InitListZero/InitListOne • Definition of the name pattern of c-init-data struct corrected • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Changed error handling in EcuM • Starting and stopping of WakeupSources on CAN now involves CanSM • Editorial changes

Document Change History

Date	Release	Changed by	Change Description
2013-10-31	4.1.2	AUTOSAR Release Management	• Changed behavior of EcuM_KillAllRUNRequests • Added API to kill POST_RUN requests • Reworked error classification • Editorial changes • Removed chapter(s) on change documentationf
2013-03-15	4.1.1	AUTOSAR Administration	• Reworked according to the new SWS_BSWGeneral • Reworked Production Errors • Fixed wakeup indication to ComM channels if several pending events undergoing • Extended and fixed configuration classes and -variants • Added service interface blueprints
2011-12-22	4.0.3	AUTOSAR Administration	• Re-integrated EcuM_GetState • EcuM_KillAllRUNRequests does no longer clear requests POST_RUN • EcuM_RequestPOST_RUN now accepts new requests during shutdown • Fixed include structure (Don't include Rte.h but Rte_EcuM.h) • EcuMEnableDefBehaviour is deprecated for EcuM fixed

Document Change History

Date	Release	Changed by	Change Description
2010-09-30	3.1.5	AUTOSAR Administration	• Bugfixing: • Removed obsolete interfaces (e.g. CanSM_EcuM) • Deleted interface to WdgM (SWS_EcuM_00861) • Added DET errors (EcuM_GetVersionInfo, EcuM_GetBootTarget, EcuM_GetShutdownTarget) • Changed polling mechanism in SLEEP SEQUENCE II state • Fixed transition from GOSLEEP state to WAKEUP Il state • Defined binding character of the Standardized AUTOSAR Interfaces (EcuM_StateRequest, EcuM_CurrentMode, EcuM_ShutdownTarget, EcuM_BootTarget) • Clarification: • Clarification under which circumstances the error hook will be called • Added note for EcuM_SelectBootTarget / EcuM_GetBootTarget because of the default boot target • Added Appendix A (help the application software programmer to understand when to request which mode) • Added note for exit from GO SLEEP state
2010-02-02	3.1.4	AUTOSAR Administration	• Initial Release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction

There are actually two variants of AUTOSAR ECU management: flexible and fixed. Fixed ECU management continues ECU management in the form of previous AUTOSAR releases. Flexible ECU management extends the previous versions of the ECU Manager.

The ECU State Manager module for flexible ECU State Management is specified in [24]. This document specifies the ECU State Manager module for fixed ECU State Management. This specification is obsolete and will be removed from the standard in an upcoming release.

1.1 Functional Overview

The ECU State Manager is a basic software module (see [1]). It manages all aspects of the ECU related to the OFF, RUN, and SLEEP states of that ECU and the transitions (transient states) between these states like STARTUP and SHUTDOWN.
In detail, the ECU State Manager Fixed module

- is responsible for the initialization and de-initialization of all basic software modules including OS and RTE,
- cooperates with the Communication Manager, and hence indirectly with network management, to shut down the ECU when needed,
- manages all wake up events and configures the ECU for SLEEP when requested.
- allows MultiCore applications without BSW distribution.

In order to fulfill all these tasks, the ECU State Manager Fixed module provides some important protocols:

- the RUN request protocol, which is needed to coordinate whether the ECU must be kept alive or is ready to shut down,
- the wake up validation protocol to distinguish ‘real’ wake up events from ‘erratic’ ones,
- the time triggered increased inoperation protocol (TTII), which allows to put the ECU into an increasingly energy saving sleep state over time.

These protocols were specified with the following underlying constraints:

- standardization at the API side, to allow applicability to all kinds of ECUs and portability of AUTOSAR applications
- high degree of flexibility to the low side interface, mainly reached by a set of callouts
- quick startup times
- consistent programming paradigm across all mode managing modules (rubber band model[1])

[1] As long as some entity requests run, the rubber band is stretched to the RUN state, and it snaps back when it is released. Since there is only one state (namely the RUN state) to which the rubber band applies, this term is not used any further in this specification. However, it is important to understand that, if applied to resource managers, the result is a powerful and consistent concept for enhancing state machines. The Communication Manager is a module which picks up the idea of the

Summarizing all this, the ECU State Manager Fixed module will be one of the principal state machines of an AUTOSAR compliant ECU, namely that one around states with the highest priority: RUN, SLEEP, and OFF. However, it does not and shall not in future contain functionality which might be related to terms like 'vehicle modes', 'error modes', or any other kind of application related kind of states or modes. These topics shall be addressed by other state machines (application mode managers).

resource manager and of the rubber band model and henceforce fits well into landscape spawn by the ECU State Manager.

1.2 Conventions Used in this Specification

1.2.1 Font Faces

SWS_EcuM_00023 Requirements are tagged with an ID in bold font.

References to other documents or to other chapters within this document are printed in italic.

`Source code` is printed in a Courier font.

`Configuration Parameters` are printed in Courier Italic.

STATE names are written in capital letters.

1.2.2 Figures

Figure X - Title (diagram type)

Figures are typically drawn in UML. To capture the hierarchical organization of the UML diagrams, some diagrams are classified in the title (diagram type). The following types are used:

- *Top level*
  An entry diagram to the structural or behavioral domain
- *High level*
  First degree of break down below the top level
- *SUB-STATE*
  The diagram describes the behavior of the given sub-state, the diagram type is the name of the sub-state
- no class
  All other diagrams, typically detail information

In the present version of this documentation, there is only one top level diagram: The main state machine, see Figure 2 – ECU Main States (top level diagram).
The next level is covered by high level diagrams. There are five high level sequence diagrams:

Figure 4 – Startup Sequence (high level diagram)
Figure 8 – RUN State Sequence (high level diagram)
Figure 12 – Shutdown Sequence (high level diagram)
Figure 17 – Sleep Sequence (high level diagram)
Figure 20 – Wake-up Sequence (high level diagram)

These high level diagrams give an overview of the major activities in the main state and explain how the state transitions occur. High level sequence diagrams always start with a diagram reference to the preceding sequence and end with a diagram reference to the following sequence.
High level diagrams are typically broken down into SUB-STATE diagrams. They show details which are irrelevant at the high level.

2 Definitions and Acronyms

Term	***Description***
Inoperation	An artificial word to describe the ECU when it is not operational, i.e. not running. Comprises all meanings of *off, sleeping, frozen*, etc. Using this definition is beneficial since it has no predefined meaning.
Shutdown Target	The shutdown of an ECU may end up in different states, depending on what application requires or desires for the next shutdown. By selecting a shutdown target, the application can communicate its wishes to the ECU State Manager. SLEEP, OFF, and RESET are shutdown targets.
Callout	Within this document, the term 'callout' is used for function stubs which can be filled by the system designer, usually at configuration time, with the purpose to add functionality to the ECU State Manager. Callouts are separated into two classes, where one class is optional to be filled. The other class is mandatory and serves as a hardware abstraction layer.
Passive Wake up	A wake up caused from an attached bus rather than an internal event like a timer or sensor activity.
Post run	Post run is the period from when the application detects a reason to start the shutdown until the shutdown actually occurs. Typically this period starts when all network communication is put to sleep and lasts until the ECU is put to sleep.
Vital Data	Any kind of data (RAM or NVRAM) that must stay consistent to ensure correct operation of the ECU. E.g. stacks, important state variables, etc.
Wake up Event	A physical event which causes a wake up. A CAN message or a toggling IO line can be wake up events. Similarly, the internal SW representation, e.g. an interrupt, may also be called a wake up event.
Wake up Reason	The wake up reason is the wakeup event being the actual cause of the last wake up.
Wake up Source	The peripheral or ECU component which deals with wake up events is called a wake up source.
Mode	A mode is a certain set of states of the various state machines that are running in the vehicle that are relevant to a particular entity, an application or the whole vehicle. The EcuM Mode is visible to the application. The EcuM Mode offers exactly the following options • STARTUP • RUN • SLEEP • WAKE_SLEEP • POST_RUN • SHUTDOWN
State	States are not visible to the application but are used by the EcuM-internal state machine that handels the Modes. The EcuM defines also sub-states, which are also not visible to the application (e.g. GO SLEEP, PREP SHUTDOWN, ...).

Acronym	***Description***
TTII	Time-Triggered Increased Inoperation
BswM	Basic Software Mode Manager
DEM	Diagnostic Event Manager
DET	Default Error Tracer
EcuM	ECU Manager
GPT	General Purpose Timer
ICU	Input Capture Unit
MCU	Microcontroller Unit

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] Requirements on Mode Management
AUTOSAR_SRS_ModeManagement.pdf

[5] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

None

3.3 Related AUTOSAR Software Specifications

[6] Glossary
AUTOSAR_TR_Glossary.pdf

[7] Specification of Communication Manager
AUTOSAR_SWS_ComManager.pdf

[8] Specification of Watchdog Manager
AUTOSAR_SWS_WatchdogManager.pdf

[9] Specification of CAN Interface
AUTOSAR_SWS_CANInterface.pdf

[10] Specification of LIN Interface
AUTOSAR_SWS_LINInterface.pdf

[11] Specification of FlexRay Interface
AUTOSAR_SWS_FlexRayInterface.pdf

[12] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager.pdf

[13] Specification of MCU Driver
AUTOSAR_SWS_MCUDriver.pdf

[14] Specification of SPI Handler/Driver
AUTOSAR_SWS_SPIHandlerDriver.pdf

[15] Specification of EEPROM Abstraction
AUTOSAR_SWS_EEPROMAbstraction.pdf

[16] Specification of Flash Driver
AUTOSAR_SWS_FlashDriver.pdf

[17] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[18] Specification of RTE
AUTOSAR_SWS_RTE.pdf

[19] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[20] Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.pdf

[21] Specification of CAN Transceiver Driver
AUTOSAR_SWS_CANTransceiverDriver.pdf

[22] Specification of C Implementation Rules
AUTOSAR_TR_CImplementationRules.pdf

[23] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[24] Specification of ECU Manager
AUTOSAR_SWS_ECUStateManager.pdf

[25] Specification of LIN Driver
AUTOSAR_SWS_LINDriver.pdf

[26] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

AUTOSAR provides a General Specification on Basic Software modules [5] (SWS BSW General), which is also valid for ECU State Manager Fixed.

Thus, the specification SWS BSW General shall be considered as additional and required specification for ECU State Manager Fixed.

4 Constraints and Assumptions

4.1 Limitations

Requirement: Applications (SW-C's) shall not assume that it is actually possible to switch off ECUs (i.e. power consumption is zero).
Rationale: The shutdown target OFF requires special hardware on the ECU so that it can actually be reached (e.g. a power hold circuit). If this hardware is not available, this specification proposes to issue a reset instead but other default behaviors can be defined.

This specification of the ECU State Manager with fixed state machine module does support Multicore only for the following use case: "Multicore without BSW distribution". All BSW modules, except EcuM, BswM and Os, which are also located on Slave Cores, are located on the Master Core. For the use case "Multicore with BSW distribution" the ECU State Manager with flexible state machine module shall be used.

4.2 Hardware Requirements

Requirement: ECU RAM shall keep contents of vital data while ECU clock is switched off.
Rationale: This requirement is needed to implement sleep states as required in *7.6* SLEEP State.

Requirement: ECU RAM shall provide a no-init area which keeps contents over a reset cycle.

Requirement: The no-init area in ECU RAM shall only be initialized on a power on event (clamp 30).

Requirement: The system designer is responsible for establishing an initialization strategy for the no-init area in ECU RAM.

4.3 Applicability to car domains

The ECU State Manager Fixed module is applicable to all car domains.

5 Dependencies to other Modules

The following sections outline the important relationships to other modules. They also contain some requirements that these modules have to fulfill to collaborate correctly with ECU State Manager.

5.1 Mode Management Modules

5.1.1 Communication Manager

The Communication Manager is a so-called ‘Resource Manager’[2] and thus requests RUN state. Resource Managers are described in chapter *7.2.3* Resource Managers.

The Communication Manager requests RUN state when it is leaving the ‘no communication’ state and it releases RUN when it is returning to this state.

5.1.2 Watchdog Manager

The Watchdog Manager is initialized by the ECU State Manager.

The ECU State Manager Fixed module does not set any Watchdog Manager Mode; this is considered to be handled via the BSW Mode Manager.

The ECU State Manager Fixed module is one of the Supervised Entities of the Watchdog Manager.

5.1.3 Basic Software Mode Manager

[SWS_EcuMf_00013] ⌈ The ECU State Manager Fixed module shall run in parallel to the Basic Software Mode Manager. ⌋ ()

[SWS_EcuMf_00014] ⌈ The ECU State Manager Fixed module shall indicate the current ECU Operation Mode to the BswM (`BswM_EcuM_CurrentState`). ⌋ ()

[SWS_EcuMf_00015] ⌈The ECU State Manager Fixed module shall indicate the current state of a wake up source to the BswM (`BswM_EcuM_CurrentWakeup`).⌋ ()

[SWS_EcuMf_00016] ⌈The ECU State Manager Fixed module shall initialize the BswM (`BswM_Init`)
⌋ ()
[SWS_EcuMf_00017] ⌈The ECU State Manager Fixed module shall de-initialize the BswM (`BswM_Deinit`).
⌋ ()

[2] 'Resource Manager' is invented in this specification to classify BSW modules which interact with Ecu State Manager.

5.2 SPAL Modules

5.2.1 MCU Driver

The MCU Driver is the first basic software module initialized by the ECU State Manager. However, returning `MCU_Init`, the MCU and the MCU driver are not necessarily fully initialized. Additional, MCU specific steps may be needed. The ECU State Manager Fixed module provides callouts where this additional code can be placed, see chapter 8.7.2. For details on how this code should look like refer to [13].

5.2.2 Driver Dependencies and Initialization Order

BSW drivers may depend on each other. A typical example is the watchdog driver which needs the SPI driver to access an external watchdog. This means on the one hand, that drivers may be stacked (not relevant to the ECU State Manager Fixed module) but on the other hand that the underlying driver needs to be initialized first.

The system designer is responsible for defining the initialization order of the BSW drivers at configuration time.

5.3 Peripherals with Wake-up Capability

Wake up sources have to be handled and encapsulated by drivers. The implementation must follow the protocols and requirements presented in this document to ensure a seamless integration into AUTOSAR BSW.
To support the wake up and validation protocol, the driver has to fulfill the following requirements:

The driver has to notify ECU State Manager Fixed module by invoking the EcuM_SetWakeupEvent service once when a wake up event is detected. The same service should also be invoked during initialization of the driver if a pending wake up event is detected during the initialization.

The driver shall provide an explicit service to put the wake up source to sleep. This service shall put the wake up source into an energy saving and inert operation mode and re-arm the wake up notification mechanism.

If the wake up source is capable of generating faulty events[3] then the driver or the software stack consuming the driver or another appropriate BSW module shall either provide a validation callout for the wake up event under validation or directly call the wake up validation service of the ECU State Manager. If validation is not necessary, then this requirement is not applicable for the according wake up source.

[3] Faulty wakeup events may result from EMV spikes, bouncing effects on wakeup lines etc.

5.4 Operating System

ECU State Manager Fixed module starts and shuts down the AUTOSAR OS. It also defines the protocol how control is handed over to the OS after its startup and how control is handed back to the ECU State Manager Fixed module when the OS is shut down.

5.5 Runtime Environment (RTE)

Requirement: The initialization and de-initialization functions of the RTE are assumed to return.

The ECU State Manager Fixed module shall use the mode port feature of the RTE to notify about mode changes. See *chapter 8.2* Service Interfaces for more information.

5.6 BSW Scheduler

The ECU State Manager Fixed module has a twofold relation with the BSW Scheduler. It initializes the BSW Scheduler and it also contains scheduled functions. EcuM_MainFunction is scheduled to periodically evaluate run requests.

5.7 NVRAM Manager

The following operations of the NVRAM Manager [12] are executed by the ECU State Manager Fixed module .

- Initialization of NVRAM Manager after a power up or reset of the ECU
- Read-back of non-volatile data from NVRAM to ECU RAM during the initialization of the ECU
- In case of SLEEP state, storing of non-volatile data to NVRAM may prematurely be terminated upon wakeup events to ensure a quick restart of the ECU.

The ECU State Manager Fixed module does not read NVRAM during the wake up sequence since RAM contents is assumed to be still valid from the previous cycle. To verify this, the ECU State Manager Fixed module offers services to check RAM integrity[4]. The ECU State Manager Fixed module does only read NVRAM during the STARTUP phase.

The NVRAM Manager shall call the callbacks defined in chapter *8.6.1* Callbacks from NVRAM Manager to notify the ECU State Manager Fixed module about job status.

[4] See *8.7.4.6* EcuM_GenerateRamHash and *8.7.5.1* EcuM_CheckRamHash for details.

5.8 Diagnostic Event Manager

The DEM module requires the NVRAM Manager module to be operational. The DEM module is aware if the NVRAM Manager module is operational or provides limited functionality. These differences are handled within the DEM module.

5.9 Network Management

[SWS_EcuMf_00022] ⌈The initialization process has to guarantee that NM is initialized, so the ECU is not set into sleep mode if AUTOSAR CAN Generic NM is not initialized.⌋ ()

[SWS_EcuMf_00023] ⌈ Initialization of NM is only allowed after the initialization of the respective bus interface. ⌋ ()

Implementation hint: The integrator may call `Nm_Init` inside the callout EcuM_AL_DriverInitThree.

5.10 Other Basic Software Modules

[SWS_EcuMf_00028] ⌈The integrator shall place initialization code for Basic Software Modules not already mentioned in this specification in the callouts EcuM_AL_DriverInitZero, EcuM_AL_DriverInitOne, EcuM_AL_DriverInitTwo, or EcuM_AL_DriverInitThree
⌋ ()

5.11 Software Components

The ECU State Manager Fixed module handles two ECU-wide settings/variables:

- OS application modes[5]
- Setting of shutdown targets

It is assumed in this specification that these properties are set by the application (through AUTOSAR ports, represented by service interfaces), typically by some ECU specific part of the application. The ECU State Manager Fixed module does not prohibit an application overriding settings of other applications. The policy must be defined at a higher level.

The following two requirements formulate an attempt to resolve this issue.

The SW-C Template may specify a field whether the SW-C sets the shutdown target.

[5] In this context, 'application mode' is a technical term which is defined by the AUTOSAR OS specification.

The generation tool may only allow configuration that have only one SW-C accessing shutdown target.

5.12 File Structure

5.12.1 Code file structure

[SWS_EcuM_02990] ⌈The implementation of the ECU State Manager Fixed module shall provide one file `EcuM_Callout_Stubs.c` containing the stubs of the defined callouts.⌋ ()

Whether this file `EcuM_Callout_Stubs.c` has to be modified directly or includes other generated files is specific to the implementation.

5.12.2 Header file structure

[SWS_EcuM_00991] ⌈The implementation of the ECU State Manager Fixed module shall provide one file `EcuM.h` containing fix type declarations, forward declaration to generated types, and function prototypes.⌋ ()

[SWS_EcuM_02992] ⌈The implementation of the ECU State Manager Fixed module shall provide one file `EcuM_Generated_Types.h` containing generated types and fulfilling the forward declarations from `EcuM.h`.⌋ ()

[SWS_EcuM_02677]⌈ It shall only be necessary to include `EcuM_Cbk.h` to interact with the callbacks and callouts of the ECU State Manager.⌋ ()

[SWS_EcuM_00676]⌈ It shall only be necessary to include `EcuM.h` to use all services of the ECU State Manager.⌋ ()

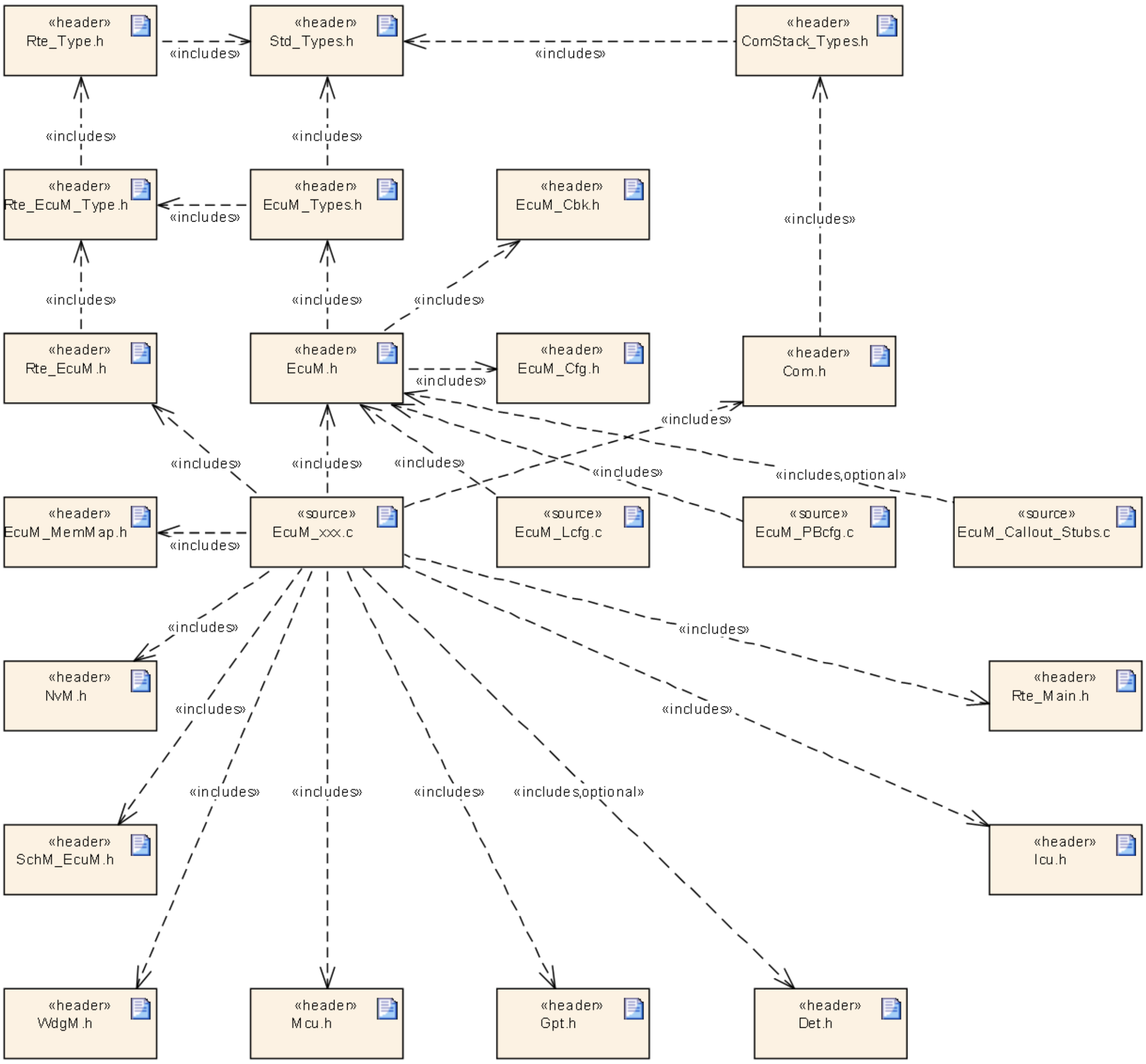


Figure 1 – Header file structure

[SWS_EcuM_02875] ⌈The ECU State Manager Fixed module shall include the `Dem.h` file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in `Dem_IntErrId.h`. ⌋ ()

Also refer to chapter *8.8* Expected Interfaces for dependencies to other modules.

6 Requirements traceability

Requirement	Description	Satisfied by

7 Functional Specification

7.1 Main States of the ECU State Manager

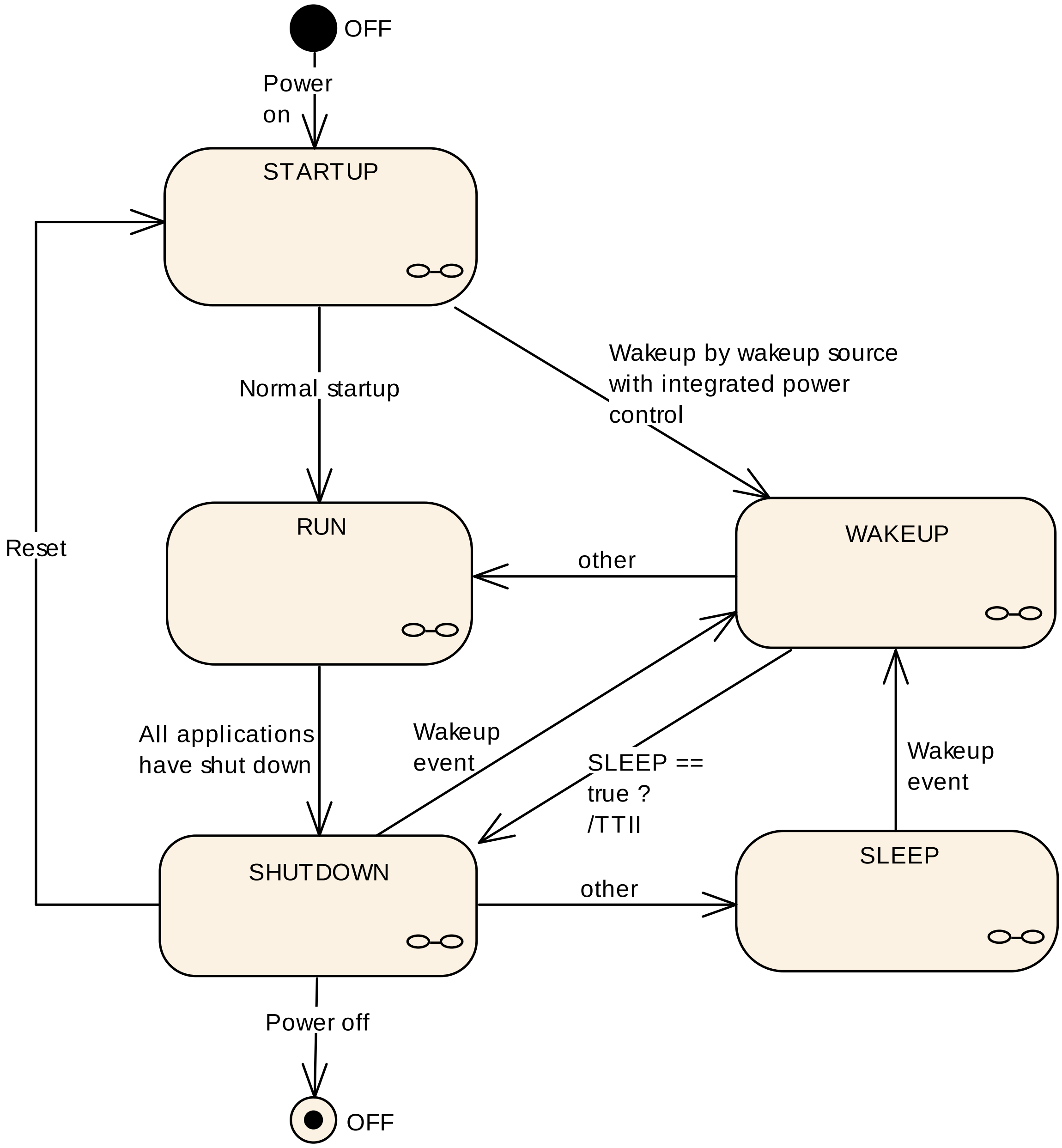


Figure 2 – ECU Main States (top level diagram)

Figure 2 – ECU Main States (top level diagram) shows the main state machine provided by the ECU State Manager Fixed module . This state machine manages the ‘life cycle’ of an ECU from `OFF` through `STARTUP` and `RUN` to `SLEEP` or `OFF`.

Please refer to the following chapters and to 8.2.3.1 Data Types for the relevant substates.

7.1.1 STARTUP State

The purpose of the STARTUP state is to initialize the basic software modules. The STARTUP state is divided into two parts, the first being the part before OS startup, the second part after OS startup (and therefore with a running OS). More details about the initialization are given in chapter *7.3* STARTUP State.

7.1.2 RUN State

The RUN State is entered by the ECU State Manager Fixed module after all modules of basic software including OS and RTE have been initialized by the ECU State Manager Fixed module .
The RUN State indicates to the SW-C's above RTE that BSW has initialized and applications start operating. Further, the RUN state provides a mechanism for synchronized shutdown of application software.
RUN state must be requested by the application explicitly or implicitly[6] whenever it is needed to keep the ECU awake. Otherwise, the ECU State Manager Fixed module will commence shutdown. In other words: a SW-C has to request the RUN state from the ECU State Manager Fixed module when the ECU needs to stay awake.
The RUN State falls into two sub-states: The regular RUN state and a POST_RUN state. The POST_RUN state can be requested by SW-C's to indicate that the need to execute cleanup or saving activities before the ECU goes to sleep. The POST_RUN state can be requested independently from the RUN state with a separate API or via System Services accordingly[7].
SW-C's shall react on state changes by interfacing with the mode port of the ECU State Manager Fixed module.
If the SW-C's primary intent is to communicate with other SW-C's, the SW-C has to request a communication state from the Communication Manager module instead.

7.1.3 SHUTDOWN State

The SHUTDOWN state handles the controlled shutdown of basic software modules and finally results in the selected shutdown target for the ECU: SLEEP, OFF, or Reset. An important activity in this state is to write non-volatile data back to NVRAM.

[6] RUN state is requested implicitly if a non-idle state is requested from a Resource Manager. E.g. requesting any state but 'no communication' from the Communication Manager will have the Communication Manager requesting RUN state from the ECU State Manager in turn. This is a request for communication which implicitly results in a request for RUN state. See also [6].

[7] In this specification RUN and POST RUN sub-states are called RUN II and RUN III.

7.1.4 SLEEP State

The `SLEEP` state is an energy saving state. Typically, no code is executed but power is still supplied, and if configured accordingly, the ECU is wakeable in this state[8]. The `SLEEP` state provides a configurable set of sleep modes which typically are a trade off between power consumption and time to restart the ECU. In terms of the API, the sleep modes are referred to as *shutdown targets*.

7.1.5 WAKEUP State

The `WAKEUP` State is entered when the ECU comes out of the SLEEP state, due to intended or unintended wake up.

The `WAKEUP` State provides a protocol to support validation of wake up events. This is necessary to differentiate between intended und unintended wake-ups. The validation itself is a cooperative process between the driver which handles the wake up source and the ECU State Manager Fixed module (see *7.8* Wake-up Validation Protocol).

7.1.6 OFF State

The `OFF` state describes the unpowered ECU. Wakeability may be required in this state but only for wake up sources with integrated power control. In any case the ECU must be startable (e.g. by reset events).

[8] Some ECU designs actually do require code execution to implement a SLEEP state (and the wakeup capability). For these ECUs, the clock speed is typically dramatically reduced. These could be implemented with a small loop inside the SLEEP state.

7.2 Structural Description of the ECU State Manager

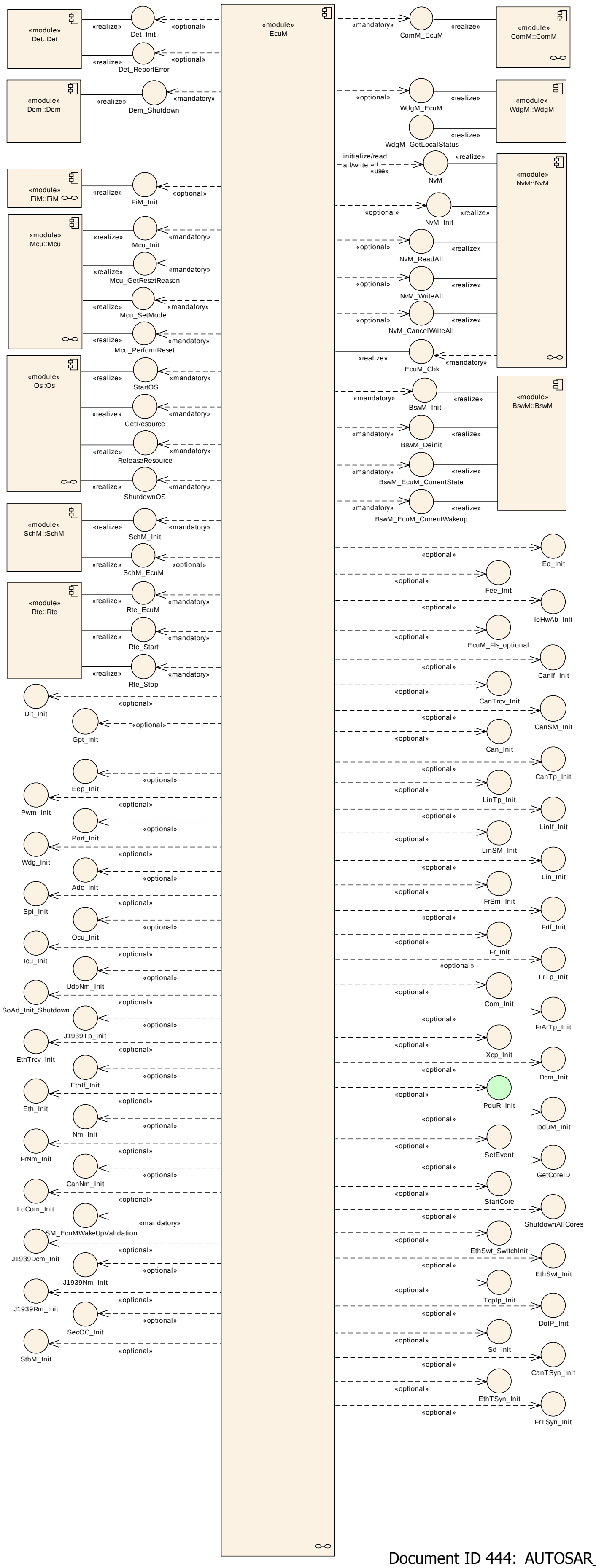

«module»
EcuM
«module»
Det::Det
Det_Init
Det_ReportError
«module»
Dem::Dem
Dem_Shutdown
«module»
FiM::FiM
FiM_Init
«module»
Mcu::Mcu
Mcu_Init
Mcu_GetResetReason
Mcu_SetMode
Mcu_PerformReset
«module»
Os::Os
StartOS
GetResource
ReleaseResource
ShutdownOS
«module»
SchM::SchM
SchM_Init
SchM_EcuM
«module»
Rte::Rte
Rte_EcuM
Rte_Start
Rte_Stop
Dlt_Init
Gpt_Init
Eep_Init
Pwm_Init
Port_Init
Wdg_Init
Adc_Init
Spi_Init
Ocu_Init
Icu_Init
UdpNm_Init
SoAd_Init_Shutdown
J1939Tp_Init
EthTrcv_Init
EthIf_Init
Eth_Init
Nm_Init
FrNm_Init
CanNm_Init
LdCom_Init
SM_EcuMWakeUpValidation
J1939Dcm_Init
J1939Nm_Init
J1939Rm_Init
SecOC_Init
StbM_Init
«module»
ComM::ComM
ComM_EcuM
«module»
WdgM::WdgM
WdgM_EcuM
WdgM_GetLocalStatus
initialize/read all/write all
«use»
NvM
«module»
NvM::NvM
NvM_Init
NvM_ReadAll
NvM_WriteAll
NvM_CancelWriteAll
EcuM_Cbk
«module»
BswM::BswM
BswM_Init
BswM_Deinit
BswM_EcuM_CurrentState
BswM_EcuM_CurrentWakeup
Ea_Init
Fee_Init
IoHwAb_Init
EcuM_Fls_optional
CanIf_Init
CanTrcv_Init
CanSM_Init
Can_Init
CanTp_Init
LinTp_Init
LinIf_Init
LinSM_Init
Lin_Init
FrSm_Init
FrIf_Init
Fr_Init
FrTp_Init
Com_Init
FrArTp_Init
Xcp_Init
Dcm_Init
PduR_Init
IpduM_Init
SetEvent
GetCoreID
StartCore
ShutdownAllCores
EthSwt_SwitchInit
EthSwt_Init
TcpIp_Init
DoIP_Init
Sd_Init
CanTSyn_Init
EthTSyn_Init
FrTSyn_Init
«realize»
«optional»
«mandatory»

Figure 3 – Module Relationship (top level diagram)

Figure 3 shows how the ECU State Manager Fixed module is related to other modules. In most cases, the ECU State Manager Fixed module is simply responsible for initialization[9]. There are however some modules that have a functional relationship with the ECU State Manager Fixed module which are explained in the following paragraphs.

7.2.1 Standardized AUTOSAR Software Modules

Basic Software modules are initialized and shut down by the ECU State Manager.
The RTE is initialized and shut down by the ECU State Manager.

The OS is initialized and shut down by the ECU State Manager.

After the OS initialization, additional initialization steps are undertaken by the ECU State Manager Fixed module before the RUN state is reached. Execution control is handed over to the ECU State Manager Fixed module after OS shutdown. Details are provided in the chapters *7.3* STARTUP State and *7.5* SHUTDOWN State.

7.2.2 Software Components

SW Components contain the application code of an AUTOSAR ECU. Software Components shall request the RUN state from the ECU State Manager Fixed module when they have the need to keep the ECU alive.
If the intent of the SW-C is primarily to communicate then it should request a communication state from the Communication Manager (see [6]). This will implicitly keep the ECU alive. A SW-C should clearly separate between the need to communicate and the need to keep an ECU alive. Mixing up these two ideas may result in an instable shutdown algorithm.
A SW-C interacts with the ECU State Manager Fixed module using AUTOSAR ports which are mapped to system Services of the ECU State Manager Fixed.

7.2.3 Resource Managers

The concept of resource managers allows adding new state machines to the BSW (as a part of new BSW modules) which behave like sub-state machines of the RUN state.

In order to collaborate correctly with the ECU State Manager Fixed module only very few requirements must be met:

A Resource Manager has to define exactly one idle state that signifies the state where the Resource Manager isn't doing anything but waiting.

[9] To be precise, “initialization” could also mean de-initialization.

A Resource Manager has to transit into its idle state after initialization. It shall request the RUN state from the ECU State Manager Fixed module whenever it leaves its idle state and it shall release the RUN state when it returns back to its idle state.

The Communication Manager module is one such resource manager.

7.3 STARTUP State

See *7.1.1* STARTUP State for an overview description.

7.3.1 High Level Sequence Diagram

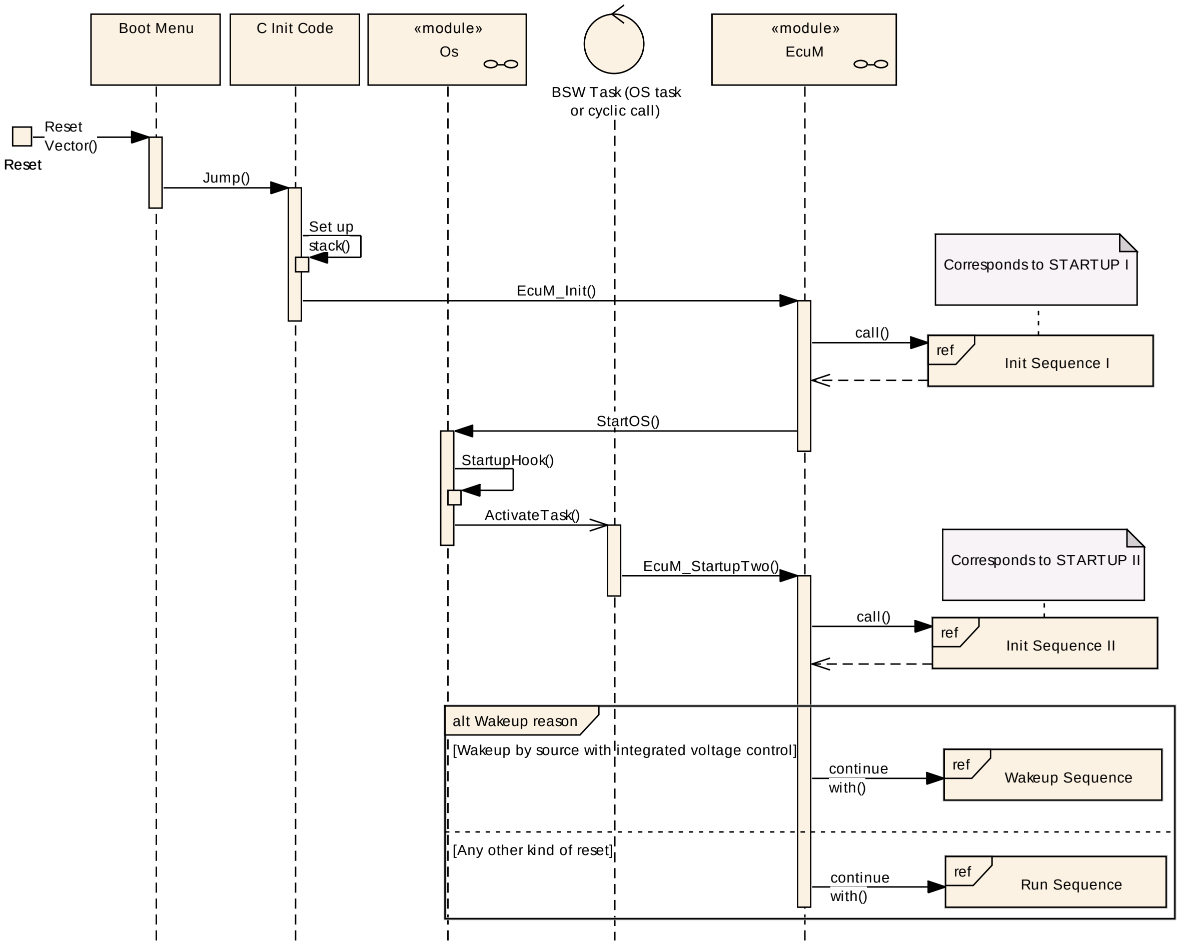


Figure 4 – Startup Sequence (high level diagram)

To see adjacent diagrams refer to
Figure 8 – RUN State Sequence
Figure 2 – ECU Main States (top level diagram)

The startup sequence in Figure 4 shows the startup behavior of the ECU. With the invocation of EcuM_Init the ECU State Manager Fixed module takes control of the startup procedure. The startup of the ECU falls into two parts.

The first part, init sequence I or STARTUP I is finished when the AUTOSAR OS is started.

The second part, init sequence II or STARTUP II is finished when RTE is started.

To distinguish services that are called before the OS is started from those that are called afterwards and to have a cleaner visualization, the ECU State Manager Fixed module is split into two parts: The initialization of the ECU State Manager Fixed module (started with a call to EcuM_Init(), which runs without OS, and the EcuM.

7.3.2 Activities before EcuM_Init

The ECU State Manager Fixed module assumes that before EcuM_Init is called a minimal initialization of the MCU has taken place, so that a stack is set up and code can be executed.

7.3.3 STARTUP Activity Overview

[SWS_EcuMf_00007] ⌈ Initialization dependencies (as defined in the BSWMDs) have to be respected. ⌋ ()

Example (to show when ComM_CommunicationAllowed shall be called):
ComM_Init()
CanXX_Init()
ComM_CommunicationAllowed(<CAN channel>, true)
And reverse for DeInit

[SWS_EcuM_02411] ⌈ The following table shows the Startup activities and the order in which they shall be executed.

Sub-state ***Initialization Activity***[10]	***Comment***	***Opt.***[11]
STARTUP I		
Callout EcuM_AL_DriverInitZero	Init block 0 This callout may only initialize BSW modules that do not use post-build configuration parameters. The callout may not only contain driver initialization but any kind of pre-OS, low level initialization code. See *7.3.5* Driver Initialization	yes
Callout EcuM_DeterminePbConfiguration	This callout is expected to return a pointer to a fully initialized EcuM_ConfigType structure containing the post-build configuration data for EcuM and all other BSW modules.	no
Check consistency of configuration data	If check fails the EcuM_ErrorHook is called. See *10.4* Checking Configuration Consistency for details on the consistency check.	No
Callout EcuM_AL_DriverInitOne	Init block I The callout may not only contain driver initialization but any kind of pre-OS, low level initialization code. See *7.3.5* Driver Initialization	Yes
Get reset reason	The reset reason is derived from a call to `Mcu_GetResetReason` and the mapping defined via	No

[10] Activities marked with × are conditional.
[11] Optional activities can be switched on or off by configuration. See chapter *10.2* Published for details.

Sub-state ***Initialization Activity***[10]	***Comment***	***Opt.***[11]
	the EcuMWakeupSource configuration containers. See *8.6.2.2* EcuM_SetWakeupEvent and *8.4.4.3* EcuM_GetValidatedWakeupEvents.	
Select default shutdown target	See SWS_EcuM_02181	No
Start OS	Start the AUTOSAR OS, see SWS_EcuM_02603	No
STARTUP II		
Init BSW Scheduler	Initialize the semaphores for critical sections used by BSW modules	No
Callout EcuM_AL_DriverInitTwo	Init block II The callout may only initialize BSW modules that need OS support but don't need access to private NvRam data (other that post-build configuration data in their <Module>_ConfigType) or manage that data on their own. See *7.3.5* Driver Initialization	Yes
Callout EcuM_OnRTEStartup		No
Start RTE	From now on SW-Cs are running. RTE will signal the (initial) mode STARTUP during start.	No
Callout EcuM_AL_DriverInitThree	Init block III The callout may initialize BSW modules that need OS support and rely on their private NvRam data (other that post-build configuration data in their <Module>_ConfigType) to be restored. See *7.3.5* Driver Initialization	Yes
Indicate mode change to RTE	Indicated mode is SLEEP if next state is WAKEUP VALIDATION, indicated mode is RUN if next state is RUN.	No

Table 1 - Initialization Activities

⌋ ()

[SWS_EcuM_02623] ⌈ The ECU State Manager shall remember the wake up source resulting from the reset reason translation (see Table 1 - Initialization Activities).
⌋ ()

7.3.4 Sub-State Descriptions

7.3.4.1 STARTUP I

The STARTUP I state is entered with a call of the API function EcuM_Init.

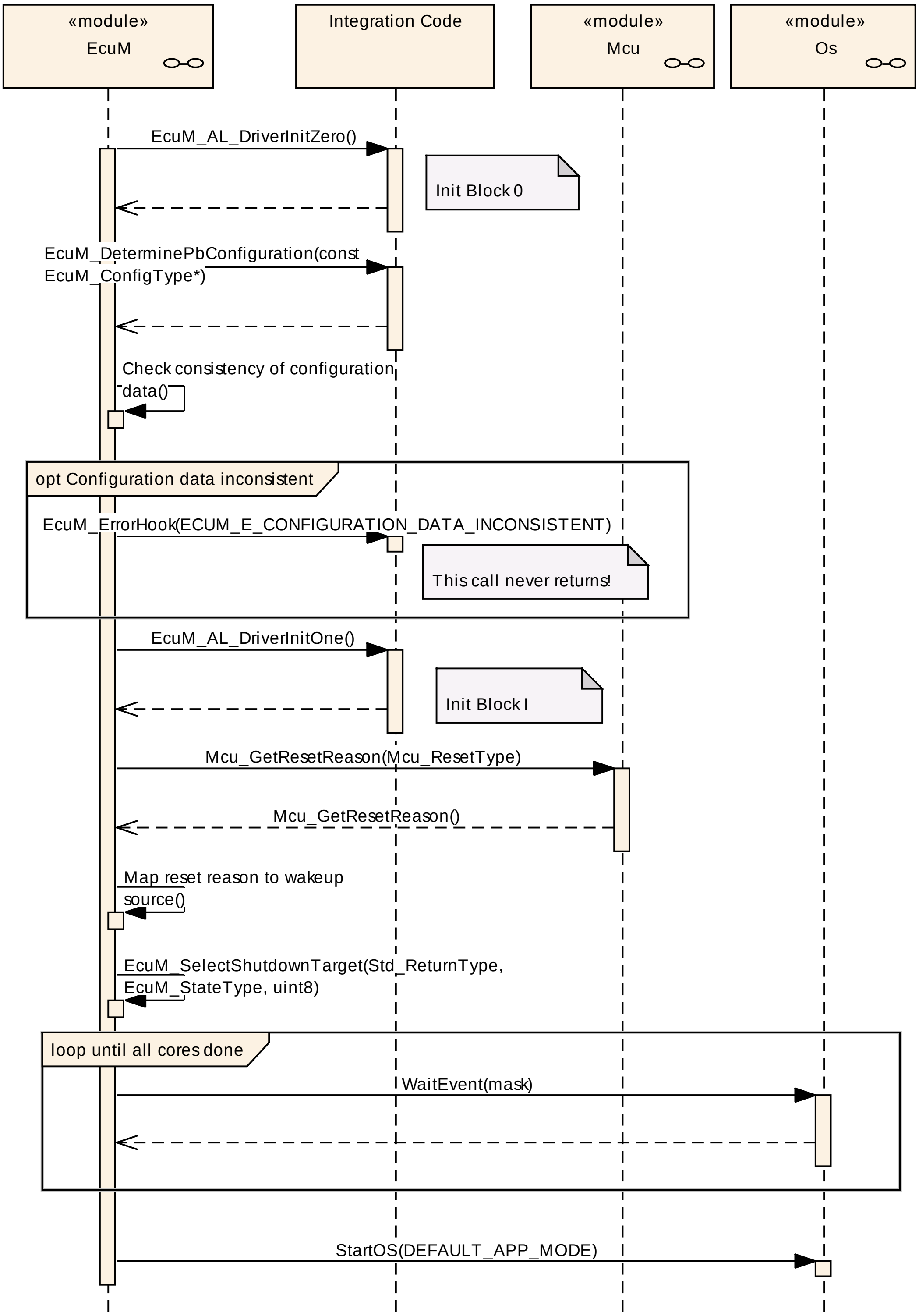

«module»
EcuM
Integration Code
«module»
Mcu
«module»
Os
EcuM_AL_DriverInitZero()
Init Block 0
EcuM_DeterminePbConfiguration(const EcuM_ConfigType*)
Check consistency of configuration data()
opt Configuration data inconsistent
EcuM_ErrorHook(ECUM_E_CONFIGURATION_DATA_INCONSISTENT)
This call never returns!
EcuM_AL_DriverInitOne()
Init Block I
Mcu_GetResetReason(Mcu_ResetType)
Mcu_GetResetReason()
Map reset reason to wakeup source()
EcuM_SelectShutdownTarget(Std_ReturnType, EcuM_StateType, uint8)
loop until all cores done
WaitEvent(mask)
StartOS(DEFAULT_APP_MODE)

Figure 5 – Init Sequence I (STARTUP I)

STARTUP I is intended for preparing the ECU to initialize the OS. The phase should be kept as short as possible. This also applies to the callouts. Initialization of drivers should be done in STARTUP II whenever possible. Interrupts should not be used in this phase. If interrupts have to be used, only category I interrupts are allowed in STARTUP phase 1[12].
Initialization of drivers and hardware abstraction modules is not strictly defined by the ECU State Manager. Two callouts EcuM_AL_DriverInitZero and EcuM_AL_DriverInitOne are provided to define the init blocks 0 and I. These blocks are initialization activities during STARTUP I, where initialization can take place. Modules needing OS support can be placed into init blocks II or III (see *7.3.4.2* STARTUP II).

`MCU_Init` does not provide complete MCU initialization. Additionally, hardware dependent steps have to be executed and must be defined at system design time. These steps are supposed to be taken within the EcuM_AL_DriverInitZero or EcuM_AL_DriverInitOne callouts. Details can be found in [13].

[SWS_EcuM_02181] ⌈ECU State Manager Fixed module must call EcuM_SelectShutdownTarget with the configured default shutdown target (see *7.6.2.1* Shutdown Targets, *7.9* Time Triggered Increased Inoperation and 10.3 Configurable Parameters. ⌋ ()

[SWS_EcuM_02603] ⌈At the end of the STARTUP I state, the ECU State Manager starts the OS. All basic software modules which are needed by the OS shall be initialized by this time. Modules left out so far may be initialized later in STARTUP II. ⌋ ()

Note: If a Watchdog Manager is configured and initialized in any of the Init Blocks, the integration code shall call the Watchdog Manager often enough to ensure correct operation of the ECU during the transient states.

For the handling of the functionalities during `SLEEP` see [8] Chapter 7.4.

[12] Category II interrupts require a running OS while category I interrupts do not. AUTOSAR OS requires each interrupt vector to be exclusively put into one category.

7.3.4.2 STARTUP II

STARTUP II is carried out by the EcuM_StartupTwo API function.

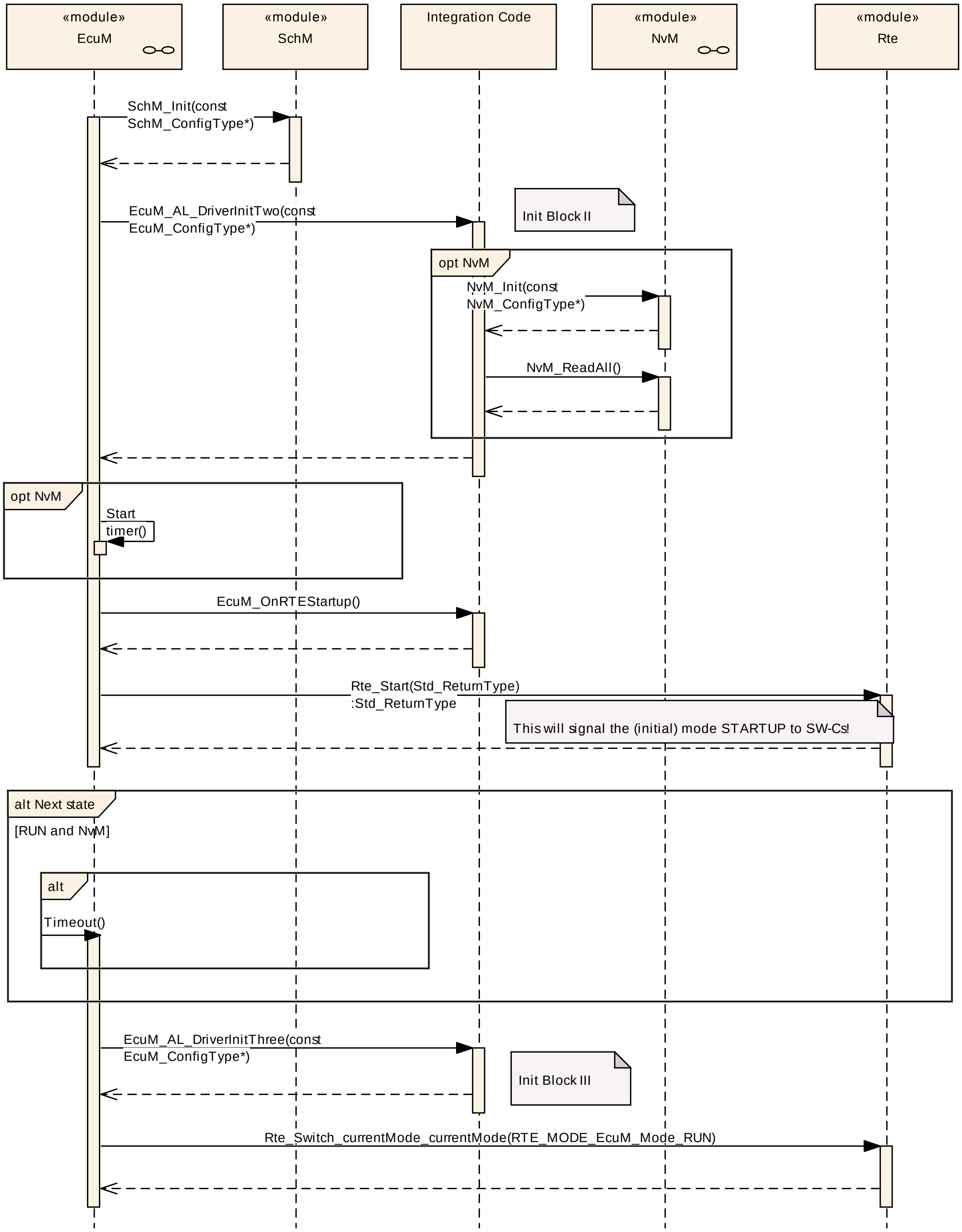

«module»
EcuM
«module»
SchM
Integration Code
«module»
NvM
«module»
Rte
SchM_Init(const SchM_ConfigType*)
EcuM_AL_DriverInitTwo(const EcuM_ConfigType*)
Init Block II
opt NvM
NvM_Init(const NvM_ConfigType*)
NvM_ReadAll()
opt NvM
Start timer()
EcuM_OnRTEStartup()
Rte_Start(Std_ReturnType) :Std_ReturnType
This will signal the (initial) mode STARTUP to SW-Cs!
alt Next state
[RUN and NvM]
alt
Timeout()
EcuM_AL_DriverInitThree(const EcuM_ConfigType*)
Init Block III
Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_RUN)

Figure 6 – Init Sequence II (STARTUP II)

The callout EcuM_AL_DriverInitTwo is provided, where initialization of those basic software modules should take place, which need OS support and need no access to NvRam data or manage the NvRam data on their own.

The callout EcuM_AL_DriverInitThree is provided, where initialization of those basic software modules should take place, which need OS support and need NvRam data to be completely restored.

[SWS_EcuM_00632] ⌈If one of the wake up sources listed in *7.8.7* Wake up Sources and Reset Reason is set, then exection shall continue with RUN state. In all other cases, execution shall continue with WAKEUP VALIDATION state. ⌋ ()

7.3.5 Driver Initialization

This chapter applies to drivers of the AUTOSAR Basic Software that are not handled directly by the ECU State Manager Fixed module.
A driver's location in the initialization process depends strongly on its implementation and the target hardware design. Drivers can be initialized from the driver init blocks I and II during STARTUP I and II respectively.

[SWS_EcuM_02559] ⌈The order inside of the blocks shall be generated from configuration information (see 10.3 Configurable Parameters `EcuMDriverInitListZero`, `EcuMDriverInitListOne`, `EcuMDriverInitListTwo`, `EcuMDriverInitListThree`, and `EcuMDriverRestartList`). ⌋ ()

[SWS_EcuM_02730] ⌈For each driver, its init function with the configured init configuration shall be called. The init parameter for the init function shall be derived from driver's configuration (see 10.3 Configurable Parameters `EcuMFixedModuleConfigurationRef`). ⌋ ()

Some drivers may need re-initialization when the ECU is woken up. This is especially true for drivers with wake up sources. For re-initialization, a restart block is defined. The restart block is part of the `WAKEUP` state.

[SWS_EcuM_02561] ⌈The restart list will typically only contain a subset of drivers. But drivers shall appear in the same order as in the combined list of init block I and init block II (see 10.3 Configurable Parameters, `EcuMDriverRestartList`). ⌋ ()

[SWS_EcuM_02562] ⌈Drivers which serve as wake up sources may need to be re-initialized in the restart block. The driver restart shall re-arm the trigger mechanism of the 'wake up detected' callback (see *7.7.4.1* WAKEUP I). ⌋ ()

[SWS_EcuM_02563] ⌈If hardware is put into a sleep mode during `SHUTDOWN` then this hardware must be restarted by its driver.
The restart list will be invoked in state WAKEUP I (see *7.1.5* WAKEUP State). ⌋ ()

The following table shows one possible (and recommended) sequence of activities for the Init Blocks 0, I, II, and III. Depending on hardware and software configuration, BSW modules may be added or left out and other sequences may also be possible.

Recommended Init Block

Init Activity	*Comment*
Init Block 0[13]	
Default Error Tracer	This always needs to be the first module to be initialized, so that other modules can report development errors.
Any drivers needed to access post-build configuration data	These drivers may themselves not need post-build configuration or OS features.
Init Block I[14]	
MCU Driver	
PORT	
Diagnostic Event Manager	
General Purpose Timer	Pre-Initialization
Watchdog Driver	
Watchdog Manager	Internal watchdogs only, external ones may need SPI
SchM	
BswM	
ADC Driver	
ICU Driver	
PWM Driver	
OCU Driver	
OCU Driver	
Init Block II[15]	
SPI Driver	
EEPROM Driver	
Flash Driver	
NVRAM Manager	
CAN Transceiver	Initialization and start NvM_ReadAll job
CAN Driver	
CAN Interface	
CAN State Manager	
CAN TP	
LIN Driver	
LIN Interface	
LIN State Manager	
LIN TP	
FlexRay Transceiver	
FlexRay Driver	
FlexRay Interface	
FlexRay State Manager	
FlexRay TP	
Synchronized Time-Base Manager	
Time Sync Over CAN	
Time Sync Over Ethernet	
Time Sync Over FlexRay	
Ethernet Driver	
Ethernet Interface	
Ethernet State Manager	

[13] Drivers in Init Block 0 are listed in the `EcuMDriverInitListZero` configuration container.
[14] Drivers in Init Block I are listed in the `EcuMDriverInitListOne` configuration container.
[15] Drivers in Init Block II are listed in the `EcuMDriverInitListTwo` configuration container.

Recommended Init Block		
	Init Activity	***Comment***
	Ethernet Switch Driver Ethernet Transceiver Driver TCP/IP Stack SAE J1939 Diagnostic Communication Manager SAE J1939 Request Manager SAE J1939 Transport Layer Service Discovery Socket Adaptor Large Data COM PDU Router CAN NM FlexRay NM NM Interface UDP Network Management SAE J1939 Network Management Diagnostic over IP I-PDU Multiplexer COM Diagnostic Communication Manager Diagnostic Communication Manager	
Init Block III[16]		
	Communication Manager Diagnostic Event Manager Function Inhibition Manager Secure Onboard Communication	Full initialization

Table 2 - Driver Initialization Details, Sample Configuration

[SWS_EcuMf_00037]⌈ Depending on the post-build tooling approach described in the TPS_EcuConfiguration document, a BSW module shall be configured using EcuMModuleParameter, EcuMModuleRef and EcuMModuleService of EcuMDriverInitItem.
⌋ ()

Example of Dem Initialization in EcuM_DriverInitItemsOne:

Configuration parameters:
- EcuMDriverInitItem: DemPreInit
    - EcuMModuleParameter: VOID
    - EcuMModuleRef: ..../EcucModuleConfigurationValues/Dem
    - EcuMModuleServiceId: "PreInit"

- EcuMDriverInitItem: DemInit

[16] Drivers in Init Block III are listed in the *EcuMDriverInitListThree* configuration container.

- EcuMModuleRef: .../EcucModuleConfigurationValues/Dem
- EcuMServiceId: "" => Can be empty because it's Init
- EcuMModuleParameter: "POSTBUILD_PTR"

Resulting code in DriverInitListOne:
Dem_PreInit();
Dem_Init(&Dem_Config);

[SWS_EcuM_02719] ⌈A configuration tool shall fill the callout EcuM_AL_DriverRestart with initialization calls to the listed drivers in the order in which they occur in the list. ⌋ ()

[SWS_EcuM_02720] ⌈Entries in this list must appear in the same order as in the combined list of EcuM_DriverInitListOne and EcuM_DriverInitListTwo. This list may be a real subset though. In all other cases, the generation tool shall report an error. ⌋ ()

7.3.6 DET Initialization

The Default Error Tracer is a software module for debugging purposes.

[SWS_EcuM_02783] ⌈DET shall be initialized early during STARTUP I by the ECU State Manager. ⌋ ()

[SWS_EcuM_02634] ⌈DET is not *started* by default but the system designer has to configure the point where DET is started, preferably into one of the callouts EcuM_AL_DriverInitOne or EcuM_AL_DriverInitTwo. The best point for starting DET depends on its implementation and behavior. DET is started by invoking `Det_Start`. ⌋ ()

7.4 RUN State

See *7.1.2* RUN State for an overview description.
All activities in the RUN state described in this chapter are carried out in the EcuM_MainFunction service.

7.4.1 State Breakdown Structure

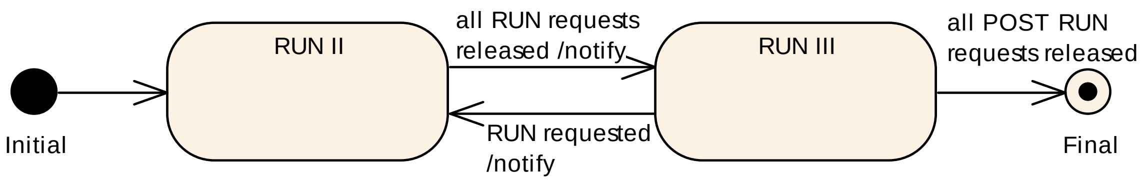

Initial
RUN II
all RUN requests released /notify
RUN requested /notify
RUN III
all POST RUN requests released
Final

Figure 7 – RUN State Breakdown

7.4.2 High Level Sequence Diagram

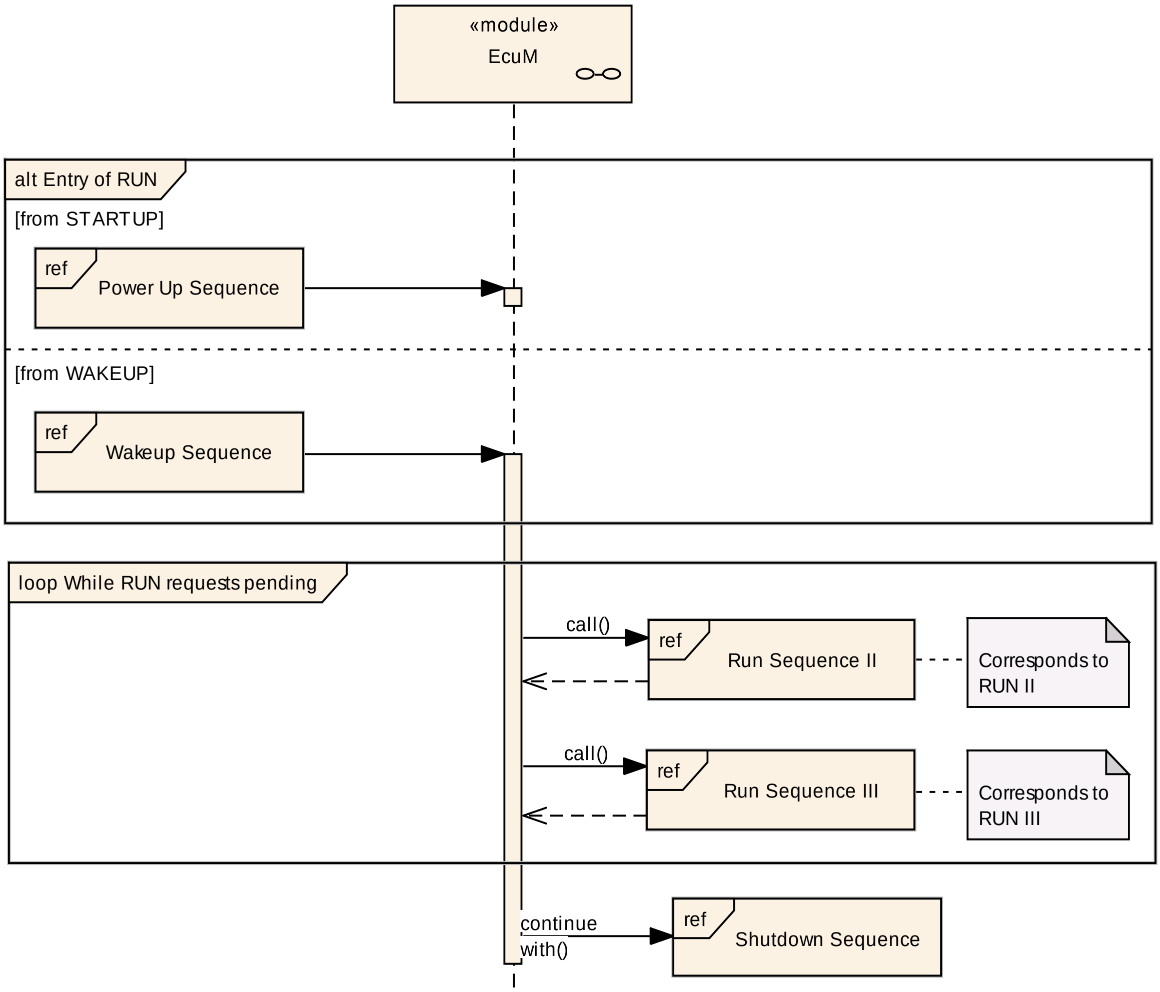


Figure 8 – RUN State Sequence (high level diagram)

To see adjacent diagrams refer to
Figure 4 – Startup Sequence (high level diagram)
Figure 20 – Wake-up Sequence (high level diagram)
Figure 12 – Shutdown Sequence (high level diagram)
Figure 2 – ECU Main States (top level diagram)

7.4.3 Sub-State Description

7.4.3.1 RUN II

RUN II is the state in which SW-C's should execute their regular tasks.

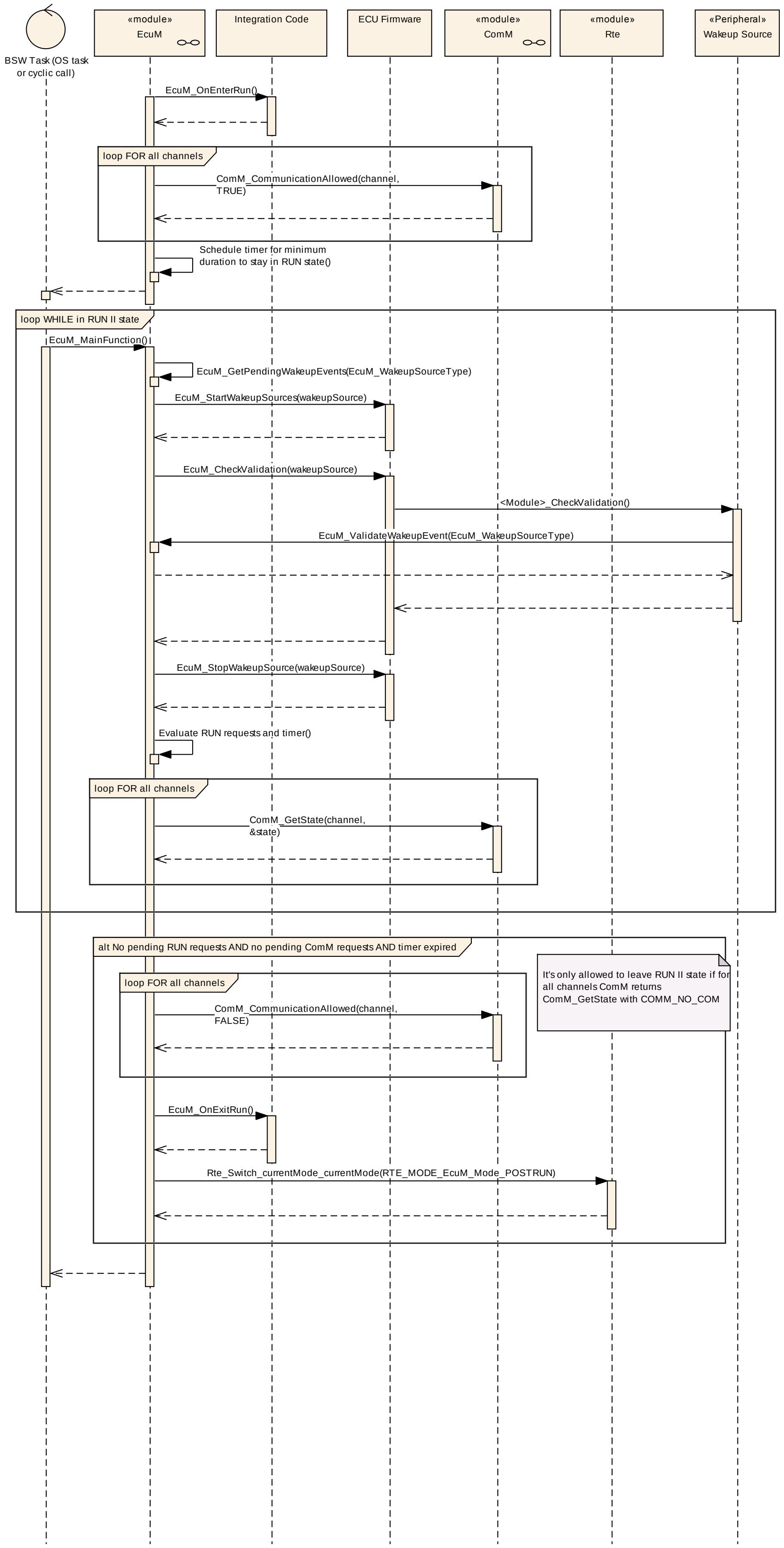

BSW Task (OS task or cyclic call)
«module» EcuM
Integration Code
ECU Firmware
«module» ComM
«module» Rte
«Peripheral» Wakeup Source
EcuM_OnEnterRun()
loop FOR all channels
ComM_CommunicationAllowed(channel, TRUE)
Schedule timer for minimum duration to stay in RUN state()
loop WHILE in RUN II state
EcuM_MainFunction()
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
EcuM_StartWakeupSources(wakeupSource)
EcuM_CheckValidation(wakeupSource)
<Module>_CheckValidation()
EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType)
EcuM_StopWakeupSource(wakeupSource)
Evaluate RUN requests and timer()
loop FOR all channels
ComM_GetState(channel, &state)
alt No pending RUN requests AND no pending ComM requests AND timer expired
It's only allowed to leave RUN II state if for all channels ComM returns ComM_GetState with COMM_NO_COM
loop FOR all channels
ComM_CommunicationAllowed(channel, FALSE)
EcuM_OnExitRun()
Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_POSTRUN)

Figure 9 – RUN II State Sequence

7.4.3.2 Entering RUN II State

On entering RUN II state, the steps as shown in Figure 9 must be performed.

[SWS_EcuM_00308] ⌈When entering RUN II state, the callout EcuM_OnEnterRun shall be invoked and RUN mode shall be indicated. ⌋ ()

[SWS_EcuMf_00008] ⌈ The ECU State Manager Fixed module shall call `ComM_CommunicationAllowed` when it becomes possible to communicate on that channel (parameter `TRUE`) and when it becomes impossible to communciate (parameter `FALSE`).
Exception is LIN communication in "sleep mode"⌋ ()

[SWS_EcuMf_00018] ⌈The call to `ComM_CommunicationAllowed` shall be configurable (It must be possible to define within the configuration for which ComM channels the ECU State Manager Fixed module should call `ComM_CommunicationAllowed`.)⌋ ()

[SWS_EcuMf_00019] ⌈For all channels for which this is defined, the ECU State Manager Fixed module must call `ComM_CommunicationAllowed(channel, TRUE)` immediately after entering RUN mode.⌋ ()

[SWS_EcuMf_00020] ⌈For all channels for which this is defined, the ECU State Manager Fixed module must call `ComM_CommunicationAllowed(channel, FALSE)` immediately before leaving RUN mode.⌋ ()

[SWS_EcuM_00310] ⌈The ECU State Manager Fixed module shall remain in `RUN` state for a configurable minimum duration (see 10.3 Configurable Parameters parameter EcuMRunMinimumDuration). ⌋ ()

The minimum duration of RUN state is needed to give the SW-Cs a chance to request `RUN`. Otherwise the ECU State Manager Fixed module will immediately leave `RUN` again.

[SWS_EcuMf_00027] ⌈ECU in `RUN` state shall also perform wake up validation of sleeping busses
⌋ ()

7.4.3.3 Leaving RUN II State

[SWS_EcuM_00311] ⌈When the last RUN request has been released, ECU State Manager Fixed module shall advance to the RUN III state. The evaluation is done with the next cyclic invocation of EcuM_MainFunction. ⌋ ()

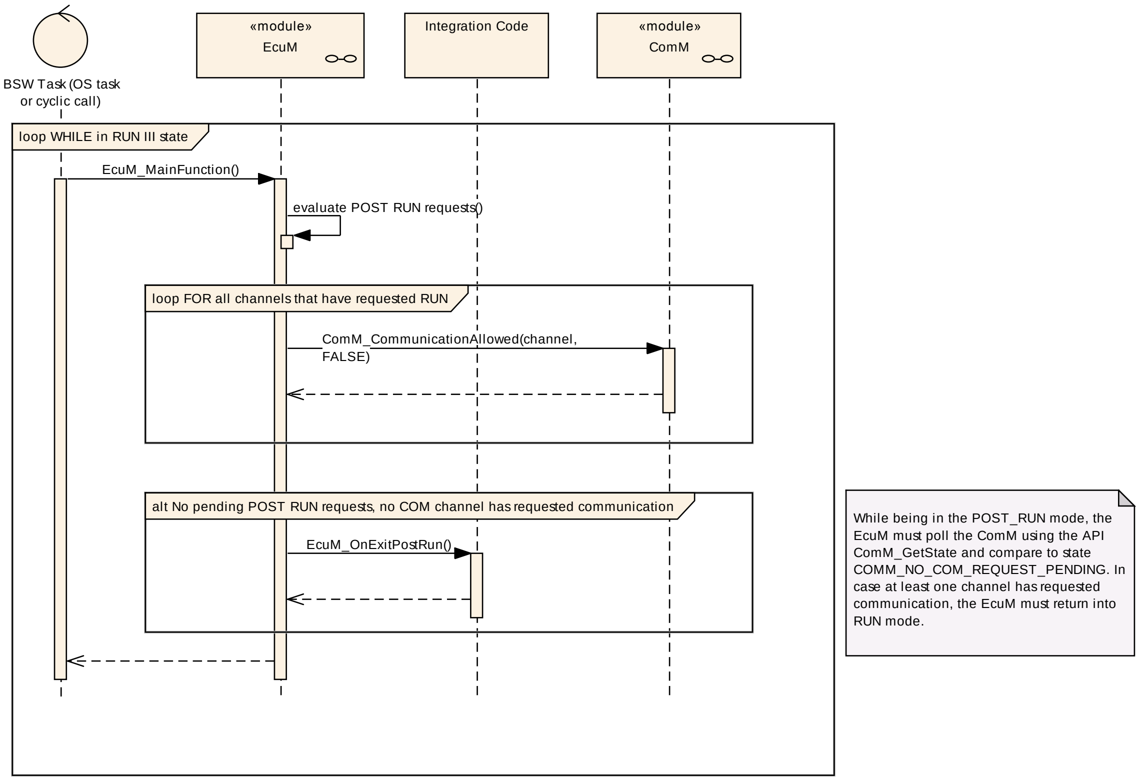
[SWS_EcuM_00865] ⌈When leaving RUN II state, the callout EcuM_OnExitRun shall be invoked and POST_RUN mode shall be indicated. ⌋ ()

If a SW-C needs post run activity during RUN III (e.g. shutdown preparation), then it must request POST_RUN before releasing the RUN request. Otherwise it is not guaranteed that this SW-C will get a chance to run its POST_RUN code.

The Communication Manager will not release `RUN` unless the no communication state is reached.

7.4.3.4 RUN III

RUN III state provides a post run phase for SW-C's and allows them to save important data or switch off peripherals before the ECU State Manager Fixed module continues with the shutdown process.


BSW Task (OS task or cyclic call)
«module» EcuM
Integration Code
«module» ComM
loop WHILE in RUN III state
EcuM_MainFunction()
evaluate POST RUN requests()
loop FOR all channels that have requested RUN
ComM_CommunicationAllowed(channel, FALSE)
alt No pending POST RUN requests, no COM channel has requested communication
EcuM_OnExitPostRun()

While being in the POST_RUN mode, the EcuM must poll the ComM using the API ComM_GetState and compare to state COMM_NO_COM_REQUEST_PENDING. In case at least one channel has requested communication, the EcuM must return into RUN mode.

Figure 10 – RUN III State Sequence

7.4.3.5 Leaving RUN III State

[SWS_EcuM_00761] ⌈When the last POST_RUN request has been released and no RUN request has been issued, the ECU State Manager Fixed module shall advance to the SHUTDOWN state and shall invoke the callout EcuM_OnExitPostRun. The evaluation is done with the next cyclic invocation of EcuM_MainFunction. ⌋ ()

[SWS_EcuM_00866] ⌈While in RUN III state, if a RUN request is received, the ECU State Manager Fixed module shall immediately enter RUN II state again. ⌋ ()

7.5 SHUTDOWN State

Refer to *7.1.3* SHUTDOWN State for an overview description.

[SWS_EcuM_02188] ⌈When `SHUTDOWN` state is entered and shutdown target is `SLEEP`, no wake up event shall be missed. If a valid wake up event occurs while the ECU is in transition to `SLEEP` the ECU shall as quickly as possible proceed to the `WAKEUP` state and shall not enter the `SLEEP` state. ⌋ ()

[SWS_EcuM_02756] ⌈When a wake up event occurs during the shutdown phase and the shutdown target is `OFF` or `RESET`, then the shutdown shall complete but the ECU shall restart immediately thereafter. ⌋ ()

7.5.1 State Breakdown Structure

When the `SHUTDOWN` state is entered, applications have de-initialized and the communication stack has been put into the no communication state[17]. Please refer to *7.4.3.3* Leaving RUN II State for details.

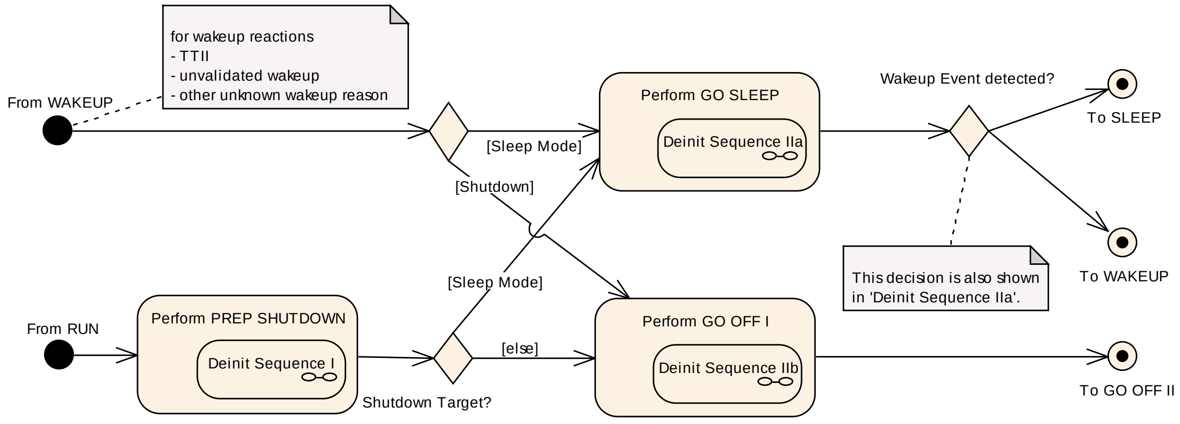


Figure 11 – Fine Structure of SHUTDOWN

[17] This statement is only true for SW-Cs which are registered users of the ECU State or Communication Manager. All other SW-C may be terminated by the system without warning.

7.5.2 High Level Sequence Diagram

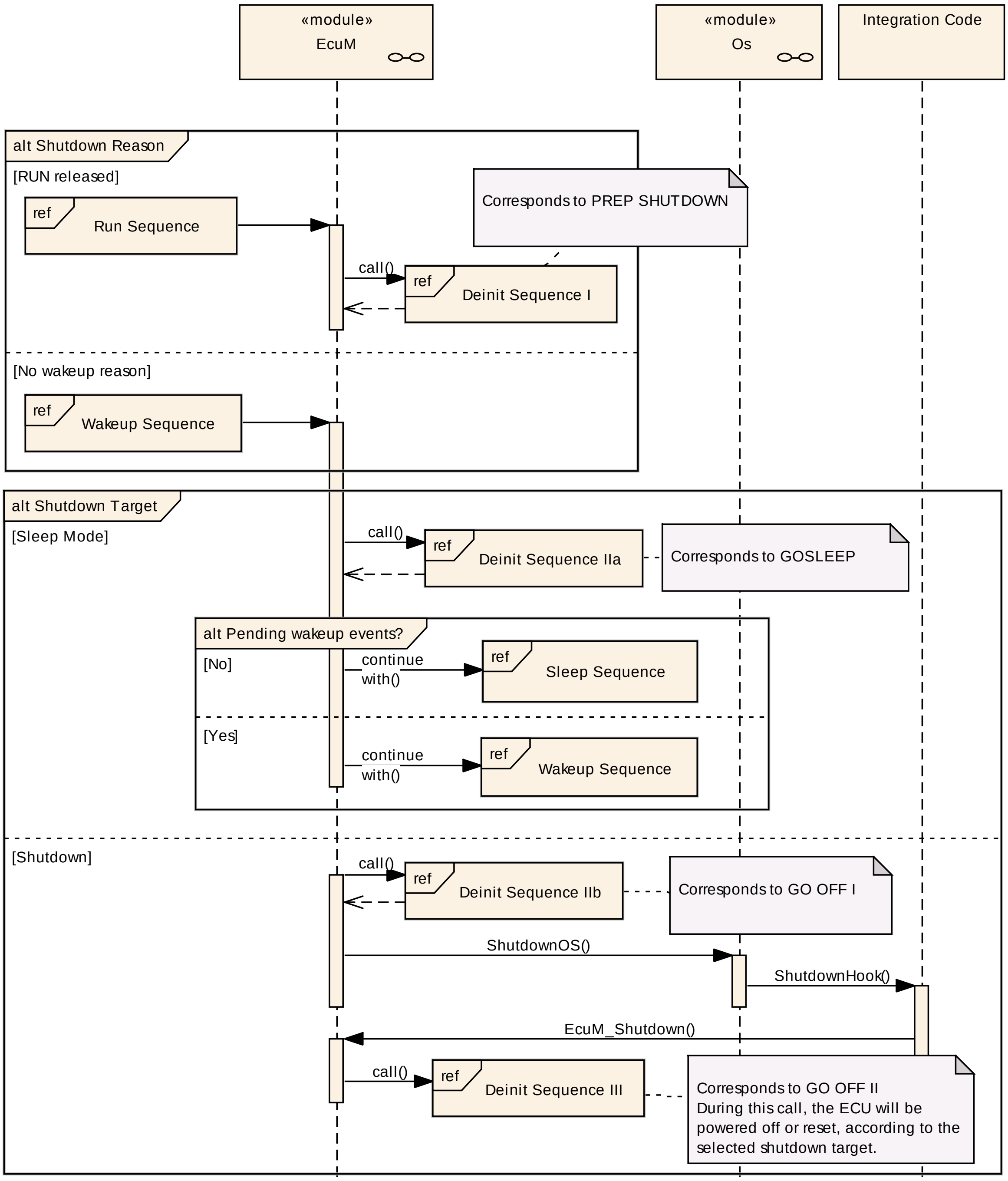


Figure 12 – Shutdown Sequence (high level diagram)

To see adjacent diagrams refer to

Figure 8 – RUN State Sequence (high level diagram)
Figure 20 – Wake-up Sequence (high level diagram)
Figure 17 – Sleep Sequence (high level diagram)
Figure 2 – ECU Main States (top level diagram)

7.5.3 SHUTDOWN Activity Overview

Sub-state[18]	***Shutdown Activity***	***Comment***	***Optional***[19]
PREP SHUTDOWN			
	Callout `EcuM_OnPrepShutdown`		
	Shutdown Diagnostic Event Manager		yes
	Indicate mode change to RTE	Indicated mode is `SLEEP` if next state is GO SLEEP, indicated mode is `SHUTDOWN` if next state is GO OFF I.	
GO SLEEP			
	Callout EcuM_OnGoSleep		
	Save persistent data to NVRAM	An incoming wake up event will cancel an ongoing write job	yes
	Check for pending wake up events	Purpose is to detect wake up events that occurred while interrupts were disabled	
	Callout EcuM_EnableWakeupSources	See EcuM_EnableWakeupSources	
	Lock Scheduler	Prevent other tasks from running in SLEEP state.	
GO OFF I			
	Callout `EcuM_OnGoOffOne`		
	Stop RTE		
	Deinit Communication Manager		yes
	Save persistent data to NVRAM		yes
	Check for pending wake up events	Purpose is to detect wake up events that occurred during shutdownd	
	Set RESET as shutdown target	This action shall only be carried out when pending wake up events were detected	yes
	ShutdownOS	Last operation in this OS task	
GO OFF II			
	Callout `EcuM_OnGoOffTwo`		
	Call Mcu_PerformReset or *Callout* `EcuM_AL_SwitchOff`	Depends on the selected shutdown target (RESET or OFF)	
The following modules need not to be shut down:			
	NVRAM Manager		
All other modules are not shutdown automatically. The following basic software modules must not be shut down at all.			
	None		

Table 3 - Shutdown Activities

[18] Rows marked with × are conditional.

[19] Optional activities can be switched on or off by configuration. It shall be the system designer's choice if a module is compiled in or not for an ECU design. See chapter *10.2* Published for details.

7.5.4 Sub-State Descriptions

7.5.4.1 PREP SHUTDOWN

PREP SHUTDOWN is a state common for all shutdown targets, i.e. `SLEEP`, `OFF`, reset, etc. During this state, handlers and managers of the basic software are shut down.

[SWS_EcuM_00288] **⌈**If the shutdown target is not any of the sleep modes, then control has to be handed over to GO OFF I (see *7.5.4.3* GO OFF I) after activities of this state have finished. ⌋ ()

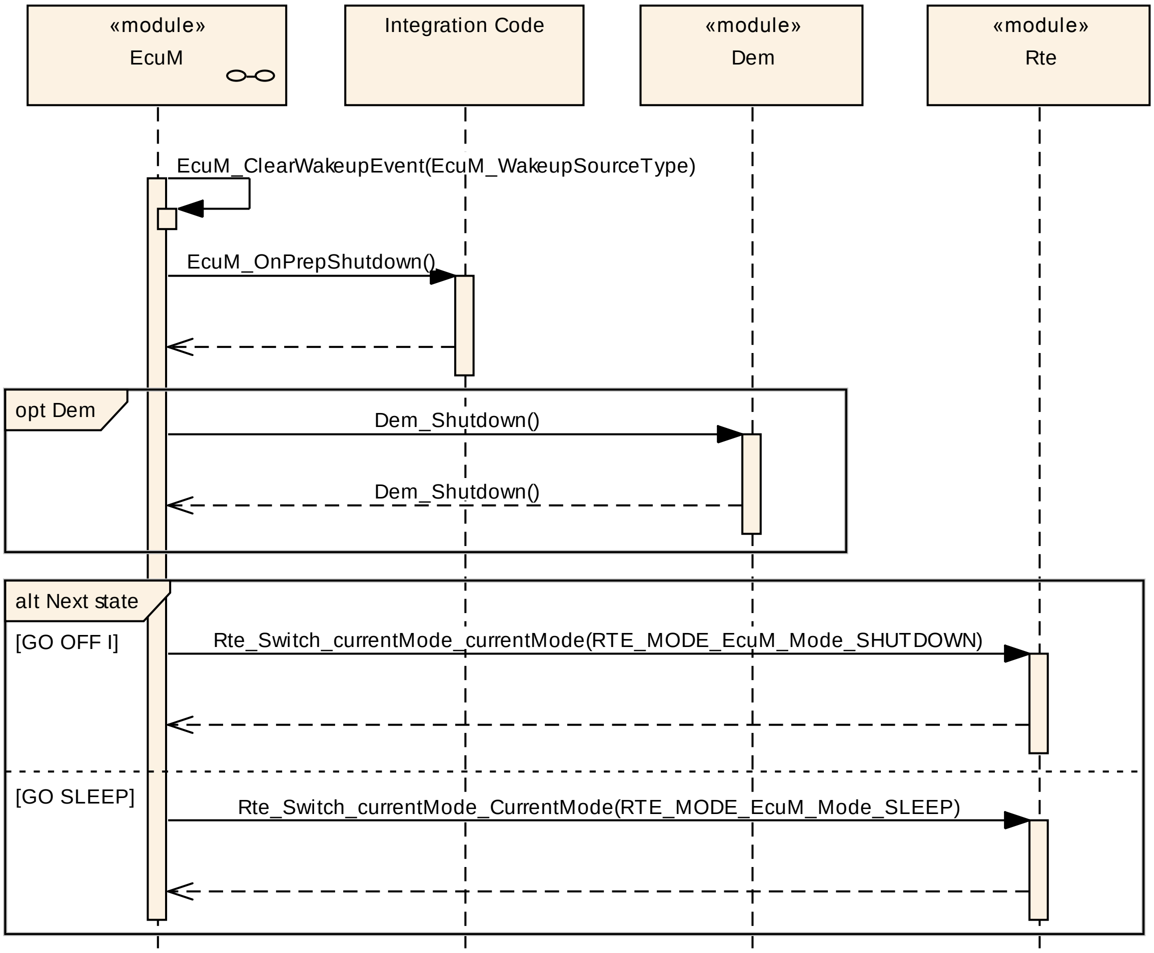

«module»
EcuM
Integration Code
«module»
Dem
«module»
Rte
EcuM_ClearWakeupEvent(EcuM_WakeupSourceType)
EcuM_OnPrepShutdown()
opt Dem
Dem_Shutdown()
Dem_Shutdown()
alt Next state
[GO OFF I]
Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_SHUTDOWN)
[GO SLEEP]
Rte_Switch_currentMode_CurrentMode(RTE_MODE_EcuM_Mode_SLEEP)

Figure 13 – Deinitialization Sequence I (PREP SHUTDOWN)

7.5.4.2 GO SLEEP

Purpose of GO SLEEP is to configure hardware for the following sleep phase and to setup the ECU for the next wake up event.

[SWS_EcuM_02389] ⌈To set up the wake up sources for the next sleep mode, the ECU State Manager Fixed module shall execute the callout EcuM_EnableWakeupSources for each wake up source that is configured in the target sleep mode.⌋ ()

In contrast to shutdown, the OS is not shut down when entering the sleep state. The sleep mode shall be transparent to the OS.

Note:
In case of pending wake up events, after calling `NvM_CancelWriteAll()` the transition shall go to WAKEUP VALIDATION as for the "Power On Sequence" (see also Figure 32).

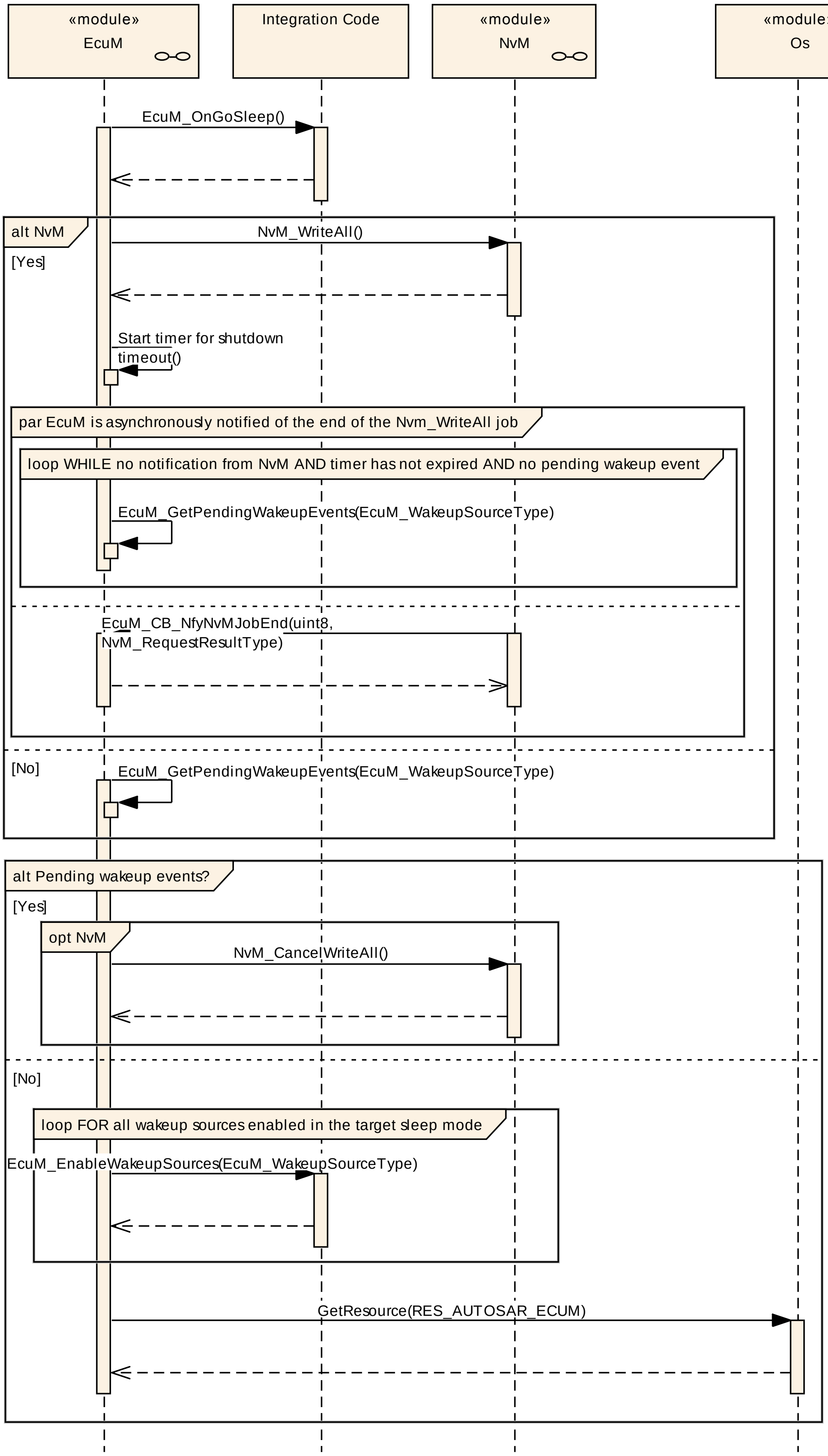

«module»
EcuM
Integration Code
«module»
NvM
«module»
Os
EcuM_OnGoSleep()
alt NvM
[Yes]
NvM_WriteAll()
Start timer for shutdown
timeout()
par EcuM is asynchronously notified of the end of the Nvm_WriteAll job
loop WHILE no notification from NvM AND timer has not expired AND no pending wakeup event
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
EcuM_CB_NfyNvMJobEnd(uint8,
NvM_RequestResultType)
[No]
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
alt Pending wakeup events?
[Yes]
opt NvM
NvM_CancelWriteAll()
[No]
loop FOR all wakeup sources enabled in the target sleep mode
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(RES_AUTOSAR_ECUM)

Figure 14 – Deinitialization Sequence IIa (GOSLEEP)

7.5.4.3 GO OFF I

GO OFF I is carried out under OS control and is implemented by the EcuM_MainFunction service.

[SWS_EcuM_00328] **⌈**As its last activity, the `ShutdownOS` service shall be called. This service will end up in the shutdown hook. The shutdown hook in turn shall call EcuM_Shutdown to terminate the shutdown process. EcuM_Shutdown will not return but switch off the ECU or issue a reset. ⌋ ()

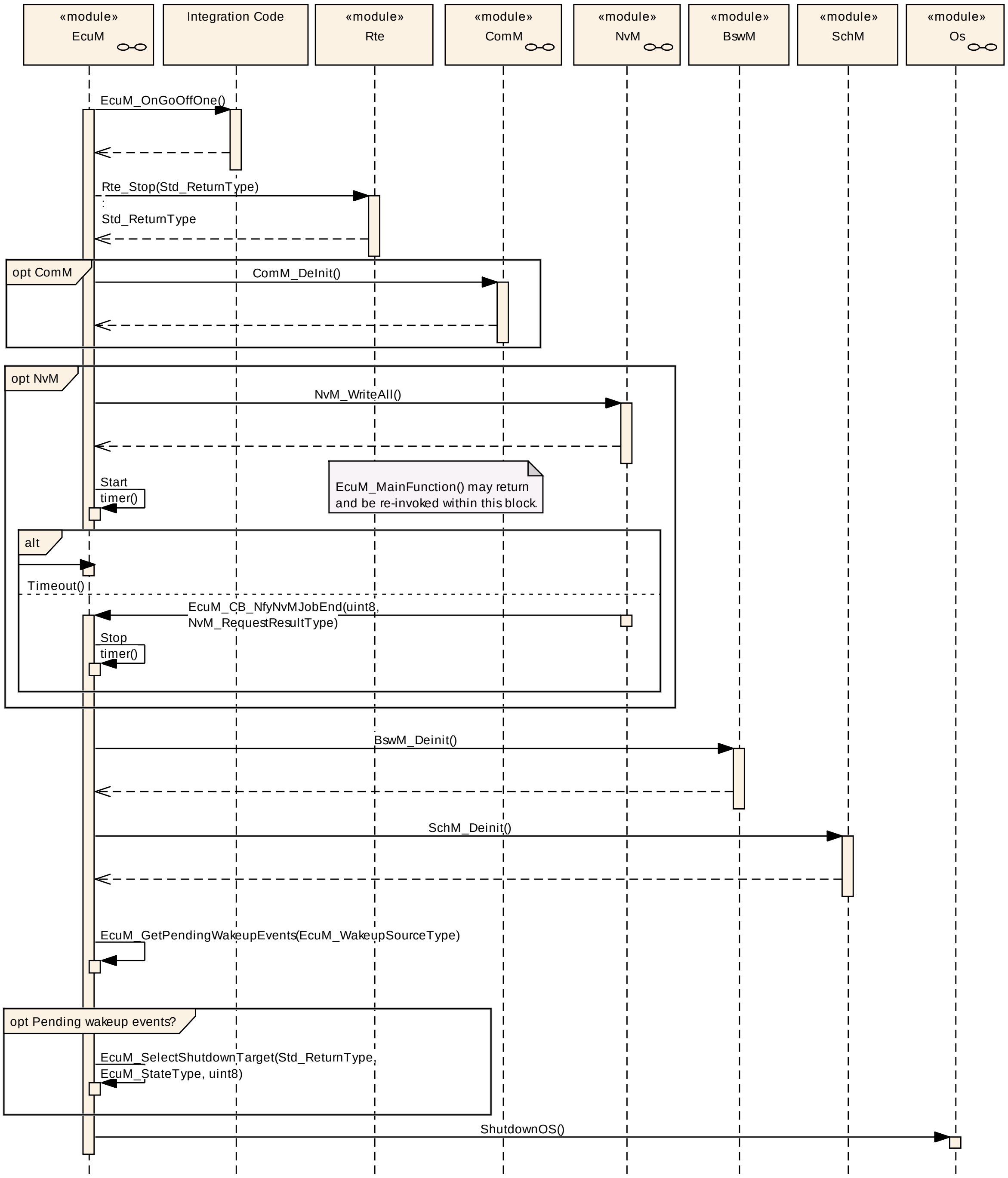


Figure 15 – Deinitialization Sequence IIb (GO OFF I)

7.5.4.4 GO OFF II

This state implements the final steps to reach the shutdown target after the OS has been shut down.

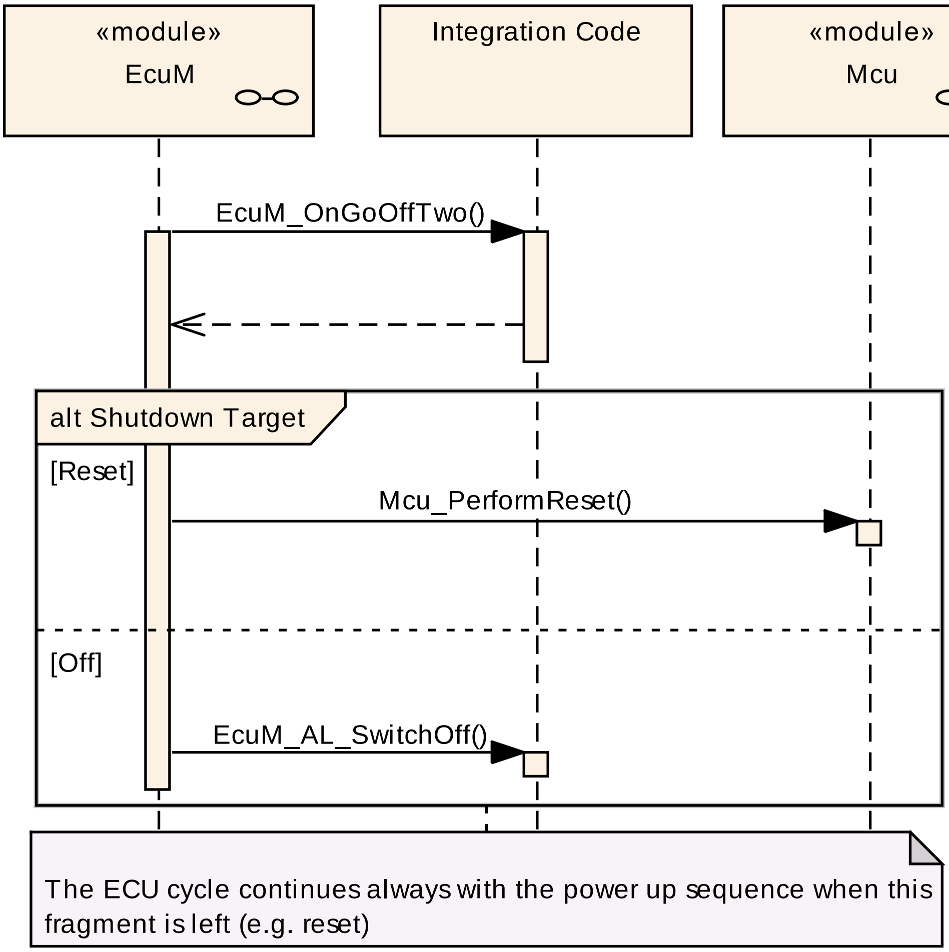


Figure 16 – Deinitialization Sequence III (GO OFF II)

The shutdown target RESET is reached by invoking the `Mcu_PerformReset` service of the MCU driver (see [13]).
The shutdown target OFF is implemented by the EcuM_AL_SwitchOff callout which must be filled at configuration time. See *8.7.4.7* EcuM_AL_SwitchOff for details.

7.6 SLEEP State

Refer To chapter *7.1.4* SLEEP State for an overview description.

[SWS_EcuMf_00025] ⌈The ECU State Manager Fixed module shall not put the ECU into `SLEEP` state before all run requests are released.⌋ ()

[SWS_EcuMf_00026] ⌈The ECU State Manager Fixed module shall put all communication interfaces to standby state and shall arm the wake up source before the ECU State Manager Fixed module may put the ECU into `SLEEP` state.
⌋ ()

7.6.1 High Level Sequence Diagram

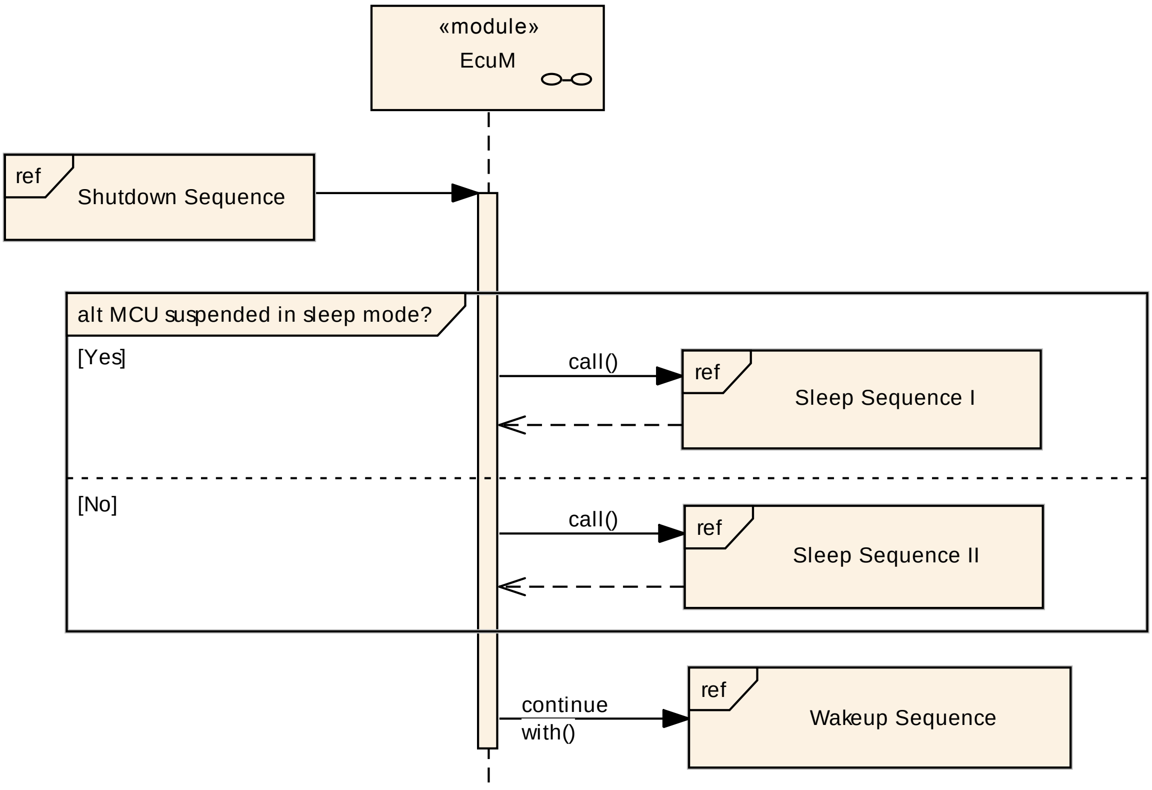


Figure 17 – Sleep Sequence (high level diagram)

To see adjacent diagrams refer to
Figure 12 – Shutdown Sequence (high level diagram)
Figure 20 – Wake-up Sequence (high level diagram)
Figure 2 – ECU Main States (top level diagram)

7.6.2 Sub-State Descriptions

7.6.2.1 Shutdown Targets

Shutdown Targets is a descriptive term for all states and their modes or sub-states where no code is executed. They are called shutdown targets because it is the final state where the state machine will drive to when RUN state is left. The following states are shutdown targets:

- OFF[20]
- SLEEP
- Reset
  is only a transient a state, but also can be selected as shutdown target.

[SWS_EcuM_00232] ⌈The default shutdown target shall be defined by configuration. This shutdown target shall be overridden by calling EcuM_SelectShutdownTarget. ⌋ ()

The SLEEP state can define a configurable set of sleep modes, where each mode itself is a shutdown target (the bullet list above is a simplification). These sleep modes are hardware dependent and differ typically in clock settings or other low power features provided by the hardware. These different features are accessible through the MCU driver as so called MCU modes (see [13]). The ECU State Manager Fixed module allows to map these MCU modes to ECU sleep modes and hence they are addressable as shutdown targets. Further the configuration allows defining aliases for shutdown targets to simplify portability of code across different ECUs. See 10.3 Configurable Parameters container EcuMSleepMode for details.

[20] The OFF state requires the capability of the ECU to switch off itself. This is not granted for all hardware designs.

7.6.2.2 Sleep Sequence I

Sleep Sequence I is executed in sleep modes that halt the microcontroller. In these sleep modes no code is executed.

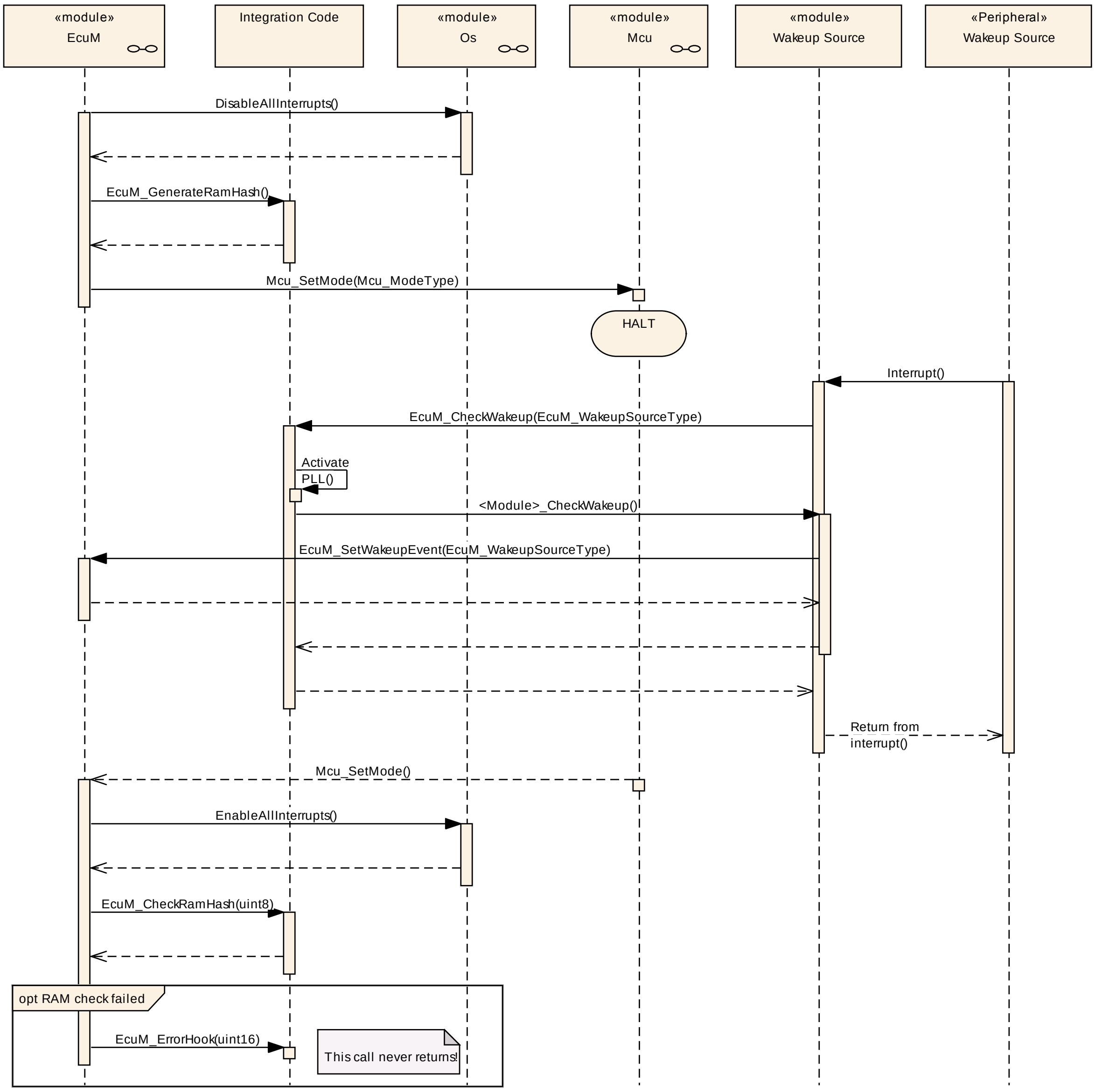


Figure 18 – Sleep Sequence I

A callout is invoked where the system designer can place a RAM integrity check. See also EcuM_GenerateRamHash and EcuM_CheckRamHash.

[SWS_EcuM_02863] ⌈The ECU Manager module shall invoke the callout EcuM_GenerateRamHash (see SWS_EcuM_02919) before halting the microcontroller and the callout EcuM_CheckRamHash (see SWS_EcuM_02921) after the processor returns from halt. ⌋ ()

Rationale for SWS_EcuM_02863: RAM memory may become corrupted when an ECU is held in SLEEP mode for a long time. The RAM memory's integrity should therefore be checked to prevent unforeseen behavior. The system designer may choose an adequate checksum algorithm to perform the check.

7.6.2.3 Sleep Sequence II

[SWS_EcuM_02962] ⌈The ECU State Manager Fixed module shall execute the Poll Sequence in sleep modes that reduce the power consumption of the microcontroller but still execute code. ⌋ ()

[SWS_EcuM_03020] ⌈In the Poll sequence the ECU State Manager Fixed module shall call the callouts `EcuM_SleepActivity()` and `EcuM_CheckWakeup()` in a blocking loop until a pending wake up event is reported. ⌋ ()

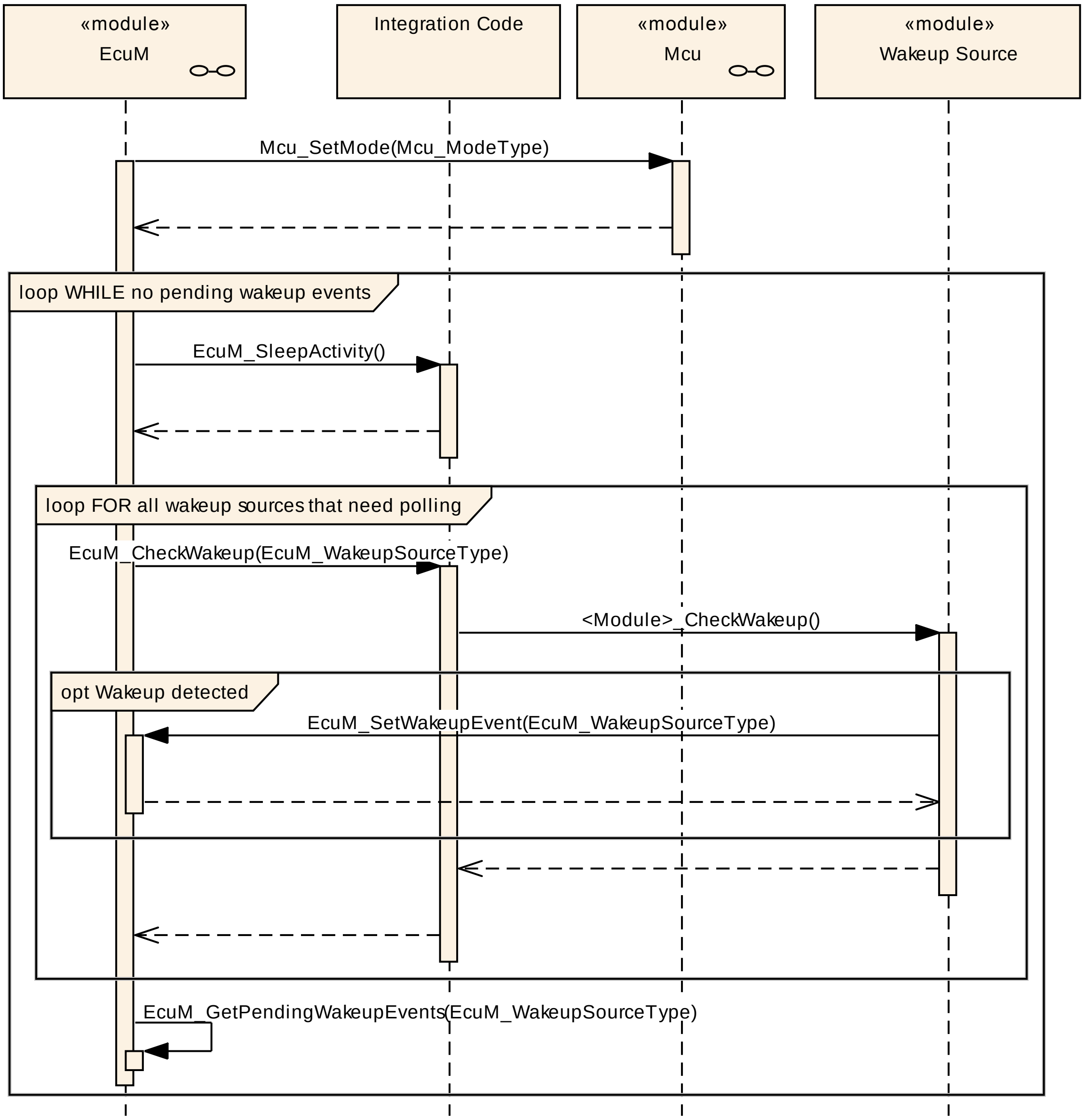

«module»
EcuM
Integration Code
«module»
Mcu
«module»
Wakeup Source
Mcu_SetMode(Mcu_ModeType)
loop WHILE no pending wakeup events
EcuM_SleepActivity()
loop FOR all wakeup sources that need polling
EcuM_CheckWakeup(EcuM_WakeupSourceType)
<Module>_CheckWakeup()
opt Wakeup detected
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)

Figure 19 – Sleep Sequence II

7.6.3 Leaving SLEEP State

Regular exits of the SLEEP state are a result of a wake up event (toggling a wake up line, communication on a CAN bus etc.). An ISR may be invoked to handle the event, but this is specific to hardware and driver implementation. Finally, the MCU_SetMode service of the MCU driver will return and the ECU State Manager Fixed module will regain control. Execution then continues with the WAKEUP state.

Irregular events are a hardware reset or a power cycle. In this case, the ECU will restart from the STARTUP state.

7.7 WAKEUP State

7.7.1 High Level Sequence Diagram

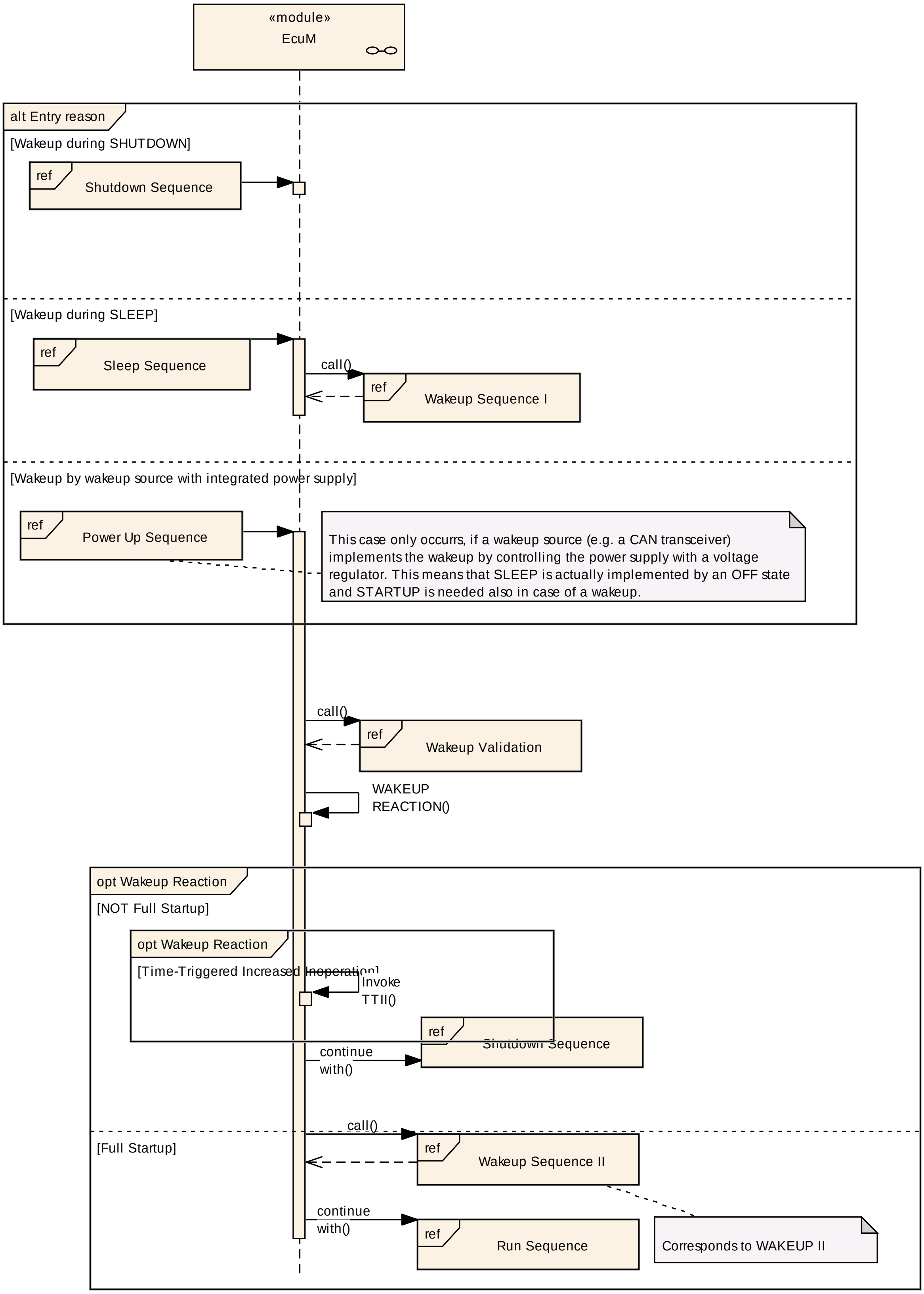


Figure 20 – Wake-up Sequence (high level diagram)

To see adjacent diagrams, refer to

Figure 12 – Shutdown Sequence (high level diagram)
Figure 8 – RUN State Sequence (high level diagram)
Figure 2 – ECU Main States (top level diagram)

7.7.2 State Breakdown Structure

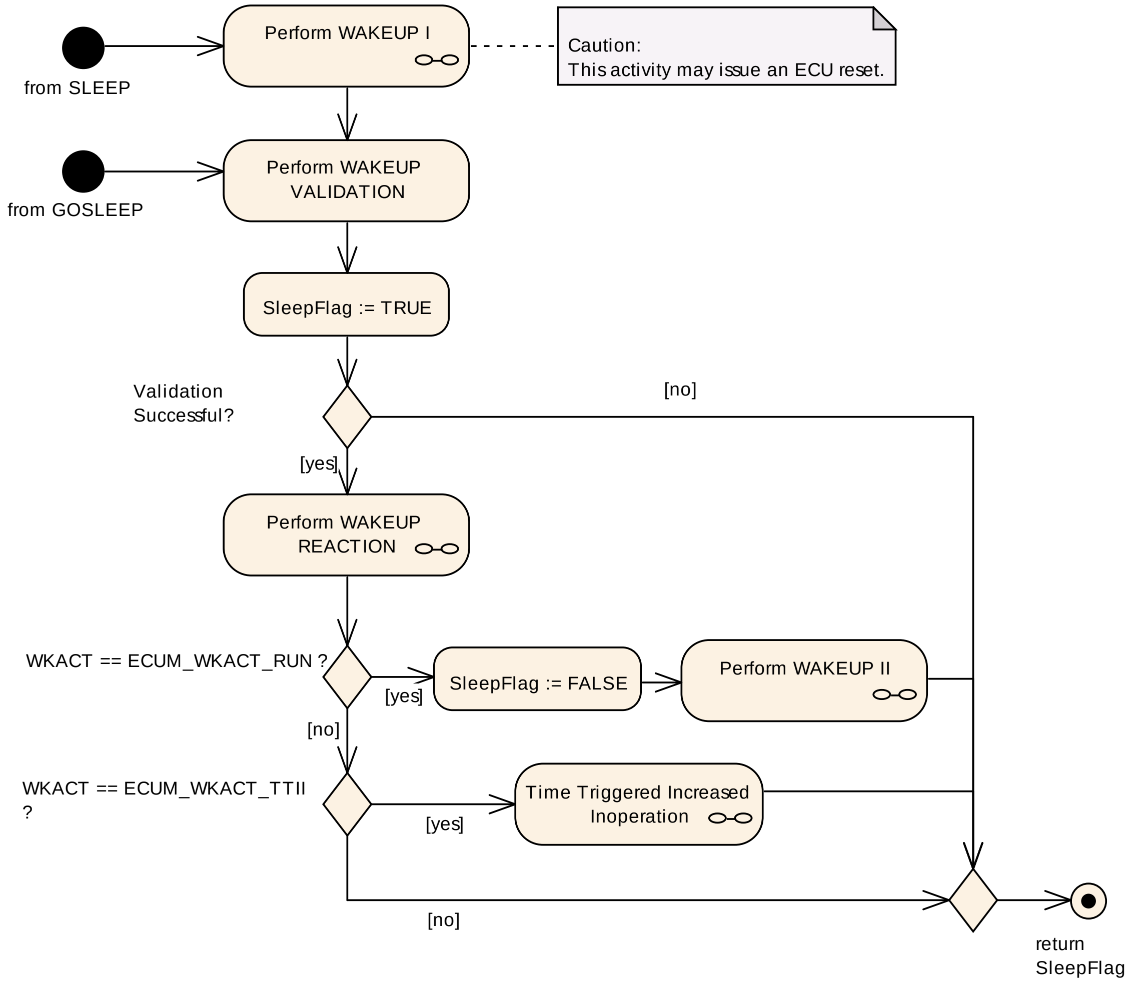


Figure 21 – WAKEUP State Breakdown

7.7.3 WAKEUP Activity Overview

Sub-state[21]	*Wake-up Activity*	*Comment*	*Opt.*
WAKEUP I			
	Restore MCU normal mode	Selected MCU mode is configured in parameter EcuMNormalMcuModeRef	
	Get the pending wake up sources		
	Callout EcuM_DisableWakeupSources	Disable currently pending wake up source but leave the others armed so that later wake-ups are possible.	
	Callout EcuM_AL_DriverRestart	Initialize drivers that need restarting	
	Unlock Scheduler	From this point on, all other tasks may run again	
WAKEUP VALIDATION		see chapter *7.7.4.2* WAKEUP VALIDATION	
WAKEUP REACTION			
	Compute wake up reaction	see chapter 7.7.4.3 unterhalb	
	Callout EcuM_OnWakeupReaction	if the wakeup reaction is ECUM_WKACT_SHUTDOWN	
×	Indicate mode change to RTE		
×	Invoke TTII protocol	see chapter 7.9 unterhalb	
WAKEUP II			
	Initialize Diagnostic Event Manager	Conditional: a) If the System comes out of SLEEP, the Dem shall be initialized b) If this is not the case the EcuM shall wait for EcuM_CB_NfyNvMJobEnd() and then execute `EcuMDriverInitListThree`	yes
×	Indicate mode change to RTE		

Table 4 – Wake-up Activities

7.7.4 Sub-State Descriptions

7.7.4.1 WAKEUP I

The EcuM_AL_DriverRestart callout is invoked. This callout is intended for re-initializing drivers. Re-initialization is typically required for drivers with wake up sources, at least. For more details on driver initialization refer to *7.3.5* Driver Initialization.

[SWS_EcuM_02539] ⌈During re-initialization, a driver must check if one of its assigned wake up sources was the reason for the previous wake up. If this test is true, it must invoke its 'wake up detected' callback (see [21] for an example), which in turn has to call the EcuM_SetWakeupEvent service. As a result, when WAKEUP I has finished, the ECU State Manager Fixed module has a list of wake up source candidates. These wake up source candidates still may need validation. See also *7.8* Wake-up Validation Protocol for more information.⌋ ()

[21] Rows marked with × are conditional.

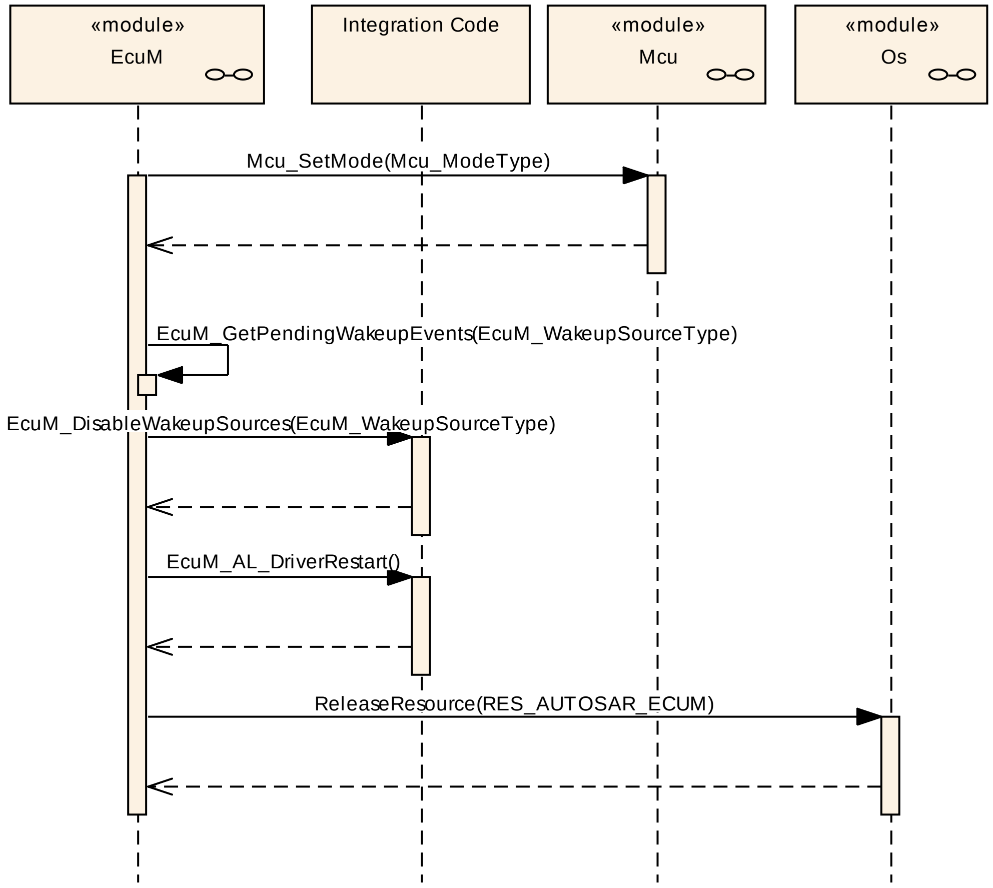

«module»
EcuM
Integration Code
«module»
Mcu
«module»
Os
Mcu_SetMode(Mcu_ModeType)
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
EcuM_AL_DriverRestart()
ReleaseResource(RES_AUTOSAR_ECUM)

Figure 22 – Wake-up Sequence I

[SWS_EcuM_00545] ⌈The driver should be implemented in a way that it only invokes the wake up callback once and then requires a dedicated service call to re-arm this mechanism. The driver then needs to be re-armed to fire the callback again. ⌋ ()

7.7.4.2 WAKEUP VALIDATION

Because wake up events can be generated unintended (e.g. EVM spike on CAN line), it is necessary to validate wake-ups before the ECU takes up its full operation. The validation mechanism is the same for all wake up sources. When a wake up event occurs, the ECU is woken up from its SLEEP state and execution resumes within the `MCU_SetMode` service of the MCU driver[22]. When WAKEUP I is left, the ECU State Manager Fixed module will have a list of pending wake up events which need to be validated.

[22] Actually, the first code to be executed may be an ISR, e.g. a wakeup ISR. However, this is specific to hardware and/or driver implementation.

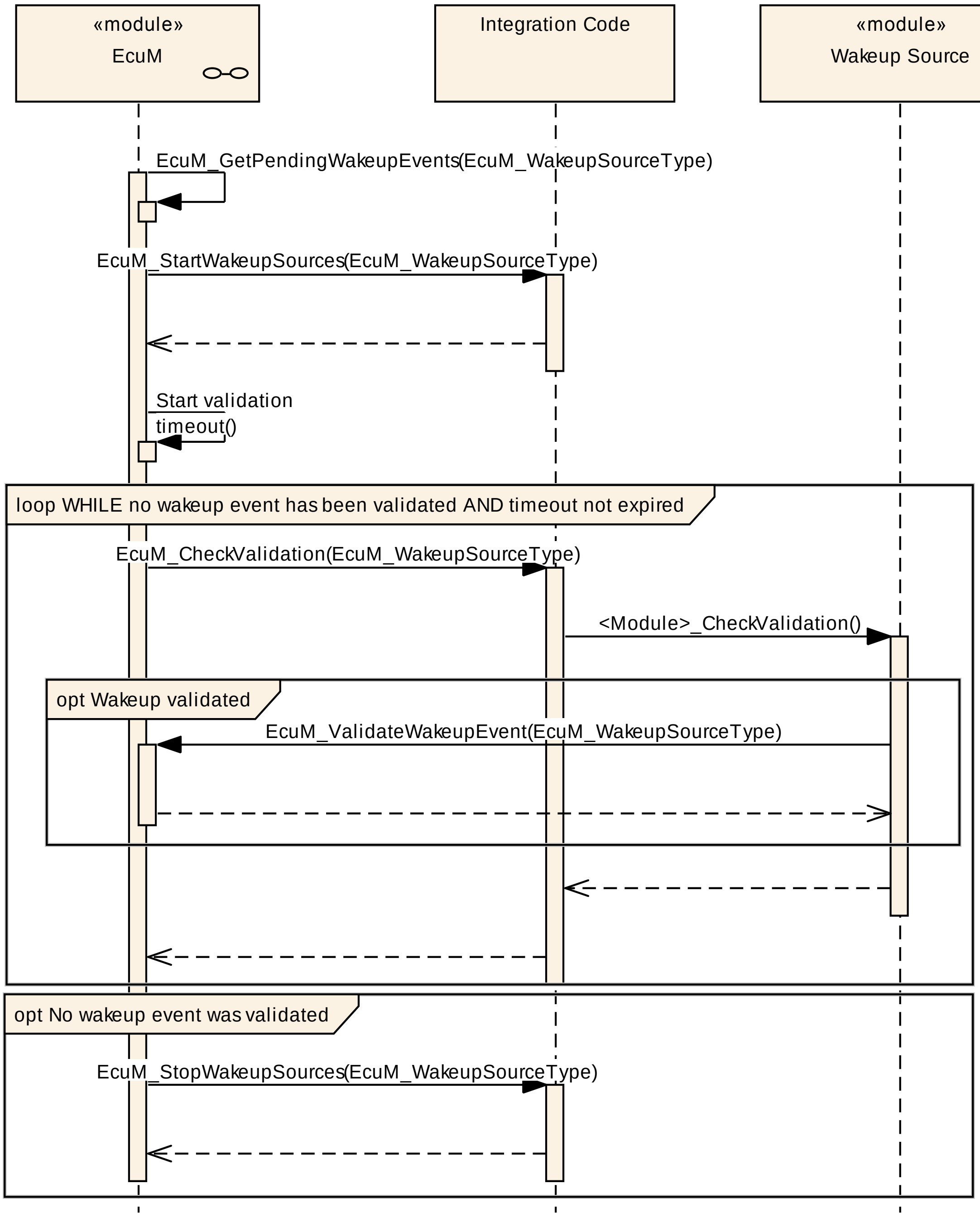

«module»
EcuM
Integration Code
«module»
Wakeup Source
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
EcuM_StartWakeupSources(EcuM_WakeupSourceType)
Start validation timeout()
loop WHILE no wakeup event has been validated AND timeout not expired
EcuM_CheckValidation(EcuM_WakeupSourceType)
<Module>_CheckValidation()
opt Wakeup validated
EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType)
opt No wakeup event was validated
EcuM_StopWakeupSources(EcuM_WakeupSourceType)

Figure 23 – Wake-up Validation Sequence

[SWS_EcuM_02566] ⌈Wake up validation shall apply only to those wake up sources where it is required by configuration. If the validation protocol is not configured, then a call to EcuM_SetWakeupEvent shall also imply a call to EcuM_ValidateWakeupEvent. ⌋ ()

[SWS_EcuM_02565] ⌈For each pending wake up event, for which validation is required, a validation timeout shall be started. The timeout is event specific and can be defined by configuration. Strictly spoken, it is sufficient for an implementation to provide only one timer, which is prolonged to the largest timeout when new wake up events are reported. ⌋ ()

[SWS_EcuM_00567] ⌈If the last timeout expires without validation then the wake up validation is considered to have failed. ⌋ ()

[SWS_EcuM_00568] ⌈If at least one of the pending events is validated then the entire validation has passed.
Pending events are validated with a call to EcuM_ValidateWakeupEvent. This call must be placed in the driver or the consuming stack on top of the driver (e.g. the handler). The best place to put this depends on hardware and software design. See also *7.8.5* Requirements for drivers with wake up sources. ⌋ ()

7.7.4.3 WAKEUP REACTION

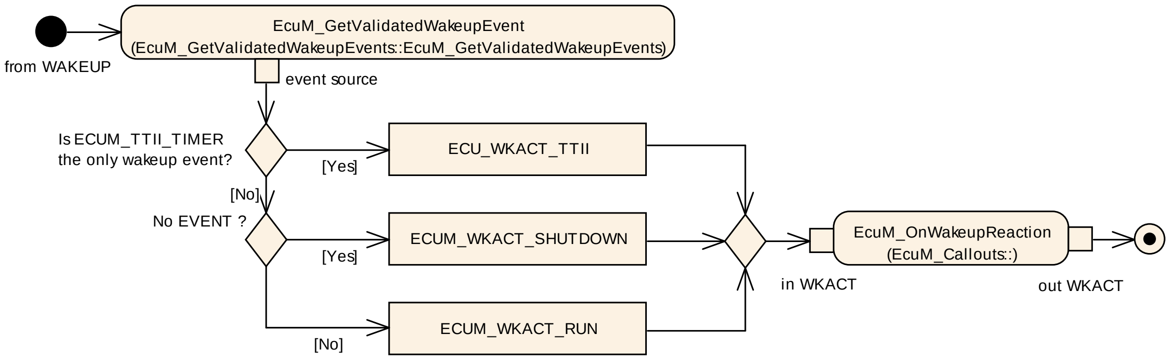


Figure 24 – Activity Diagram of WAKEUP REACTION

The WAKEUP REACTION state determines the appropriate wake up reaction (see 8.3.5 EcuM_WakeupReactionType) according to the wake up source (see 8.3.3 EcuM_WakeupSourceType).
As can be seen from, Figure 24 – Activity Diagram of WAKEUP REACTION there are the following wake up reactions:

- Execution of the TTII protocol (see *7.9* Time Triggered Increased Inoperation)
- Proceed to RUN state (full startup)
- Shutdown
  If none of the above cases is chosen, the ECU will be shut down again by default. The exact behavior depends on the selected shutdown target.

The callout of this state may be used to override the wake up reaction and provide an ECU specific algorithm.
In case of an ECU Reset, the ECU State Manager Fixed module will perform a full initialization.

After a failed wake up validation the EcuM shall put the ECU into the same state as before the wake up event which failed, i.e. into "SLEEP" or "OFF". The state before the wake up event can be determined by calling
`"EcuM_GetLastShutdownTarget()"`.

7.7.4.4 WAKEUP II

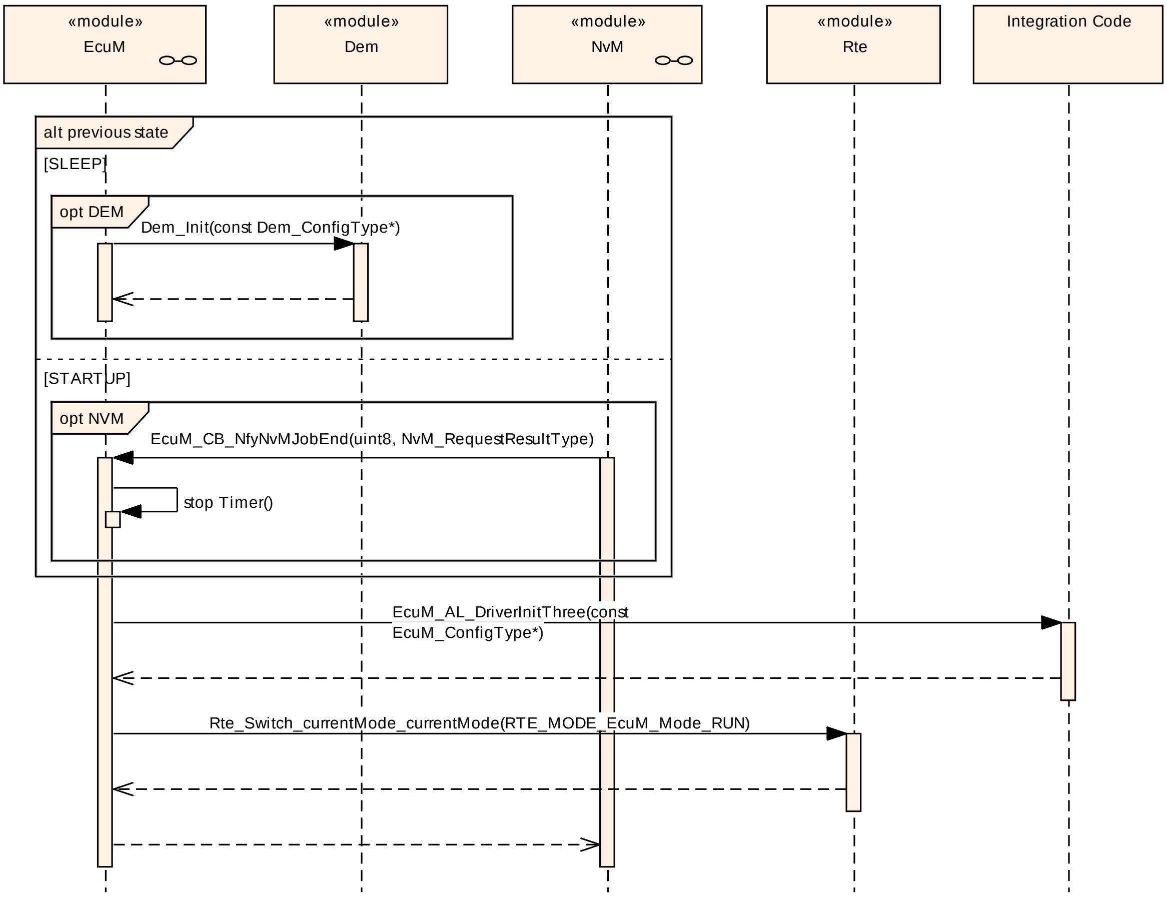

«module»
EcuM
«module»
Dem
«module»
NvM
«module»
Rte
Integration Code
alt previous state
[SLEEP]
opt DEM
Dem_Init(const Dem_ConfigType*)
[STARTUP]
opt NVM
EcuM_CB_NfyNvMJobEnd(uint8, NvM_RequestResultType)
stop Timer()
EcuM_AL_DriverInitThree(const EcuM_ConfigType*)
Rte_Switch_currentMode_currentMode(RTE_MODE_EcuM_Mode_RUN)

Figure 25 – Wake-up Sequence II

7.8 Wake-up Validation Protocol

7.8.1 Wake-up of Communication Channels

Communication channels have their own state machines including run and also sleep states. This is necessary since an ECU may have interfaces to several communication busses and busses can go to sleep independently from the ECU. Consider the following example:

> An ECU may have two bus interfaces A and B. The ECU may be awake, bus A is in full communication state, but bus B is sleeping.

The state machines of the communication channels are completely provided by the Communication Manager, see [6] for details.
According to the specification, the Communication Manager autonomously can fulfill the following tasks:

- Drive a channel from full communication in no communication mode in collaboration with Network Management.
- Put the bus transceiver into standby mode by using the Bus State Manager according to the bus interface type. This will configure the bus transceiver to generate wake up events when bus traffic occurs.

The Communication Manager however will not drive the wake up process since wake up events will be directed to the ECU State Manager Fixed module which in turn will notify the Communication Manager if and only if appropriate.

[SWS_EcuM_00478] ⌈If a wake up occurs on a communication channel, the according bus transceiver driver shall notify the ECU State Manager Fixed module by invoking the EcuM_SetWakeupEvent service. Requirements for this notification are described in *5.3* Peripherals with Wake-up Capability. ⌋ ()

[SWS_EcuM_02479] ⌈The ECU State Manager Fixed module shall execute the wake up validation protocol according to *7.8.3* Interaction of wake up Sources and the later in this chapter. ⌋ ()

[SWS_EcuM_00480] ⌈If validation is successful, the ECU State Manager Fixed module shall inform the Communication Manager about the wake up event by invoking the Communication Manager's `ComM_EcuM_WakeUpIndication` service with the according channel as parameter. In turn, the Communication Manager will use this event to bring the channel into full communication mode. ⌋ ()

[SWS_EcuMf_00044]⌈ If at least one valid wake up is detected the ECU shall perform a startup as fast as possible. ⌋ ()

[SWS_EcuMf_00045] ⌈If in addition to a validated wake up an "invalid" wake up occurs as well, it is tolerable to indicate it too. This does not contradict SRS_ModeMgm_09097, since the ECU has to start anyway. ⌋ ()
⌋ ()

7.8.2 Wake-up of the Entire ECU

Before the ECU State Manager Fixed module can put the ECU into `SLEEP` state, the Communication Manager must have released all run requests[23], see **[SWS_EcuMf_00025**. This will only happen, if all communication state machines are in 'no communication' mode.

But this, taking into account the previous paragraphs, implies that all communication interfaces (i.e. all bus transceivers) must have been put to standby state and the wake up source must have been armed. Thus, when a wake up occurs, all communication channels are in no communication state and there are no RUN requests.

The wake up procedure is identical to the previous chapter.

[23] This statement can be extended to any resource manager which may be added in future versions of the AUTOSAR Basic Software.

7.8.3 Interaction of wake up Sources and the ECU State Manager Fixed module

All wake up sources must be treated in the same way. The procedure shall be as follows:

Upon occurrence of a wake up event, the responsible driver shall invoke an indication to notify the ECU State Manager Fixed module about the wake up.

This step can happen in several scenarios. The most likely are:

- After exiting the SLEEP state. In this scenario, the ECU State Manager Fixed module would issue a re-initialization of the relevant drivers which in turn get a chance to scan their hardware e.g. for pending wake up interrupts.
- If the wake up source is actually in sleep mode, then the driver shall scan autonomously for wake up events. The driver may do this interrupt driven or in polling mode, whichever is the preferred way for implementing it.

[SWS_EcuM_00494] ⌈If wake up validation is required for this event, then the validation protocol applies. Otherwise the event is valid immediately. ⌋ ()

[SWS_EcuM_00495] ⌈If the valid event is a wake up event from a communication interface then it is propagated to the Communication Manager. ⌋ ()

[SWS_EcuM_02975]⌈ If a wake up event requires validation then the ECU Manager module shall invoke the validation protocol. ⌋ ()

[SWS_EcuM_02976]⌈ If a wake up event does not require validation, the ECU Manager module shall issue a mode switch request to set the event's mode to ECUM_WKSTATUS_VALIDATED**.** ⌋ ()

[SWS_EcuM_02496]⌈ If the wake up event is validated (either immediately or by the wake up validation protocol), it is labelled as a wake up source and this information is made available by the EcuM_GetValidatedWakeupEvents service. ⌋ ()

7.8.4 Wake up validation timeout

It is the implementer's choice whether he wants to provide a single wake up validation timeout timer or one timer per wake up source. The following requirements apply:

[SWS_EcuM_02709] ⌈The timer shall be started when the service EcuM_SetWakeupEvent is called. ⌋ ()

[SWS_EcuM_02710] ⌈The timer shall be stopped and the validation is set to "passed" when the service EcuM_ValidateWakeupEvent is called. ⌋ ()

[SWS_EcuM_00711] ⌈When the timer expires, validation is set to "failed". ⌋ ()

[SWS_EcuM_02712] ⌈Subsequent calls to EcuM_SetWakeupEvent for the same wake up source shall not prolong the timeout. ⌋ ()

If only one timer is used, the following approach is proposed:

[SWS_EcuM_00714] ⌈If EcuM_SetWakeupEvent is called for a wake up source which did not fire yet during the same wake up cycle then the timeout should be prolonged for the validation timeout of that wake up source.
Wake up timeouts are defined by configuration in chapter 10.3 Configurable Parameters. ⌋ ()

7.8.5 Requirements for drivers with wake up sources

The driver shall invoke the EcuM_SetWakeupEvent service with a configurable parameter identifying the source of the wake up once when the wake up event is detected.

[SWS_EcuM_02572] ⌈Wake-ups which occurred prior to driver initialization shall be detectable. This applies to initialization from `SLEEP` or from `OFF` state. ⌋ ()

The driver shall provide an API to configure the wake up source for the `SLEEP` state, to enable or disable the wake up source, and to put the related peripherals to sleep. This requirement only applies if hardware provides these capabilities.

The callback invocation shall be enabled by calling the driver initialization service.

7.8.6 Requirements for Wake-up Validation

If the wake up source requires validation, this may be done by any but only by one appropriate module of the basic software. This may be a driver, an interface, a handler, or a manager.

Validation is done by calling the EcuM_ValidateWakeupEvent service.

7.8.7 Wake up Sources and Reset Reason

The API of the ECU State Manager Fixed module API only provides one type (EcuM_WakeupSourceType) which can describe all reasons why the ECU starts or wakes up.

[SWS_EcuM_02625] ⌈The following wake up sources shall not require validation under no circumstances:

- `ECUM_WKSOURCE_POWER`
- `ECUM_WKSOURCE_RESET`
- `ECUM_WKSOURCE_INTERNAL_RESET`
- `ECUM_WKSOURCE_INTERNAL_WDG`

- ECUM_WKSOURCE_EXTERNAL_WDG

⌋ ()

7.8.8 Wake up Sources with Integrated Power Control

This section applies if the sleep state is realized by a system chip which controls the MCU's power supply. Typical examples are CAN transceivers with integrated power supplies. These transceivers switch off power upon application request and switch on power upon CAN activity.
As a consequence, the sleep state looks like the OFF state for the ECU State Manager. This distinction is rather philosophical and not of practical importance. The practical impact is that a passive wake up on CAN will look like a power on reset to the ECU. Hence, the ECU will continue with the startup sequence after a wake up event. Nevertheless, wake up validation is required. In order to make this work, the system designer has to consider the following topics:

- The CAN transceiver is initialized during one of the driver initialization blocks (Init Block II by default). This is configured or generated code, i.e. code which is under control of the system designer.
- The CAN transceiver driver API provides services to find out if it was the CAN transceiver, due to a passive wake up, which started the ECU. It is the system designer's responsibility to check the CAN transceiver for wake up reasons and give this information to the ECU State Manager Fixed module by using the EcuM_SetWakeupEvent and EcuM_ClearWakeupEvent services.
- If the system designer sets the CAN transceiver as the wake up source, then the ECU State Manager Fixed module will not continue with the RUN state when STARTUP II is finished. Instead it will continue with the WAKEUP VALIDATION state.

This behavior can be applied to all kinds of wake up sources. The CAN transceiver only serves as an example here.

Waking up from a sleep state which is implemented by unpowering the MCU is not fully transparent to the SW-Cs. First of all the BSW modules are brought back into their default states after initialization. Second, when starting RTE the SW-Cs will be initialized and STARTUP state is signaled for a very short time. When the MCU is unpowered, it is inevitable that the ECU State Manager Fixed module carries out the STARTUP state. The ECU State Manager Fixed module offers support by detecting this case and then branching into wake up validation and from there (if validation is successful) into RUN state. If wake up validation is not successful, the ECU State Manager Fixed module supports branching into SHUTDOWN state. During wake up validation the ECU State Manager Fixed module will signal SLEEP state to the SW-Cs so that afterwards it appears as if they were woken up from a normal SLEEP state.

7.8.9 Activity Diagram

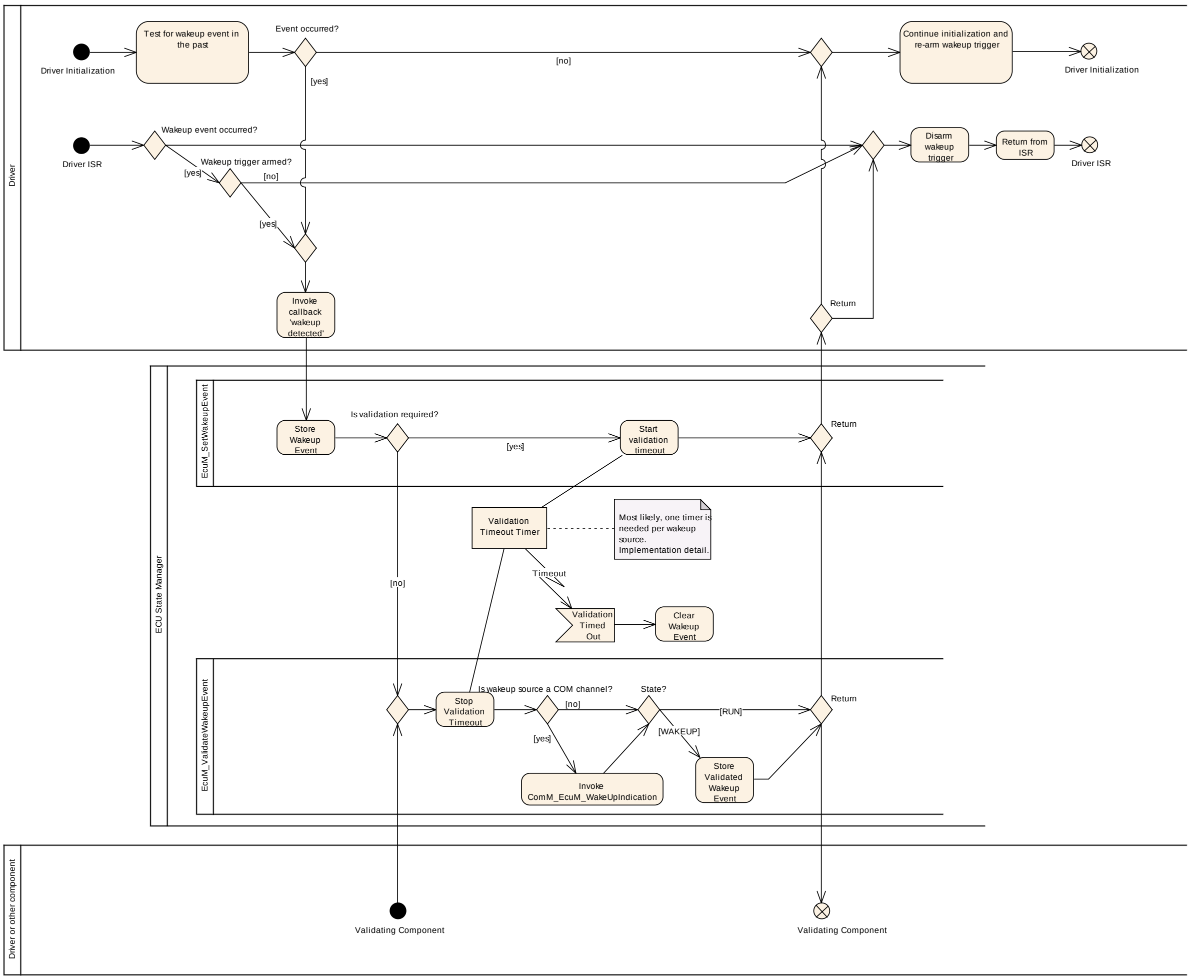


Figure 26 – Wake up Validation Protocol

7.9 Time Triggered Increased Inoperation

[SWS_EcuM_00653] ⌈TTII shall manage a list of all sleep modes (shutdown targets). These sleep modes can be defined at configuration time. Typically the sleep modes are ordered to deepen the sleep phase of the ECU (decreased power consumption). ⌋ ()

[SWS_EcuM_00654] ⌈An entry of the sleep mode list shall contain the following properties:

- A description of the ECU sleep mode
- A reference to the successor sleep mode
- A divisor counter which tells how often the ECU must be woken up before the successor sleep mode is selected.

These properties shall be defined at configuration time (see 10.3 Configurable Parameters container EcuMFixedTTII). ⌋ ()

The TTII protocol is executed during the WAKEUP REACTION sub-state. Refer to chapter *7.7.4.3* WAKEUP REACTION and Figure 24 – Activity Diagram of WAKEUP REACTION.

[SWS_EcuM_00223] ⌈The entire TTII feature can be completely disabled by setting the *ECUM_TTII_ENABLED* configuration parameter to false. All further described activities are only applicable if TTII is enabled. ⌋ ()

[SWS_EcuM_00222] ⌈A wake up source must be selected by configuration (*ECUM_TTII_WKSOURCE* configurable parameter) for use by the TTII protocol. Typically, the wake up source will be a timer, which serves as a timebase for TTII. Whenever the ECU is woken up by this configured wake up source, then the TTII protocol shall be executed. ⌋ ()

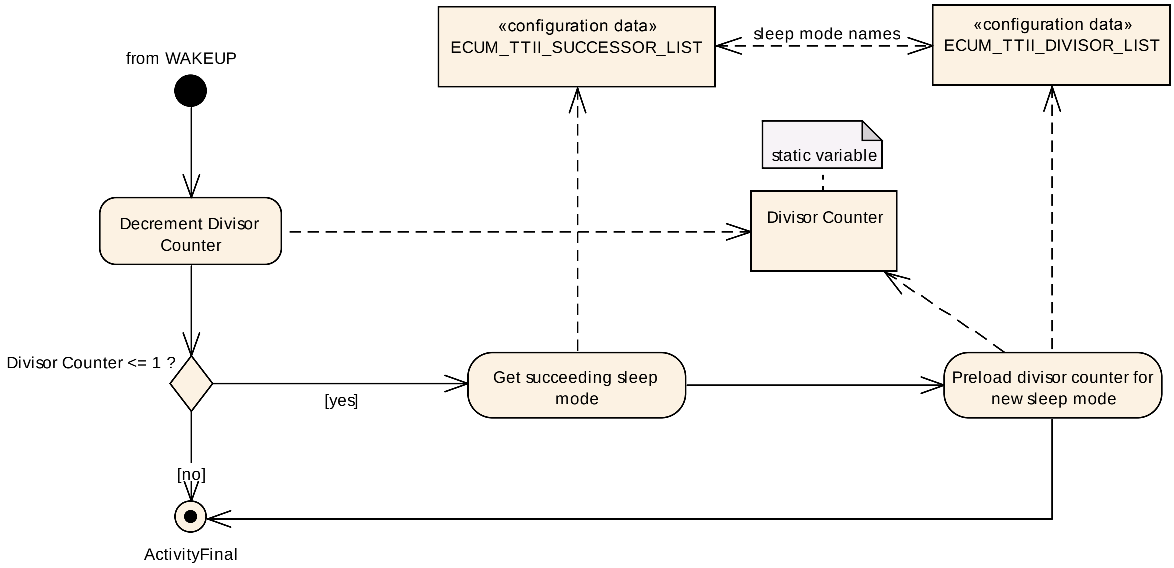


Figure 27 – Activity Diagram of TTII

[SWS_EcuM_02785] ⌈Whenever RUN mode is reached, the TTII protocol shall be reset to use the wakeup source referenced by this parameter. This configuration parameter is a human readable name for a TTII wakeup source which is only needed by the configuration tool. For imlementation on the ECU, this parameter may be dropped and replaced by a generated list index of EcuM_TTII. ⌋ ()

7.10 Advanced Topics

7.10.1 OS Application Modes

OS Application Modes is a feature of the OS which allows defining different configurations, e.g. sets of tasks which will be started initially. The application mode is an in parameter of the `StartOS` service [6]. Since the ECU State Manager Fixed module is responsible for starting the OS, it has also responsibility for managing the application mode.

[SWS_EcuMf_00010] ⌈ Since AUTOSAR RTE does not use application modes, the ECU State Manager Fixed module always has to start the OS with the value `DEFAULT_APP_MODE` (defined by the operating system). ⌋ ()

[SWS_EcuM_00243] ⌈The default application mode is set in the STARTUP I state in case of unintended restarts[24], see chapter *7.3.4.1* STARTUP I. ⌋ ()

7.10.2 Relation to Bootloader

The Bootloader is not part of AUTOSAR. Still, the application needs an interface to activate the bootloader. For this purpose, two functions are provided: EcuM_SelectBootTarget and EcuM_GetBootTarget.

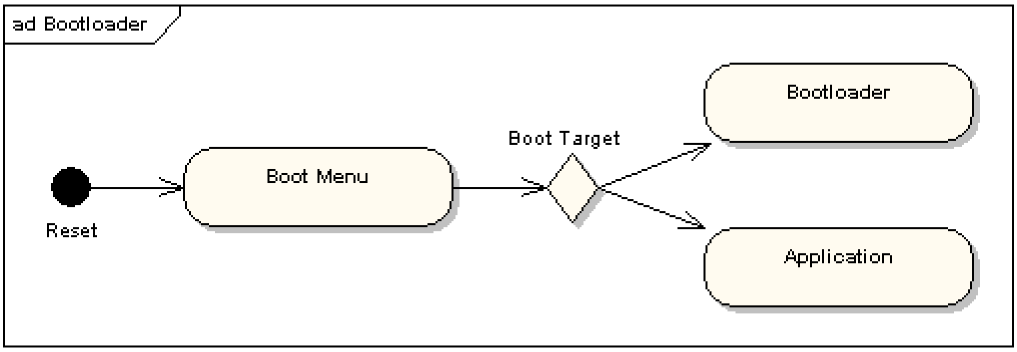


Figure 28 – Selection of Boot Targets

Bootloader and application are two separated programs which in many cases even can be flashed separately. The only way to get from one image to another is through

[24] e.g. like watchdog reset

reset. The boot menu will branch into the one or other image depending on the selected boot target.

7.10.3 Relation to Complex Drivers

If the complex driver handles a wake up source, it must obey all rules of this specification which are related to handling wake up events.

A complex driver may issue RUN requests.

7.10.4 Handling Errors during Startup and Shutdown

The ECU State Manager Fixed module will ignore all types of errors that occur during initialization, e.g. as return values of init functions. Initialization is a configuration issue and henceforth cannot be standardized.
If errors occur during the initialization of a BSW module and this error is worthwhile being reported, then it is in the responsibility of that BSW module to report this error directly to DEM or DET and not the responsibility of the ECU State Manager.
If special error reactions are necessary, then also this is in the responsibility of the BSW module.

7.10.5 Configuration Alternative for Providing Wake-Sleep Operation

In rare use cases, an ECU has to wake up cyclically (e.g. each second), execute a very simple task (like blinking an LED) and go back to sleep. For most operations, the normal WAKEUP/SHUTDOWN behavior as defined by the ECU State Manager Fixed module will be sufficient. Sometimes, however, the software has to be written very specific to maximize energy savings. Because the use case is so rare, there is no built-in feature in the ECU State Manager. However, the system designer can achieve this by using the ECU State Manager Fixed module in the following way:

- Define a wake up source to be used for the wake-sleep-operation (typically a timer)
- Check the wake up source in the EcuM_AL_DriverInitOne callout and, if it was the reason, execute the necessary task
- Finally, put the ECU back to sleep or perform a startup

The code needed for this behavior is custom code which is located below the RTE.

7.10.6 Selecting Scheduling Schemes for Startup and Shutdown

On some ECU designs, it will be necessary to change the scheduling tables for startup and shutdown of the ECU, e.g. to improve speed for reading or writing non-volatile data. Unless other mechanisms are provided by basic software, the notification to switch the schedule table shall preferably be done from the EcuM_OnEnterRun and EcuM_OnExitRun callouts.

7.11 Error Handling

7.11.1 Error Hook

AUTOSAR BSW modules normaly report their errors to Det (development errors) or Dem (production errors). The EcuM handles errors differently and does not report its errors to Dem/Det. If a reporting of errors to Dem/Det is needed the user can perform these actions in the EcuM_ErrorHook().

The following table contains all error codes which might be reported from the EcuM (besides those individual error codes defined integrator)

Type or error	*Related error code*	*Value [hex]*
The RAM check during wake up failed	`ECUM_E_RAM_CHECK_FAILED`	Assigned by Implementation
Configuration data is inconsistent	`ECUM_E_CONFIGURATION_DATA_INCONSISTENT`	Assigned by Implementation

Table 5 - Runtime Errors

7.11.2 Development Errors

There are no development errors.

7.11.3 Transient Faults

There are no transient faults.

7.11.4 Production Errors

There are no production errors.

7.11.5 Extended Production Errors

There are no extended production errors.

7.12 Debug Support

In order to support debugging AUTOSAR implementations must publish information which can be used for debugging purpose. As start-up and shut-down are crucial system phases, sufficient information to track the current state of the ECU State Manager Fixed module needs to be provided by implementations.

7.13 Multicore without BSW distribution

The following figure illustrates the architecture of EcuMFixed in a multi core environment.

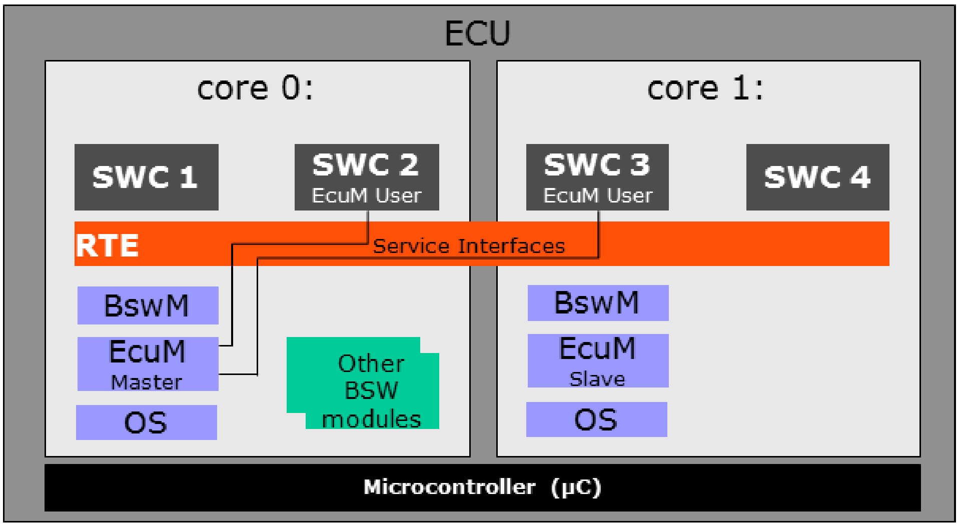


Figure 29 - MultiCore Use Case

The behavior of EcuMFixed depends on the core it is running on.
Master EcuM has the same scope of functionality as on a single core. Furthermore, it shall synchronize all Satellite EcuMs during state transitions.
A Slave EcuM is basically needed during startup, shutdown and sleep modes, when all cores have to be synchronized.
A Slave EcuM has no service interfaces, this kind of interfaces are routed by the RTE to the Master EcuM (SWS_EcuM_04094), see Figure 29. The EcuM_MainFunction on a slave core does not arbitrate RUN requests and does not evaluate wakeup events as no other BSW modules are located on the slave core.

If the same image of EcuM is executed on every core of the ECU, the ECU Manager’s behavior has to differ on the different cores. This can be accomplished by the ECU Manager by testing first whether it is on a master or a slave core and act appropriately.

[SWS_EcuMf_00116]⌈ The EcuM_MainFunction shall run in all EcuM instances.⌋ ()
[SWS_EcuMf_00117]⌈ The ECU Manager module shall start the SchM and the OS on every core.⌋ ()
[SWS_EcuMf_00118]⌈ The ECU Manager module shall call BswM_Init for all core local BswMs on the master and all slave cores.⌋ ()

Every core contains an EcuM. The communication between the ECU State Managers on every core is done by the Master/Satellite-Approach. The scale of distribution is vendor-specific. Master EcuM contains the global State Machine and is responsible for the transitions. All satellite EcuMs are controlled by the Master EcuM using synchronization points.

7.13.1 Master Core

There is one explicit master core. Which core the master core is, is determined by the boot loader. The EcuM of the master core gets started as first BSW module and performs initialization actions.

Then it starts all other cores with all other EcuMs.

When these are started, it initializes together with each satellite EcuM the core local OS and BswM.

7.13.2 Master Core – Slave Core Signalling

This section discusses the general mechanisms with which BSW can communicate over cores. It presupposed general knowledge of the SchM, which is described and specified in the RTE.

7.13.2.1 BSW Level

The Operating System provides a basic mechanism for synchronizing the starts of the operating systems on the master and slave cores. The Scheduler Manager provides basic mechanisms for communication of BSW modules across partition boundaries. One BSW Mode Manager per core is responsible for starting and stopping the RTE.

7.13.2.2 EcuM Level

Before calling `ShutdownAllCores`, the Master EcuM must start the shutdown of all "slave" ECU Manager Modules and has to wait until all modules have de-initialized the BSW modules for which they are responsible and successfully shutdown.

Therefore the master ECU Manager Module sets a shutdown flag which can be read by all slave modules. After that the master calls the main routine *EcuM_MainFunction()* of every slave ECU Manager Module. The slave modules read the flag inside the main routine and shutdown if requested.

The Operating System extends the OSEK SetEvent function across cores. A task on one core can wait for an event set on another core. Figure 30 illustrates how this applies to the problem of synchronizing the cores before calling ShutdownAllCores (whereby the de-intialization details have been omitted). The Set/WaitEvent functions accept a bitmask which can be used to indicate shutdown-readiness on the individual slave cores. Each SetEvent call from a "slave" ECU Manager module will stop the "master" ECU Manager module's wait. The "master" ECU Manager module must therefore track the state of the individual slave cores and set the wait until all cores have registered their readiness.

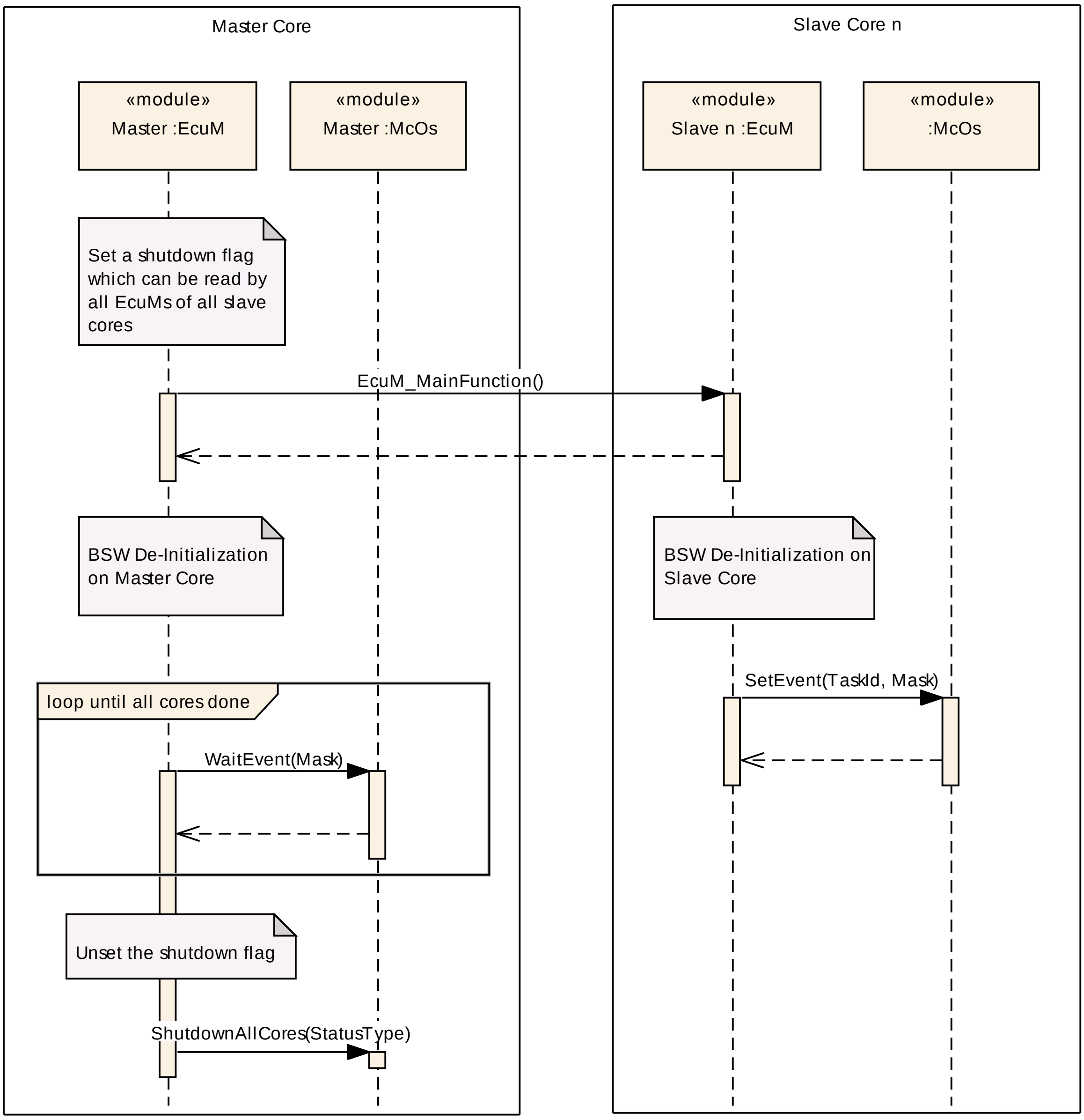

Master Core
Slave Core n
«module»
Master :EcuM
«module»
Master :McOs
«module»
Slave n :EcuM
«module»
:McOs
Set a shutdown flag which can be read by all EcuMs of all slave cores
EcuM_MainFunction()
BSW De-Initialization on Master Core
BSW De-Initialization on Slave Core
SetEvent(TaskId, Mask)
loop until all cores done
WaitEvent(Mask)
Unset the shutdown flag
ShutdownAllCores(StatusType)

Figure 30 – Master / Slave Core Shutdown Synchronization

The following sequences describe the extension of EcuMFixed for multi core use cases: single core, master core and slave core. The following sequences are based on the use case “Multi Core Application without BSW Distribution”.

7.13.2.3 Startup State

The ECU Manager module functions nearly identically on all cores. That is, as for the single-core case, the ECU Manager module performs the steps specified for Startup; most importantly starting the OS, initializing the SchM and starting the core local BswMs.

The master EcuM activates all slave cores after calling InitBlock 1 and doing the reset / wakeup housekeeping. After being activated, the slave cores execute their startup routines, which call EcuM_Init on their core.

After each EcuM has called StartOs on its core, the OS synchronizes the cores before executing the core-individual startup hooks and synchronizes the cores again before executing the first tasks on each core.

StartPreOS Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - SingleCore
GetCoreID()		
EcuM_AL_DriverInitZero()		EcuM_AL_DriverInitZero()
EcuM_DeterminePbConfiguration()		EcuM_DeterminePbConfiguration()
Check consistency of configuration data		Check consistency of configuration data
EcuM_AL_DriverInitOne()		EcuM_AL_DriverInitOne()
Mcu_GetResetReason()		Mcu_GetResetReason()
Map reset reason to wakeup source		Map reset reason to wakeup source
EcuM_SelectShutdownTarget()		EcuM_SelectShutdownTarget()
EcuM_LoopDetection()		EcuM_LoopDetection()
StartCore()	GetCoreID()	
StartOS()	StartOS()	StartOS()

StartPostOs Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - SingleCore	Comment
GetCoreID()	GetCoreID()		
SchM_Init()	SchM_Init()	SchM_Init()	
EcuM_AL_DriverInitTwo()	EcuM_AL_DriverInitTwo()	EcuM_AL_DriverInitTwo()	Needed for starting BswM_Init, could be restricted to only start BswM_Init on Slave Core
EcuM_OnRTEStartup()		EcuM_OnRTEStartup()	
WaitEvent until all cores done	Signal to Master Core		
RTE_Start()	Rte_Start()	RTE_Start()	
EcuM_AL_DriverInitThree()		EcuM_AL_DriverInitThree()	
Rte_Switch_currentMode(RUN)		Rte_Switch_currentMode(RUN)	As the RUN Mode is transmitted to the SWCs only by the MasterCore

7.13.2.4 RUN State

RUN II (APP_RUN) Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - Single Core	Comment
EcuM_OnEnterRun()		EcuM_OnEnterRun()	
ComM_CommunicationAllowed(TRUE)		ComM_CommunicationAllowed(TRUE)	For all Channels
Schedule timer for minimum duration to stay in RUN state		Schedule timer for minimum duration to stay in RUN state	
EcuM_GetPendingWakeupEvent()		EcuM_GetPendingWakeupEvent()	While in RUN II State, this is done in the EcuM_Mainfunction
EcuM_StartWakeupSources()		EcuM_StartWakeupSources()	
EcuM_CheckValidation()		EcuM_CheckValidation()	
EcuM_ValidateWakeupeEvent()		EcuM_ValidateWakeupeEvent()	
EcuM_StopWakeupSource()		EcuM_StopWakeupSource()	
Evaluate RUN requests and timer()		Evaluate RUN requests and timer()	
ComM_GetState()		ComM_GetState()	
ComM_CommunicationAllowed(FALSE)		ComM_CommunicationAllowed(FALSE)	No pending RUN and ComM requests
EcuM_OnExitRun()		EcuM_OnExitRun()	
Rte_Switch_currentMode(POSTRUN)		Rte_Switch_currentMode(POSTRUN)	

There are no tasks to be done in the MainFunction of the EcuMFixed on a slave core.

RUN III (POST_RUN) Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - Single Core	Comment
Evaluate POST RUN requests		Evaluate POST RUN requests	
EcuM_OnExitPostRun()		EcuM_OnExitPostRun()	

7.13.2.5 SHUTDOWN State

Individual core shutdown (i.e. while the rest of the ECU continues to run) is currently not supported. All cores are shut down simultaneously.

When the ECU shall be shut down, the master ECU Manager module calls ShutdownAllCores rather than somehow calling ShutdownOs on the individual cores. The ShutdownAllCores stops the OS on all cores and stops all cores as well.

Since the master core could issue the ShutdownAllCores before all slave cores are finished processing, the cores must be synchronized before entering SHUTDOWN.

GO Off One Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - Single Core	Comment
GetCoreID()	GetCoreID()		
EcuM_OnGoOffOne()		EcuM_OnGoOffOne()	
WaitEvent until all cores done	Signal Master Core		
Rte_Stop()	Rte_Stop()	Rte_Stop()	
ComM_Deinit()		ComM_Deinit()	
NvM_WriteAll()		NvM_WriteAll()	
EcuM_CB_NvmJobEnd()		EcuM_CB_NvmJobEnd()	
BswM_Deinit()	Bswm_Deinit()	BswM_Deinit()	
SchM_Deinit()	SchM_Deinit()	SchM_Deinit()	
EcuM_GetPendingWakeupEvents()		EcuM_GetPendingWakeupEvents()	
EcuM_SelectShutdown()		EcuM_SelectShutdown()	
ShutdownAllCores()	SetEvent()	ShutdownOS()	

Go Off Two Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - Single Core	Comment
GetCoreID()	GetCoreID()		
EcuM_OnGoOffTwo()		EcuM_OnGoOffTwo()	
EcuM_AL_SwitchOff()		EcuM_AL_SwitchOff()	

Prep Shutdown Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - Single Core	Comment
GetCoreID()	GetCoreID()		
EcuM_ClearWakeupEvent()		EcuM_ClearWakeupEvent()	
EcuM_OnPrepShutdown()		EcuM_OnPrepShutdown()	
Dem_Shutdown()		Dem_Shutdown()	
Rte_Switch(SHUTDOWN)		Rte_Switch(SHUTDOWN)	

7.13.2.6 SLEEP State

When the shutdown target `Sleep` is requested, all cores are put to sleep simultaneously.

If the ECU is put to sleep, the "halt"s must be synchronized so that all slave cores are halted before the master core computes the checksum. The ECU Manager module on the master core uses the same "signal" mechanism as for synchronizing cores on GoOff.

Similarly, the ECU Manager module on the master core must validate the checksum before releasing the slave cores from the sleep state.

Go Sleep Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - Single Core	Comment
GetCoreID()	GetCoreID()		
EcuM_OnGoSleep()		EcuM_OnGoSleep()	
NvM_WriteAll()		NvM_WriteAll()	
Start timer for shutdown timeout		Start timer for shutdown timeout	
EcuM_GetPendingWakeupEvents()		EcuM_GetPendingWakeupEvents()	
EcuM_CB_NfyNvMJobEnd()		EcuM_CB_NfyNvMJobEnd()	
EcuM_EnableWakeupSources()		EcuM_EnableWakeupSources()	
GetResource()	GetResource()		

Sleep I Sequence

EcuMFixed - Master	EcuMFixed - Slave	EcuMFixed - Single Core	Comment
GetCoreID()	GetCoreID()		
DisableAllInterrupts()		DisableAllInterrupts()	
EcuM_GenerateRamHash()		EcuM_GenerateRamHash()	
Mcu_SetMode()		Mcu_SetMode()	
Interrupt()		Interrupt()	
EcuM_CheckWakeup()		EcuM_CheckWakeup()	
Activate PLL		Activate PLL	
<Module>_CheckWakeup()		<Module>_CheckWakeup()	
EcuM_SetWakeupEvent()		EcuM_SetWakeupEvent()	
EnableInterrupts()		EnableInterrupts()	
EcuM_CheckRamHash()		EcuM_CheckRamHash()	

Sleep II Sequence

EcuMFixed - Master Core	EcuMFixed - Slave Core	EcuMFixed - Single Core	Comment
GetCoreID()	GetCoreID()		
Mcu_SetMode()		Mcu_SetMode()	
EcuM_SleepActivity()		EcuM_SleepActivity()	
EcuM_CheckWakeup()		EcuM_CheckWakeup()	
<Module>_CheckWakeup()		<Module>_CheckWakeup()	
EcuM_SetWakeupEvent()		EcuM_SetWakeupEvent()	
EcuM_GetPendingWakeupEvents()		EcuM_GetPendingWakeupEvents()	

Wakeup I Sequence

EcuMFixed - Master Core	EcuMFixed - Slave Core	EcuMFixed - Single Core	Comment
Mcu_SetMode()		Mcu_SetMode()	

EcuM_GetPendingWakeupEvents		EcuM_GetPendingWakeupEvents	
EcuM_DisableWakeupSources		EcuM_DisableWakeupSources	
EcuM_AL_DriverRestart		EcuM_AL_DriverRestart	
ReleaseResource()	ReleaseResource()	ReleaseResource()	

Wakeup Validation Sequence

EcuMFixed - Master Core	**EcuMFixed - Slave Core**	**EcuMFixed - Single Core**	**Comment**
GetCoreID()	GetCoreID()		
EcuM_GetPendingWakeupEvents()		EcuM_GetPendingWakeupEvents()	
EcuM_StartWakeupSources()		EcuM_StartWakeupSources()	
Start validation timeout		Start validation timeout	
EcuM_CheckValidation()		EcuM_CheckValidation()	
EcuM_StopWakeupSources()		EcuM_StopWakeupSources()	

Wakeup II Sequence

EcuMFixed - Master Core	**EcuMFixed - Slave Core**	**EcuMFixed - Single Core**	**Comment**
Dem_Init()		Dem_Init()	
EcuM_CB_NfyNvMJobEnd()		EcuM_CB_NfyNvMJobEnd()	
EcuM_AL_DriverInitThree()		EcuM_AL_DriverInitThree()	
Rte_Switch(RUN)		Rte_Switch(RUN)	

8 API specification

8.1 Imported Types

In this chapter all types included from the following files are listed:

[SWS_EcuM_02810] ⌈

Module	*Imported Type*
Adc	Adc_ConfigType
BswM	BswM_ConfigType
Can	Can_ConfigType
CanIf	CanIf_ConfigType
CanNm	CanNm_ConfigType
CanSM	CanSM_ConfigType
CanTSyn	CanTSyn_ConfigType
CanTp	CanTp_ConfigType
CanTrcv	CanTrcv_ConfigType
Com	Com_ConfigType
ComM	ComM_ConfigType
	ComM_StateType
ComStack_Types	NetworkHandleType
Dcm	Dcm_ConfigType
Dem	Dem_ConfigType
Det	Det_ConfigType
Dlt	Dlt_ConfigType
DoIP	DoIP_ConfigType
Ea	Ea_ConfigType
Eep	Eep_ConfigType
Eth	Eth_ConfigType
EthIf	EthIf_ConfigType
EthSwt	EthSwt_ConfigType
EthTSyn	EthTSyn_ConfigType
Eth_GeneralTypes	EthTrcv_ConfigType
Fee	Fee_ConfigType
FiM	FiM_ConfigType
Fls	Fls_ConfigType
Fr	Fr_ConfigType
FrArTp	FrArTp_ConfigType
FrIf	FrIf_ConfigType
FrNm	FrNm_ConfigType
FrSm	FrSM_ConfigType
FrTSyn	FrTSyn_ConfigType
FrTp	FrTp_ConfigType
Gpt	Gpt_ConfigType
Icu	Icu_ConfigType
IoHwAb	IoHwAb<Init_Id>_ConfigType
IpduM	IpduM_ConfigType
J1939Dcm	J1939Dcm_ConfigType
J1939Nm	J1939Nm_ConfigType
J1939Rm	J1939Rm_ConfigType
J1939Tp	J1939Tp_ConfigType

LdCom	LdCom_ConfigType
Lin	Lin_ConfigType
LinIf	LinIf_ConfigType
	LinTp_ConfigType
LinSM	LinSM_ConfigType
McOs	AppModeType
	CoreIdType
Mcu	Mcu_ConfigType
	Mcu_ModeType
	Mcu_ResetType
Nm	Nm_ConfigType
NvM	NvM_ConfigType
	NvM_RequestResultType
Ocu	Ocu_ConfigType
Os	StatusType
PduR	PduR_PBConfigType
Port	Port_ConfigType
Pwm	Pwm_ConfigType
Rte	Rte_ModeType_EcuM_Mode
SchM	SchM_ConfigType
Sd	Sd_ConfigType
SecOC	SecOC_ConfigType
SoAd	SoAd_ConfigType
Spi	Spi_ConfigType
StbM	StbM_ConfigType
Std_Types	Std_ReturnType
	Std_VersionInfoType
TcpIp	TcpIp_ConfigType
UdpNm	UdpNm_ConfigType
Wdg	Wdg_ConfigType
WdgM	WdgM_ConfigType
Xcp	Xcp_ConfigType

⌋ ()

8.2 Service Interfaces

8.2.1 Use Cases for System Services

[SWS_EcuM_00762] ⌈The ECU State Manager Fixed module shall provide System Services for the following functionalities:

- requesting RUN
- releasing RUN
- requesting POST_RUN
- releasing POST_RUN

⌋ ()

[SWS_EcuM_02763] ⌈The ECU State Manager Fixed module shall provide also System Services for the following functionality:

- selecting and getting shutdown target
- selecting and getting boot targets

⌋ ()

8.2.2 Port Interface for Interface EcuM_StateRequest

A SW-C which needs to keep the ECU alive or needs to execute any operations before the ECU is shut down shall require the client-server interface `EcuM_StateRequest`.
This interface uses port-defined argument values to identify the user that requests modes. See [SWS_Rte_1360] in [18] for a description of port-defined argument values.

8.2.2.1 Data Types

[SWS_EcuMf_00048] ⌈

Name	EcuM_UserType
Kind	Type
Derived from	uint8
Description	Unique value for each user.
Variation	--

⌋ ()

[SWS_EcuM_00487] ⌈For each user, a unique value must be defined at system generation time, please refer to 10.3 Configurable Parameters. ⌋ ()

8.2.2.2 EcuM_StateRequest

[SWS_EcuMf_00030] ⌈

Name	EcuM_StateRequest	
Comment	Interface to request a specific ECU state	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

ReleasePOSTRUN		
Comments	--	
Variation	--	
Possible	E_OK	The request was accepted by EcuM

Errors	E_NOT_OK	The request was not accepted by EcuM, a detailed error condition was sent to DET
ReleaseRUN		
Comments	--	
Variation	--	
Possible Errors	E_OK	The request was accepted by EcuM
	E_NOT_OK	The request was not accepted by EcuM, a detailed error condition was sent to DET
RequestPOSTRUN		
Comments	--	
Variation	--	
Possible Errors	E_OK	The request was accepted by EcuM
	E_NOT_OK	The request was not accepted by EcuM, a detailed error condition was sent to DET
RequestRUN		
Comments	--	
Variation	--	
Possible Errors	E_OK	The request was accepted by EcuM
	E_NOT_OK	The request was not accepted by EcuM, a detailed error condition was sent to DET

⌋ ()

The ECU State Manager Fixed module provides additional calls which would typically be made by one management instance on the ECU as they have a global impact. The function "`EcuM_KillAllRUNRequests()`" unconditionally undoes all requests to RUN. Because of this, calling `EcuM_RequestRUN` does not necessarily guarantee that the ECU will stay awake until calling `EcuM_ReleaseRUN` (e.g. a `KillAllRUNRequests`-call can override the wish of individual users for the ECU to stay awake). The function "`EcuM_KillAllRUNRequests()`" is not accessible over the RTE and thus can not be used by SW-Cs.

The following activity chart is not normative and shall help the application software programmer to understand when to request which state via EcuM_StateRequest.

- RequestRUN is actually prohibiting shutdown.
- RequestPOSTRUN is actually requesting the opportunity to de-initialize.
- The application software components shall only listen to the EcuM state over the mode switch port.
- RUN state can be requested (RequestRUN) and released (ReleaseRUN) arbitrarily often during operation.

The EcuM is the mode manager, but does not have a requestmode (RTE ModeGroup) interface.

When all RUN requests are released, the ECU shuts down, so the active Software Components have to request RUN (RequestRUN) right after their initialization, otherwise the ECU may shut down immediately.

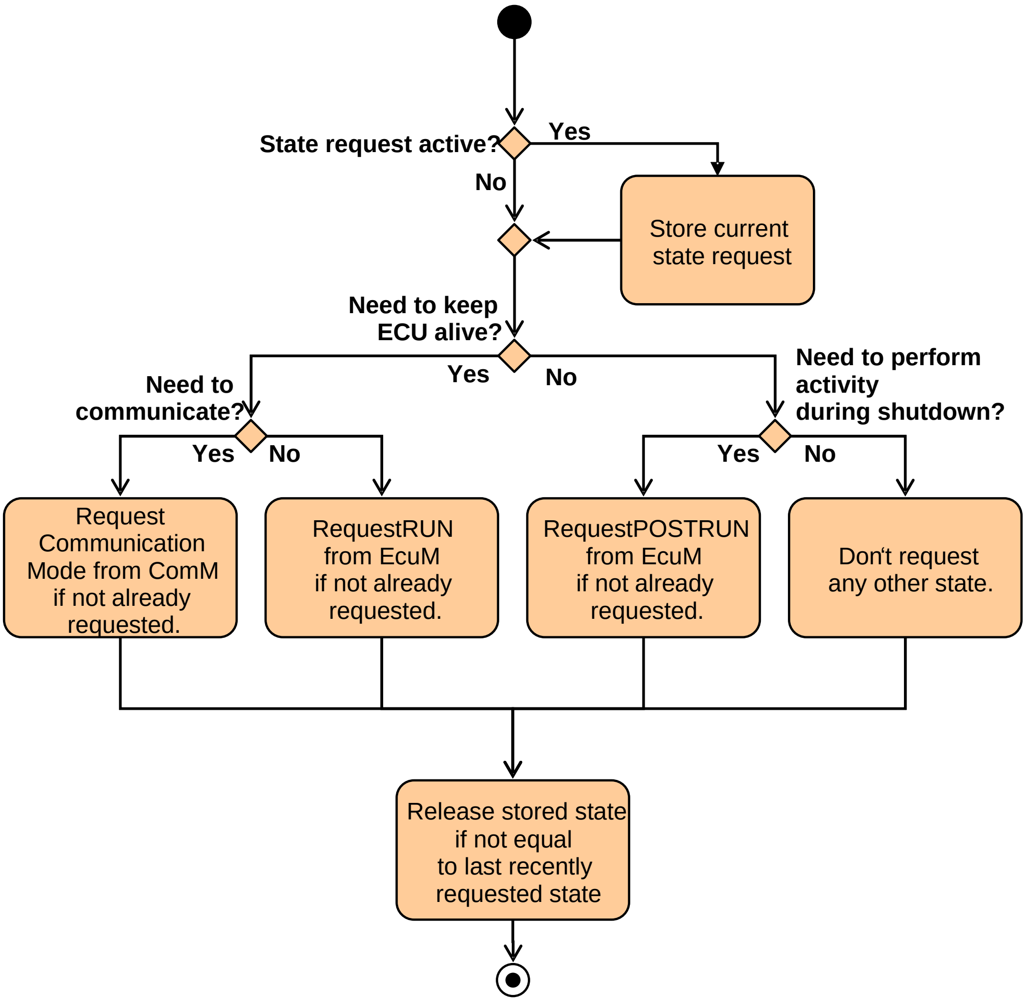

State request active?
Yes
No
Store current state request
Need to keep ECU alive?
Yes
No
Need to communicate?
Need to perform activity during shutdown?
Yes
No
Yes
No
Request Communication Mode from ComM if not already requested.
RequestRUN from EcuM if not already requested.
RequestPOSTRUN from EcuM if not already requested.
Don't request any other state.
Release stored state if not equal to last recently requested state

Figure 31 – Activity chart for SW-C

8.2.2.3 Port Interface for Interface EcuM_CurrentMode

[SWS_EcuM_00749] **⌈**The mode port of the ECU State Manager Fixed module shall declare the following modes:

- STARTUP
- RUN
- POST_RUN
- SLEEP
- WAKE_SLEEP
- SHUTDOWN

⌋ ()

This definition is a simplified view of ECU Modes that applications do need to know. It does not restrict or limit in any way how application modes could be defined. Applications modes are completely handled by the application itself.

[SWS_EcuM_00750] **⌈**Mode changes shall be notified to SW-Cs through the RTE mode ports when the mode change occurs. The ECU State Manager Fixed module shall not wait until the RTE has performed the mode switch completely. ⌋ ()

This specification assumes that the port name is currentMode and that the direct API of RTE will be used. Under these conditions mode changes signaled by invoking

```
Rte_StatusType Rte_Switch_currentMode_currentMode(
        Rte_ModeType_EcuM_Mode mode)
```

where `mode` is the new mode to be notified. The value range is specified by the previous requirement. The return value shall be ignored.

A SW-C which wants to be notified of mode changes should require the mode switch interface `EcuM_CurrentMode`.

The following figure shows how the defined modes are mapped to the states of the ECU State Manager Fixed module and when the notifications shall occur.

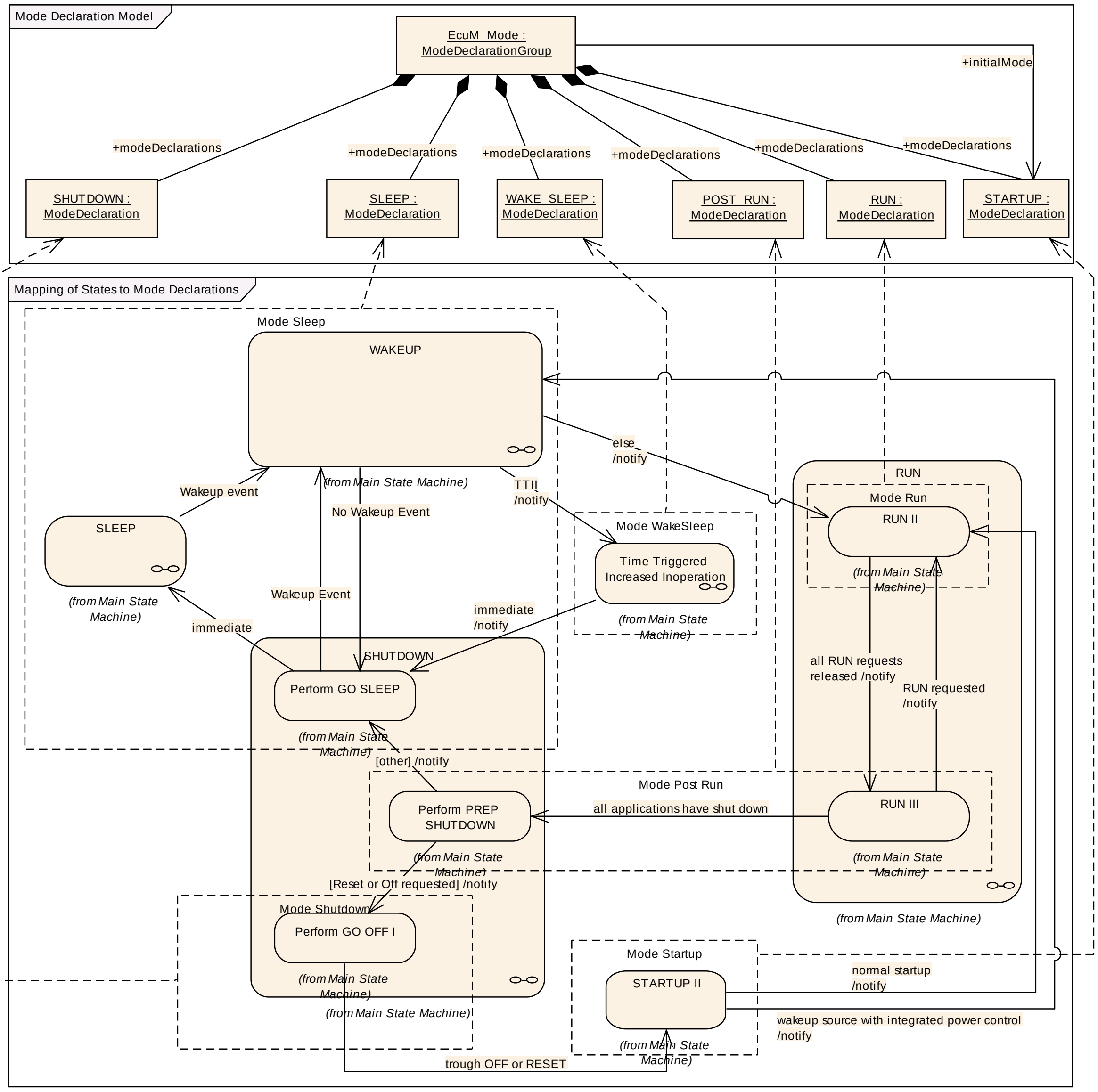

Mode Declaration Model
EcuM_Mode : ModeDeclarationGroup
+initialMode
+modeDeclarations
+modeDeclarations
+modeDeclarations
+modeDeclarations
+modeDeclarations
+modeDeclarations
SHUTDOWN : ModeDeclaration
SLEEP : ModeDeclaration
WAKE_SLEEP : ModeDeclaration
POST_RUN : ModeDeclaration
RUN : ModeDeclaration
STARTUP : ModeDeclaration
Mapping of States to Mode Declarations
Mode Sleep
WAKEUP
else /notify
TTII /notify
(from Main State Machine)
No Wakeup Event
Wakeup event
SLEEP
(from Main State Machine)
Mode WakeSleep
Time Triggered Increased Inoperation
(from Main State Machine)
Wakeup Event
immediate
immediate /notify
SHUTDOWN
Perform GO SLEEP
(from Main State Machine)
[other] /notify
RUN
Mode Run
RUN II
(from Main State Machine)
all RUN requests released /notify
RUN requested /notify
Mode Post Run
all applications have shut down
Perform PREP SHUTDOWN
(from Main State Machine)
RUN III
(from Main State Machine)
(from Main State Machine)
[Reset or Off requested] /notify
Mode Shutdown
Perform GO OFF I
(from Main State Machine)
(from Main State Machine)
Mode Startup
STARTUP II
(from Main State Machine)
normal startup /notify
wakeup source with integrated power control /notify
trough OFF or RESET

Figure 32 – Mapping of Declared Modes to states of ECU State Manager Fixed module

[SWS_EcuM_00752] ⌈The ECU State Manager Fixed module shall notify WakeSleep mode and Sleep mode when transiting from `WAKEUP` to `SHUTDOWN`, but only if the selected shutdown target is `SLEEP`.
This allows the system designer to trigger runnables for TTII. ⌋ ()

8.2.2.4 Data Types

The mode declaration group `EcuM_Mode` represents the modes of the ECU State Manager Fixed module that will be notified to the SW-Cs.

[SWS_EcuMf_00104] ⌈

Name	EcuM_Mode

Kind	ModeDeclarationGroup	
Category	ALPHABETIC_ORDER	
Initial mode	STARTUP	
On transition value	--	
Modes	POST_RUN	--
	RUN	--
	SHUTDOWN	--
	SLEEP	--
	STARTUP	--
	WAKE_SLEEP	--
Description	--	

⌋ ()

8.2.2.5 EcuM_CurrentMode

[SWS_EcuMf_00031] ⌈

Name	EcuM_CurrentMode	
Comment	Interface to read the current ECU mode	
IsService	true	
Variation	--	
ModeGroup	currentMode	EcuM_Mode

⌋ ()

8.2.3 Ports and Port Interface for Interface EcuM_ShutdownTarget

A SW-C which wants to select a shutdown target should require the client-server interface `EcuM_ShutdownTarget`.

8.2.3.1 Data Types

This data type represents the states of the ECU State Manager Fixed module and thus includes the shutdown targets.

[SWS_EcuMf_00105] ⌈

Name	EcuM_StateType
Kind	Type

Derived from	uint8		
Description	ECU State Manager states.		
Range	ECUM_SUBSTATE_MASK	0x0f	--
	ECUM_STATE_STARTUP	0x10	--
	ECUM_STATE_STARTUP_ONE	0x11	--
	ECUM_STATE_STARTUP_TWO	0x12	--
	ECUM_STATE_WAKEUP	0x20	--
	ECUM_STATE_WAKEUP_ONE	0x21	--
	ECUM_STATE_WAKEUP_VALIDATION	0x22	--
	ECUM_STATE_WAKEUP_REACTION	0x23	--
	ECUM_STATE_WAKEUP_TWO	0x24	--
	ECUM_STATE_WAKEUP_WAKESLEEP	0x25	--
	ECUM_STATE_WAKEUP_TTII	0x26	--
	ECUM_STATE_RUN	0x30	--
	ECUM_STATE_APP_RUN	0x32	--
	ECUM_STATE_APP_POST_RUN	0x33	--
	ECUM_STATE_SHUTDOWN	0x40	--
	ECUM_STATE_PREP_SHUTDOWN	0x44	--
	ECUM_STATE_GO_SLEEP	0x49	--
	ECUM_STATE_GO_OFF_ONE	0x4d	--
	ECUM_STATE_GO_OFF_TWO	0x4e	--
	ECUM_STATE_SLEEP	0x50	--
	ECUM_STATE_OFF	0x80	--
	ECUM_STATE_RESET	0x90	--
Variation	--		

⌋ ()

[SWS_EcuM_00507] ⌈Encodes states and sub-states of the ECU State Manager. States are encoded in the high-nibble, sub-state in the low-nibble. The sub-state can be determined by ANDing the state value with `ECUM_SUBSTATE_MASK`. ⌋ ()

[SWS_EcuM_02664] ⌈The ECU State Manager Fixed module shall define all states as listed in the `EcuM_StateType`. ⌋ ()

8.2.3.2 EcuM_ShutdownTarget

[SWS_EcuMf_00032] ⌈

Name	EcuM_ShutdownTarget	
Comment	Interfaces to manage the shutdown target	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

GetLastShutdownTarget			
Comments	GetLastShutdownTarget returns the shutdown target of the previous shutdown process.		
Variation	--		
Parameters	shutdownTarget	Comment	--
		Type	EcuM_StateType
		Variation	--
		Direction	OUT
	sleepMode	Comment	--
		Type	EcuM_SleepModeType
		Variation	--
		Direction	OUT
Possible Errors	E_OK	Operation successful	
	E_NOT_OK	--	
GetShutdownTarget			
Comments	GetShutdownTarget returns the currently selected shutdown target as set by SelectShutdownTarget		
Variation	--		
Parameters	shutdownTarget	Comment	--
		Type	EcuM_StateType
		Variation	--

		Direction	OUT
	sleepMode	Comment	--
		Type	EcuM_SleepModeType
		Variation	--
		Direction	OUT
Possible Errors	E_OK	Operation successful	
	E_NOT_OK	--	
SelectShutdownTarget			
Comments	Select a new shutdown target		
Variation	--		
Parameters	shutdownTarget	Comment	--
		Type	EcuM_StateType
		Variation	--
		Direction	IN
	sleepMode	Comment	--
		Type	EcuM_SleepModeType
		Variation	--
		Direction	IN
Possible Errors	E_OK	The new shutdown target was set.	
	E_NOT_OK	The new shutdown target was not set	

⌋ ()

The parameter mode determines the concrete sleep mode. This parameter shall only be used if the target parameter equals to ECUM_STATE_SLEEP, otherwise it will be ignored.

8.2.4 Port Interface for Interface EcuM_BootTarget

A SW-C which wants to select a boot target shall require the client-server interface `EcuM_BootTarget`.

8.2.4.1 Data Types

This data type represents the boot targets the ECU State Manager Fixed module can be configured with.

[SWS_EcuMf_00036] ⌈

Name	EcuM_BootTargetType		
Kind	Type		
Derived from	uint8		
Description	This type represents the boot targets the ECU Manager module can be configured with. The default boot target is ECUM_BOOT_TARGET_OEM_BOOTLOADER.		
Range	ECUM_BOOT_TARGET_APP	0	The ECU will boot into the application
	ECUM_BOOT_TARGET_OEM_BOOTLOADER	1	The ECU will boot into the OEM bootloader
	ECUM_BOOT_TARGET_SYS_BOOTLOADER	2	The ECU will boot into the system supplier bootloader
Variation	--		

⌋ ()

8.2.4.2 EcuM_BootTarget

[SWS_EcuMf_00033] ⌈

Name	EcuM_BootTarget	
Comment	Interfaces to manage the boot target	
IsService	true	
Variation	--	
Possible Errors	0	E_OK
	1	E_NOT_OK

Operations

GetBootTarget			
Comments	Read the current boot target		
Variation	--		
Parameters	target	Comment	--
		Type	EcuM_BootTargetType
		Variation	--

		Direction	OUT
Possible Errors	E_OK	Operation successful (the service always succeeds)	
	E_NOT_OK	--	
SelectBootTarget			
Comments	Select a new boot target		
Variation	--		
Parameters	target	Comment	--
		Type	EcuM_BootTargetType
		Variation	--
		Direction	IN
Possible Errors	E_OK	The new boot target was accepted by EcuM	
	E_NOT_OK	The new boot target was not accepted by EcuM	

⌋ ()

8.2.5 Definition of the Service ECU State Manager

This section provides guidance on the definition of the ECU State Manager service. Note that these definitions can only be completed during ECU configuration (because it depends on certain configuration parameters of the ECU State Manager Fixed module which determine the number of ports provided by the ECU State Manager service). Also note that the implementation of a SW-C does *not* depend on these definitions.
There are ports on both sides of the RTE: This description of the ECU State Manager service defines the ports below the RTE. Each SW-Component, which uses the service, must contain “service ports” in its own SW-C description which will be connected to the ports of the ECU State Manager, so that the RTE can be generated.

[SWS_EcuMf_00112] ⌈

Name	BootTarget_{UserName}		
Kind	ProvidedPort	Interface	EcuM_BootTarget
Description	Provides an interface to SW-Cs to select a new boot target and query the current boot target.		
Variation	UserName = {ecuc(EcuM/EcuMConfiguration/EcuMFixedConfiguration/EcuMFixedUserConfig/EcuMFixedUser.SHORT-NAME)}		

⌋ ()

[SWS_EcuMf_00113] ⌈

Name	currentMode		
Kind	ProvidedPort	Interface	EcuM_CurrentMode
Description	Provides an interface to SW-Cs to get notified about state changes of the ECU.		
Variation	--		

⌋ ()

[SWS_EcuMf_00114] ⌈

Name	ShutdownTarget_{UserName}		
Kind	ProvidedPort	Interface	EcuM_ShutdownTarget
Description	Provides an interface to SW-Cs to select a new shutdown target and query the current shutdown target.		
Variation	UserName = {ecuc(EcuM/EcuMConfiguration/EcuMFixedConfiguration/ EcuMFixedUserConfig/EcuMFixedUser.SHORT-NAME)}		

⌋ ()

[SWS_EcuMf_00115] ⌈

Name	StateRequest_{UserName}		
Kind	ProvidedPort	Interface	EcuM_StateRequest
Description	Provides an interface to SW-Cs to request state changes of the ECU state. The port uses port-defined argument values to identify the user.		
Port Defined Argument Value(s)	Type	EcuM_UserType	
	Value	{ecuc(EcuM/EcuMConfiguration/EcuMFixedConfiguration/ EcuMFixedUserConfig/EcuMFixedUser.value)}	
Variation	UserName = {ecuc(EcuM/EcuMConfiguration/EcuMFixedConfiguration/ EcuMFixedUserConfig/EcuMFixedUser.SHORT-NAME)}		

⌋ ()

8.2.6 Runnables and Entry points

8.2.6.1 Internal behavior

This is the inside description of the ECU State Manager. This detailed description is only needed for the configuration of the local RTE.

```
InternalBehavior EcuStateManager {

      // Runnable entities of the EcuStateManager
      RunnableEntity RequestRUN
            symbol "EcuM_RequestRUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity ReleaseRUN
            symbol "EcuM_ReleaseRUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity RequestPOSTRUN
            symbol “EcuM_RequestPOST_RUN”
            canbeInvokedConcurrently = TRUE
      RunnableEntity ReleasePOSTRUN
            symbol “EcuM_ReleasePOST_RUN”
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectShutdownTarget
            symbol “EcuM_SelectShutdownTarget”
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetShutdownTarget
            symbol “EcuM_GetShutdownTarget”
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetLastShutdownTarget
            symbol “EcuM_GetLastShutdownTarget”
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectBootTarget
            symbol “EcuM_SelectBootTarget”
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetBootTarget
            symbol “EcuM_GetBootTarget”
            canbeInvokedConcurrently = TRUE

      // Port present for each user. There are NU users
      SR000.RequestRUN -> RequestRUN
      SR000.ReleaseRUN -> ReleaseRUN
      SR000.RequestPOSTRUN -> RequestPOSTRUN
      SR000.ReleasePOSTRUN -> RequestPOSTRUN
      PortArgument {port=SR000, value.type=EcuM_UserType,
      value.value=EcuM_User[0].User}
      (...)
      SRnnn.RequestRUN -> RequestRUN
      SRnnn.ReleaseRUN -> ReleaseRUN
      SRnnn.RequestPOSTRUN -> RequestPOSTRUN
      SRnnn.ReleasePOSTRUN -> RequestPOSTRUN
      PortArgument {port=SRnnn, value.type=EcuM_UserType,
      value.value=EcuM_User[nnn].User}

      shutDownTarget.SelectShutdownTarget -> SelectShutdownTarget
      shutDownTarget.GetShutdownTarget -> GetShutdownTarget
      shutDownTarget.GetLastShutdownTarget -> GetLastShutdownTarget
      bootTarget.SelectBootTarget -> SelectBootTarget
      bootTarget.GetBootTarget -> GetBootTarget
};
```

8.3 Type definitions

8.3.1 EcuM_ConfigType

[SWS_EcuMf_00046] ⌈

Name:	`EcuM_ConfigType`	
Type:	`Structure`	
Range:	–	The content of this structure depends on the post-build configuration of EcuM.
Description:	A pointer to such a structure shall be provided to the ECU State Manager initialization routine for configuration.	

⌋ ()

[SWS_EcuM_02801] ⌈This structure shall hold the post-build configuration parameters for the ECU State Manager Fixed module as well as pointers to all `ConfigType` structures of modules that are initialized by the ECU State Manager. ⌋ ()

[SWS_EcuM_00793] ⌈The ECU State Manager Configuration Tool shall specifically generate this structure for a given set of basic software modules that comprise the ECU configuration. The set of basic software modules is derived from the corresponding *EcuMFixedModuleConfiguration* parameters. ⌋ ()

[SWS_EcuM_02794] ⌈This structure shall contain an additional post-build configuration variant identifier (uint8/uint16/uint32 depending on algorithm to compute the identifier). See also chapter *10.4* Checking Configuration Consistency*.* ⌋ ()

[SWS_EcuM_02795] ⌈This structure shall contain an additional hash code with is tested against the configuration parameter *EcuMConfigConsistencyHash* for checking consistency of the configuration data. See also chapter *10.4* Checking Configuration Consistency*.* ⌋ ()

[SWS_EcuM_00800] ⌈The ECU State Manager Configuration Tool shall also generate for each given ECU configuration an instance of this structure that is filled with the post-build configuration parameters of the ECU State Manager Fixed module as well as pointers to instances of configuration structures for the modules mentioned in **SWS_EcuM_00793**. The pointers are derived from the corresponding *EcuMFixedModuleConfiguration* parameters. ⌋ ()

8.3.2 EcuM_SleepModeType

[SWS_EcuMf_00119] ⌈

Name	EcuM_SleepModeType
Kind	Type
Derived from	uint8
Description	Configured Sleep Modes
Variation	--

⌋ ()

8.3.3 EcuM_WakeupSourceType

[SWS_EcuMf_00049] ⌈

Name:	`EcuM_WakeupSourceType`		
Type:	`uint32`		
Range:	`ECUM_WKSOURCE_POWER`	--	Power cycle (bit 0)

	`ECUM_WKSOURCE_RESET`
	`(default)`
		--	Hardware reset (bit 1).
			If the Mcu driver cannot distinguish between a power cycle and a reset reason, then this shall be the default wakeup source.

	`ECUM_WKSOURCE_INTERNAL_RESET`	--	Internal reset of µC (bit 2) The internal reset typically only resets the µC core but not peripherals or memory controllers. The exact behavior is hardware specific. This source may also indicate an unhandled exception.
	`ECUM_WKSOURCE_INTERNAL_WDG`	--	Reset by internal watchdog (bit 3)
	`ECUM_WKSOURCE_EXTERNAL_WDG`	--	Reset by external watchdog (bit 4), if detection supported by hardware
Description:	EcuM_WakeupSourceType defines a bitfield with 5 pre-defined positions (see Range). The bitfield provides one bit for each wakeup source. In WAKEUP, all bits cleared indicates that no wakeup source is known. In STARTUP, all bits cleared indicates that no reason for restart or reset is known. In this case, ECUM_WKSOURCE_RESET shall be assumed.		

⌋ ()

[SWS_EcuM_02165] ⌈The list can be extended by configuration⌋ ()

[SWS_EcuM_02166] ⌈Extension values (see chapter 10.3 Configurable Parameters) must define single additional bits. The bit assignment shall be done by the configuration tool. ⌋ ()

[SWS_EcuMf_00002] ⌈The following rule applies for extension values of type `EcuM_WakeupSourceType`:
`EcuMWakeupSourceId` defines the bit position of the corresponding wake up source in `EcuM_WakeupSourceType`.
Values 0 to 4 are not allowed for `EcuMWakeupSourceId` (pre-defined values in `EcuM_WakeupSourceType`)
Values 5 up to 31 in `EcuMWakeupSourceId` implies `0x00000020` up to `0x8000000000` in `EcuM_WakeupSourceType` (i.e. bit 5 up to bit 31).⌋ ()

[SWS_EcuM_02601] ⌈If hardware cannot detect a specific wake up source, then the ECU State Manager Fixed module shall report `ECUM_WKSOURCE_RESET` instead. ⌋ ()

8.3.4 EcuM_WakeupStatusType

[SWS_EcuMf_00050] ⌈

Name:	`EcuM_WakeupStatusType`
Type:	`uint8`

Range:	ECUM_WKSTATUS_NONE	0	No pending wakeup event was detected
	ECUM_WKSTATUS_PENDING	1	The wakeup event was detected but not yet validated
	ECUM_WKSTATUS_VALIDATED	2	The wakeup event is valid
	ECUM_WKSTATUS_EXPIRED	3	The wakeup event has not been validated and has expired therefore
	ECUM_WKSTATUS_ENABLED	6	The wakeup source is enabled (armed) and is ready to call EcuM_SetWakeupEvent().
Description:	The type describes the possible states of a wakeup source.		

⌋ ()

See also *8.4.4.5* EcuM_GetStatusOfWakeupSource.

8.3.5 EcuM_WakeupReactionType

[SWS_EcuMf_00051] ⌈

Name:	EcuM_WakeupReactionType		
Type:	uint8		
Range:	ECUM_WKACT_RUN	0x00	Initialization into RUN state
	ECUM_WKACT_TTII	0x02	Execute time triggered increased inoperation protocol and shutdown
	ECUM_WKACT_SHUTDOWN	0x03	Immediate shutdown
Description:	The type describes the possible outcomes of the WAKEUP REACTION state.		

⌋ ()

8.4 Function Definitions

8.4.1 General

8.4.1.1 EcuM_GetVersionInfo

[SWS_EcuM_02813] ⌈

Service name:	EcuM_GetVersionInfo	

Syntax:	`void EcuM_GetVersionInfo(`
	`    Std_VersionInfoType* versioninfo`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Returns the version information of this module.	

⌋ (SRS_BSW_00407,SRS_BSW_00411)

[SWS_EcuMf_00034] ⌈Parameter `versioninfo` of the function `EcuM_GetVersionInfo`: An implementation shall cope with NULL pointers by returning immediately without any further action. ⌋ ()

Hint:
If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.

8.4.2 Initialization and Shutdown

8.4.2.1 EcuM_Init

[SWS_EcuM_02811] ⌈

Service name:	EcuM_Init

Syntax:	`void EcuM_Init(`
	`    void`
	`)`

Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Initializes the ECU state manager and carries out the startup procedure. The function will never return (it calls StartOS)

⌋ (SRS_BSW_00358,SRS_BSW_00414,SRS_BSW_00101)

8.4.2.2 EcuM_StartupTwo

[SWS_EcuM_02838] ⌈

Service name:	EcuM_StartupTwo

Syntax:	`void EcuM_StartupTwo(`
	`    void`
	`)`

Service ID[hex]:	0x1a
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This function implements the STARTUP II state.

⌋ ()

[SWS_EcuM_02806] ⌈This function must be called from a task which is started directly as a consequence of StartOS. I.e. either it must be called from an autostart task or it must be called from a task which is explicitely started. ⌋ ()

8.4.2.3 EcuM_Shutdown

[SWS_EcuM_02812] ⌈

Service name:	EcuM_Shutdown

Syntax:	`void EcuM_Shutdown(`
	`    void`
	`)`

Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Typically called from the shutdown hook, this function takes over execution control and will carry out GO OFF II activities.

⌋ (SRS_ModeMgm_09114)

8.4.3 State Management

8.4.3.1 EcuM_RequestRUN

[SWS_EcuM_04124] ⌈

Service name:	EcuM_RequestRUN	

Syntax:	`Std_ReturnType EcuM_RequestRUN(`
	`    EcuM_UserType user`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	ID of the entity requesting the RUN state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request was accepted by EcuM. E_NOT_OK: The request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes below).
Description:	Places a request for the RUN state. Requests can be placed by every user made known to the state manager at configuration time.	

⌋ ()

[SWS_EcuM_00143] ⌈Requests of `EcuM_RequestRUN` cannot be nested, i.e. one user can only place one request but not more. ⌋ ()

[SWS_EcuM_00144] ⌈An implementation must track requests for each user known on the ECU. Run requests are specific to the user. ⌋ ()

[SWS_EcuM_00668] ⌈RUN requests shall be ignored after EcuM_KillAllRUNRequests has been executed until the shutdown has completed. ⌋ ()

Configuration of `EcuM_RequestRUN`: Refer to 8.2.2.1 Data Types for more information about user IDs and their generation.

Error Codes of `EcuM_RequestRUN`: `ECUM_E_MULTIPLE_RUN_REQUESTS`: On multiple requests by the same user ID

8.4.3.2 EcuM_ReleaseRUN

[SWS_EcuM_00815] ⌈

Service name:	EcuM_ReleaseRUN	

Syntax:	`Std_ReturnType EcuM_ReleaseRUN(`
	`    EcuM_UserType user`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	ID of the entity releasing the RUN state.
Parameters (inout):	None	
Parameters (out):	None	

Return value:	Std_ReturnType	E_OK: The release request was accepted by EcuM E_NOT_OK: The release request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes below).
Description:	Releases a RUN request previously done with a call to EcuM_RequestRUN. The service is intended for implementing AUTOSAR ports.	

⌋ ()

Configuration of `EcuM_ReleaseRUN`: Refer to 8.2.2.1 Data Types for more information about user IDs and their generation.

Error Codes of `EcuM_ReleaseRUN`: `ECUM_E_MISMATCHED_RUN_RELEASE`: On releasing without a matching request.

8.4.3.3 EcuM_RequestPOST_RUN

[SWS_EcuM_00819] ⌈

Service name:	EcuM_RequestPOST_RUN	

Syntax:	`Std_ReturnType EcuM_RequestPOST_RUN(`
	`    EcuM_UserType user`
	`)`	

Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	ID of the entity requesting the POST RUN state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request was accepted by EcuM E_NOT_OK: The request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes below).
Description:	Places a request for the POST RUN state. Requests can be placed by every user made known to the state manager at configuration time. Requests for RUN and POST RUN must be tracked independently (in other words: two independent variables). The service is intended for implementing AUTOSAR ports.	

⌋ ()

All requirements of *8.4.3.1* EcuM_RequestRUN apply accordingly to the function EcuM_RequestPOST_RUN.

Configuration of `EcuM_RequestPOST_RUN`: Refer to 8.2.2.1 Data Types for more information about user IDs and their generation.

Error Codes of `EcuM_RequestPOST_RUN`: `ECUM_E_MULTIPLE_RUN_REQUESTS`: On multiple requests by the same user ID.

8.4.3.4 EcuM_ReleasePOST_RUN

[SWS_EcuM_04129] ⌈

Service name:	EcuM_ReleasePOST_RUN
Syntax:	`Std_ReturnType EcuM_ReleasePOST_RUN(`

	`EcuM_UserType user`
	`)`	

Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	ID of the entity releasing the POST RUN state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The release request was accepted by EcuM E_NOT_OK: The release request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes below).
Description:	Releases a POST RUN request previously done with a call to EcuM_RequestPOST_RUN. The service is intended for implementing AUTOSAR ports.	

⌋ (SRS_ModeMgm_09116)

Configuration of `EcuM_ReleasePOST_RUN`: Refer to to 8.2.2.1 Data Types for more information about user IDs and their generation.

Error Codes of `EcuM_ReleasePOST_RUN`: `ECUM_E_MISMATCHED_RUN_RELEASE`: On releasing without a matching request.

8.4.3.5 EcuM_KillAllRUNRequests

[SWS_EcuM_00821] ⌈

Service name:	EcuM_KillAllRUNRequests

Syntax:	`void EcuM_KillAllRUNRequests(`
	`    void`
	`)`

Service ID[hex]:	0x05
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The benefit of this function over an ECU reset is that the shutdown sequence is executed, which e.g. takes care of writing back NV memory contents.

⌋ ()

[SWS_EcuM_00872] ⌈The function unconditionally clears all requests to RUN.
Note: As an effect the ECU State Manager switches to RUN III state (see also **SWS_EcuM_00311**), which allows for a controlled shutdown. ⌋ ()

[SWS_EcuM_00600] ⌈As a consequence EcuM_RequestRUN must not accept any new requests unless the resulting shutdown has been completed. ⌋ ()

Caveat of `EcuM_KillAllRUNRequests`: Use this function with care. Side effects may occur in the application. If an implementation contains synchronization for more graceful shutdown a timeout must be provided to ensure that the shutdown process is initiated.

Error Codes of `EcuM_KillAllRUNRequests`:
`ECUM_E_ALL_RUN_REQUESTS_KILLED`: On each invocation.

8.4.3.6 EcuM_KillAllPostRUNRequests

[SWS_EcuMf_00101] ⌈

Service name:	EcuM_KillAllPostRUNRequests

Syntax:	`void EcuM_KillAllPostRUNRequests(`
	`    void`
	`)`

Service ID[hex]:	0x2a
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This function unconditionally releases all pending requests to PostRUN.

⌋ ()

[SWS_EcuMf_00102] ⌈ The function `EcuM_KillAllPostRUNRequests` unconditionally releases all pending requests to PostRUN. ⌋ ()

[SWS_EcuMf_00103] ⌈ As a consequence EcuM_RequestRUN must not accept any new requests unless the resulting shutdown has been completed. ⌋ ()

8.4.3.7 EcuM_SelectShutdownTarget

[SWS_EcuM_02822] ⌈

Service name:	EcuM_SelectShutdownTarget	

Syntax:	`Std_ReturnType EcuM_SelectShutdownTarget(`
	`    EcuM_StateType target,`
	`    uint8 mode`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	target	The selected shutdown target.
	mode	The identfier of a sleep mode (if target is ECUM_STATE_SLEEP) or a reset mechanism (if target is ECUM_STATE_RESET) as defined by configuration.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The new shutdown target was set E_NOT_OK: The new shutdown target was not set
Description:	EcuM_SelectShutdownTarget selects the shutdown target. EcuM_SelectShutdownTarget is part of the ECU Manager Module port interface.	

⌋ ()

[SWS_EcuM_00624] ⌈Parameter mode of the function EcuM_SelectShutdownTarget: The selected shutdown target. Only the following subset of the EcuM_StateType value range is accepted:

- ECUM_STATE_SLEEP
- ECUM_STATE_RESET
- ECUM_STATE_OFF

All other values will be rejected. ⌋ ()

[SWS_EcuM_02185] ⌈The parameter mode of the function `EcuM_SelectShutdownTarget` shall be the identifier of a sleep mode. The mode parameter shall only be used if the target parameter equals `ECUM_STATE_SLEEP`. In all other cases, it shall be ignored. Only sleep modes that are defined at configuration time and are stored in the EcuMSleepMode container are allowed as parameters. ⌋ ()

[SWS_EcuM_02585] ⌈An implementation of this service should not initiate any setup activities but only store the value for later use in the `SHUTDOWN` state. ⌋ ()

[SWS_EcuM_00228] ⌈The TTII-algorithm shall set the TTII divisor counter variable with the preload value defined in ECUM_TTII_DIVISOR_LIST.
The service is intended for implementing AUTOSAR ports. ⌋ ()

Caveat of EcuM_SelectShutdownTarget: The ECU State Manager Fixed module does not define any mechanism to resolve issues arising from requests from different sources. Always the last set values will be used as shutdown target. It is assumed that there will be one piece of application which is specific to the ECU and handles these kinds of issues.

8.4.3.8 EcuM_GetShutdownTarget

[SWS_EcuM_02824] ⌈

Service name:	EcuM_GetShutdownTarget	

Syntax:	`Std_ReturnType EcuM_GetShutdownTarget(`
	`    EcuM_StateType* shutdownTarget,`
	`    uint8* sleepMode`
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	shutdownTarget	One of these values is returned: * ECUM_STATE_SLEEP * ECUM_STATE_RESET * ECUM_STATE_OFF
	sleepMode	If the out parameter "shutdownTarget" is ECUM_STATE_SLEEP, sleepMode tells which of the configured sleep modes was actually chosen. If "shutdownTarget" is ECUM_STATE_RESET, sleepMode tells which of the configured reset modes was actually

		chosen.
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service has failed, e.g. due to NULL pointer being passed
Description:	EcuM_GetShutdownTarget returns the currently selected shutdown target as set by EcuM_SelectShutdownTarget. EcuM_GetShutdownTarget is part of the ECU Manager Module port interface.	

⌋ ()

[SWS_EcuM_02788] ⌈Parameter `sleepMode` and `shutdownTarget` of the function `EcuM_GetShutdownTarget`: An implementation shall cope with NULL pointers by simply ignoring the out parameter in all cases. ⌋ ()

8.4.3.9 EcuM_GetLastShutdownTarget

[SWS_EcuM_02825] ⌈

Service name:	EcuM_GetLastShutdownTarget	

Syntax:	`Std_ReturnType EcuM_GetLastShutdownTarget(`
	`    EcuM_StateType* shutdownTarget,`
	`    uint8* sleepMode`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	shutdownTarget	One of these values is returned: * ECUM_STATE_SLEEP * ECUM_STATE_RESET * ECUM_STATE_OFF
	sleepMode	If the out parameter "shutdownTarget" is ECUM_STATE_SLEEP, sleepMode tells which of the configured sleep modes was actually chosen. If "shutdownTarget" is ECUM_STATE_RESET, sleepMode tells which of the configured reset modes was actually chosen.
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service has failed, e.g. due to NULL pointer being passed
Description:	EcuM_GetLastShutdownTarget returns the shutdown target of the previous shutdown process. EcuM_GetLastShutdownTarget is part of the ECU Manager Module port interface.	

⌋ ()

[SWS_EcuM_02336] ⌈Parameter `sleepMode` of the function `EcuM_GetLastShutdownTarget`: If the return parameter is `ECUM_STATE_SLEEP`, this out parameter tells which of the configured sleep modes was actually chosen. ⌋ ()

[SWS_EcuM_02337] ⌈Parameters `sleepMode` and `shutdownTarget` of the function `EcuM_GetLastShutdownTarget`: An implementation shall cope with NULL pointers by simply ignoring the out parameter in all cases. ⌋ ()

[SWS_EcuM_02156] ⌈The return value describes the ECU state from which the last wake up or power up occurred. This function shall return always the same value until the next shutdown. ⌋ ()

[SWS_EcuM_02157] ⌈This function is intended for primary use in `STARTUP` or `RUN` state. Reasonable use cases exist there. To simplify implementation, it is acceptable if the value is set in late shutdown phase for use during the next startup. If so, implementation specific limitations must be clearly documented. ⌋ ()

8.4.3.10 EcuM_GetState

[SWS_EcuM_00823] ⌈

Service name:	EcuM_GetState	

Syntax:	`Std_ReturnType EcuM_GetState(`
	`    EcuM_StateType* state`
	`)`	

Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	state	The value of the internal state variable.
Return value:	Std_ReturnType	E_OK: The out parameter was set successfully. E_NOT_OK: The out parameter was not set.
Description:	Gets a state.	

⌋ ()

[SWS_EcuM_00423] ⌈The service must be accessible from an OS and an OS-free context as well as from an interrupt context. ⌋ ()

8.4.4 Wake up Handling

8.4.4.1 EcuM_GetPendingWakeupEvents

[SWS_EcuM_02827] ⌈

Service name:	EcuM_GetPendingWakeupEvents	

Syntax:	`EcuM_WakeupSourceType EcuM_GetPendingWakeupEvents(`
	`    void`
	`)`	

Service ID[hex]:	0x0d	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant, Non-Interruptible	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupSourceType	All wakeup events
Description:	Gets pending wakeup events.	

⌋ (SRS_ModeMgm_09126)

[SWS_EcuM_01156] ⌈Return code of the function `EcuM_GetPendingWakeupEvents`: Returns wake up events which have been set but not yet validated. ⌋ ()

[SWS_EcuM_02172] ⌈The service `EcuM_GetPendingWakeupEvents` must be callable from interrupt context, from OS context and an OS-free context. ⌋ ()

Caveat of `EcuM_GetPendingWakeupEvents`: The wake up events returned by this service are only pending

8.4.4.2 EcuM_ClearWakeupEvent

[SWS_EcuM_02828] ⌈

Service name:	EcuM_ClearWakeupEvent	

Syntax:	`void EcuM_ClearWakeupEvent(`
	`    EcuM_WakeupSourceType sources`
	`)`	

Service ID[hex]:	0x16	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant, Non-Interruptible	
Parameters (in):	sources	Events to be cleared
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Clears wakeup events.	

⌋ (SRS_ModeMgm_09126)

[SWS_EcuM_02683] ⌈`EcuM_ClearWakeupEvent` shall clear all wake up events like pending, validated and expired events. ⌋ ()

[SWS_EcuM_02807] ⌈The function must be callable from interrupt context, from OS context and an OS-free context. ⌋ ()

8.4.4.3 EcuM_GetValidatedWakeupEvents

[SWS_EcuM_02830] ⌈

Service name:	EcuM_GetValidatedWakeupEvents	

Syntax:	`EcuM_WakeupSourceType EcuM_GetValidatedWakeupEvents(`
	`    void`
	`)`	

Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant, Non-Interruptible	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupSourceType	All wakeup events
Description:	Gets validated wakeup events.	

⌋ (SRS_ModeMgm_09126)

[SWS_EcuM_02533] ⌈Return code of `EcuM_GetValidatedWakeupEvents`: Returns the value from the internal variable. ⌋ ()

[SWS_EcuM_02532] ⌈The service must be callable from interrupt context, from OS context and an OS-free context. ⌋ ()

8.4.4.4 EcuM_GetExpiredWakeupEvents

[SWS_EcuM_02831] ⌈

Service name:	EcuM_GetExpiredWakeupEvents	

Syntax:	`EcuM_WakeupSourceType EcuM_GetExpiredWakeupEvents(`
	`    void`
	`)`	

Service ID[hex]:	0x19	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant, Non-Interruptible	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupSourceType	All wakeup events: Returns all events that have been set and for which validation has failed. Events which do not need validation must never be reported by this function.
Description:	Gets expired wakeup events.	

⌋ (SRS_ModeMgm_09126)

[SWS_EcuM_02589]⌈ The service `EcuM_GetExpiredWakeupEvents` must be callable from interrupt context, from OS context and an OS-free context. ⌋ ()

8.4.4.5 EcuM_GetStatusOfWakeupSource

[SWS_EcuM_00832] ⌈

Service name:	EcuM_GetStatusOfWakeupSource	

Syntax:	`EcuM_WakeupStatusType EcuM_GetStatusOfWakeupSource(`
	`    EcuM_WakeupSourceType sources`
	`)`	

Service ID[hex]:	0x17	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	sources	The sources for which the status is returned
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupStatusType	Sum status of all wakeup sources passed in the in parameter.
Description:	The sum status shall be computed according to the following algorithm: If (EcuM_GetValidatedWakeupEvents() AND sources) is not 0 then return ECUM_WKSTATUS_VALIDATED. If (EcuM_GetPendingWakeupEvents() AND sources) is not 0 then return ECUM_WKSTATUS_PENDING. If (EcuM_GetExpiredWakeupEvents() AND sources) is not 0 then return ECUM_WKSTATUS_EXPIRED. Otherwise, return ECUM_WKSTATUS_NONE.	

⌋ ()

[SWS_EcuM_00754] ⌈When the `EcuM_GetStatusOfWakeupSource` service is called and parameter “sources” equals 0, then this service shall return `ECUM_WKSTATUS_NONE`. If parameter “sources” equals `ECUM_WKSOURCE_ALL_SOURCES`, then this service shall return the sum status of all configured wake up sources. ⌋ ()

8.4.5 Miscellaneous

8.4.5.1 EcuM_SelectBootTarget

[SWS_EcuM_02835] ⌈

Service name:	EcuM_SelectBootTarget	

Syntax:	`Std_ReturnType EcuM_SelectBootTarget(`
	`    EcuM_BootTargetType target`
	`)`	

Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	target	The selected boot target.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The new boot target was accepted by EcuM E_NOT_OK: The new boot target was not accepted by EcuM
Description:	EcuM_SelectBootTarget selects a boot target. EcuM_SelectBootTarget is part of the ECU Manager Module port interface.	

⌋ ()

[SWS_EcuM_02247] ⌈The service must store the selected target in a way which is compatible with the boot loader. This may mean format AND location. The service is intended for implementing AUTOSAR ports. ⌋ ()

Caveat of the function `EcuM_SelectBootTarget`: This service may be dependent on the available hardware and the boot loader used.

The implementation of this service will not be prescribed by AUTOSAR. The implementer of the service `EcuM_SelectBootTarget` has to ensure to place the boot target information at a safe location, which then shall be evaluated by the boot manager after a reset.

This service is only intended for use by SW-C’s related to diagnostics (boot management).

Note: In the definition of *8.2.4.1* Data Types a default boot target is defined. So even in case of E_NOT_OK, a valid boot targed is defined.

8.4.5.2 EcuM_GetBootTarget

[SWS_EcuM_02836] ⌈

Service name:	EcuM_GetBootTarget	

Syntax:	`Std_ReturnType EcuM_GetBootTarget(`
	`    EcuM_BootTargetType * target`
	`)`	

Service ID[hex]:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	target	The currently selected boot target.
Return value:	Std_ReturnType	E_OK: The service always succeeds.
Description:	EcuM_GetBootTarget returns the current boot target - see EcuM_SelectBootTarget. EcuM_GetBootTarget is part of the ECU Manager Module port interface.	

⌋ ()

Since the information of the boot target shall also be evaluated by the boot loader, the service EcuM_GetBootTarget must be available without the context of the RTE, the OS, or even the C language! If this is not implementable, the implementer has to offer and document another API which then is available for the boot loader.

[SWS_EcuMf_00035] ⌈Parameter `target` of the function `EcuM_GetBootTarget`: An implementation shall cope with NULL pointers by simply ignoring the out parameter in all cases. ⌋ ()

Note: In the definition of *8.2.4.1* Data Types a default boot target is defined. So even if EcuM_SelectBootTarget was not called beforehand, a valid boot targed is defined.

8.5 Scheduled Functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1 EcuM_MainFunction

[SWS_EcuM_02837] ⌈

Service name:	EcuM_MainFunction

Syntax:	`void EcuM_MainFunction(`
	`    void`
	`)`

Service ID[hex]:	0x18
Description:	The purpose of this service is to implement all activities of the ECU State Manager while the OS is up and running.

⌋ (SRS_BSW_00425,SRS_BSW_00373)

[SWS_EcuM_00594] ⌈This service must be called on a periodic basis from an adequate BSW task (i.e. a task under control of the BSW scheduler).
To determine the period, the system designer should consider the following timings:

- The period directly results in a possible latency for testing RUN requests. The largest acceptable reaction time will therefore limit the maximum period for invocation.
- The service will also carry out the wake up validation protocol (see *7.8* Wake-up Validation Protocol). The smallest validation timeout typically should limit the period.
- As a rule of thumb, the period of this service should be in the order of half as long as the shortest time constant mentioned in the topics above.

⌋ ()

[SWS_EcuM_00656] ⌈The service shall not be called from tasks which may invoke runnable entities. ⌋ ()

[SWS_EcuMf_00029] ⌈ If the `EcuM_MainFunction` is called without having called `EcuM_Init` in advance (so the EcuM is un-initialized) the `EcuM_MainFunction` shall return immediately without performing any functionality and without raising any errors. ⌋ ()

8.6 Callback Definitions

8.6.1 Callbacks from NVRAM Manager

8.6.1.1 EcuM_CB_NfyNvMJobEnd

[SWS_EcuM_00839] ⌈

Service name:	EcuM_CB_NfyNvMJobEnd	

Syntax:	`void EcuM_CB_NfyNvMJobEnd(`
	`    uint8 ServiceId,`
	`    NvM_RequestResultType JobResult`
	`)`	

Service ID[hex]:	0x65	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ServiceId	Unique Service ID of NVRAM manager service.
	JobResult	Covers the job result of the previous processed multi block job.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Used to notify about the end of NVRAM jobs initiated by EcuM The callback must be callable from normal and interrupt execution contexts.	

⌋ ()

Configuration of `EcuM_CB_NfyNvMJobEnd`: NVRAM manager must be configured to call this callback as a multiple block job end notification. See [12] for details.

8.6.2 Callbacks from Wake up Sources

8.6.2.1 EcuM_CheckWakeup

See *8.7.6.28.7.6.2* EcuM_CheckWakeupEcuM_CheckWakeup for a description of the service.

This service is a Callout of the ECU State Manager Fixed module as well as a Callback that wake up sources invoke when they process wake up interrupts.

8.6.2.2 EcuM_SetWakeupEvent

[SWS_EcuM_00826] ⌈

Service name:	EcuM_SetWakeupEvent	

Syntax:	`void EcuM_SetWakeupEvent(`
	`    EcuM_WakeupSourceType sources`
	`)`	

Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant, Non-Interruptible	
Parameters (in):	sources	Value to be set
Parameters	None	

(inout):	
Parameters (out):	None
Return value:	None
Description:	Sets the wakeup event.

⌋ ()

[SWS_EcuM_01117] ⌈Takes the value and stores it in an internal variable (OR-operation). ⌋ ()

[SWS_EcuM_02707] ⌈The service must start the wake up validation timeout timer according to chapter *7.8.4* Wake up validation timeout. ⌋ ()

[SWS_EcuM_02171] ⌈The function must be callable from interrupt context, from OS context and an OS-free context. ⌋ ()

[SWS_EcuMf_00038] ⌈If EcuM_SetWakeupEvent is called for the corresponding wakeup source the corresponding CheckWakeupTimer is cancelled.
⌋ ()

[SWS_EcuMf_00039] ⌈If the corresponding CheckWakeupTimer expires before EcuM_EndCheckWakeup or EcuM_SetWakeupEvent is called, the check of the this wakeup source is finished.
⌋ ()

[SWS_EcuMf_00120] ⌈ EcuM_SetWakeupEvent shall ignore all events passed in the sources parameter that are not associated to the selected sleep mode. ⌋ ()

8.6.2.3 EcuM_ValidateWakeupEvent

[SWS_EcuM_02829] ⌈

Service name:	EcuM_ValidateWakeupEvent	

Syntax:	`void EcuM_ValidateWakeupEvent(`
	`    EcuM_WakeupSourceType sources`
	`)`	

Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	sources	Events that have been validated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	After wakeup, the ECU State Manager will stop the process during the WAKEUP VALIDATION state/sequence to wait for validation of the wakeup event.This API service is used to indicate to the ECU Manager module that the wakeup events indicated in the sources parameter have been validated.	

⌋ (SRS_BSW_00359,SRS_BSW_00360,SRS_BSW_00440)

[SWS_EcuM_00344] ⌈The validation shall be valid when ANDing the parameter `events` with the internal variable of pending wake up events results in a value other than null. ⌋ ()

[SWS_EcuM_02645] ⌈The service shall invoke `ComM_EcuM_WakeUpIndication` of the Communication Manager for each wake up event if the EcuMComMChannelRef parameter in the EcuMWakeupSource configuration container for the corresponding wake up source is configured. ⌋ ()

[SWS_EcuM_02345] ⌈The function must be callable from interrupt context, from OS context, and an OS-free context. ⌋ ()

[SWS_EcuM_02790] ⌈The service shall return without effect for all sources except communication channels when called while ECU State Manager Fixed module is NOT in one of the states: SHUTDOWN, SLEEP, WAKEUP I, WAKEUP VALIDATION, and STARTUP. ⌋ ()

[SWS_EcuM_02791] ⌈The service shall have full effect in any state other than ECUM_STATE_APP_RUN for those sources which correspond to a communication channel (see **SWS_EcuM_02645**). ⌋ ()

8.7 Callout Definitions

Callouts are pieces of code that have to be added to the ECU State Manager Fixed module during ECU integration. The content of most callouts is hand-written code, for some callouts the ECU State Manager Fixed module configuration tool shall generate a default implementation that is manually edited by the integrator. Conceptually, these callouts belong to the ECU State Manager Fixed module .

Since callouts are no services of the ECU State Manager Fixed module they do not have an assigned Service ID.

8.7.1 Generic Callouts

8.7.1.1 EcuM_ErrorHook

[SWS_EcuM_02904] ⌈

Service name:	EcuM_ErrorHook	

Syntax:	`void EcuM_ErrorHook(`
	`    uint16 reason`
	`)`	

Service ID[hex]:	0x30	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	reason	Reason for calling the error hook
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The ECU State Manager will call the error hook if the error codes "ECUM_E_RAM_CHECK_FAILED" or "ECUM_E_CONFIGURATION_DATA_INCONSISTENT" occur. In this situation it is not possible to continue processing and the ECU must be stopped. The integrator may choose the modality how the ECU is stopped, i.e. reset, halt, restart, safe state etc.	

⌋ ()

Invocation of `EcuM_ErrorHook`: in all states

Class of `EcuM_ErrorHook`: Mandatory

EcuM_ErrorHook is integration code and the integrator is free to define additional individual error codes to be passed as the `reason` parameter. These error codes shall not conflict with the standard error codes i.e. E_OK, E_NOT_OK, etc.

8.7.2 Callouts from STARTUP

8.7.2.1 EcuM_AL_DriverInitZero

[SWS_EcuM_02905] ⌈

Service name:	EcuM_AL_DriverInitZero

Syntax:	`void EcuM_AL_DriverInitZero(`
	`    void`
	`)`

Service ID[hex]:	0x31
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callout shall provide driver initialization and other hardware-related startup activities for loading the post-build configuration data. Beware: Here only pre-compile and link-time configurable modules may be used.

⌋ ()
Invocation of `EcuM_AL_DriverInitZero`: Early in STARTUP I

The ECU State Manager Fixed module configuration tool shall generate a default implementation of the EcuM_AL_DriverInitZero callout from the sequence of modules defined in the EcuMDriverInitListZero configuration container. See SWS_EcuM_02559 and SWS_EcuM_02730.

8.7.2.2 EcuM_DeterminePbConfiguration

[SWS_EcuM_02906] ⌈

Service name:	EcuM_DeterminePbConfiguration	

Syntax:	`const EcuM_ConfigType* EcuM_DeterminePbConfiguration(`
	`    void`
	`)`	

Service ID[hex]:	0x32	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	const EcuM_ConfigType*	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
Description:	This callout should evaluate some condition, like port pin or NVRAM value, to determine which post-build configuration shall be used in the remainder of the startup process. It shall load this configuration data into a piece of memory that is accessible by all BSW modules and shall return a pointer to the EcuM post-build configuration as a base for all BSW module post-build configrations.	

⌋ ()
Invocation of `EcuM_DeterminePbConfiguration`: Early in STARTUP I

Content is manually written.

8.7.2.3 EcuM_AL_DriverInitOne

[SWS_EcuM_02907] ⌈

Service name:	EcuM_AL_DriverInitOne

Syntax:	`void EcuM_AL_DriverInitOne(`
	`    void`
	`)`

Service ID[hex]:	0x33
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callout shall provide driver initialization and other hardware-related startup activities in case of a power on reset.

⌋ ()
Invocation of `EcuM_AL_DriverInitOne`: In STARTUP I

The ECU State Manager Fixed module configuration tool shall generate a default implementation of the `EcuM_AL_DriverInitOne` callout from the sequence of modules defined in the `EcuMDriverInitListOne` configuration container. See SWS_EcuM_02559 and SWS_EcuM_02730.

Besides driver initialization, the following initialization sequences should be considered in this block: MCU initialization according to AUTOSAR_SWS_Mcu_Driver chapter 9.1.

8.7.2.4 EcuM_AL_DriverInitTwo

[SWS_EcuM_00908] ⌈

Service name:	EcuM_AL_DriverInitTwo	

Syntax:	`void EcuM_AL_DriverInitTwo(`
	`    const EcuM_ConfigType* ConfigPtr`
	`)`	

Service ID[hex]:	0x34	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout shall provide driver initialization of drivers which need OS and do not need to wait for the NvM_ReadAll job to finish.	

⌋ ()
Invocation of `EcuM_AL_DriverInitTwo`: In STARTUP II

The ECU State Manager Fixed module configuration tool shall generate a default implementation of the `EcuM_AL_DriverInitTwo` callout from the sequence of modules defined in the `EcuMDriverInitListTwo` configuration container. See SWS_EcuM_02559 and SWS_EcuM_02730.

8.7.2.5 EcuM_AL_DriverInitThree

[SWS_EcuM_00909] ⌈

Service name:	EcuM_AL_DriverInitThree	

Syntax:	`void EcuM_AL_DriverInitThree(`
	`    const EcuM_ConfigType* ConfigPtr`
	`)`	

Service ID[hex]:	0x35	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout shall provide driver initialization of drivers which need OS and need to wait for the NvM_ReadAll job to finish.	

⌋ ()

Invocation of `EcuM_AL_DriverInitThree`: In STARTUP II

The ECU State Manager Fixed module configuration tool shall generate a default implementation of the `EcuM_AL_DriverInitThree` callout from the sequence of modules defined in the `EcuMDriverInitListThree` configuration container. See SWS_EcuM_02559 and SWS_EcuM_02730.

8.7.2.6 EcuM_OnRTEStartup

[SWS_EcuM_00910] ⌈

Service name:	EcuM_OnRTEStartup

Syntax:	`void EcuM_OnRTEStartup(`
	`    void`
	`)`

Service ID[hex]:	0x36
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	--

⌋ ()

Invocation of `EcuM_OnRTEStartup`: Just before calling RTE_Start

8.7.3 Callouts from RUN State

8.7.3.1 EcuM_OnEnterRun

[SWS_EcuM_00911] ⌈

Service name:	EcuM_OnEnterRun

Syntax:	`void EcuM_OnEnterRun(`
	`    void`
	`)`

Service ID[hex]:	0x37
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	On entry of RUN state is very similar to "just after startup". This call allows the system designer to notify that RUN state has been reached.

⌋ ()

Invocation of `EcuM_OnEnterRun`: On entry of RUN state.

8.7.3.2 EcuM_OnExitRun

[SWS_EcuM_00912] ⌈

Service name:	EcuM_OnExitRun

Syntax:	`void EcuM_OnExitRun(`
	`    void`
	`)`

Service ID[hex]:	0x38
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the APP RUN state is about to be left.

⌋ ()

Invocation of `EcuM_OnExitRun`: By ECU State Manager Module upon detection that the last run request has been released.

8.7.3.3 EcuM_OnExitPostRun

[SWS_EcuM_00913] ⌈

Service name:	EcuM_OnExitPostRun

Syntax:	`void EcuM_OnExitPostRun(`
	`    void`

	)
Service ID[hex]:	0x39
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the APP POST RUN state is about to be left.

⌋ ()

Invocation of `EcuM_OnExitPostRun`: ECU State Manager Module upon detection that the last POST_RUN request has been released.

8.7.4 Callouts from SHUTDOWN

8.7.4.1 EcuM_OnPrepShutdown

[SWS_EcuM_00914] ⌈

Service name:	EcuM_OnPrepShutdown

Syntax:	`void EcuM_OnPrepShutdown(`
	`    void`
	`)`

Service ID[hex]:	0x3A
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the PREP SHUTDOWN state is about to be entered.

⌋ ()

Invocation of `EcuM_OnPrepShutdown`: On entry of PREP SHUTDOWN

8.7.4.2 EcuM_OnGoSleep

[SWS_EcuM_00915] ⌈

Service name:	EcuM_OnGoSleep

Syntax:	`void EcuM_OnGoSleep(`
	`    void`
	`)`

Service ID[hex]:	0x3B
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the GO SLEEP state is about to be entered.

⌋ ()

Invocation of `EcuM_OnGoSleep`: On entry of GO SLEEP

8.7.4.3 EcuM_OnGoOffOne

[SWS_EcuM_02916] ⌈

Service name:	EcuM_OnGoOffOne

Syntax:	`void EcuM_OnGoOffOne(`
	`    void`
	`)`

Service ID[hex]:	0x3C
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the GO OFF I state is about to be entered.

⌋ ()

Invocation of `EcuM_OnGoOffOne`: On entry of GO OFF I

8.7.4.4 EcuM_OnGoOffTwo

[SWS_EcuM_02917] ⌈

Service name:	EcuM_OnGoOffTwo

Syntax:	`void EcuM_OnGoOffTwo(`
	`    void`
	`)`

Service ID[hex]:	0x3D
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the GO OFF II state is about to be entered.

⌋ ()

Invocation of `EcuM_OnGoOffTwo`: On entry of GO OFF II

8.7.4.5 EcuM_EnableWakeupSources

[SWS_EcuM_02918] ⌈

Service name:	EcuM_EnableWakeupSources	

Syntax:	`void EcuM_EnableWakeupSources(`
	`    EcuM_WakeupSourceType wakeupSource`
	`)`	

Service ID[hex]:	0x3F	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The ECU Manager Module calls EcuM_EnableWakeupSource to allow the system designer to notify wakeup sources defined in the wakeupSource bitfield that SLEEP will be entered and to adjust their source accordingly.	

⌋ ()

[SWS_EcuM_02546] ⌈The ECU State Manager Fixed module needs to derive the wake up sources to be enabled for the from configuration information. ⌋ ()

Invocation of `EcuM_EnableWakeupSources`: From GOSLEEP II

8.7.4.6 EcuM_GenerateRamHash

[SWS_EcuM_02919] ⌈

Service name:	EcuM_GenerateRamHash

Syntax:	`void EcuM_GenerateRamHash(`
	`    void`
	`)`

Service ID[hex]:	0x40
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	see EcuM_CheckRamHash

⌋ ()

8.7.4.7 EcuM_AL_SwitchOff

[SWS_EcuM_02920] ⌈

Service name:	EcuM_AL_SwitchOff

Syntax:	`void EcuM_AL_SwitchOff(`
	`    void`
	`)`

Service ID[hex]:	0x3E
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters	None

(inout):	
Parameters (out):	None
Return value:	None
Description:	This callout shall take the code for shutting off the power supply of the ECU. If the ECU cannot unpower itself, a reset may be an adequate reaction.

⌋ ()

Invocation of `EcuM_AL_SwitchOff`: Last activity in SHUTDOWN II

Note: In some cases of HW/SW concurrency, it may happen that during the power down in `EcuM_AL_SwitchOff` (endless loop) some hardware (e.g. a CAN transceiver) switches on the ECU again. In this case the ECU may be in a deadlock until the hardware watchdog resets the ECU. To reduce the time until the hardware watchdog fixes this deadlock, the integrator code in `EcuM_AL_SwitchOff` as last action can limit the endless loop and after a sufficient long time reset the ECU using `Mcu_PerformReset().`

8.7.5 Callouts from WAKEUP

8.7.5.1 EcuM_CheckRamHash

[SWS_EcuM_02921] ⌈

Service name:	EcuM_CheckRamHash	

Syntax:	`uint8 EcuM_CheckRamHash(`
	`    void`
	`)`	

Service ID[hex]:	0x43	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	uint8	0: RAM integrity test failed else: RAM integrity test passed
Description:	This callout is intended to provide a RAM integrity test. The goal of this test is to ensure that after a long SLEEP duration, RAM contents is still consistent. The check does not need to be exhaustive since this would consume quite some processing time during wakeups. A well designed check will execute quickly and detect RAM integrity defects with a sufficient probability. This specification does not make any assumption about the algorithm chosen for a particular ECU. The areas of RAM which will be checked have to be chosen carefully. It depends on the check algorithm itself and the task structure. Stack contents of the task executing the RAM check e.g. very likely cannot be checked. It is good practice to have the hash generation and checking in the same task and that this task is not preemptible and that there is only little activity between hash generation and hash check. The RAM check itself is provided by the system designer. In case of applied multi core and existence of Satellite-EcuM(s): this API will be called by the Master-EcuM only.	

⌋ ()

8.7.5.2 EcuM_DisableWakeupSources

[SWS_EcuM_02922] ⌈

Service name:	EcuM_DisableWakeupSources	

Syntax:	`void EcuM_DisableWakeupSources(`
	`    EcuM_WakeupSourceType wakeupSource`
	`)`	

Service ID[hex]:	0x44	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The ECU Manager Module calls EcuM_DisableWakeupSources to set the wakeup source(s) defined in the wakeupSource bitfield so that they are not able to wake the ECU up.	

⌋ ()
Invocation of `EcuM_DisableWakeupSources`: In WAKEUP I

8.7.5.3 EcuM_AL_DriverRestart

[SWS_EcuM_02923] ⌈

Service name:	EcuM_AL_DriverRestart

Syntax:	`void EcuM_AL_DriverRestart(`
	`    void`
	`)`

Service ID[hex]:	0x45
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callout shall provide driver initialization and other hardware-related startup activities in the wakeup case.

⌋ ()
Invocation of `EcuM_AL_DriverRestart`: In WAKEUP I

The ECU State Manager Fixed module configuration tool shall generate a default implementation of the `EcuM_AL_DriverRestart` callout from the sequence of modules defined in the `EcuMDriverRestartList` configuration container. See SWS_EcuM_02561, SWS_EcuM_02559 and SWS_EcuM_02730.

8.7.5.4 EcuM_StartWakeupSources

[SWS_EcuM_02924] ⌈

Service name:	EcuM_StartWakeupSources

Syntax:	`void EcuM_StartWakeupSources(`
	`    EcuM_WakeupSourceType wakeupSource`
	`)`	

Service ID[hex]:	0x46	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The callout shall start the given wakeup source(s) so that they are ready to perform wakeup validation.	

⌋ ()

Invocation of `EcuM_StartWakeupSources`: In WAKEUP VALIDATION

8.7.5.5 EcuM_CheckValidation

[SWS_EcuM_02925] ⌈

Service name:	EcuM_CheckValidation	

Syntax:	`void EcuM_CheckValidation(`
	`    EcuM_WakeupSourceType wakeupSource`
	`)`	

Service ID[hex]:	0x47	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout is called by the EcuM to validate a wakeup source. If a valid wakeup has been detected, it shall be reported to EcuM via EcuM_ValidateWakeupEvent().	

⌋ ()

Invocation of `EcuM_CheckValidation`: In WAKEUP VALIDATION

8.7.5.6 EcuM_StopWakeupSources

[SWS_EcuM_02926] ⌈

Service name:	EcuM_StopWakeupSources	

Syntax:	`void EcuM_StopWakeupSources(`
	`    EcuM_WakeupSourceType wakeupSource`
	`)`	

Service ID[hex]:	0x48	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	

Return value:	None
Description:	The callout shall stop the given wakeup source(s) after unsuccessful wakeup validation.

⌋ ()

Invocation of `EcuM_StopWakeupSources`: In WAKEUP VALIDATION

8.7.5.7 EcuM_OnWakeupReaction

[SWS_EcuM_00927] ⌈

Service name:	EcuM_OnWakeupReaction	

Syntax:	`EcuM_WakeupReactionType EcuM_OnWakeupReaction(`
	`    EcuM_WakeupReactionType wact`
	`)`	

Service ID[hex]:	0x49	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wact	The wakeup reaction computed by ECU State Manager
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupReactionType	All values: The desired wakeup reaction.
Description:	This callout gives the system designer the chance to intercept the automatic boot behavior and to override the wakeup reaction computed from wakeup source.	

⌋ ()

Invocation of `EcuM_OnWakeupReaction`: In WAKEUP REACTION after default computation of wake up reaction.

8.7.5.8 EcuM_ StartCheckWakeup

[SWS_EcuM_04096] ⌈

Service name:	EcuM_StartCheckWakeup	

Syntax:	`void EcuM_StartCheckWakeup(`
	`    EcuM_WakeupSourceType WakeupSource`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	WakeupSource	For this wakeup source the corresponding CheckWakeupTimer shall be started.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This API is called by the ECU Firmware to start the CheckWakeupTimer for the corresponding WakeupSource. If EcuMCheckWakeupTimeout > 0 the CheckWakeupTimer for the WakeupSource is started. If EcuMCheckWakeupTimeout ≤ 0 the API call is ignored by the EcuM.	

⌋ ()

[SWS_EcuMf_00042] ⌈The function EcuM_ StartCheckWakeup must be callable from interrupt context, and from OS context.⌋ ()

8.7.5.9 EcuM_ EndCheckWakeup

[SWS_EcuM_02927] ⌈

Service name:	EcuM_EndCheckWakeup	

Syntax:	`void EcuM_EndCheckWakeup(`
	`    EcuM_WakeupSourceType WakeupSource`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	WakeupSource	For this wakeup source the corresponding CheckWakeupTimer shall be canceled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This API is called by any SW Module whose wakeup source is checked asynchronously (e.g. asynchronous Can Trcv Driver) and the Check of the Wakeup returns a negative Result (no Wakeup by this Source). The API cancels the CheckWakeupTimer for the WakeupSource. If the correponding CheckWakeupTimer is canceled the check of this wakeup source is finished.	

⌋ ()

[SWS_EcuMf_00043] ⌈The function EcuM_ EndCheckWakeup must be callable from interrupt context, and from OS context.⌋ ()

8.7.6 Callouts from SLEEP State

8.7.6.1 EcuM_SleepActivity

[SWS_EcuM_02928] ⌈

Service name:	EcuM_SleepActivity

Syntax:	`void EcuM_SleepActivity(`
	`    void`
	`)`

Service ID[hex]:	0x41
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callout is invoked periodically in all reduced clock sleep modes. It is explicitely allowed to poll wakeup sources from this callout and to call wakeup notification functions to indicate the end of the sleep state to the ECU State Manager.

⌋ ()

Invocation of `EcuM_SleepActivity`: Periodically in SLEEP state if the MCU is not halted (i.e. clock is reduced)

Note: If called from the poll sequence the EcuM calls this callout functions in a blocking loop at maximum frequency. The callout implementation must ensure by other means if callout code shall be executed with a lower period. The integrator may choose any method to control this, e.g. with the help of OS counters, OS alarms,
or Gpt timers.

8.7.6.2 EcuM_CheckWakeup

[SWS_EcuM_02929] ⌈

Service name:	EcuM_CheckWakeup	

Syntax:	`void EcuM_CheckWakeup(`
	`    EcuM_WakeupSourceType wakeupSource`
	`)`	

Service ID[hex]:	0x42	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout is called by the EcuM to poll a wakeup source. It shall also be called by the ISR of a wakeup source to set up the PLL and check other wakeup sources that may be connected to the same interrupt.	

⌋ ()

Invocation of `EcuM_CheckWakeup`: Periodically in SLEEP state if the MCU is not halted, or when handling a wake up interrupt

Note: If called from the poll sequence the EcuM calls this callout functions in a blocking loop at maximum frequency. The callout implementation must ensure by other means if callout code shall be executed with a lower period. The integrator may choose any method to control this, e.g. with the help of OS counters, OS alarms,
or Gpt timers.

8.8 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.8.1 Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module.

[SWS_EcuM_02858] ⌈

API function	***Description***
BswM_Deinit	Deinitializes the BSW Mode Manager.
BswM_EcuM_CurrentState	Function called by EcuM to indicate the current ECU Operation Mode.
BswM_EcuM_CurrentWakeup	Function called by EcuM to indicate the current state of a wakeup source.
BswM_Init	Initializes the BSW Mode Manager.
CanSM_StartWakeupSource	This function shall be called by EcuM when a wakeup source shall be started.
CanSM_StopWakeupSource	This function shall be called by EcuM when a wakeup source shall be stopped.
ComM_CommunicationAllowed	EcuM or BswM shall indicate to ComM when communication is allowed. If EcuM/Fixed is used: EcuM/Fixed. If EcuM/Flex is used: BswM
ComM_DeInit	This API de-initializes the AUTOSAR Communication Manager.
ComM_EcuM_WakeUpIndication	Notification of a wake up on the corresponding channel.
ComM_GetState	Return current state, including sub-state, of the ComM channel state machine. Usage of function only valid if EcuM/Fixed is used: To leave RUN: state/sub-state need to be COMM_NO_COM_NO_PENDING_REQUEST (No communication and no pending request to start communication) In POST RUN to return to RUN: state/sub-state need to be in COMM_NO_COM_REQUEST_PENDING (No communication, but a pending request to start communication) If EcuM/Flex and BswM is used, BswM instead use received mode indications from ComM (BswM_ComM_RequestedMode(..)).
ComM_Init	Initializes the AUTOSAR Communication Manager and restarts the internal state machines.
Dem_Init	Initializes or reinitializes this module.
Dem_PreInit	Initializes the internal states necessary to process events reported by BSW-modules.
Dem_Shutdown	Shuts down this module.
GetResource	--
Mcu_GetResetReason	The service reads the reset type from the hardware, if supported.
Mcu_Init	This service initializes the MCU driver.
Mcu_PerformReset	The service performs a microcontroller reset.
Mcu_SetMode	This service activates the MCU power modes.

ReleaseResource	--
Rte_Start	Rte_Start is intended to allocate and initialize system resources and communication resources used by the RTE.
Rte_Stop	Rte_Stop is used to finalize the RTE on the core it is called. This service releases all system and communication resources allocated by the RTE on that core.
Rte_Switch_currentMode_currentMode	--
SchM_Init	SchM_Init is intended to allocate and initialize system resources used by the Basic Software Scheduler part of the RTE for the core on which it is called.
ShutdownOS	--
StartOS	--

⌋ ()

Table 6 - Mandatory interfaces

Remark: The OS service `GetResource` needs a resource name. Therefore the ECU State Manager Fixed module has to define one OS resource name. The name of this OS resource is up to the implementation of the ECU State Manager, nevertheless this document assumes the name “`RES_AUTOSAR_ECUM`”, which will be used in all figures in this document.

8.8.2 Optional Interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of the module.

[SWS_EcuM_02859] ⌈

API function	***Description***
Adc_Init	Initializes the ADC hardware units and driver.
Can_Init	This function initializes the module.
CanIf_Init	This service Initializes internal and external interfaces of the CAN Interface for the further processing.
CanNm_Init	Initialize the CanNm module.
CanSM_Init	This service initializes the CanSM module
CanTp_Init	This function initializes the CanTp module.
CanTrcv_Init	Initializes the CanTrcv module.
CanTSyn_Init	This function initializes the Time Synchronization over CAN.
Com_Init	This service initializes internal and external interfaces and variables of the AUTOSAR COM module layer for the further processing. After calling this function the inter-ECU communication is still disabled.
Dcm_Init	Service for basic initialization of DCM module.
Dem_Init	Initializes or reinitializes this module.
Dem_PreInit	Initializes the internal states necessary to process events reported by BSW-modules.
Dem_Shutdown	Shuts down this module.
Det_Init	Service to initialize the Default Error Tracer.
Det_ReportError	Service to report development errors.
Det_Start	Service to initialize the Default Error Tracer.
Dlt_Init	Dlt is using the NVRamManager and is to be initialized very late in the ECU startup phase. The Dlt_Init() function should be called after the NVRamManager

	is initialized.
DoIP_Init	This service initializes all global variables of the DoIP module. After return of this service the DoIP module is operational.
Ea_Init	Initializes the EEPROM abstraction module.
Eep_Init	Service for EEPROM initialization.
Eth_Init	Initializes the Ethernet Driver
EthIf_Init	Initializes the Ethernet Interface
EthSwt_Init	Initializes the Ethernet Switch Driver
EthSwt_SwitchInit	Initializes the indexed swtich with a given configuration for the switch index
EthTrcv_Init	Initializes the Ethernet Transceiver Driver
EthTSyn_Init	This function initializes the Time Synchronization over Ethernet.
Fee_Init	Service to initialize the FEE module.
FiM_Init	This service initializes the FIM.
Fls_Init	Initializes the Flash Driver.
Fr_Init	Initializes the Fr.
FrArTp_Init	This service initializes all global variables of the FlexRay AUTOSAR Transport Layer and sets all states to idle.
FrIf_Init	Initializes the FlexRay Interface.
FrNm_Init	Initializes the FlexRay NM and its internal state machine.
FrSm_Init	Initializes the FlexRay State Manager.
FrTp_Init	This service initializes all global variables of a FlexRay Transport Layer instance and set it in the idle state. It has no return value because software errors in initialisation data shall be detected during configuration time (e.g. by configuration tool).
FrTSyn_Init	This function initializes the Time Synchronization over FlexRay.
GetCoreID	The function returns a unique core identifier.
Gpt_Init	Initializes the GPT driver.
Icu_Init	This function initializes the driver.
IoHwAb_Init<Init_Id>	Initializes either all the IO Hardware Abstraction software or is a part of the IO Hardware Abstraction.
IpduM_Init	Initializes the I-PDU Multiplexer.
J1939Dcm_Init	This function initializes the J1939 Diagnostic Communication Manager.
J1939Nm_Init	This function initializes the J1939 Network Management module.
J1939Rm_Init	This function initializes the J1939 Request Manager.
J1939Tp_Init	This function initializes the J1939Tp module.
LdCom_Init	This service initializes internal and external interfaces and variables of the AUTOSAR LdCom module for the further processing.
Lin_Init	Initializes the LIN module.
LinIf_Init	Initializes the LIN Interface.
LinSM_Init	This function initializes the LinSM.
LinTp_Init	Initializes the LIN Transport Layer.
Nm_Init	Initializes the NM Interface.
NvM_CancelWriteAll	Service to cancel a running NvM_WriteAll request.
NvM_Init	Service for resetting all internal variables.
NvM_ReadAll	Initiates a multi block read request.
NvM_WriteAll	Initiates a multi block write request.
Ocu_Init	Service for OCU initialization.
PduR_Init	Initializes the PDU Router
Port_Init	Initializes the Port Driver module.
Pwm_Init	Service for PWM initialization.
SchM_Enter_EcuM	--
SchM_Exit_EcuM	--

Sd_Init	Initializes the Service Discovery.
SecOC_Init	Initializes the the SecOC module. Successful initialization leads to state SecOC_INIT.
ShutdownAllCores	After this service the OS on all AUTOSAR cores is shut down. Allowed at TASK level and ISR level and also internally by the OS. The function will never return. The function will force other cores into a shutdown.
SoAd_Init	Initializes the Socket Adaptor.
Spi_Init	Service for SPI initialization.
StartCore	It is not supported to call this function after StartOS(). The function starts the core specified by the parameter CoreID. The OUT parameter allows the caller to check whether the operation was successful or not. If a core is started by means of this function StartOS shall be called on the core.
StbM_Init	Initializes the Synchronized Time-base Manager
TcpIp_Init	This service initializes the TCP/IP Stack. TcpIp_Init may not block the start-up process for an indefinite amount of time. Caveats: The call of this service is mandatory before using the TcpIp instance for further processing.
UdpNm_Init	Initialize the complete UdpNm module, i.e. all channels which are activated at configuration time are initialized. A UDP socket shall be set up with the TCP/IP stack. Caveats: This function has to be called after initialization of the TCP/IP stack. Configuration: Mandatory
Wdg_Init	Initializes the module.
WdgM_DeInit	De-initializes the Watchdog Manager.
WdgM_Init	Initializes the Watchdog Manager.
Xcp_Init	This service initializes interfaces and variables of the AUTOSAR XCP layer.

⌋ ()

Table 7 - Optional Interfaces

8.8.3 Configurable interfaces

There are no configurable interfaces.

9 Sequence Charts

9.1 State Sequences

Sequence charts showing the behavior of the ECU State Manager Fixed module in various states are contained in the flow of the specification text. The following list shows all sequence charts presented in this specification.

- Figure 4 – Startup Sequence (high level diagram)
- Figure 5 – Init Sequence I (STARTUP I)
- Figure 6 – Init Sequence II (STARTUP II)
- Figure 8 – RUN State Sequence (high level diagram)
- Figure 9 – RUN II State Sequence
- Figure 10 – RUN III State Sequence
- Figure 12 – Shutdown Sequence (high level diagram)
- Figure 13 – Deinitialization Sequence I (PREP SHUTDOWN)
- Figure 14 – Deinitialization Sequence IIa (GOSLEEP
- Figure 15 – Deinitialization Sequence IIb (GO OFF I)
- Figure 16 – Deinitialization Sequence III (GO OFF II)
- Figure 17 – Sleep Sequence (high level diagram)
- Figure 18 – Sleep Sequence I
- Figure 19 – Sleep Sequence II
- Figure 20 – Wake-up Sequence (high level diagram)
- Figure 22 – Wake-up Sequence I
- Figure 23 – Wake-up Validation Sequence
- Figure 25 – Wake-up Sequence II

9.2 Wake-up Sequences

The Wake-up Sequences show how a number of modules cooperate to put the ECU into a sleep state to be able to wake up and startup the ECU when a wake up event has occurred.

9.2.1 GPT Wake-up Sequences

The General Purpose Timer (GPT) is one of the possible wake up sources. Usually the GPT is started before the ECU is put to sleep and the hardware timer causes an interrupt when it expires. The interrupt wakes the microcontroller, and executes the interrupt handler in the GPT module. It informs the ECU State Manager Fixed module that a GPT wake up has occurred. In order to distinguish different GPT channels that caused the wake up, the integrator can assign a different wake up source identifier to each GPT channel. Figure 33 shows the corresponding sequence of calls.

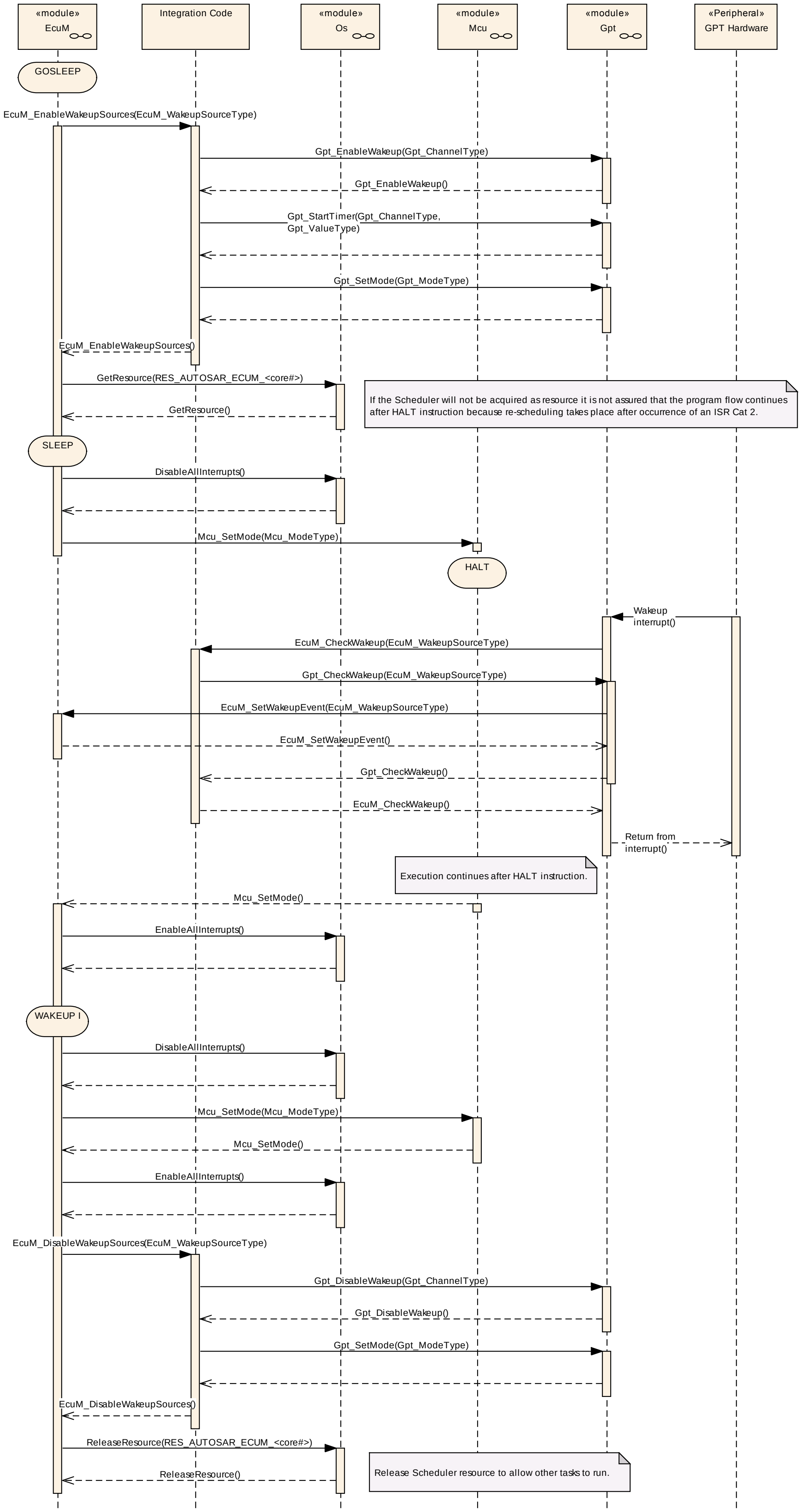

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Gpt
«Peripheral» GPT Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Gpt_EnableWakeup(Gpt_ChannelType)
Gpt_EnableWakeup()
Gpt_StartTimer(Gpt_ChannelType, Gpt_ValueType)
Gpt_SetMode(Gpt_ModeType)
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_<core#>)
GetResource()
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
Gpt_CheckWakeup(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
Gpt_CheckWakeup()
EcuM_CheckWakeup()
Return from interrupt()
Execution continues after HALT instruction.
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Gpt_DisableWakeup(Gpt_ChannelType)
Gpt_DisableWakeup()
Gpt_SetMode(Gpt_ModeType)
EcuM_DisableWakeupSources()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
ReleaseResource()
Release Scheduler resource to allow other tasks to run.

Figure 33 – GPT wake up by interrupt

If the GPT hardware is capable of latching timer overruns, it is also possible to poll the GPT for wake-ups as shown in Figure 34.

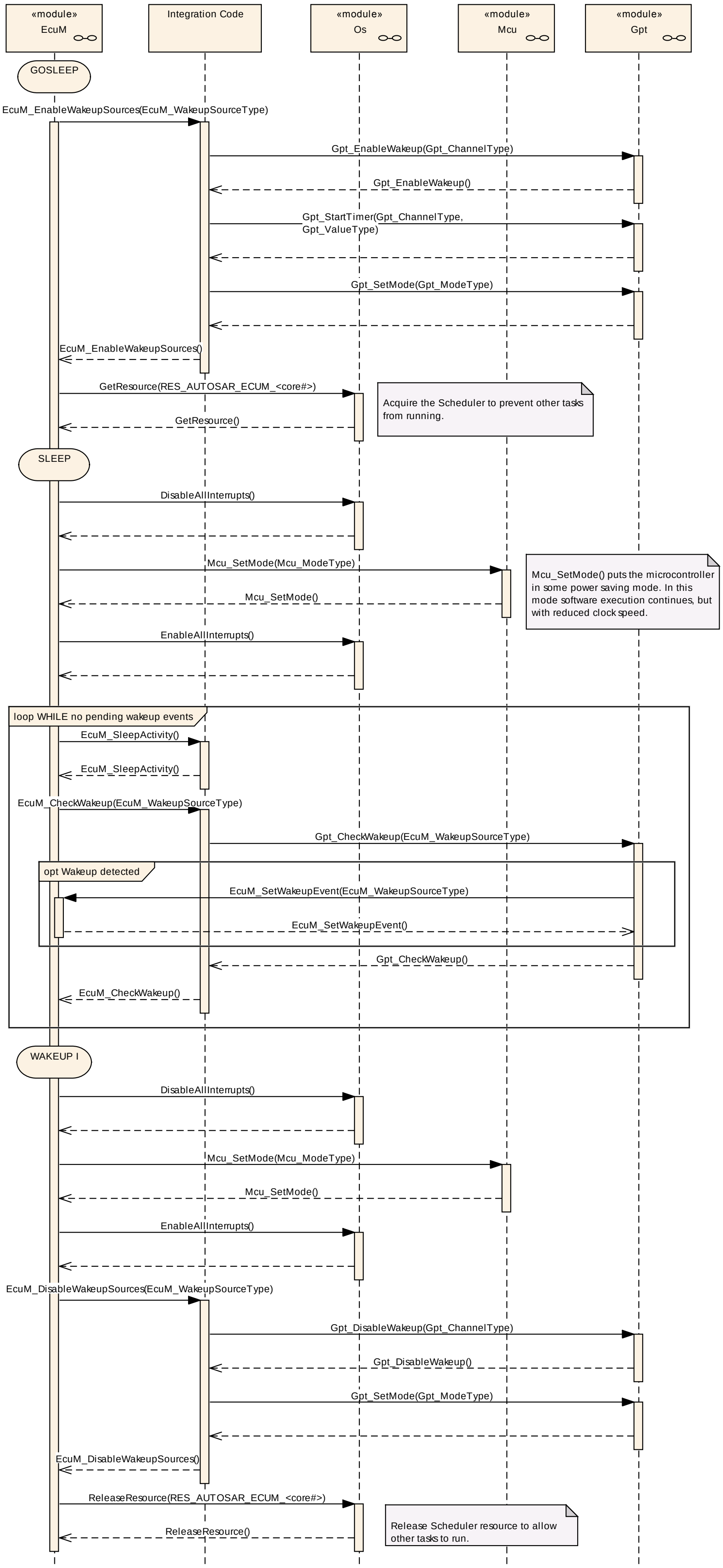

«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Gpt
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Gpt_EnableWakeup(Gpt_ChannelType)
Gpt_EnableWakeup()
Gpt_StartTimer(Gpt_ChannelType,
Gpt_ValueType)
Gpt_SetMode(Gpt_ModeType)
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_<core#>)
Acquire the Scheduler to prevent other tasks from running.
GetResource()
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode() puts the microcontroller in some power saving mode. In this mode software execution continues, but with reduced clock speed.
Mcu_SetMode()
EnableAllInterrupts()
loop WHILE no pending wakeup events
EcuM_SleepActivity()
EcuM_SleepActivity()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
Gpt_CheckWakeup(EcuM_WakeupSourceType)
opt Wakeup detected
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
Gpt_CheckWakeup()
EcuM_CheckWakeup()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Gpt_DisableWakeup(Gpt_ChannelType)
Gpt_DisableWakeup()
Gpt_SetMode(Gpt_ModeType)
EcuM_DisableWakeupSources()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
Release Scheduler resource to allow other tasks to run.
ReleaseResource()

Figure 34 – GPT wake up by polling

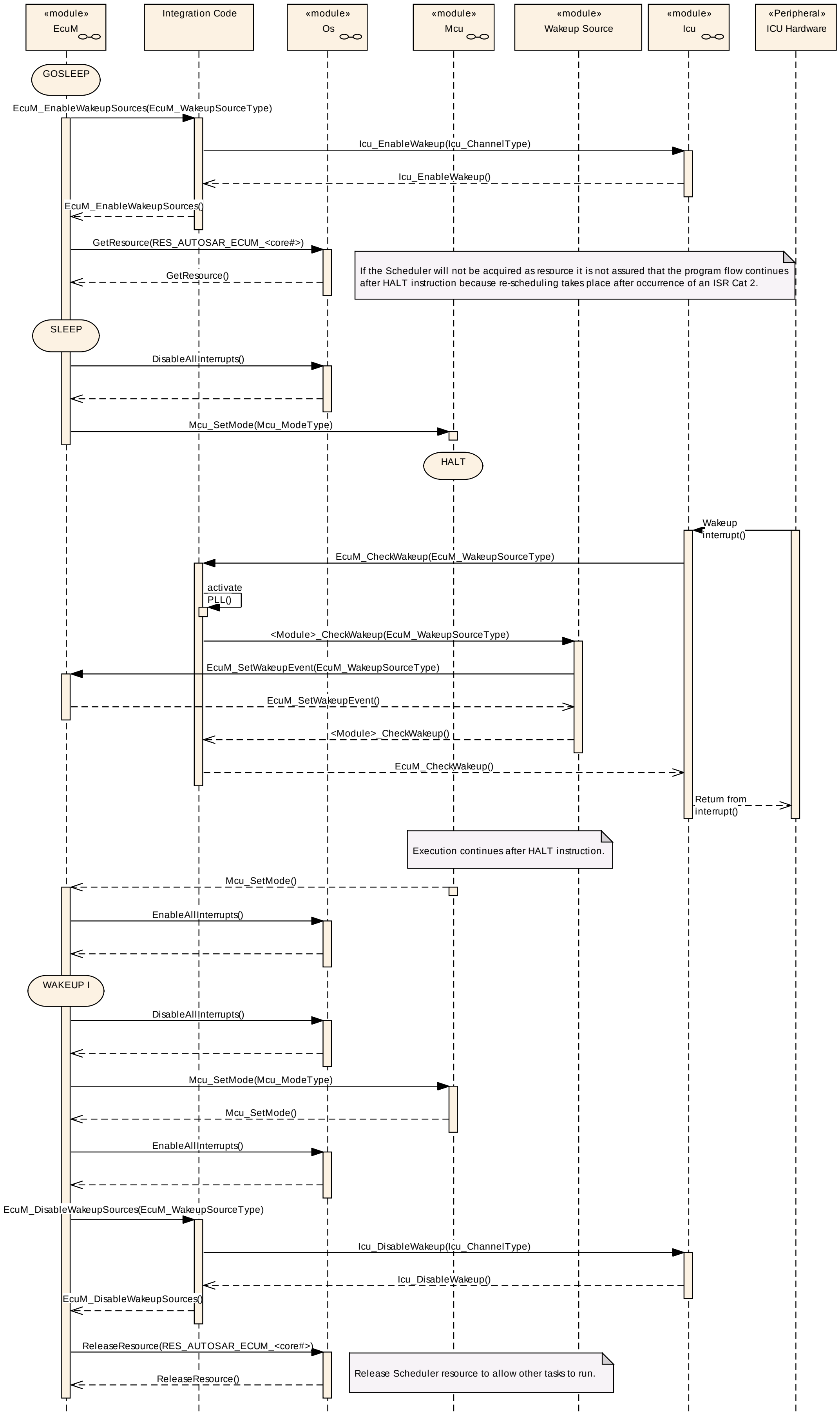
9.2.2 ICU Wake-up Sequences

The Input Capture Unit (ICU) is another wake up source. In contrast to GPT, the ICU driver is not itself the wake up source. It is just the module that processes the wake up interrupt. Therefore, only the driver of the wake up source can tell if it was responsible for that wake up. This makes it necessary for `EcuM_CheckWakeup` to ask the module that is the actual wake up source. In order to know which module to ask, the ICU has to pass the identifier of the wake up source to `EcuM_CheckWakeup`.

For shared interrupts the Integration code may have to check multiple wake up sources within `EcuM_CheckWakeup`. To this end, the ICU has to pass the identifiers of all wake up sources that may have caused this interrupt to `EcuM_CheckWakeup`. Note that, `EcuM_WakeupSourceType` contains one bit for each wake up source, so that multiple wake up sources can be passed in one call.

Figure 35 shows the resulting sequence of calls.

Since the ICU is only responsible for processing the wake up interrupt, polling the ICU is not sensible. For polling the wake up sources have to be checked directly as shown in Figure 19 – Sleep Sequence II.


«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Wakeup Source
«module»
Icu
«Peripheral»
ICU Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Icu_EnableWakeup(Icu_ChannelType)
Icu_EnableWakeup()
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_<core#>)
GetResource()
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup
interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
activate
PLL()
<Module>_CheckWakeup(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
<Module>_CheckWakeup()
EcuM_CheckWakeup()
Return from
interrupt()
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Icu_DisableWakeup(Icu_ChannelType)
Icu_DisableWakeup()
EcuM_DisableWakeupSources()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
ReleaseResource()

If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.

Execution continues after HALT instruction.

Release Scheduler resource to allow other tasks to run.

Figure 35 – ICU wake up by interrupt

9.2.3 CAN Wake-up Sequences

On CAN a wake up can be detected by the transceiver or the communication controller using either an interrupt or polling. Wake up source identifiers should be shared between transceiver and controller as the ECU State Manager Fixed module only needs to know the network that has woken up and passes that on to the Communication Manager module.

In interrupt case or in shared interrupt case it is not clear which specific wake up source (CAN controller, CAN transceiver, LIN controller etc.) detected the wake up. Therefore the integrator has to assign the derived wakeupSource of EcuM_CheckWakeup(wakeupSource), which could stand for a shared interrupt or just for a interrupt channel, to specific wake up sources which are passed to CanIf_CheckWakeup(WakeupSource). So here the parameters wakeupSource from EcuM_CheckWakeup() could be different to WakeupSource of CanIf_CheckWakeup or they could equal. It depends on the hardware topology and the implementation in the integrator code of EcuM_CheckWakeup().

During CanIf_CheckWakeup(WakeupSource) the CAN Interface module (CanIf) will check if any device (CAN communication controller or transceiver) is configured with the value of “WakeupSource”. If this is the case, the device is checked for wake up via the corresponding device driver module. If the device detected a wake up, the device driver informs EcuM via EcuM_SetWakeupEvent(sources). The parameter “sources” is set to the configured value at the device. Thus it is set to the value CanIf_CheckWakeup() was called with.

Multiple devices might be configured with the same wake up source value. But if devices are connected to different bus medium and they are wake-able, it makes sense to configure them with different wake up sources.

The following CAN Wake-up Sequences are partly optional, because there is no specification for the “Integration Code”. Thus it is implementation specific if e.g. during EcuM_CheckWakeup() the CanIf is called to check the wake up source.

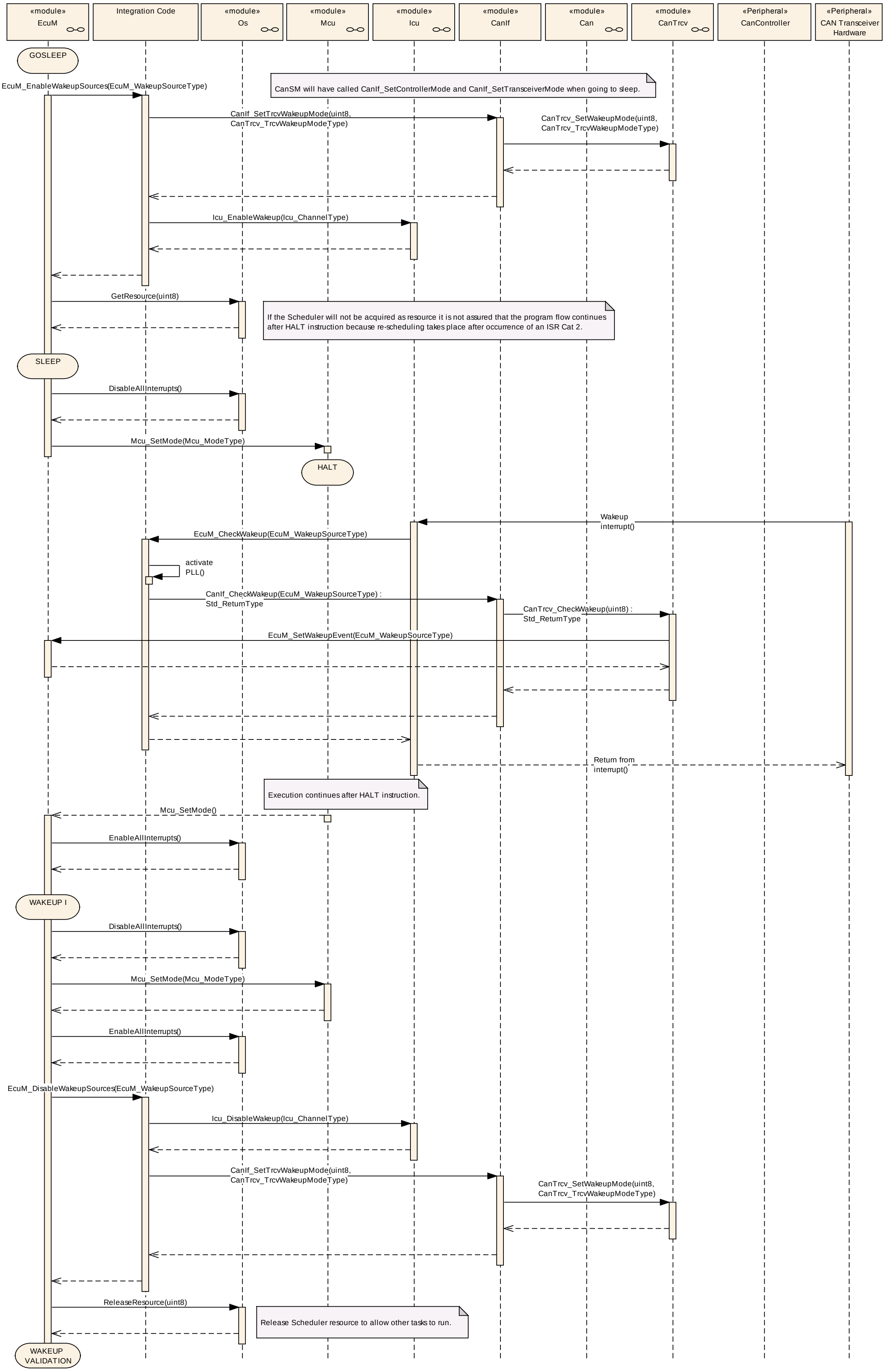
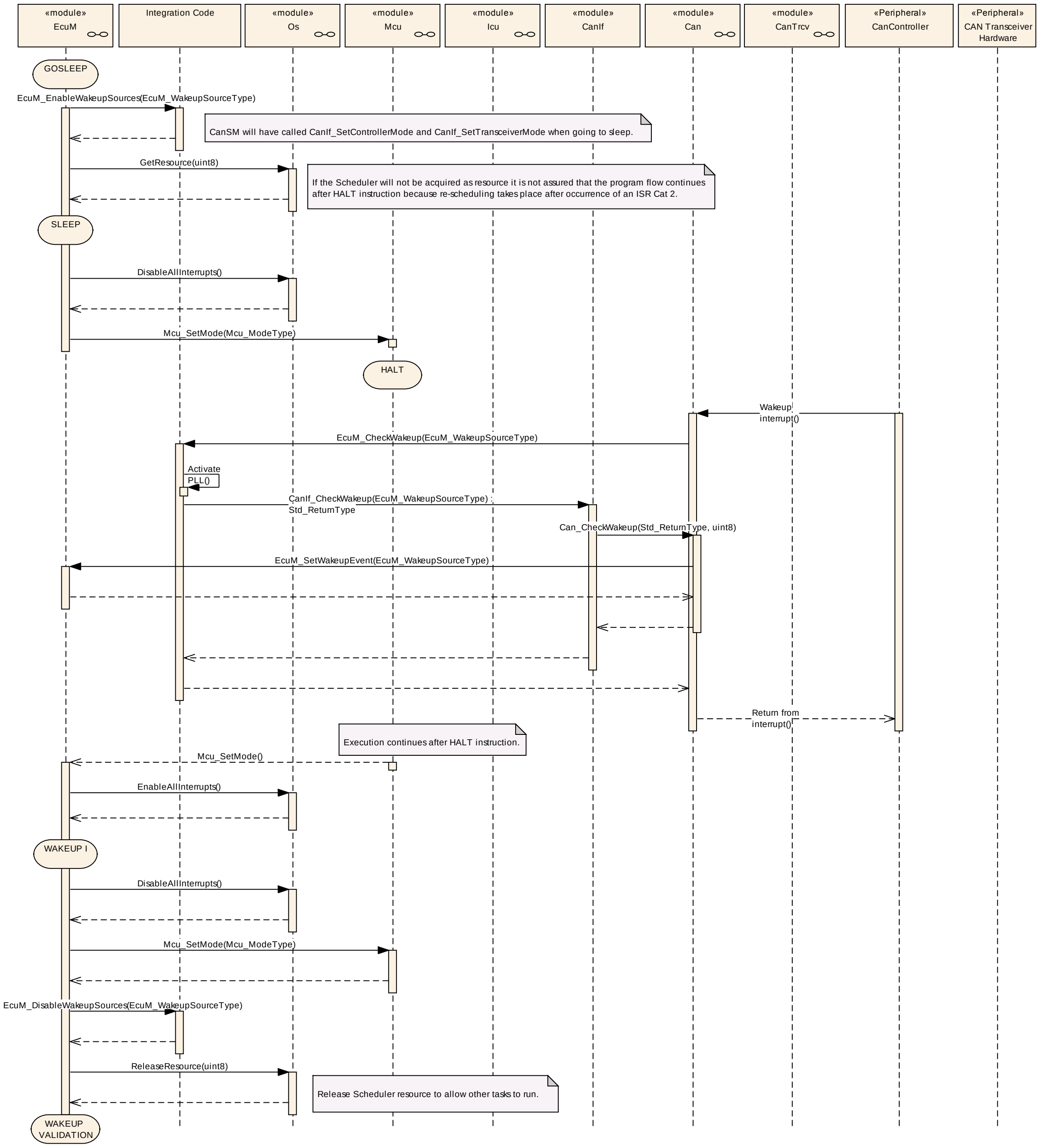

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode when going to sleep.
CanIf_SetTrcvWakeupMode(uint8, CanTrcv_TrcvWakeupModeType)
CanTrcv_SetWakeupMode(uint8, CanTrcv_TrcvWakeupModeType)
Icu_EnableWakeup(Icu_ChannelType)
GetResource(uint8)
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
activate PLL()
CanIf_CheckWakeup(EcuM_WakeupSourceType) : Std_ReturnType
CanTrcv_CheckWakeup(uint8) : Std_ReturnType
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Execution continues after HALT instruction.
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Icu_DisableWakeup(Icu_ChannelType)
CanIf_SetTrcvWakeupMode(uint8, CanTrcv_TrcvWakeupModeType)
CanTrcv_SetWakeupMode(uint8, CanTrcv_TrcvWakeupModeType)
ReleaseResource(uint8)
Release Scheduler resource to allow other tasks to run.
WAKEUP VALIDATION

Figure 36 – CAN transceiver wake up by interrupt

Figure 36 shows the CAN transceiver wake up via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.

A CAN controller wake up by interrupt works similar to the GPT wake up. Here the interrupt handler and the CheckWakeup functionality are both encapsulated in the CAN Driver module, as shown in Figure 37.


«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(uint8)
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
Activate PLL()
CanIf_CheckWakeup(EcuM_WakeupSourceType) : Std_ReturnType
Can_CheckWakeup(Std_ReturnType, uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(uint8)
WAKEUP VALIDATION

CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode when going to sleep.

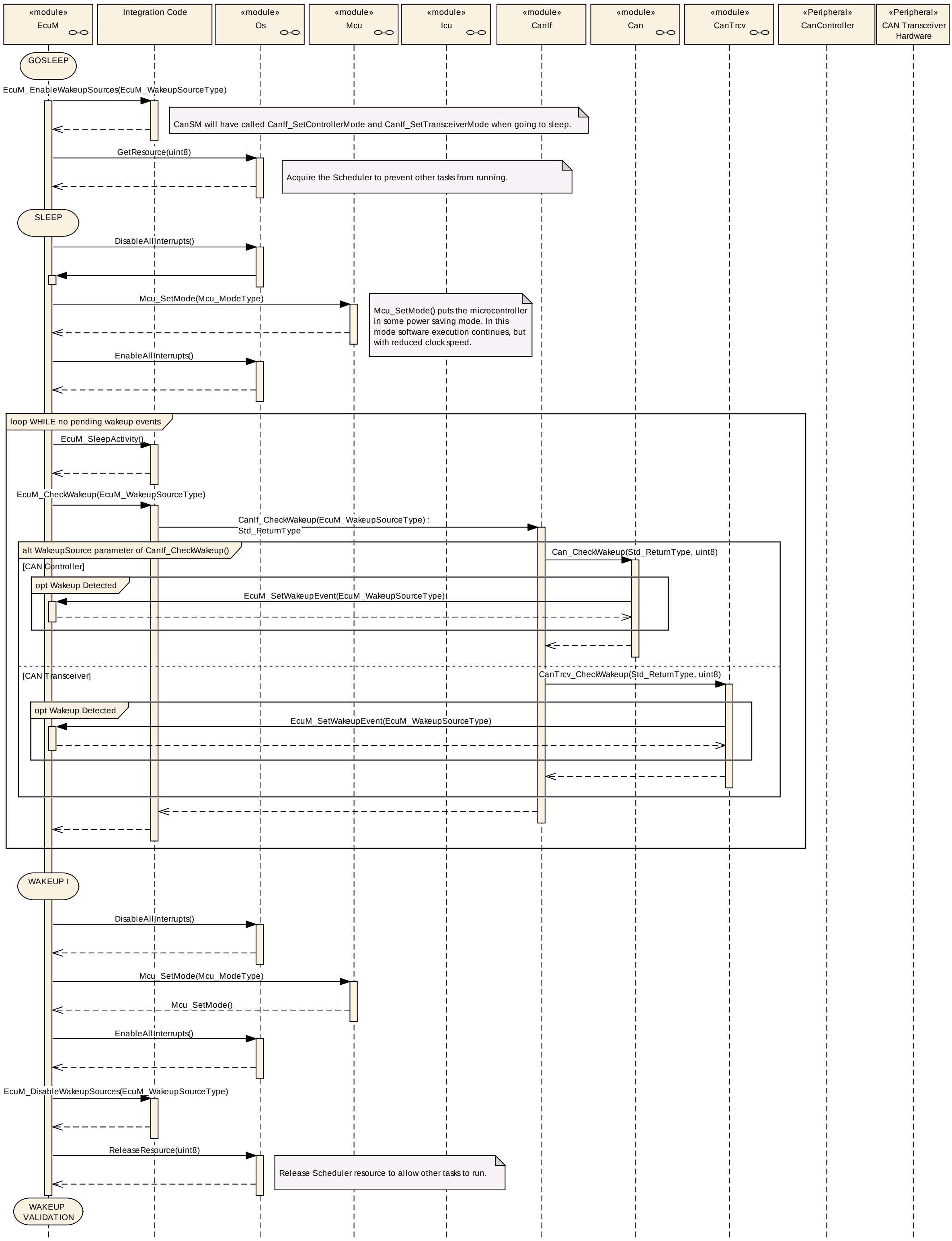
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.

Execution continues after HALT instruction.

Release Scheduler resource to allow other tasks to run.

Figure 37 – CAN controller wake up by interrupt

Wake up by polling is possible both for CAN transceiver and CAN controller. The ECU State Manager Fixed module will regularly check the CAN Interface module, which in turn asks either the CAN Driver module or the CAN Transceiver Driver module depending on the wake up source parameter passed to the CAN Interface module, as shown in Figure 38.


«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(uint8)
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
loop WHILE no pending wakeup events
EcuM_SleepActivity()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
CanIf_CheckWakeup(EcuM_WakeupSourceType) : Std_ReturnType
alt WakeupSource parameter of CanIf_CheckWakeup()
[CAN Controller]
Can_CheckWakeup(Std_ReturnType, uint8)
opt Wakeup Detected
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
[CAN Transceiver]
CanTrcv_CheckWakeup(Std_ReturnType, uint8)
opt Wakeup Detected
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(uint8)
WAKEUP VALIDATION

CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode when going to sleep.

Acquire the Scheduler to prevent other tasks from running.

Mcu_SetMode() puts the microcontroller in some power saving mode. In this mode software execution continues, but with reduced clock speed.

Release Scheduler resource to allow other tasks to run.

Figure 38 – CAN controller or transceiver wake up by polling

After the detection of a wake up event from the CAN transceiver or CAN controller by either interrupt or polling, the wake up event can be validated. This is done by switching on the corresponding CAN transceiver and CAN controller in `EcuM_StartWakeupSources`. It depends on the used CAN transceivers and controllers, which function calls in Integrator Code EcuM_StartWakeupSource are necessary. In Figure 39 e.g. the needed function calls to start and stop the wake up sources from CAN state manager module are mentioned.

Note that, although controller and transceiver are switched on, no CAN message will be forwarded by the CAN Interface module to any upper layer module.

Only when the corresponding PDU channel modes of the CAN Interface module are set to “Online”, it will forward CAN messages.

The CAN Interface module recognizes the successful reception of at least one message and records it as a successful validation. During validation the ECU State Manager Fixed module regularly checks the CAN Interface module in Integrator Code `EcuM_CheckValidation`.

The ECU State Manager Fixed module will, after successful validation, continue the normal startup of the CAN network via the Communication Manager module.

Otherwise, it will shutdown the CAN controller and CAN transceiver in `EcuM_StopWakeupSources` and go back to sleep. The resulting sequence is shown in Figure 39.

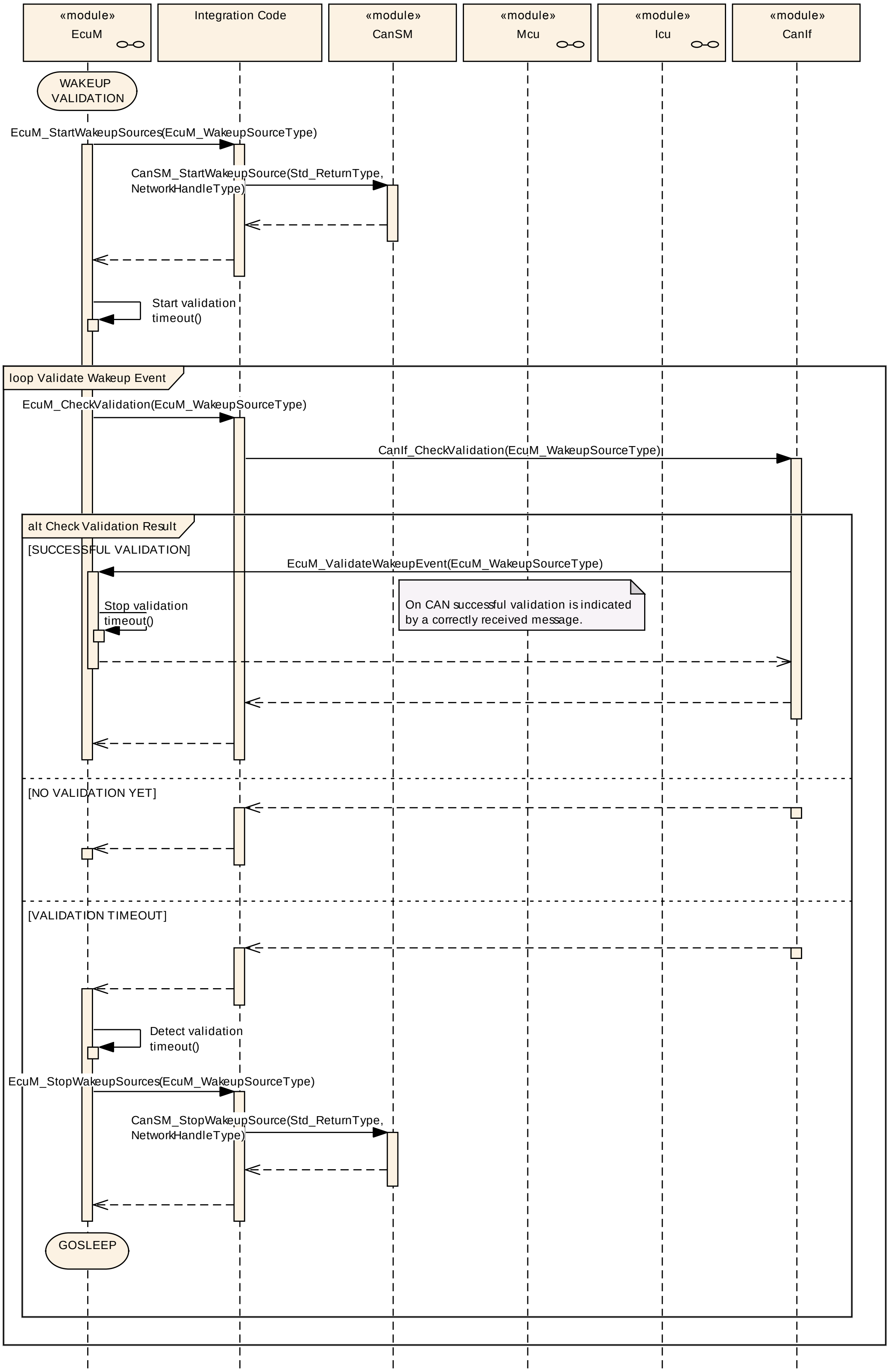
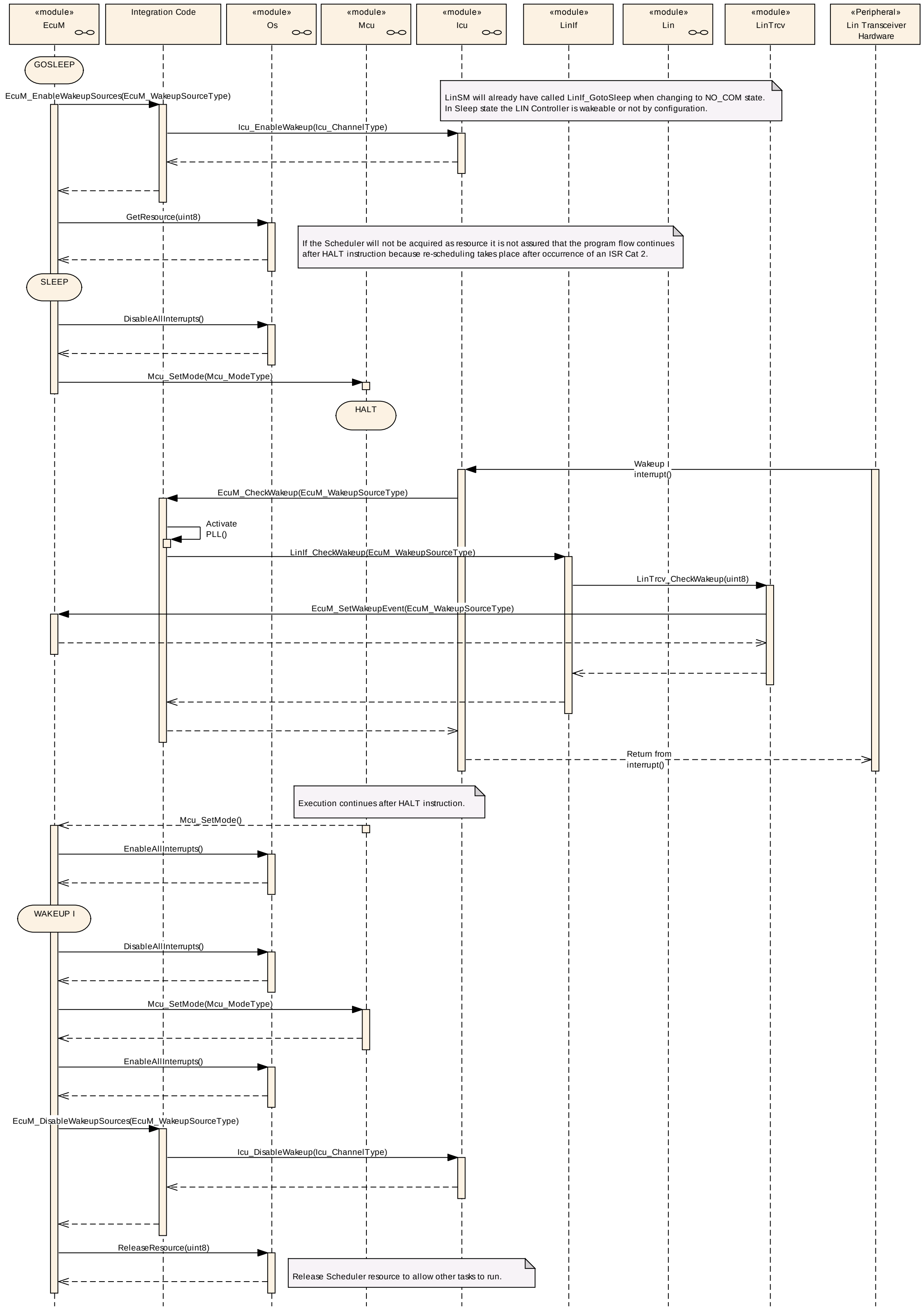

«module»
EcuM
Integration Code
«module»
CanSM
«module»
Mcu
«module»
Icu
«module»
CanIf
WAKEUP VALIDATION
EcuM_StartWakeupSources(EcuM_WakeupSourceType)
CanSM_StartWakeupSource(Std_ReturnType, NetworkHandleType)
Start validation timeout()
loop Validate Wakeup Event
EcuM_CheckValidation(EcuM_WakeupSourceType)
CanIf_CheckValidation(EcuM_WakeupSourceType)
alt Check Validation Result
[SUCCESSFUL VALIDATION]
EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType)
Stop validation timeout()
On CAN successful validation is indicated by a correctly received message.
[NO VALIDATION YET]
[VALIDATION TIMEOUT]
Detect validation timeout()
EcuM_StopWakeupSources(EcuM_WakeupSourceType)
CanSM_StopWakeupSource(Std_ReturnType, NetworkHandleType)
GOSLEEP

Figure 39 – CAN wake up validation

9.2.4 LIN Wake-up Sequences

Figure 40 shows the LIN transceiver wake up via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.


«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» LinIf
«module» Lin
«module» LinTrcv
«Peripheral» Lin Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Icu_EnableWakeup(Icu_ChannelType)
GetResource(uint8)
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup I interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
Activate PLL()
LinIf_CheckWakeup(EcuM_WakeupSourceType)
LinTrcv_CheckWakeup(uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Icu_DisableWakeup(Icu_ChannelType)
ReleaseResource(uint8)

LinSM will already have called LinIf_GotoSleep when changing to NO_COM state. In Sleep state the LIN Controller is wakeable or not by configuration.

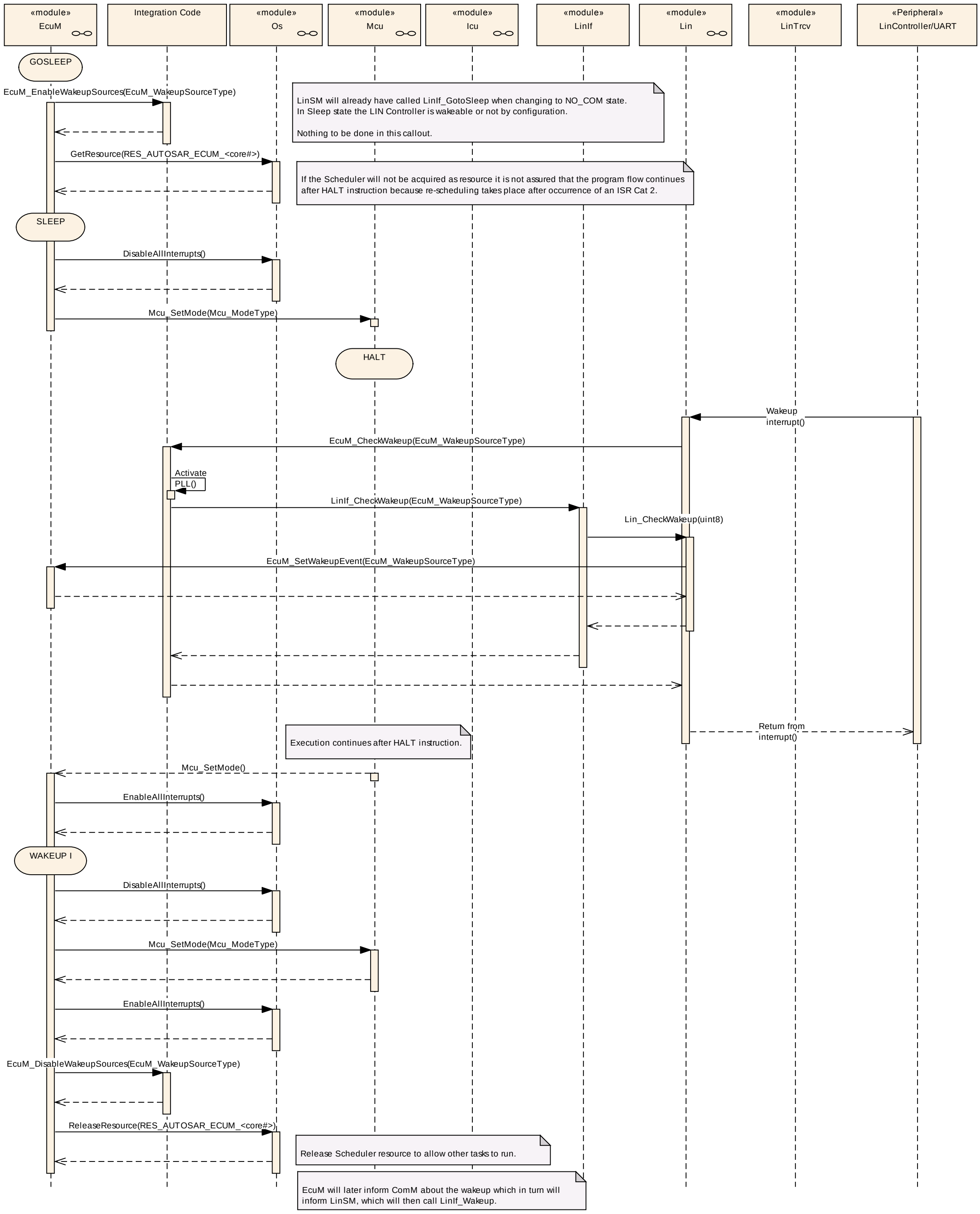
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.

Execution continues after HALT instruction.

Release Scheduler resource to allow other tasks to run.

Figure 40 – LIN transceiver wake up by interrupt

As shown in Figure 41, the LIN controller wake up by interrupt works similar to the CAN controller wake up by interrupt. In both cases the Driver module encapsulates the interrupt handler.


«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» LinIf
«module» Lin
«module» LinTrcv
«Peripheral» LinController/UART
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(RES_AUTOSAR_ECUM_<core#>)
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
Activate PLL()
LinIf_CheckWakeup(EcuM_WakeupSourceType)
Lin_CheckWakeup(uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)

LinSM will already have called LinIf_GotoSleep when changing to NO_COM state. In Sleep state the LIN Controller is wakeable or not by configuration.

Nothing to be done in this callout.

If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.

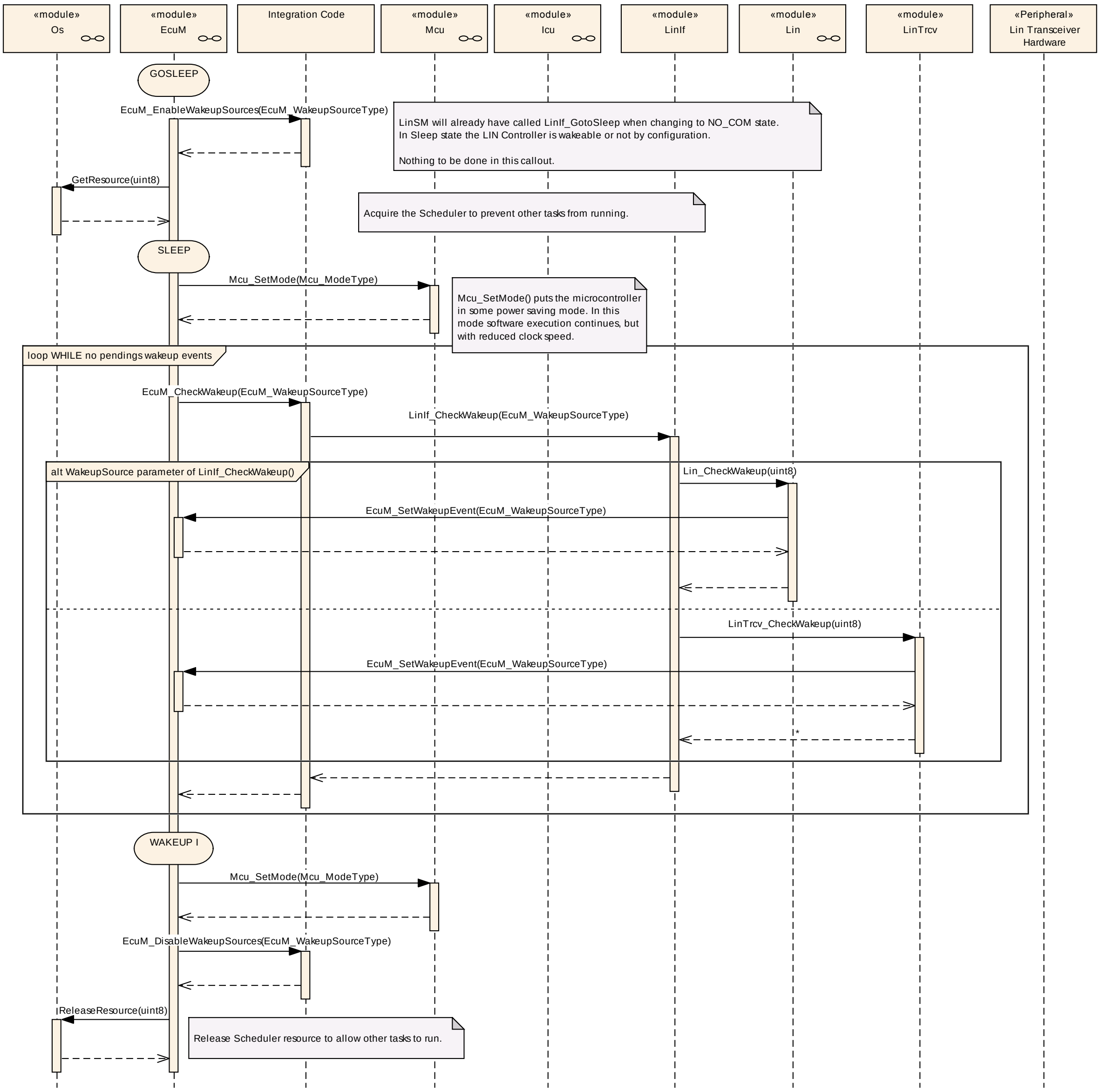
Execution continues after HALT instruction.

Release Scheduler resource to allow other tasks to run.

EcuM will later inform ComM about the wakeup which in turn will inform LinSM, which will then call LinIf_Wakeup.

Figure 41 – LIN Controller wake up by Interrupt

Wake up by polling is possible for LIN transceiver and LIN controller. The ECU State Manager Fixed module will regularly check the LIN Interface module, which in turn asks either the LIN Driver module or the LIN Transceiver Driver module, as shown in Figure 42.


«module» Os
«module» EcuM
Integration Code
«module» Mcu
«module» Icu
«module» LinIf
«module» Lin
«module» LinTrcv
«Peripheral» Lin Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(uint8)
SLEEP
Mcu_SetMode(Mcu_ModeType)
loop WHILE no pendings wakeup events
EcuM_CheckWakeup(EcuM_WakeupSourceType)
LinIf_CheckWakeup(EcuM_WakeupSourceType)
alt WakeupSource parameter of LinIf_CheckWakeup()
Lin_CheckWakeup(uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
LinTrcv_CheckWakeup(uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
WAKEUP I
Mcu_SetMode(Mcu_ModeType)
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(uint8)

LinSM will already have called LinIf_GotoSleep when changing to NO_COM state.
In Sleep state the LIN Controller is wakeable or not by configuration.

Nothing to be done in this callout.

Acquire the Scheduler to prevent other tasks from running.

Mcu_SetMode() puts the microcontroller in some power saving mode. In this mode software execution continues, but with reduced clock speed.

Release Scheduler resource to allow other tasks to run.

Figure 42 – LIN controller or transceiver wake up by polling

Note that LIN does not require wake up validation.

9.2.5 FlexRay Wake-up Sequences

For FlexRay a wake up is only possible via the FlexRay transceivers. There are two transceivers for the two different channels in a FlexRay cluster. They are treated as belonging to one network and thus, there should be only one wake up source identifier configured for both channels.

Figure 43 shows the FlexRay transceiver wake up via interrupt. The interrupt is usually handled by the ICU Driver module as described in Chapter 9.2.2.

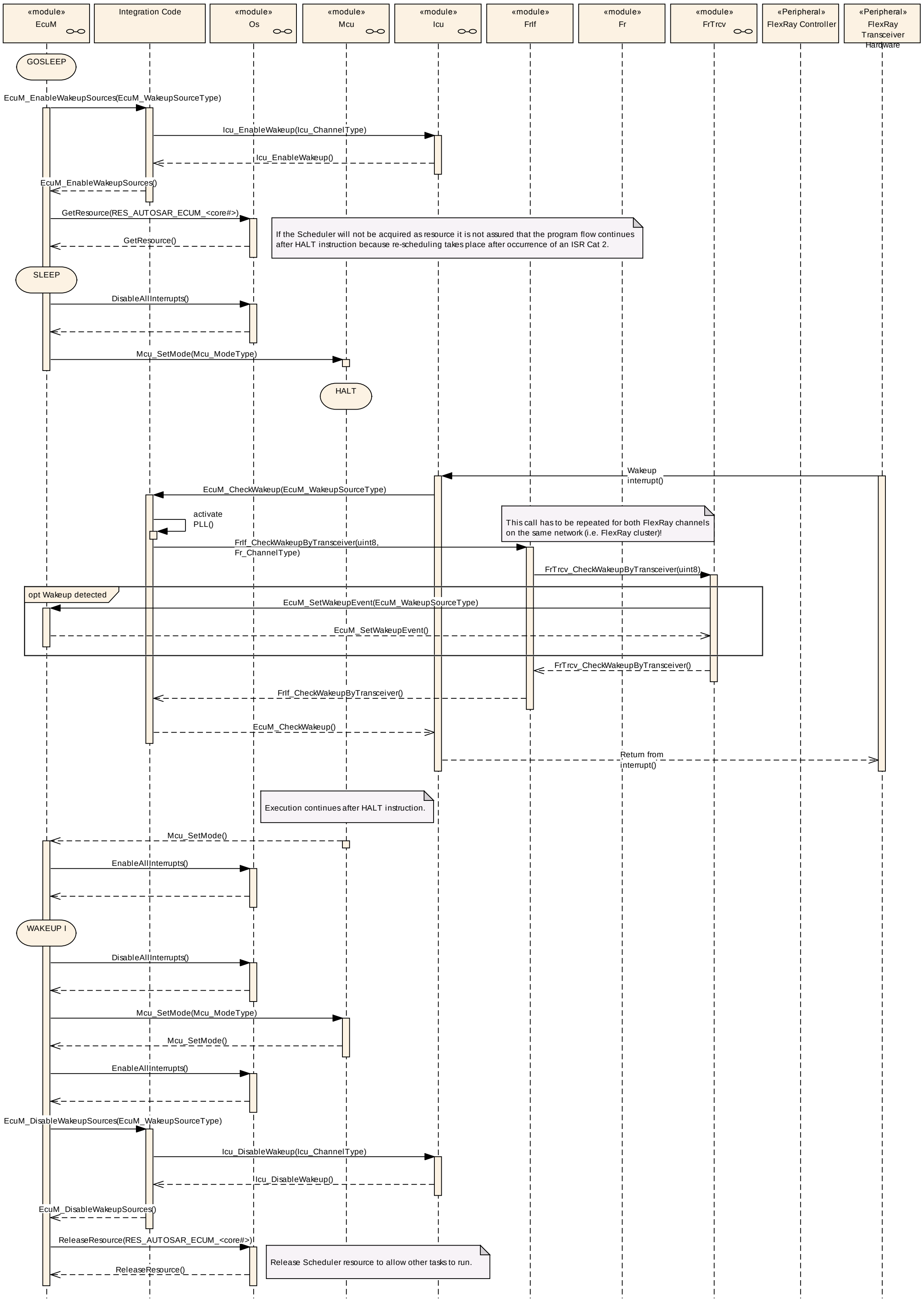

«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Icu
«module»
FrIf
«module»
Fr
«module»
FrTrcv
«Peripheral»
FlexRay Controller
«Peripheral»
FlexRay Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Icu_EnableWakeup(Icu_ChannelType)
Icu_EnableWakeup()
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_<core#>)
GetResource()
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
activate PLL()
This call has to be repeated for both FlexRay channels on the same network (i.e. FlexRay cluster)!
FrIf_CheckWakeupByTransceiver(uint8, Fr_ChannelType)
FrTrcv_CheckWakeupByTransceiver(uint8)
opt Wakeup detected
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
FrTrcv_CheckWakeupByTransceiver()
FrIf_CheckWakeupByTransceiver()
EcuM_CheckWakeup()
Return from interrupt()
Execution continues after HALT instruction.
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Icu_DisableWakeup(Icu_ChannelType)
Icu_DisableWakeup()
EcuM_DisableWakeupSources()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
ReleaseResource()
Release Scheduler resource to allow other tasks to run.

Figure 43 – FlexRay transceiver wake up by interrupt

Note that in EcuM_CheckWakeup there need to be two separate calls to `FrIf_CheckWakeupByTransceiver`, one for each FlexRay channel.

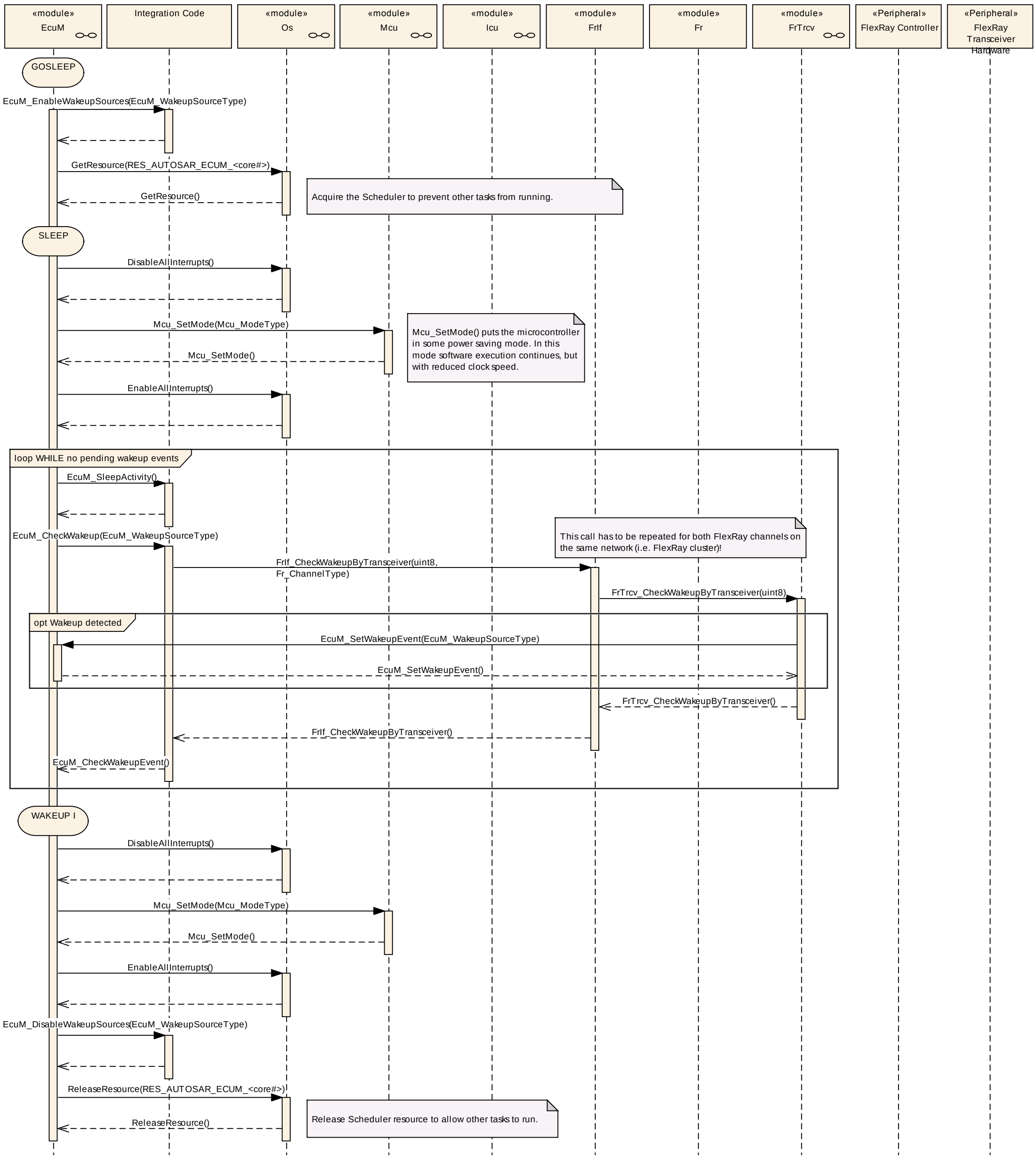

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» FrIf
«module» Fr
«module» FrTrcv
«Peripheral» FlexRay Controller
«Peripheral» FlexRay Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(RES_AUTOSAR_ECUM_<core#>)
GetResource()
Acquire the Scheduler to prevent other tasks from running.
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
Mcu_SetMode() puts the microcontroller in some power saving mode. In this mode software execution continues, but with reduced clock speed.
EnableAllInterrupts()
loop WHILE no pending wakeup events
EcuM_SleepActivity()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
This call has to be repeated for both FlexRay channels on the same network (i.e. FlexRay cluster)!
FrIf_CheckWakeupByTransceiver(uint8, Fr_ChannelType)
FrTrcv_CheckWakeupByTransceiver(uint8)
opt Wakeup detected
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
FrTrcv_CheckWakeupByTransceiver()
FrIf_CheckWakeupByTransceiver()
EcuM_CheckWakeupEvent()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
ReleaseResource()
Release Scheduler resource to allow other tasks to run.

Figure 44 – FlexRay transceiver wake up by polling

10 Configuration specification

10.1 Containers and configuration parameters

For details refer to the chapter 10.1.1 “Configuration and configuration parameters” in *SWS_BSWGeneral.*

10.2 Published Information

For details refer to the chapter 10.3 “Published Information” in *SWS_BSWGeneral.*

10.3 Configurable Parameters

[SWS_EcuM_00809] ⌈The following containers contain various references to initialization structures of BSW modules. NULL shall be a valid reference meaning ‘no configuration data available’ but only if the implementation of the initialized BSW module supports this. ⌋ ()

10.3.1 EcuM

SWS Item	**ECUC_EcuM_00225 :**
Module Name	*EcuM*
Module Description	Configuration of the EcuM (ECU State Manager) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMConfiguration	1	This container contains the configuration (parameters) of the ECU State Manager.
EcuMFixedGeneral	0..1	This container holds the general, pre-compile configuration parameters for the EcuMFixed. Only applicable if EcuMFixed is implemented. **Tags:** atp.Status=obsolete
EcuMFlexGeneral	0..1	This container holds the general, pre-compile configuration parameters for the EcuMFlex. Only applicable if EcuMFlex is implemented.
EcuMGeneral	1	This container holds the general, pre-compile configuration parameters.

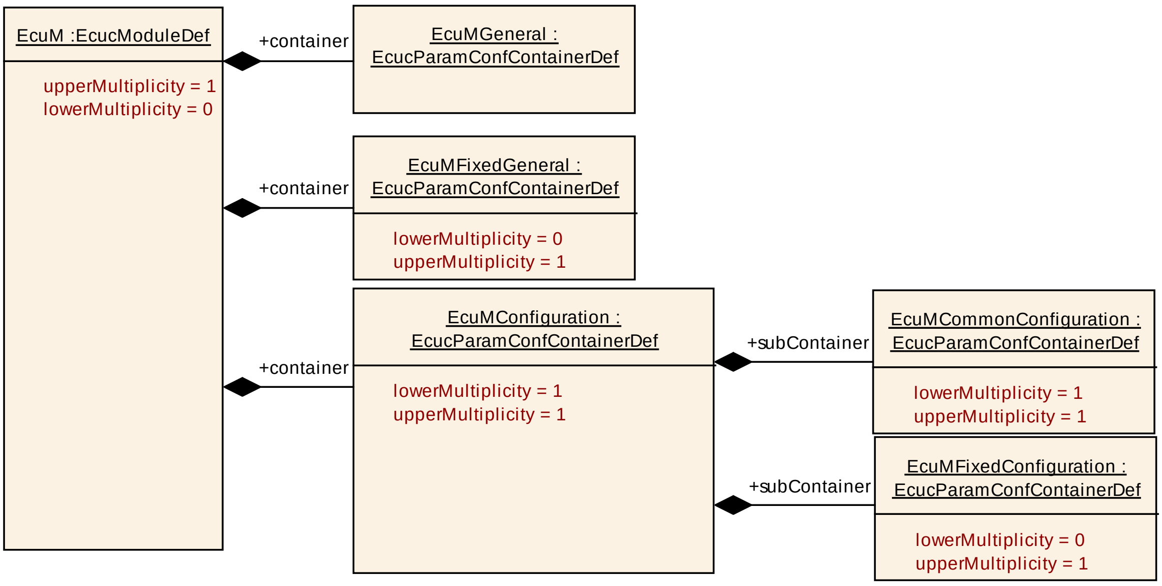


Figure 45 – EcuM Fixed Containers

10.3.2 EcuMGeneral

SWS Item	**ECUC_EcuM_00116 :**
Container Name	EcuMGeneral
Description	This container holds the general, pre-compile configuration parameters.
Configuration Parameters	

SWS Item	**ECUC_EcuM_00108 :**		
Name	EcuMDevErrorDetect		
Parent Container	EcuMGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00118 :**
Name	EcuMIncludeDet
Parent Container	EcuMGeneral
Description	If defined, the according BSW module will be initialized by the ECU State Manager
Multiplicity	1
Type	EcucBooleanParamDef

Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00121 :**		
Name	EcuMMainFunctionPeriod		
Parent Container	EcuMGeneral		
Description	This parameter defines the schedule period of EcuM_MainFunction. Unit: [s]		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Dependency is applicable for EcuMFixed: SWS_EcuM_00594. No Dependency for EcuMFlex.		

SWS Item	**ECUC_EcuM_00149 :**		
Name	EcuMVersionInfoApi		
Parent Container	EcuMGeneral		
Description	Switches the version info API on or off		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

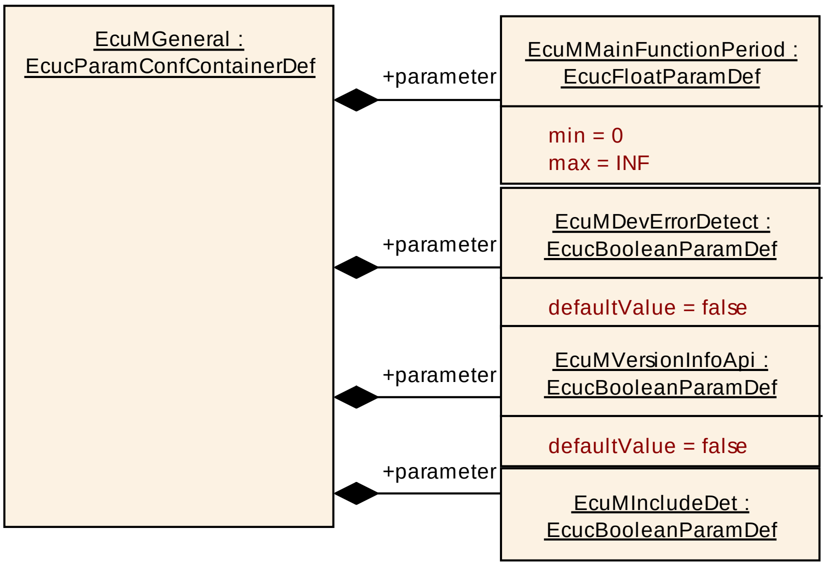


Figure 46 – Container EcuMGeneral

10.3.3 EcuMFixedGeneral

SWS Item	**ECUC_EcuM_00166 : (Obsolete)**
Container Name	EcuMFixedGeneral
Description	This container holds the general, pre-compile configuration parameters for the EcuMFixed. Only applicable if EcuMFixed is implemented. **Tags:** atp.Status=obsolete
Configuration Parameters	

SWS Item	**ECUC_EcuM_00189 : (Obsolete)**		
Name	EcuMIncludeComM		
Parent Container	EcuMFixedGeneral		
Description	This configuration parameter defines whether the communication manager is supported by EcuM. This feature is presented for development purpose to compile out the communication manager in the early debugging phase. **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00144 : (Obsolete)**
Name	EcuMTTIIEnabled
Parent Container	EcuMFixedGeneral
Description	Boolean switch to enable / disable TTII **Tags:**

	atp.Status=obsolete		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00145 : (Obsolete)**		
Name	EcuMTTIIWakeupSourceRef		
Parent Container	EcuMFixedGeneral		
Description	This configuration parameter references the initial sleep mode to be used by TTII when TTII is activated after a RUN mode. **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	Symbolic name reference to [EcuMWakeupSource]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

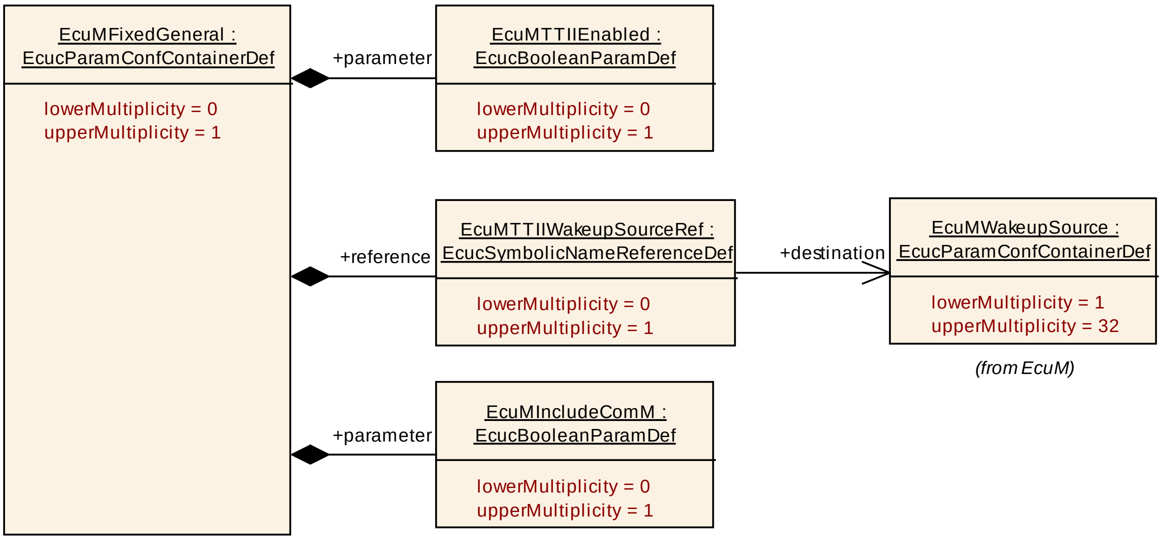


Figure 47 – Container EcuMFixedGeneral

10.3.4 EcuMFixedConfiguration

SWS Item	**ECUC_EcuM_00165 : (Obsolete)**
Container Name	EcuMFixedConfiguration
Description	This container contains the configuration (parameters) of the EcuMFixed. Only applicable if EcuMFixed is implemented.

	Tags: atp.Status=obsolete
Configuration Parameters	

SWS Item	**ECUC_EcuM_00126 : (Obsolete)**		
Name	EcuMNvramReadallTimeout		
Parent Container	EcuMFixedConfiguration		
Description	Period given in seconds for which the ECU State Manager will wait until it considers a ReadAll job of the NVRAM Manager as failed. **Tags:** atp.Status=obsolete		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00127 : (Obsolete)**		
Name	EcuMNvramWriteallTimeout		
Parent Container	EcuMFixedConfiguration		
Description	Period given in seconds for which the ECU State Manager will wait until it considers a WriteAll job of the NVRAM Manager as failed. **Tags:** atp.Status=obsolete		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00129 : (Obsolete)**		
Name	EcuMRunMinimumDuration		
Parent Container	EcuMFixedConfiguration		
Description	Duration given in seconds for which the ECU State Manager will stay in RUN state even when no one requests RUN. This duration should be long at least as long as a SW-Cs needs to request RUN. **Tags:** atp.Status=obsolete		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00191 : (Obsolete)**		
Name	EcuMComMCommunicationAllowedList		
Parent Container	EcuMFixedConfiguration		
Description	These parameters contain references to the ComMChannels for which EcuM has to call ComM_CommunicationAllowed. **Tags:** atp.Status=obsolete		
Multiplicity	0..*		
Type	Symbolic name reference to [ComMChannel]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00125 : (Obsolete)**		
Name	EcuMNormalMcuModeRef		
Parent Container	EcuMFixedConfiguration		
Description	This parameter is a reference to the normal MCU mode to be restored after a sleep. **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	Symbolic name reference to [McuModeSettingConf]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitListThree	0..1	Container for Init Block III. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order. All modules in this list are initialized after the OS is started and so these modules may use OS support. These modules may also rely on the Nvram ReadAll job to have provided all data. **Tags:** atp.Status=obsolete
EcuMDriverInitListTwo	0..1	Container for Init Block II. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order. All modules in this list are initialized after the OS is started and so these modules may use OS support. These modules may not rely on the Nvram ReadAll job to have provided all data. **Tags:** atp.Status=obsolete
EcuMFixedUserConfig	0..*	These containers describe the identifiers that are needed to refer to a software component or another appropriate entity in

		the system which is designated to request the RUN state. Application requestors refer to entities above RTE, system requestors to entities below RTE (e.g. Communication Manager). **Tags:** atp.Status=obsolete
EcuMTTII	0..*	These containers describe the structures and the following configuration items describe its elements. These structures are concatenated to build a list as indicated by Figure 27 - Configuration Container Diagram. The list must contain at least one element when ECUM_TTII_ENABLED is set to true. **Tags:** atp.Status=obsolete

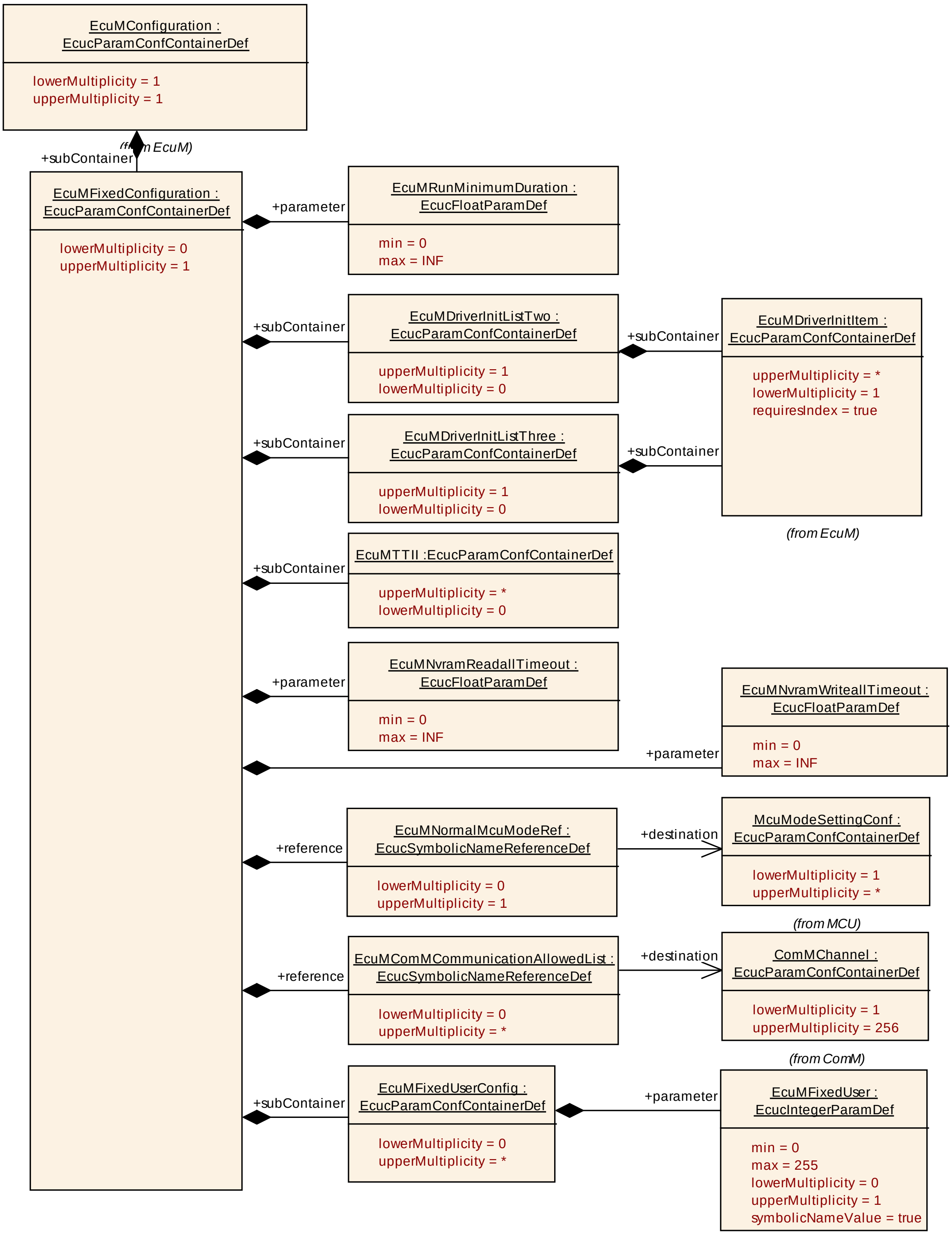


Figure 48 – Container EcuMFixedConfiguration

10.3.5 EcuMCommonConfiguration

SWS Item	**ECUC_EcuM_00181 :**
Container Name	EcuMCommonConfiguration
Description	This container contains the common configuration (parameters) of the

	ECU State Manager.
Configuration Parameters	

SWS Item	**ECUC_EcuM_00102 :**		
Name	EcuMConfigConsistencyHash		
Parent Container	EcuMCommonConfiguration		
Description	In the pre-compile and link-time configuration phase a hash value is generated across all pre-compile and link-time parameters of all BSW modules. In the post-build phase a hash value is generated across all pre-compile and link-time parameters, except for parameters located in EcucParamConfContainerDef instances or subContainers which have been introduced at post-build configuration time. This hash value is compared against each other and allows checking the consistency of the entire configuration.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	--	
	Link time	X	VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00104 :**		
Name	EcuMDefaultAppMode		
Parent Container	EcuMCommonConfiguration		
Description	The default application mode loaded when the ECU comes out of reset.		
Multiplicity	1		
Type	Reference to [OsAppMode]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00183 :**		
Name	EcuMOSResource		
Parent Container	EcuMCommonConfiguration		
Description	This parameter is a reference to a OS resource which is used to bring the ECU into sleep mode. In case of multi core each core shall have an own OsResource.		
Multiplicity	1..*		
Type	Reference to [OsResource]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDefaultShutdownTarget	1	This container describes the default shutdown target to be selected by EcuM. The actual shutdown target may be overridden by the EcuM_SelectShutdownTarget service.
EcuMDriverInitListOne	0..1	Container for Init Block I. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order. All modules in this list are initialized before the OS is started and so these modules require no OS support.
EcuMDriverInitListZero	0..1	Container for Init Block 0. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order. All modules in this list are initialized before the post-build configuration has been loaded and the OS is initialized. Therefore, these modules may not use post-build configuration.
EcuMDriverRestartList	0..1	List of modules to be initialized.
EcuMSleepMode	1..256	These containers describe the configured sleep modes. The names of these containers specify the symbolic names of the different sleep modes.
EcuMWakeupSource	1..32	These containers describe the configured wakeup sources.

10.3.6 EcuMDefaultShutdownTarget

SWS Item	**ECUC_EcuM_00105 :**
Container Name	EcuMDefaultShutdownTarget
Description	This container describes the default shutdown target to be selected by EcuM. The actual shutdown target may be overridden by the EcuM_SelectShutdownTarget service.
Configuration Parameters	

SWS Item	**ECUC_EcuM_00107 :**	
Name	EcuMDefaultShutdownTarget	
Parent Container	EcuMDefaultShutdownTarget	
Description	This parameter describes the state part of the default shutdown target selected when the ECU comes out of reset. If EcuMShutdownTargetSleep is selected, the parameter EcuMDefaultSleepModeRef selects the specific sleep mode.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	EcuMShutdownTargetOff	Corresponds to ECUM_SHUTDOWN_TARGET_OFF in EcuM_ShutdownTargetType.
	EcuMShutdownTargetReset	Corresponds to ECUM_SHUTDOWN_TARGET_RESET in EcuM_ShutdownTargetType. This literal is only be applicable for EcuMFlex.
	EcuMShutdownTargetSleep	Corresponds to ECUM_SHUTDOWN_TARGET_SLEEP in EcuM_ShutdownTargetType.
Post-Build	true	

Variant Value			
Value Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00205 :**		
Name	EcuMDefaultResetModeRef		
Parent Container	EcuMDefaultShutdownTarget		
Description	If EcuMDefaultShutdownTarget is EcuMShutdownTargetReset, this parameter selects the default reset mode. Otherwise this parameter may be ignored.		
Multiplicity	0..1		
Type	Symbolic name reference to [EcuMResetMode]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00106 :**		
Name	EcuMDefaultSleepModeRef		
Parent Container	EcuMDefaultShutdownTarget		
Description	If EcuMDefaultShutdownTarget is EcuMShutdownTargetSleep, this parameter selects the default sleep mode. Otherwise this parameter may be ignored.		
Multiplicity	0..1		
Type	Symbolic name reference to [EcuMSleepMode]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

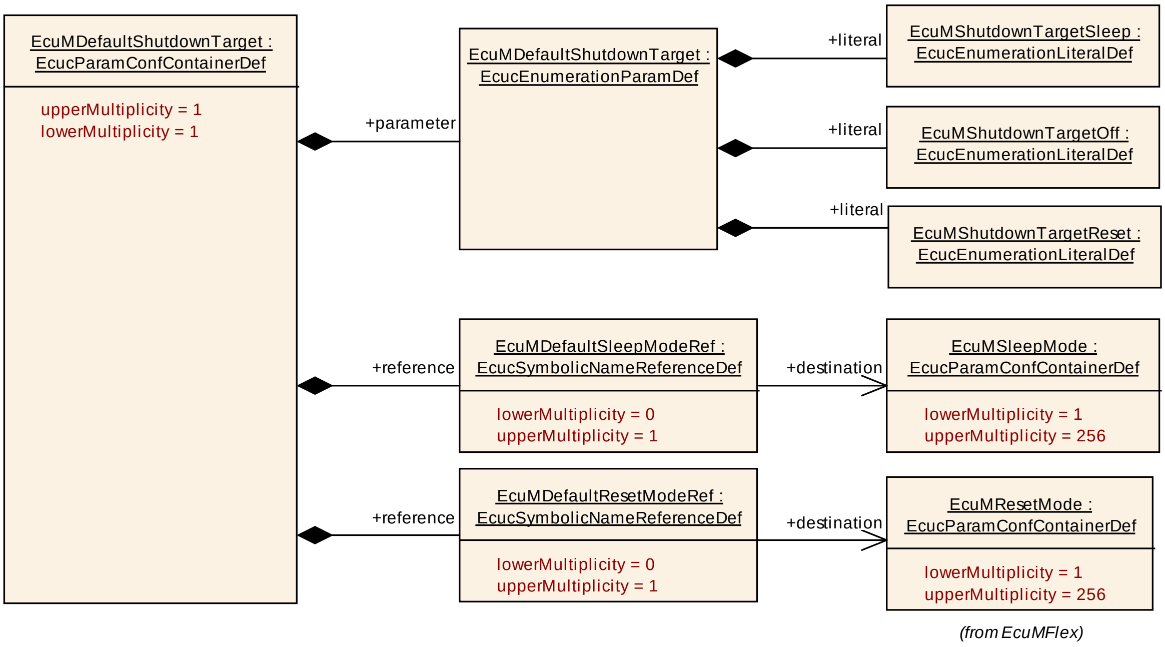


Figure 49 – EcuM Default Shutdown Target

10.3.7 EcuMDriverInitItem

SWS Item	**ECUC_EcuM_00110 :**
Container Name	EcuMDriverInitItem
Description	These containers describe the entries in a driver init list. **Attributes:** requiresIndex=true
Configuration Parameters	

SWS Item	**ECUC_EcuM_00224 :**		
Name	EcuMModuleParameter		
Parent Container	EcuMDriverInitItem		
Description	Definition of the function prototype and the parameter passed to the function.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	NULL_PTR	If NULL_PTR is configured EcuM expects as prototype: void <Mip>_<EcuMModuleService>(const <Mip>_ConfigType* <Mip>_Config). EcuM shall call this function with NULL Pointer: <Mip>_<EcuMModuleService>(NULL).	
	POSTBUILD_PTR	If POSTBUILD_PTR is configured EcuM expects as prototype: void <Mip>_<EcuMModuleService>(const <Mip>_ConfigType* <Mip>_Config). EcuM shall call this function with a valid pointer: <Mip>_<EcuMModuleService>(&<Mip>_Config[Predefinedvariant.shortName]).	
	VOID	If VOID is configured EcuM expects as prototype: void <Mip>_<EcuMModuleService>(void). EcuM will call <Mip>_<EcuMModuleService>().	
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: local

SWS Item	**ECUC_EcuM_00124 :**		
Name	EcuMModuleService		
Parent Container	EcuMDriverInitItem		
Description	The service to be called to initialize that module, e.g. Init, PreInit, Start etc. If nothing is defined "Init" is taken by default.		
Multiplicity	0..1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00223 :**		
Name	EcuMModuleRef		
Parent Container	EcuMDriverInitItem		
Description	Foreign reference to the configuration of a module instance which shall be initialized by EcuM		
Multiplicity	1		
Type	Foreign reference to [ECUC-MODULE-CONFIGURATION-VALUES]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

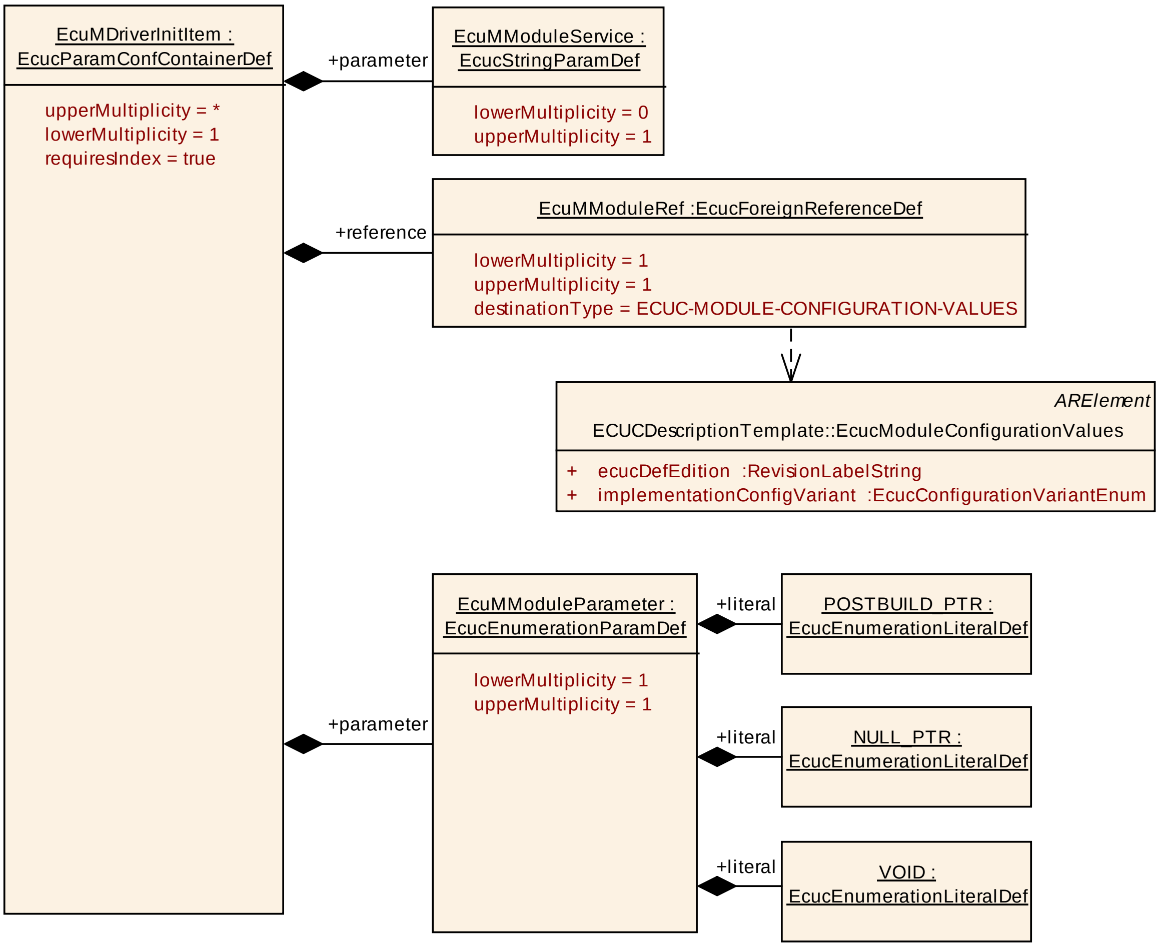

EcuMDriverInitItem : EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
requiresIndex = true
+parameter
EcuMModuleService : EcucStringParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
EcuMModuleRef :EcucForeignReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 1
destinationType = ECUC-MODULE-CONFIGURATION-VALUES
ARElement
ECUCDescriptionTemplate::EcucModuleConfigurationValues
+ ecucDefEdition :RevisionLabelString
+ implementationConfigVariant :EcucConfigurationVariantEnum
+parameter
EcuMModuleParameter : EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
+literal
POSTBUILD_PTR : EcucEnumerationLiteralDef
+literal
NULL_PTR : EcucEnumerationLiteralDef
+literal
VOID : EcucEnumerationLiteralDef

Figure 50 – EcuM Driver Init Item

10.3.8 EcuMDriverInitListZero

SWS Item	**ECUC_EcuM_00114 :**
Container Name	EcuMDriverInitListZero
Description	Container for Init Block 0. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order. All modules in this list are initialized before the post-build configuration has been loaded and the OS is initialized. Therefore, these modules may not use post-build configuration.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	These containers describe the entries in a driver init list.

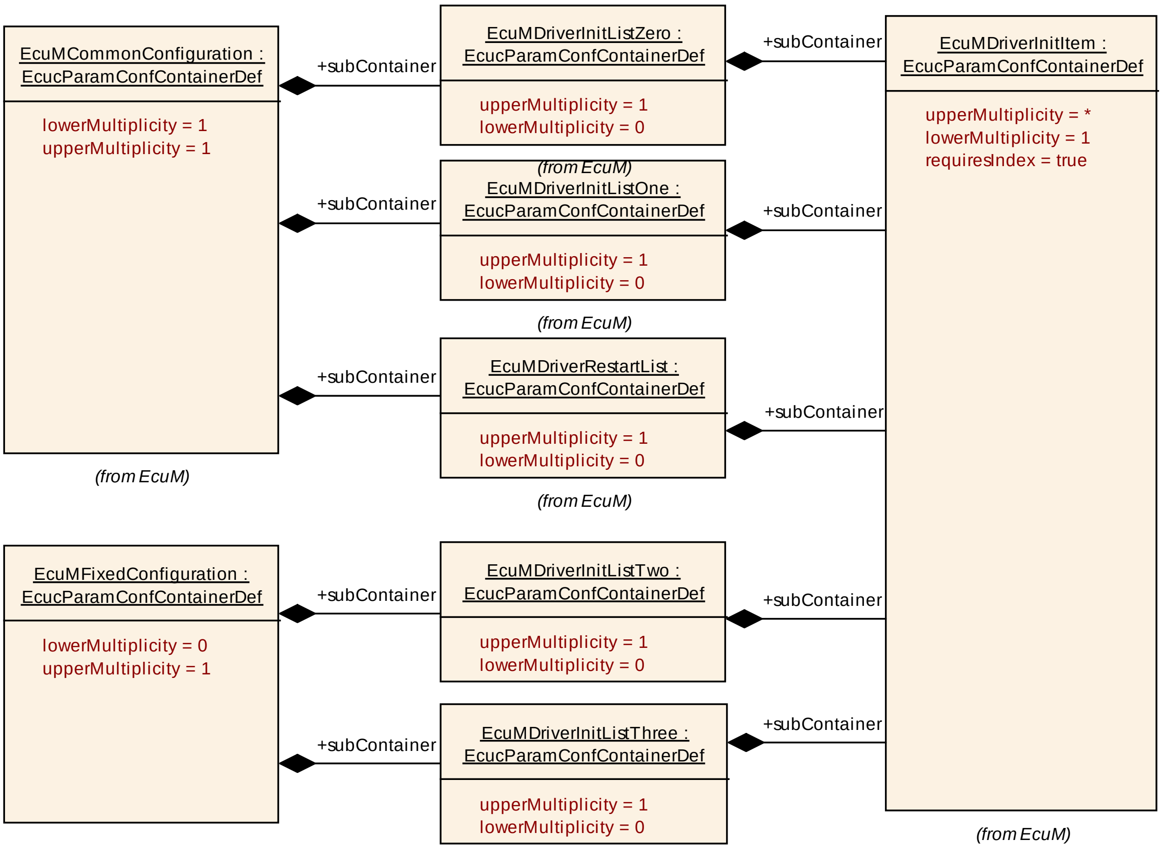


Figure 51 – EcuM Fixed Init Lists

10.3.9 EcuMDriverInitListOne

SWS Item	**ECUC_EcuM_00111 :**
Container Name	EcuMDriverInitListOne
Description	Container for Init Block I. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order. All modules in this list are initialized before the OS is started and so these modules require no OS support.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	These containers describe the entries in a driver init list.

10.3.10 EcuMDriverInitListTwo

SWS Item	**ECUC_EcuM_00113 : (Obsolete)**
Container Name	EcuMDriverInitListTwo
Description	Container for Init Block II. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order.

	All modules in this list are initialized after the OS is started and so these modules may use OS support. These modules may not rely on the Nvram ReadAll job to have provided all data. **Tags:** atp.Status=obsolete
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	These containers describe the entries in a driver init list.

10.3.11 EcuMDriverInitListThree

SWS Item	**ECUC_EcuM_00112 : (Obsolete)**
Container Name	EcuMDriverInitListThree
Description	Container for Init Block III. This container holds a list of modules to be initialized. Each module in the list will be called for initialization in the list order. All modules in this list are initialized after the OS is started and so these modules may use OS support. These modules may also rely on the Nvram ReadAll job to have provided all data. **Tags:** atp.Status=obsolete
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	These containers describe the entries in a driver init list.

10.3.12 EcuMSleepMode

SWS Item	**ECUC_EcuM_00131 :**
Container Name	EcuMSleepMode
Description	These containers describe the configured sleep modes. The names of these containers specify the symbolic names of the different sleep modes.
Configuration Parameters	

SWS Item	**ECUC_EcuM_00132 :**		
Name	EcuMSleepModeId		
Parent Container	EcuMSleepMode		
Description	This ID identifies this sleep mode in services like EcuM_SelectShutdownTarget.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_EcuM_00136 :**		
Name	EcuMSleepModeSuspend		
Parent Container	EcuMSleepMode		
Description	Flag, which is set true, if the CPU is suspended, halted, or powered off in the sleep mode. If the CPU keeps running in this sleep mode, then this flag must be set to false.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00133 :**		
Name	EcuMSleepModeMcuModeRef		
Parent Container	EcuMSleepMode		
Description	This parameter is a reference to the corresponding MCU mode for this sleep mode.		
Multiplicity	1		
Type	Symbolic name reference to [McuModeSettingConf]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00152 :**		
Name	EcuMWakeupSourceMask		
Parent Container	EcuMSleepMode		
Description	These parameters are references to the wakeup sources that shall be enabled for this sleep mode.		
Multiplicity	1..*		
Type	Symbolic name reference to [EcuMWakeupSource]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

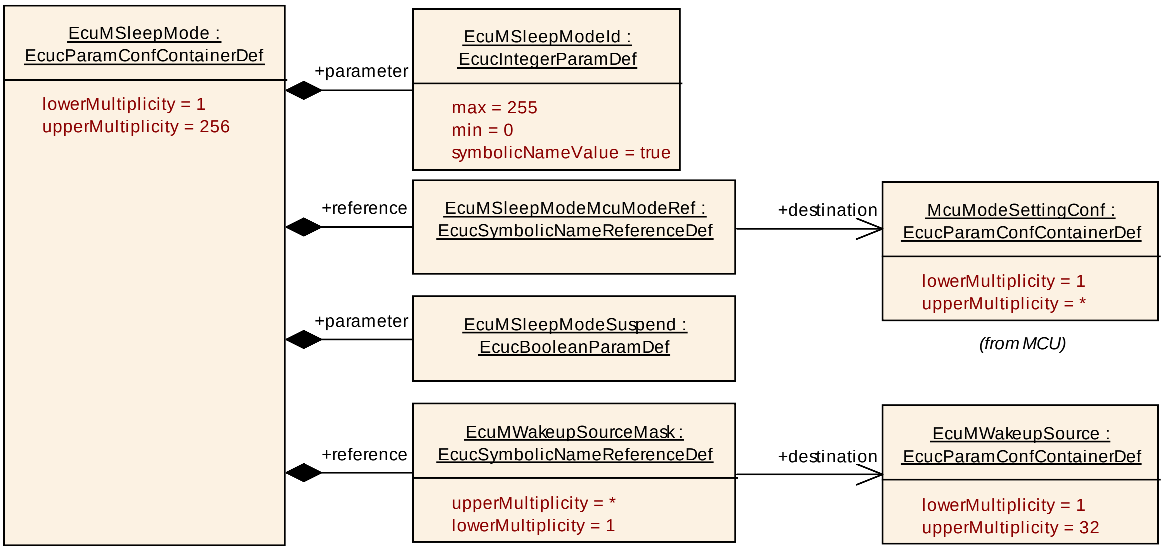

EcuMSleepMode : EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 256
+parameter
EcuMSleepModeId : EcucIntegerParamDef
max = 255
min = 0
symbolicNameValue = true
+reference
EcuMSleepModeMcuModeRef : EcucSymbolicNameReferenceDef
+destination
McuModeSettingConf : EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = *
(from MCU)
+parameter
EcuMSleepModeSuspend : EcucBooleanParamDef
+reference
EcuMWakeupSourceMask : EcucSymbolicNameReferenceDef
upperMultiplicity = *
lowerMultiplicity = 1
+destination
EcuMWakeupSource : EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 32

Figure 52 – EcuM Sleep Mode

10.3.13 EcuMWakeupSource

SWS Item	**ECUC_EcuM_00150 :**
Container Name	EcuMWakeupSource
Description	These containers describe the configured wakeup sources.
Configuration Parameters	

SWS Item	**ECUC_EcuM_00208 :**		
Name	EcuMCheckWakeupTimeout		
Parent Container	EcuMWakeupSource		
Description	This Parameter is the initial Value for the Time of the EcuM to delay shut down of the ECU if the check of the Wakeup Source is done asynchronously (CheckWakeupTimer). The unit is in seconds.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. 10]		
Default value	0		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00148 :**
Name	EcuMValidationTimeout
Parent Container	EcuMWakeupSource
Description	The validation timeout (period for which the ECU State Manager will wait for the validation of a wakeup event) can be defined for each wakeup source independently. The timeout is specified in seconds.

	When the timeout is not instantiated, there is no validation routine and the ECU Manager shall not validate the wakeup source.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00151 :**		
Name	EcuMWakeupSourceId		
Parent Container	EcuMWakeupSource		
Description	This parameter defines the identifier of this wakeup source.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 31		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_EcuM_00153 :**		
Name	EcuMWakeupSourcePolling		
Parent Container	EcuMWakeupSource		
Description	This parameter describes if the wakeup source needs polling.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00101 :**
Name	EcuMComMChannelRef
Parent Container	EcuMWakeupSource
Description	This parameter is a reference to a Network (channel) defined in the Communication Manager. No reference indicates that the wakeup source is not a communication channel.
Multiplicity	0..1
Type	Symbolic name reference to [ComMChannel]
Post-Build Variant Multiplicity	false
Post-Build Variant Value	false

Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00228 :**		
Name	EcuMComMPNCRef		
Parent Container	EcuMWakeupSource		
Description	This is a reference to a one or more PNC's defined in the Communication Manager. No reference indicates that the wakeup source is not assigned to a partial network.		
Multiplicity	0..*		
Type	Symbolic name reference to [ComMPnc]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00128 :**		
Name	EcuMResetReasonRef		
Parent Container	EcuMWakeupSource		
Description	This parameter describes the mapping of reset reasons detected by the MCU driver into wakeup sources.		
Multiplicity	0..*		
Type	Symbolic name reference to [McuResetReasonConf]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Value Configuration Class	**Pre-compile time**	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

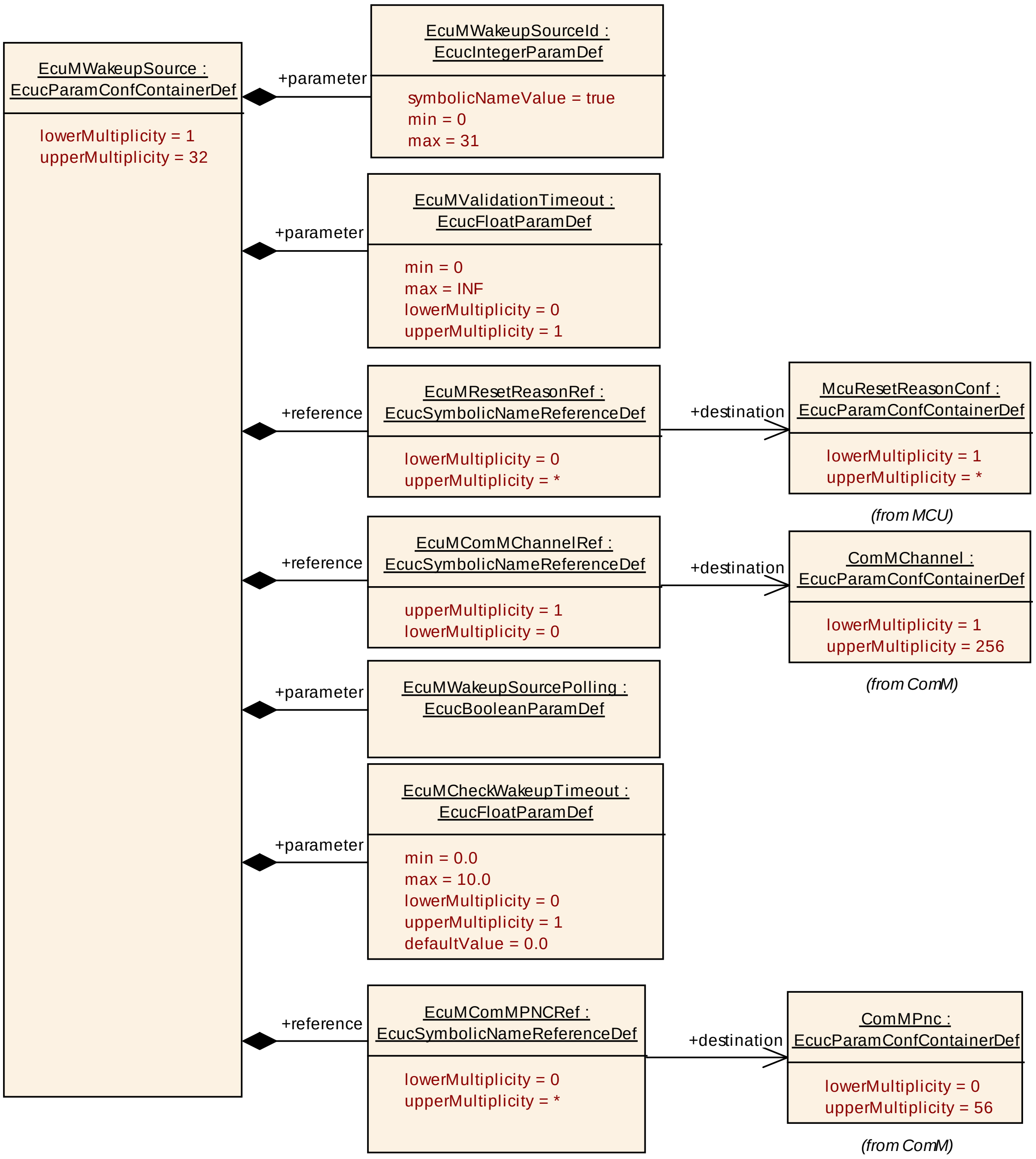


Figure 53 – EcuM Wakeup Source

10.3.14 EcuMFixedUserConfig

SWS Item	**ECUC_EcuM_00147 : (Obsolete)**
Container Name	EcuMFixedUserConfig
Description	These containers describe the identifiers that are needed to refer to a software component or another appropriate entity in the system which is designated to request the RUN state. Application requestors refer to entities above RTE, system requestors to entities below RTE (e.g. Communication Manager). **Tags:** atp.Status=obsolete
Configuration Parameters	

SWS Item	**ECUC_EcuM_00202 : (Obsolete)**		
Name	EcuMFixedUser		
Parent Container	EcuMFixedUserConfig		
Description	Parameter used to identify one user. **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

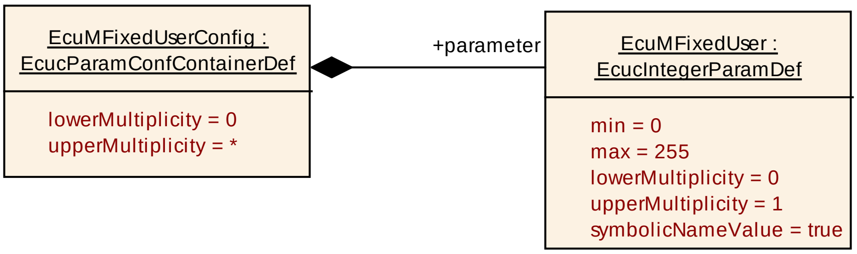


Figure 54 – EcuM Fixed User Config

10.3.15 EcuMDriverRestartList

SWS Item	**ECUC_EcuM_00115 :**
Container Name	EcuMDriverRestartList
Description	List of modules to be initialized.
Configuration Parameters	

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
EcuMDriverInitItem	1..*	These containers describe the entries in a driver init list.

10.3.16 EcuMTTII

SWS Item	**ECUC_EcuM_00143 : (Obsolete)**
Container Name	EcuMTTII
Description	These containers describe the structures and the following configuration items describe its elements. These structures are concatenated to build a list as indicated by Figure 27 - Configuration Container Diagram. The list must contain at least one element when ECUM_TTII_ENABLED is set to true. **Tags:** atp.Status=obsolete
Configuration Parameters	

SWS Item	**ECUC_EcuM_00109 : (Obsolete)**		
Name	EcuMDivisor		
Parent Container	EcuMTTII		
Description	This parameter defines the divisor preload value. **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00135 : (Obsolete)**		
Name	EcuMSleepModeRef		
Parent Container	EcuMTTII		
Description	This configuration parameter is a reference to a configured sleep mode that is used for TTII. **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	Symbolic name reference to [EcuMSleepMode]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_EcuM_00141 : (Obsolete)**		
Name	EcuMSuccessorRef		
Parent Container	EcuMTTII		
Description	This parameter is a reference to the next sleep mode in the TTII protocol. **Tags:** atp.Status=obsolete		
Multiplicity	0..1		
Type	Symbolic name reference to [EcuMSleepMode]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

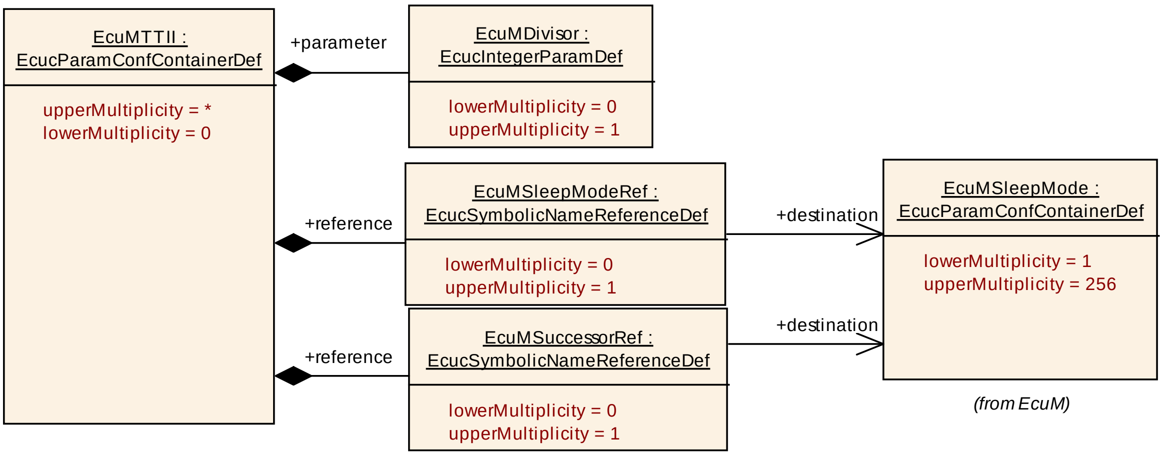


Figure 55 – EcuM TTII

10.4 Checking Configuration Consistency

10.4.1 The Necessity for Checking Configuration Consistency

In an AUTOSAR ECU several configuration parameters are set and put into the ECU at different times. Pre-compile parameters are set, put into the generated source code and compiled into object code. When the source code has been compiled, link-time parameters are set, compiled, and linked with the previously configured object code into an image that is put into the ECU. Finally, post-build parameters are set, compiled, linked, and put into the ECU at a different time. All these parameters must match to obtain a stable ECU.

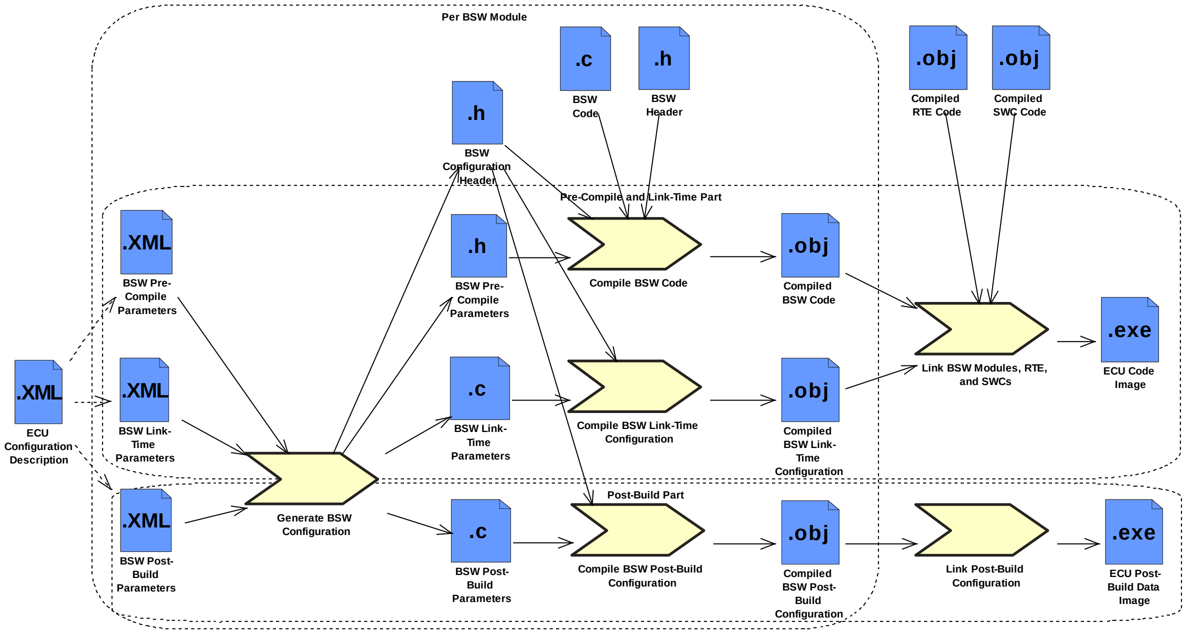

Per BSW Module
.c
BSW Code
.h
BSW Header
.obj
Compiled RTE Code
.obj
Compiled SWC Code
.h
BSW Configuration Header
Pre-Compile and Link-Time Part
.XML
BSW Pre-Compile Parameters
.h
BSW Pre-Compile Parameters
Compile BSW Code
.obj
Compiled BSW Code
Link BSW Modules, RTE, and SWCs
.exe
ECU Code Image
.XML
ECU Configuration Description
.XML
BSW Link-Time Parameters
.c
BSW Link-Time Parameters
Compile BSW Link-Time Configuration
.obj
Compiled BSW Link-Time Configuration
Generate BSW Configuration
Post-Build Part
.XML
BSW Post-Build Parameters
.c
BSW Post-Build Parameters
Compile BSW Post-Build Configuration
.obj
Compiled BSW Post-Build Configuration
Link Post-Build Configuration
.exe
ECU Post-Build Data Image

Figure 56 – BSW Configuration Steps

Checking consistency of parameters at configuration time can be done within the configuration tool itself. At compilation time, parameter errors may be detected by the compiler and at link time, the linker may find additional errors. Unfortunately, finding configuration errors in post-build parameters is very difficult. This can only be achieved at run-time by checking that

- the pre-compile and link-time parameter settings used when compiling the code

are exactly the same as

- the pre-compile and link-time parameter settings used when configuring and compiling the post-build parameters.

This can only be done at run-time.

[SWS_EcuM_02796] ⌈To avoid multiple checks scattered over the different BSW modules, the ECU State Manager Fixed module shall check the consistency once before initializing the first BSW module. This also implies that the ECU State

Manager Fixed module must not only check the consistency of its own parameters but of all post-build configurable BSW modules. ⌋ ()

The ECU configuration tool shall compute a hash value over all pre-compile and link-time configuration parameters of all BSW modules and put that into the link-time configuration parameter *ECUM_CONFIGCONSISTENCY_HASH.* The hash value is necessary for two reasons. First, the pre-compile and link-time parameters are not accessible anymore at run-time. Second, the check must be very efficient at run-time. Comparing hundreds of parameters would cause an unacceptable delay in the ECU startup process.

[SWS_EcuM_02798] ⌈The ECU State Manager Fixed module configuration tool shall put the current value of the configuration parameter *ECUM_CONFIGCONSISTENCY_HASH* into a field in the EcuM_ConfigType structure, which contains the root of all post-build configuration parameters. The ECU State Manager Fixed module shall check in EcuM_Init that the field in the structure is equal to the value of *ECUM_CONFIGCONSISTENCY_HASH.* ⌋ ()

By computing both hash values at configuration time and comparing them at run-time the code of the ECU State Manager Fixed module becomes very efficient and independent of a certain hash computation algorithm. This allows for the use of complex hash computation algorithms, e.g. cryptographically strong hash functions.

Note that the same hash algorithm can be used to produce the value for the post-build configuration identifier in the EcuM_ConfigType structure. Then the hash algorithm is applied to the post-build parameters instead of the pre-compile and link-time parameters.

The used hash computation algorithm shall always produce the same hash value for the same set of configuration data, regardless of the order of configuration parameters in the XML files.

10.4.2 Example Hash Computation Algorithm

Note: This chapter is non-normative. It describes one possible way of computing hash values.

A simple CRC over the values of configuration parameters will not serve as a good hash algorithm. It only detects global changes, e.g. one parameter has changed from 1 to 2. But if another parameter changed from 2 to 1, the CRC might stay the same.

Additionally, not only the values of the configuration parameters but also their names must be taken into account in the hash algorithm. One possibility is to build a text file that contains the names of the configuration parameters and containers, separate them from the values using a delimiter, e.g. a colon, and putting each parameter as a line into a text file. For the above Watchdog Manager example only one parameter will be included because only this one is pre-compile configured. The text file would then contain the line:

```
/WdgMConfiguration/WdgM_Trigger/WDGM_NUMBER_OF_WATCHDOG_INSTANCES:2
```

If there are multiple containers of the same type, each container name can be appended with a number, e.g. “_0”, “_1” and so on.

To make the hash value independent of the order in which the parameters are written into the text file, the lines in the file must now be sorted lexicographically.

Finally, a cryptographically strong hash function, e.g. MD5, can be run on the text file to produce the hash value. These hash functions produce completely different hash values for slightly changed input files.