Document Title	# Specification of ECU State Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	78

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R20-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2020-11-30	R20-11	AUTOSAR Release Management	• Corrected broken chapter structure • EcuM_UserType handling improved • Minor content changes, clarifications (multi-core, configuration, values)
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Reworked BswM interface through EcuM_GoDownHaltPoll • Removed EcuM fixed version references
2017-12-08	4.3.1	AUTOSAR Release Management	• Adapt API Can_CheckWakeup • Removed ConfigPtr parameter • Removed Default error • Removed unused DIO driver • EcuM AUTOSAR service configure on service partition only
2016-11-30	4.3.0	AUTOSAR Release Management	• Partial Network Cluster Support • Initialization BSW scheduler slipt • Added a driver initialization list • Removed EcuM_StateType

2015-07-31	4.2.2	AUTOSAR Release Management	• Reworked slave core poll sequence • Reviewed multicore shutdown synchronization • Reclassified error types • Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Added switch configuration • Defined initialization order for InitListZero/InitListOne • Definition of the name pattern of c-init-data struct corrected • Type conflicts solved • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• EcuM errors reworked • Inconsistencies between APIs and Interfaces resolved • Type conflicts solved • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Added API table for service interfaces • Fixed traceability topics • General clean-up of requirements (reviewed different interfaces, operations, descriptions and figures). • Editorial changes

2013-03-15	4.1.1	AUTOSAR Administration	• Specified reset mode to use in case of pending wakeup events during shutdown • Added callout for Reset Loop Detection • Extended specification of parameter "time" of function " EcuM_GetMostRecentShutdown" • Improved configuration description • Added new APIs to enable asynchronous Trcv handling for CAN/FR Wakeup • Adaption of EcuM Flex to support BSW modules distributed over multiple partitions • Reclassified which Production Errors are Extended Production Errors • Added possible error to operations of Client/Server-Interfaces, where no errors where defined • Enhancement of configuration to initialize BSW modules by the EcuM Flex
2011-12-22	4.0.3	AUTOSAR Administration	• Fixed interoperability problems between EcuM and BswM • Terminology of ECU State Manager Flexible more consistently described • Modification of sleep sequences to minimize misses of wakeup interrupts
2010-09-30	3.1.5	AUTOSAR Administration	• Updated pseudo code for AUTOSAR Services • Update startup procedure for multi core systems
2010-02-02	3.1.4	AUTOSAR Administration	• Removed state machine to accommodate mode-dependent scheduling • Added Multi-Core support • Added Alarm Clock feature • Revised disclaimer
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised

2007-12-21	3.0.1	AUTOSAR Administration	• Fixed Wakeup mechanisms • Included optional triggering of Watchdog Manager during Startup, Shutdown, and Sleep • Extended startup sequence to have more flexibility and to directly initialize all other BSW modules • Generated APIs from BSW UML model • Generated configuration from Meta Model • Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• Corrected startup flow and wakeup concept. • Added specification for AUTOSAR ports. • Modified configuration for compliance with variant management. • Added new API services. • Legal disclaimer revised • Release Notes added • " Advice for users" revised • " Revision Information" added
2006-05-16	2.0	AUTOSAR Administration	• Initial Release

Disclaimer

Table of Contents

Known Limitations

- The ECU Manager module interfaces must be specified as reentrant in the Multi-Core context.

1 Introduction and Functional Overview

The ECU Manager module (as specified in this document) is a basic software module (see [1]) that manages common aspects of ECU states. Specifically, the ECU Manager module:

- Initializes and de-initializes the OS, the SchM and the BswM as well as some basic software driver modules.
- configures the ECU for SLEEP and SHUTDOWN when requested.
- manages all wakeup events on the ECU

The ECU Manager module provides the wakeup validation protocol to distinguish 'real' wakeup events from 'erratic' ones.

Furthermore:

- Partial or fast startup where he ECU starts up with limited capabilities and later, as determined by the application, continues startup step by step.
- Interleaved startup where the ECU starts minimally and then starts the RTE to execute functionality in SW-Cs as soon as possible. It then continues to start further BSW and SW-Cs, thus interleaving BSW and application functionality..
- Multiple operational states where the ECU has more than one RUN state. This, among other things, refines the notion of a spectrum of SLEEP states to RUN states. There can now be a continuum of operational states spanning from the classic RUN (fully operational) to the deepest SLEEP (processor halted).
- Multi-Core ECUs: STARTUP, SHUTDOWN, SLEEP and WAKEUP are coordinated on all cores of the ECU.

Flexible ECU management employs the generic mode management facilities provided by the following modules:

- RTE and BSW Scheduler module [2] are now amalgamated into one module: This module supports freely configurable BSW and application modes and their mode-switching facilities.
- BSW Mode Manager module [3]: This module implements configurable rules and action lists to evaluate the conditions for switching ECU modes and to implement the necessary actions to do so.

Thus with Flexible ECU Management, most ECU states are no longer implemented in the ECU Manager module itself. In general, the ECU Manager module takes over control when the generic mode management facilities are unavailable in:

- Early STARTUP phases,
- Late SHUTDOWN phases,
- SLEEP phases where the facilities are locked out by the scheduler.

During the UP Phase of the ECU Manager module the BSW Mode Manager is responsible for further actions. Whereas, the ECU Manager module arbitrates RUN and POST_RUN Requests from SW-Cs and notifies BswM about the status of the modes.

1.1 Backwards Compatibility to Previous ECU Manager Module Versions

Flexible ECU management is backward compatible to previous ECU Manager versions if it is configured accordingly.

For more information about a configuration in respect to compatibility see the "Guide to Mode Management" [4].

2 Definitions and Abbreviations

This chapter defines terms that are of special significance to the ECU Manager and the acronyms of related modules.

2.1 Definitions

Term	Description
Callback	Refer to the Glossary [5]
Callout	'Callouts' are function stubs that the system designer can replace with code, usually at configuration time, to add functionality to the ECU Manager module. Callouts are separated into two classes. One class provides mandatory ECU Manager module functionality and serves as a hardware abstraction layer. The other class provides optional functionality.
Integration Code	Refer to the Glossary [5]
Mode	A Mode is a certain set of states of the various state machines (not only of the ECU Manager) that are running in the vehicle and are relevant to a particular entity, an application or the whole vehicle
Passive Wakeup	A wakeup caused from an attached bus rather than an internal event like a timer or sensor activity.
Phase	A logical or temporal assembly of ECU Manager's actions and events, e.g. STARTUP, UP, SHUTDOWN, SLEEP, ... Phases can consist of Sub-Phases which are often called Sequences if they above all exist to group sequences of executed actions into logical units. Phases in this context are not the phases of the AUTOSAR Methodology.
Shutdown Target	The ECU must be shut down before it is put to sleep, before it is powered off or before it is reset. SLEEP, OFF, and RESET are therefore valid shutdown targets. By selecting a shutdown target, an application can communicate its wishes for the ECU behavior after the next shutdown to the ECU Manager module.
State	States are internal to their respective BSW component and thus not visible to the application. So they are only used by the BSW's internal state machine. The States inside the ECU Manager build the phases and therefore handle the modes.
Wakeup Event	A physical event which causes a wakeup. A CAN message or a toggling IO line can be wakeup events. Similarly, the internal SW representation, e.g. an interrupt, may also be called a wakeup event.
Wakeup Reason	The wakeup reason is the wakeup event that is the actual cause of the last wakeup.
Wakeup Source	The peripheral or ECU component which deals with wakeup events is called a wakeup source.

2.2 Abbreviations

Abbreviation	Description
BswM	Basic Software Mode Manager
Dem	Diagnostic Event Manager
Det	Default Error Tracer
EcuM	ECU Manager
Gpt	General Purpose Timer
Icu	Input Capture Unit
ISR	Interrupt Service Routine
Mcu	Microcontroller Unit
NVRAM	Non-volatile random access memory
Os	Operating System
Rte	Runtime Environment
VFB	Virtual Function Bus

3 Related documentation

3.1 Input documents & related standards and norms

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[2] Specification of RTE Software
AUTOSAR_SWS_RTE

[3] Specification of Basic Software Mode Manager
AUTOSAR_SWS_BSWModeManager

[4] Guide to Mode Management
AUTOSAR_EXP_ModeManagementGuide

[5] Glossary
AUTOSAR_TR_Glossary

[6] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[7] Virtual Functional Bus
AUTOSAR_EXP_VFB

[8] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[9] Requirements on Mode Management
AUTOSAR_SRS_ModeManagement

[10] Specification of MCU Driver
AUTOSAR_SWS_MCUDriver

[11] Specification of CAN Transceiver Driver
AUTOSAR_SWS_CANTransceiverDriver

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules (see [6]), which is also valid for ECU State Manager. Thus, the specification [6] shall be considered as additional and required specification for ECU State Manager.

4 Constraints and Assumptions

4.1 Limitations

ECUs cannot always be switched off (i.e. zero power consumption).

Rationale: The shutdown target OFF can only be reached using ECU special hardware (e.g. a power hold circuit). If this hardware is not available, this specification proposes to issue a reset instead. Other default behaviors are permissible, however.

4.2 Hardware Requirements

In this section, the term "EcuM RAM" refers to a block of RAM reserved for use by the ECU Manager module.

The EcuM RAM shall keep contents of vital data while the ECU clock is switched off.

Rationale: This requirement is needed to implement sleep states as required in section 7.5 SLEEP Phase.

The EcuM RAM shall provide a no-init area that keeps contents over a reset cycle.

The no-init area of the EcuM RAM (see EcuM2869) shall only be initialized on a power on event (clamp 30).

The system designer is responsible for establishing an initialization strategy for the no init area of the ECU RAM.

4.3 Applicability to car domains

The ECU Manager 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 must fulfill to collaborate correctly with the ECU Manager module.

If data pointers are passed to a BSW module, the address needs to point to a location in the shared part of the memory space.

5.1 SPAL Modules

5.1.1 MCU Driver

The MCU Driver is the first basic software module initialized by the ECU Manager module. When MCU_Init returns (see [SWS_EcuM_02858]), the MCU module and the MCU Driver module are not necessarily fully initialized, however. Additional MCU module specific steps may be needed to complete the initialization. The ECU Manager module provides two callout where this additional code can be placed. Refer to section 7.3.2 Activities in StartPreOS Sequence for details.

5.1.2 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 Manager module) and on the other hand that the called module must be initialized before the calling module is initialized.

The system designer is responsible for defining the initialization order at configuration time in `EcuMDriverInitListZero`, `EcuMDriverInitListOne`, `EcuMDriver-RestartList` and in `EcuMDriverInitListBswM`.

5.2 Peripherals with Wakeup Capability

Wakeup sources must be handled and encapsulated by drivers.

These drivers must follow the protocols and requirements presented in this document to ensure a seamless integration into the AUTOSAR BSW. Basically, the protocol is as follows:

The driver must invoke EcuM_SetWakeupEvent (see [SWS_EcuM_02826]) to notify the ECU Manager module that a pending wakeup event has been detected. The driver must not only invoke EcuM_SetWakeupEvent while the ECU is waiting for a wakeup event during a sleep phase but also during the driver initialization phase and during normal operation when EcuM_MainFunction is running.

The driver must provide an explicit function to put the wakeup source to sleep. This function shall put the wakeup source into an energy saving and inert operation mode and rearm the wakeup notification mechanism.

If the wakeup source is capable of generating spurious events[1] then either

- the driver or
- the software stack consuming the driver or
- another appropriate BSW module

must either provide a validation callout for the wakeup event or call the ECU Manager module's validation function. If validation is not necessary, then this requirement is not applicable for the corresponding wakeup source.

5.3 Operating System

The ECU Manager module starts the AUTOSAR OS and also shuts it down. The ECU Manager module defines the protocol how control is handled before the OS is started and how control is handled after the OS has been shut down.

5.4 BSW Scheduler

The ECU Manager module initializes the BSW Scheduler and the ECU Manager module also contains EcuM_MainFunction (see [SWS_EcuM_02837]) which is scheduled to periodically evaluate wakeup requests and update the Alarm Clock.

5.5 BSW Mode Manager

ECU states are generally implemented as AUTOSAR modes and the BSW Mode Manager is responsible for monitoring changes in the ECU and affecting the corresponding changes to the ECU state machine as appropriate. Refer to the Specification of the Virtual Function Bus [7] for a discussion of AUTOSAR mode management and to the Guide to Mode Management [4] for ECU state machine implementation details and for guidelines about how to configure the BSW Mode Manager to implement the ECU state machine

The BSW Mode Manager can only manage the ECU state machine after mode management is operational - that is, after the SchM has been initialized and until the SchM

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

is de-initialised or halted. The ECU Manager module takes control of the ECU when the BSW Mode manager is not operational.

The ECU Manager module therefore takes control immediately after the ECU has booted and relegates control to the BSW Mode Manager after initializing the SchM and the BswM.

The BswM passes control of the ECU back to the ECU Manager module to lock the operating system and handle wakeup events.

The BswM also passes control back to the ECU Manager immediately before the OS is stopped on shutdown.

When wakeup sources are being validated, the ECU Manager module indicates wakeup source state changes to the BswM through mode switch requests.

5.6 Software Components

The ECU Manager module handles the following ECU-wide properties:

- Shutdown targets.

This specification assumes that SW-Cs set these properties (through AUTOSAR ports), typically by some ECU specific part of the SW-C. The ECU Manager does not prevent a SW-C from overrighting settings made by SW-Cs. The policy must be defined at a higher level.

The following measures might help to resolve this issue.

- The SW-C Template may contain a field to indicate whether the SW-C sets the shutdown target.
- The generation tool may only allow configurations that have one SW-C accessing the shutdown target.

5.7 File Structure

5.7.1 Code file structure

This specification does not define the code file structure completely.

[SWS_EcuM_02990] ⌈The ECU Manager module implementation shall provide a single `EcuM_Callout_Stubs.c` file which contains the stubs of the callouts realized in this implementation.⌋*()*

See also section 8.5 Callout Definitions for a list of the callouts that could possibly be implemented.

Whether `EcuM_Callout_Stubs.c` can be edited manually or is composed only of other generated files depends on the implementation.

5.7.2 Header file structure

Also refer to chapter 8.7 Expected Interfaces for dependencies to other modules.

6 Requirements Tracing

The following tables reference the requirements specified in [8] and [9] and links to the fulfillment of these. Please note that if column "Satisfied by" is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00005]	Modules of the μC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	[SWS_EcuM_NA_00000]
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_EcuM_NA_00000]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_EcuM_02811]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_EcuM_NA_00000]
[SRS_BSW_00160]	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	[SWS_EcuM_NA_00000]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_EcuM_NA_00000]
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_EcuM_NA_00000]
[SRS_BSW_00164]	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	[SWS_EcuM_NA_00000]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_EcuM_NA_00000]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_EcuM_NA_00000]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_EcuM_02836]
[SRS_BSW_00307]	Global variables naming convention	[SWS_EcuM_NA_00000]
[SRS_BSW_00308]	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	[SWS_EcuM_NA_00000]
[SRS_BSW_00309]	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	[SWS_EcuM_NA_00000]
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_EcuM_NA_00000]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_EcuM_03009]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_EcuM_NA_00000]
[SRS_BSW_00327]	Error values naming convention	[SWS_EcuM_04032]
[SRS_BSW_00330]	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	[SWS_EcuM_NA_00000]
[SRS_BSW_00331]	All Basic Software Modules shall strictly separate error and status information	[SWS_EcuM_91005]
[SRS_BSW_00333]	For each callback function it shall be specified if it is called from interrupt context or not	[SWS_EcuM_02171] [SWS_EcuM_02345]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_EcuM_NA_00000]
[SRS_BSW_00337]	Classification of development errors	[SWS_EcuM_04032]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_EcuM_02987]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_EcuM_NA_00000]
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_EcuM_NA_00000]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_EcuM_NA_00000]
[SRS_BSW_00350]	All AUTOSAR Basic Software Modules shall allow the enabling/ disabling of detection and reporting of development errors.	[SWS_EcuM_04032]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_EcuM_NA_00000]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_EcuM_02811]
[SRS_BSW_00359]	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	[SWS_EcuM_02826] [SWS_EcuM_02829]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_EcuM_02826] [SWS_EcuM_02829]
[SRS_BSW_00361]	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	[SWS_EcuM_NA_00000]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_EcuM_02837]
[SRS_BSW_00385]	List possible error notifications	[SWS_EcuM_04032]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_EcuM_NA_00000]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_EcuM_02813]
[SRS_BSW_00410]	Compiler switches shall have defined values	[SWS_EcuM_NA_00000]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_EcuM_02813]
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_EcuM_NA_00000]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_EcuM_02811]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_EcuM_NA_00000]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_EcuM_02559]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	[SWS_EcuM_NA_00000]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the DEM	[SWS_EcuM_NA_00000]
[SRS_BSW_00425]	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	[SWS_EcuM_02837]
[SRS_BSW_00426]	BSW Modules shall ensure data consistency of data which is shared between BSW modules	[SWS_EcuM_NA_00000]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_EcuM_NA_00000]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_EcuM_NA_00000]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_EcuM_NA_00000]
[SRS_BSW_00439]	Enable BSW modules to handle interrupts	[SWS_EcuM_NA_00000]

▽

△

Requirement	Description	Satisfied by
[SRS_BSW_00440]	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	[SWS_EcuM_02826] [SWS_EcuM_02829]
[SRS_BSW_00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	[SWS_EcuM_NA_00000]
[SRS_BSW_00450]	A Main function of a un-initialized module shall return immediately	[SWS_EcuM_NA_00000]
[SRS_BSW_00453]	BSW Modules shall be harmonized	[SWS_EcuM_NA_00000]
[SRS_ModeMgm_-09072]	ECU shutdown shall be forced	[SWS_EcuM_03022]
[SRS_ModeMgm_-09098]	Storing the wake-up reasons shall be available	[SWS_EcuM_02826]
[SRS_ModeMgm_-09100]	Selection of wake-up sources shall be configurable	[SWS_EcuM_02389]
[SRS_ModeMgm_-09104]	ECU State Manager shall take over control after OS shutdown	[SWS_EcuM_02952] [SWS_EcuM_02953]
[SRS_ModeMgm_-09113]	Initialization of Basic Software modules shall be done	[SWS_EcuM_02932]
[SRS_ModeMgm_-09114]	Starting/invoking the shutdown process shall be provided	[SWS_EcuM_00624] [SWS_EcuM_02185] [SWS_EcuM_02585] [SWS_EcuM_02812] [SWS_EcuM_02822]
[SRS_ModeMgm_-09116]	Requesting and releasing the RUN state shall be provided	[SWS_EcuM_04115] [SWS_EcuM_04116] [SWS_EcuM_04117] [SWS_EcuM_04118] [SWS_EcuM_04119] [SWS_EcuM_04120] [SWS_EcuM_04121] [SWS_EcuM_04123] [SWS_EcuM_04125] [SWS_EcuM_04126] [SWS_EcuM_04127] [SWS_EcuM_04128] [SWS_EcuM_04129] [SWS_EcuM_04130] [SWS_EcuM_04132]
[SRS_ModeMgm_-09126]	An API for querying the wake-up reason shall be provided	[SWS_EcuM_02827] [SWS_EcuM_02828] [SWS_EcuM_02830] [SWS_EcuM_02831]
[SRS_ModeMgm_-09127]	The ECU State Manager shall de-initialize Basic Software modules where appropriate during the shutdown process	[SWS_EcuM_03021]
[SRS_ModeMgm_-09128]	Several shutdown targets shall be supported	[SWS_EcuM_02822] [SWS_EcuM_02824] [SWS_EcuM_02825]
[SRS_ModeMgm_-09136]	The ECU State Manager shall be the receiver of all wake-up events	[SWS_EcuM_04091]
[SRS_ModeMgm_-09186]	Alarm Clock shall be active while the ECU is powered	[SWS_EcuM_04054] [SWS_EcuM_04055] [SWS_EcuM_04056] [SWS_EcuM_04057] [SWS_EcuM_04058] [SWS_EcuM_04059] [SWS_EcuM_04060]
[SRS_ModeMgm_-09187]	In Case of wakeup, all the alarm clock shall be canceled	[SWS_EcuM_04009]
[SRS_ModeMgm_-09188]	In Case of startup, all the alarm clock shall be canceled	[SWS_EcuM_04010]
[SRS_ModeMgm_-09190]	The alarm clock service shall allow setting an alarm relative to the current time using a time resolution of seconds	[SWS_EcuM_04054]

▽

△

Requirement	Description	Satisfied by
[SRS_ModeMgm_-09194]	The alarm clock service shall allow setting the clock	[SWS_EcuM_04064]
[SRS_ModeMgm_-09199]	The alarm clock service shall allow setting an alarm absolute by using an absolute time with a resolution of seconds	[SWS_EcuM_04057]
[SRS_ModeMgm_-09234]	The EcuM shall handle the initialization of Basic Software modules	[SWS_EcuM_02559] [SWS_EcuM_02730] [SWS_EcuM_02947]
[SRS_ModeMgm_-09235]	The ECU State Manager shall offer two targets for shutting down the ECU	[SWS_EcuM_00624] [SWS_EcuM_02156] [SWS_EcuM_02822] [SWS_EcuM_02824] [SWS_EcuM_02825]
[SRS_ModeMgm_-09239]	To shutdown, ShutdownAllCores shall be called on the master core after synchronizing all cores	[SWS_EcuM_04024]

Table 6.1: RequirementsTracing

7 Functional Specification

Chapter 1 introduced the new, more flexible approach to ECU state management.

However, this flexibility comes at the price of responsibility. There are no standard ECU modes, or states. The integrator of an ECU must decide which states are needed and also configure them.

When ECU Mode Handling is used, the standard states RUN and POST_RUN are arbitrated by the RUN Request Protocol and propagated to the BswM. The system designer has to make sure that pre-conditions of respective states are met when setting an EcuM Mode by BswM actions.

Note that neither the BSW nor SW-Cs will be able to rely on certain ECU modes or states, although previous versions of the BSW have largely not relied on them..

This document only specifies the functionality that remains in the ECU Manager module. For a complete picture of ECU State Management, refer to the specifications of the other relevant modules, i.e., RTE and BSW Scheduler module [2] and BSW Mode Manager module [3].

Refer to the Guide to Mode Management [4] for some example use cases for ECU states and the interaction between the involved BSW modules.

The ECU Manager module manages the state of wakeup sources in the same way as it has in the past. The APIs to set/clear/validate wakeup events remain the same - with the notable difference that these APIs are Callbacks.

It was always intended that wakeup source handling take place not only during wakeup but continuously, in parallel to all other EcuM activities. This functionality is now fully decoupled from the rest of ECU management via mode requests.

7.1 Phases of the ECU Manager Module

Previous versions of the ECU Manager Module specification have differentiated between ECU states and ECU modes.

ECU modes were longer-lasting periods of operational ECU activities that were visible to applications and provided orientation to them, i.e. starting up, shutting down, going to sleep and waking up.

The ECU Manager states were generally continuous sequences of ECU Manager Module operations terminated by waiting until external conditions were fulfilled. Startup1, for example, contained all BSW initialization before the OS was started and terminated when the OS returned control to the ECU Manager module.

For the current Flexible ECU Manager there exist *States, Modes* and *Phases* which are defined in Definitions and Acronyms.

Here the ECU state machine is implemented as general modes under the control of the BSW Mode Manager module. This creates a terminology problem as the old ECU *States* now become *Modes* that are visible through the RTE_Mode port interface and the old ECU *Modes* become *Phases*.

Because *Modes* as defined by the VFB and used in the RTE are only available in the UP phase (where the ECU Manager is passive) the change of terminology from *Modes* to *Phases* got necessary.

Figure 7.1 shows an overview over the phases of the Flexible ECU Manager module.

The STARTUP phase lasts until the mode management facilities are running. Basically the STARTUP phase consists of the minimal activities needed to start mode management: initializing low-level drivers, starting the OS and initializing the BSW Scheduler and the BSW Mode Manager modules. Similarly the SHUTDOWN phase is the reverse of the STARTUP phase is where mode management is de-initialized.

The UP phase consists of all states that are not highlighted. During that phase, the ECU goes from *State* to *State* and from *Mode* to *Mode*, as dictated by the Integrator-defined state machine.

The UP phase contains default Modes in case ECU Mode Handling is used. The transition between these Modes is done by cooperation between the ECU State Manager module and the BSW Mode Manager module.

Note that the UP phase contains some former sleep states. The mode management facilities do not operate from the point where the OS Scheduler has been locked to prevent other tasks from running in sleep to the point where the MCU mode that puts the ECU to sleep has been exited. The ECU Manager module provides wakeup handling support at this time.

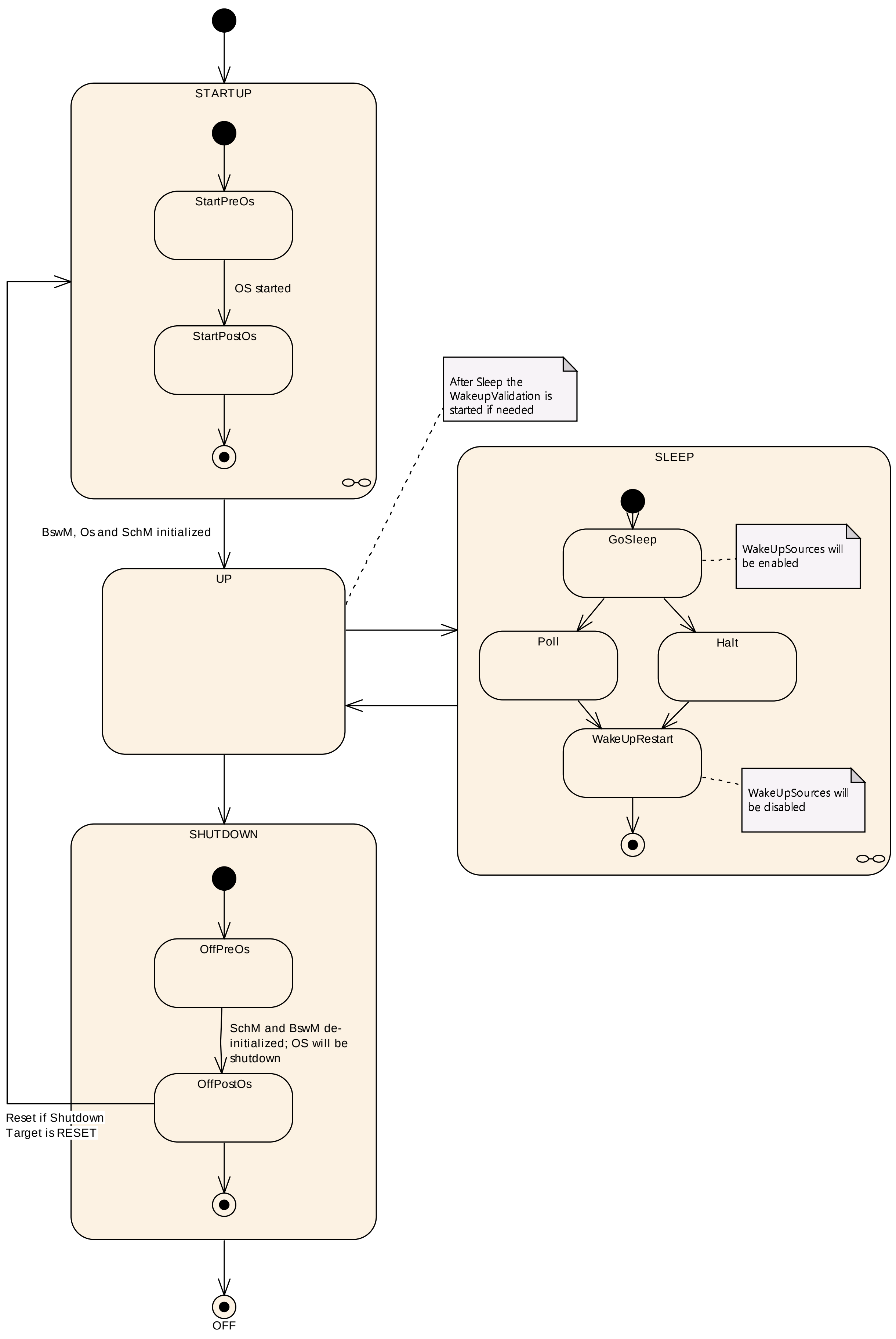

STARTUP
StartPreOs
OS started
StartPostOs
BswM, Os and SchM initialized
UP
After Sleep the WakeupValidation is started if needed
SLEEP
GoSleep
WakeUpSources will be enabled
Poll
Halt
WakeUpRestart
WakeUpSources will be disabled
SHUTDOWN
OffPreOs
SchM and BswM de-initialized; OS will be shutdown
OffPostOs
Reset if Shutdown Target is RESET
OFF

Figure 7.1: Phases of the ECU Manager

7.1.1 STARTUP Phase

The purpose of the STARTUP phase is to initialize the basic software modules to the point where Generic Mode Management facilities are operational. For more details about the initialization see chapter 7.3.

7.1.2 UP Phase

Essentially, the UP phase starts when the BSW Scheduler has started and `BswM_Init` has been called. At that point, memory management is not initialized, there are no communication stacks, no SW-C support (RTE) and the SW-Cs have not started. Processing starts in a certain mode (the next one configured after Startup) with corresponding runnables, i.e. the BSW MainFunctions, and continues as an arbitrary combination of mode changes which cause the BswM to execute actions as well as triggering and disabling corresponding runnables.

From the ECU Manager Module perspective, the ECU is "up", however. The BSW Mode Manager Module then starts mode arbitration and all further BSW initialization, starting the RTE and (implicitly) starting SW-Cs becomes code executed in the BswM's action lists or driven by mode-dependent scheduling, effectively under the control of the integrator.

Initializing the NvM and calling NvM_Readall therefore also becomes integration code. This means that the integrator is responsible for triggering the initialization of Com, DEM and FIM at the end of NvM_ReadAll. The NvM will notify the BswM when NvM_ ReadAll has finished.

Note that the RTE can be started after NvM and COM have been initialized. Note also that the communication stack need not be fully initialized before COM can be initialized.

These changes initialize BSW modules as well as starting SW-Cs in arbitrary order until the ECU reaches full capacity and the changes continue to determine the ECU capabilities thereafter as well.

Ultimately mode switches stop SW-Cs and de-initialize the BSW so that the Up phase ends when the ECU reaches a state where it can be powered off.

So, as far as the ECU Manager module is concerned, the BSW and SW-Cs run until they are ready for the ECU to be shut down or put to sleep.

Refer to the Guide to Mode Management [4] for guidance on how to design mode-driven ECU management and for configuring the BSW Mode Manager accordingly.

7.1.3 SHUTDOWN Phase

[SWS_EcuM_03022] ⌈The SHUTDOWN phase handles the controlled shutdown of basic software modules and finally results in the selected shutdown target OFF or RESET.⌋*(SRS_ModeMgm_09072)*

7.1.4 SLEEP Phase

The ECU saves energy in the SLEEP phase. Typically, no code is executed but power is still supplied, and if configured accordingly, the ECU is wakeable in this state[1]. The ECU Manager module provides a configurable set of (hardware) sleep modes which typically are a trade off between power consumption and time to restart the ECU.

The ECU Manager module wakes the ECU up in response to intended or unintended wakeup events. Since unintended wakeup events should be ignored, the ECU Manager module provides a protocol to validate wakeup events. The protocol specifies a cooperative process between the driver which handles the wakeup source and the ECU Manager (see section 7.6.4).

7.1.5 OFF Phase

The ECU enters the OFF state when it is powered down. The ECU may be wakeable in this state but only for wakeup sources with integrated power control. In any case the ECU must be startable (e.g. by reset events).

[1]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 Manager

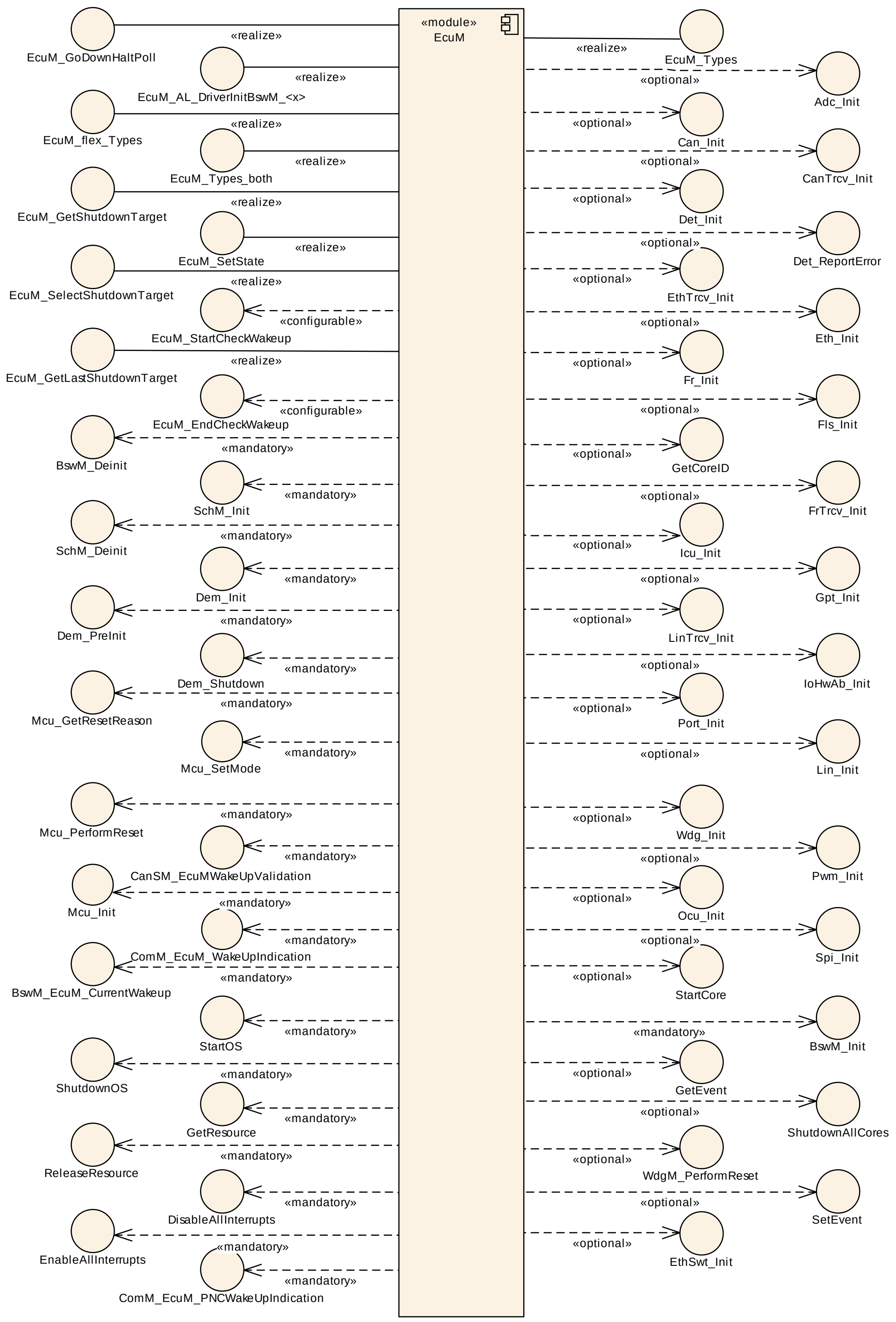


Figure 7.2: ECU Manager Module Relationships

Figure 7.2 illustrates the ECU Manager module's relationship to the interfaces of other BSW modules. In most cases, the ECU Manager module is simply responsible for initialization[2]. There are however some modules that have a functional relationship with the ECU Manager module, which is explained in the following paragraphs.

7.2.1 Standardized AUTOSAR Software Modules

Some Basic Software driver modules are initialized, shut down and re-initialized upon wakeup by the ECU Manager module.

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

After the OS initialization, additional initialization steps are undertaken by the ECU Manager module before passing control to the BswM. The BswM hands execution control back to the ECU Manager module immediately before OS shutdown. Details are provided in the chapters 7.3 STARTUP and 7.4 SHUTDOWN .

7.2.2 Software Components

SW-Components contain the AUTOSAR ECU's application code.

A SW-C interacts with the ECU Manager module using AUTOSAR ports.

7.3 STARTUP Phase

See Chapter 7.1.1 for an overview description of the STARTUP phase.

[2]To be precise, "initialization" could also mean de-initialization.

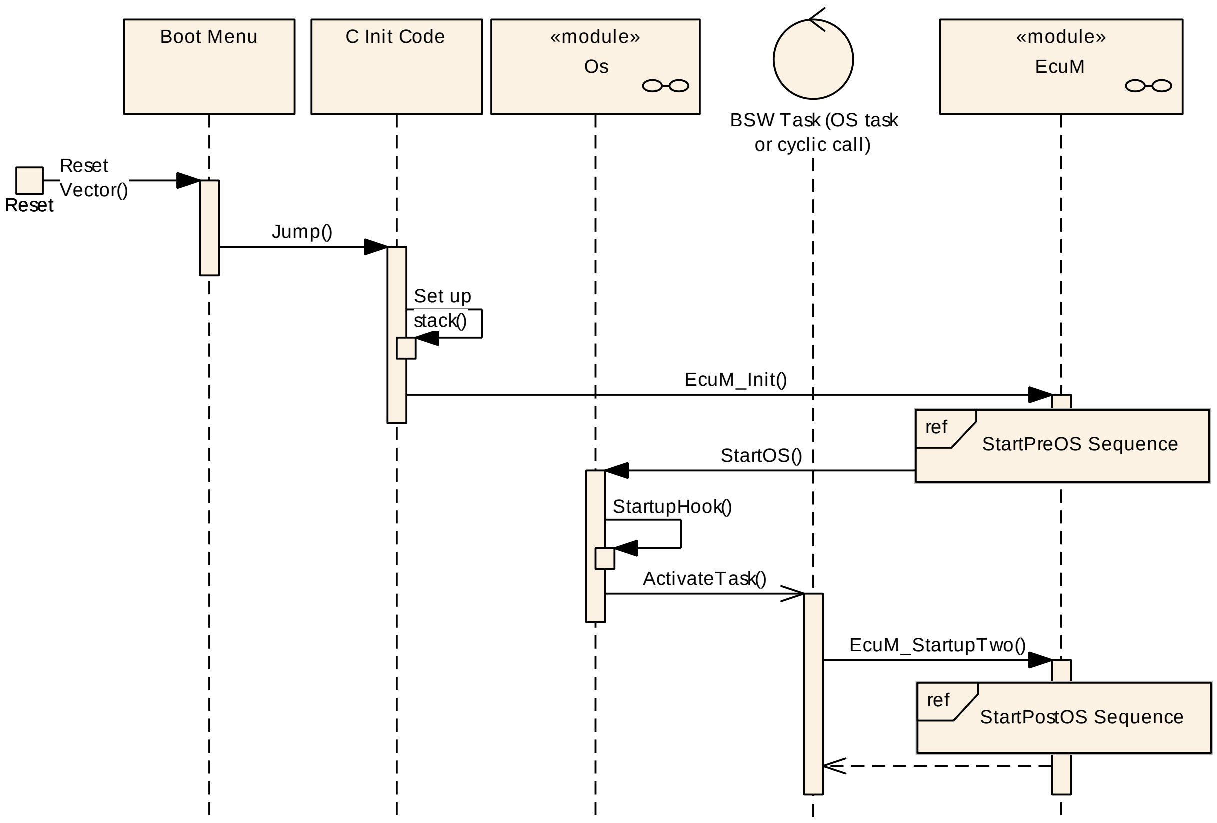


Figure 7.3: STARTUP Phase

Figure 7.3 shows the startup behavior of the ECU. When invoked through `EcuM_Init`, the ECU Manager module takes control of the ECU startup procedure. With the call to `StartOS`, the ECU Manager module temporarily relinquishes control. To regain control, the Integrator has to implement an OS task that is automatically started and calls `EcuM_StartupTwo` as its first action.

7.3.1 Activities before EcuM_Init

The ECU Manager module assumes that before `EcuM_Init` (see [SWS_EcuM_02811]) is called a minimal initialization of the MCU has taken place, so that a stack is set up and code can be executed, also that C initialization of variables has been performed.

7.3.2 Activities in StartPreOS Sequence

[SWS_EcuM_02411] ⌈Table *StartPreOS Sequence* shows the activities in StartPre OS Sequence and the order in which they shall be executed in `EcuM_Init` (see [SWS_EcuM_02811]).⌋*()*

StartPreOS Sequence		
Initialization Activity	**Comment**	**Opt.**
Callout EcuM_AL_SetProgrammableInterrupts	On ECUs with programmable interrupt priorities, these priorities must be set before the OS is started.	yes
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 also 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 the ECU Manager module and all other BSW modules.	no
Check consistency of configuration data	If check fails the EcuM_ErrorHook is called. See 7.3.4 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 the EcuMWakeupSource configuration containers. See 8.4.1.2 EcuM_SetWakeupEvent and 8.3.5.3 EcuM_GetValidatedWakeupEvents (see [SWS_EcuM_02830])	no
Select default shutdown target	See [SWS_EcuM_02181]	no
Callout EcuM_LoopDetection	If Loop Detection is enabled, this callout is called on every startup.	yes
Start OS	Start the AUTOSAR OS, see [SWS_EcuM_02603]	no

Table 7.1: StartPreOS Sequence

Note to column *Opt.* : Optional activities can be switched on or off by configuration. See section 10.1 Common Containers and configuration parameters for details.

[SWS_EcuM_02623] ⌈The ECU Manager module shall remember the wakeup source resulting from the reset reason translation (see table *StartPreOS Sequence*).⌋ *()*

Rationale for [SWS_EcuM_02623]: The wakeup sources must be validated by the EcuM_MainFunction (see section 7.6.4 Activities in the WakeupValidation Sequence).

[SWS_EcuM_02684] ⌈When activated through the EcuM_Init (see [SWS_EcuM_02811]) function, the ECU Manager module shall perform the actions in the StartPreOS Sequence (see table *StartPreOS Sequence*).⌋ *()*

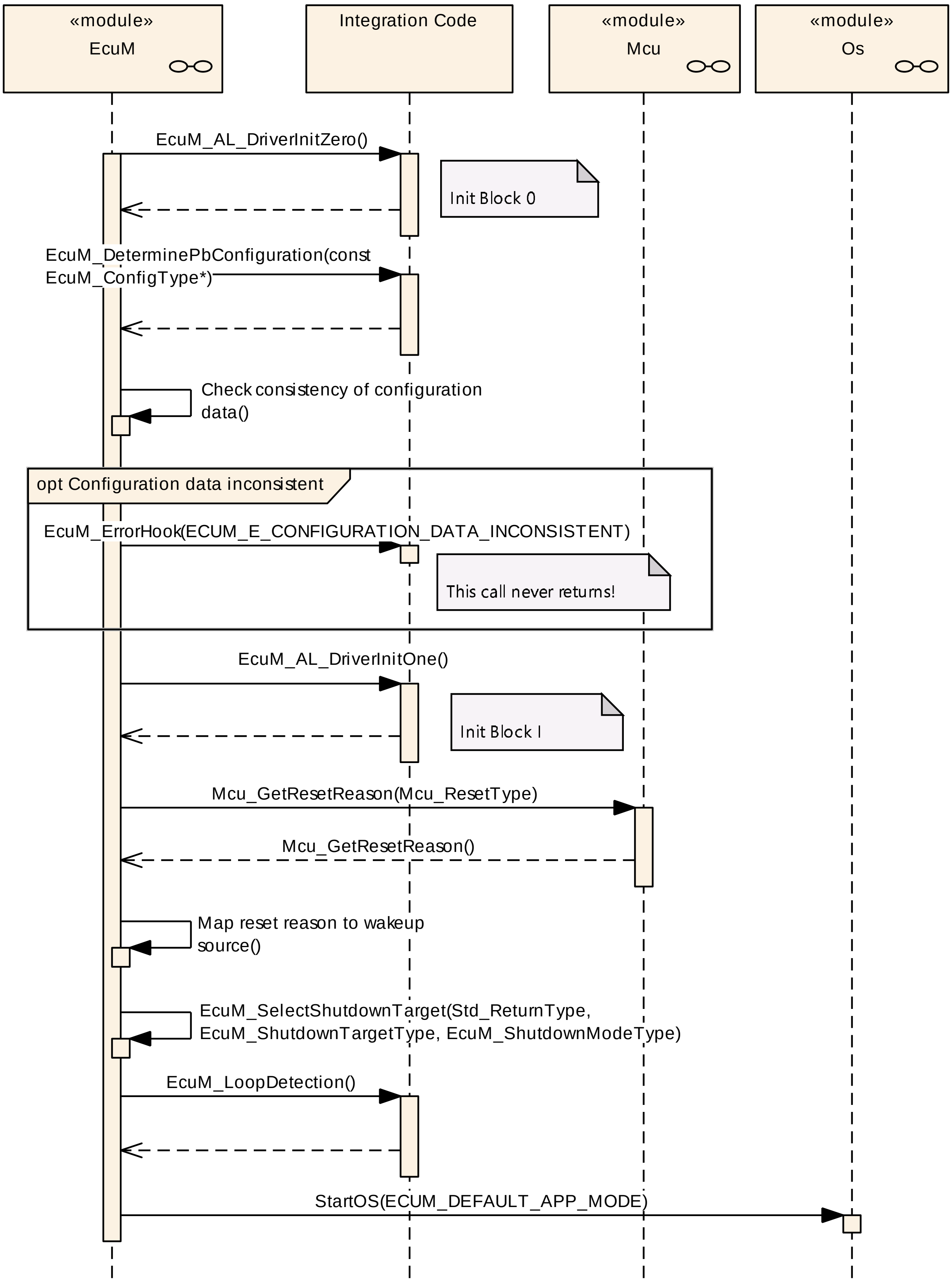

«module»
EcuM
Integration Code
«module»
Mcu
«module»
Os
EcuM_AL_DriverInitZero()
Init Block 0
EcuM_DeterminePbConfiguration(const EcuM_ConfigType*)
Check consistency of configuration data()
opt Configuration data inconsistent
EcuM_ErrorHook(ECUM_E_CONFIGURATION_DATA_INCONSISTENT)
This call never returns!
EcuM_AL_DriverInitOne()
Init Block I
Mcu_GetResetReason(Mcu_ResetType)
Mcu_GetResetReason()
Map reset reason to wakeup source()
EcuM_SelectShutdownTarget(Std_ReturnType, EcuM_ShutdownTargetType, EcuM_ShutdownModeType)
EcuM_LoopDetection()
StartOS(ECUM_DEFAULT_APP_MODE)

Figure 7.4: StartPreOS Sequence

The StartPreOS Sequence is intended to prepare the ECU to initialize the OS and should be kept as short as possible. Drivers should be initialised in the UP phase when possible and the callouts should also be kept short. Interrupts should not be used during this sequence. If interrupts have to be used, only category I interrupts are allowed in the StartPreOS Sequence 1[3] .

Initialization of drivers and hardware abstraction modules is not strictly defined by the ECU Manager. Two callouts EcuM_AL_DriverInitZero (see [SWS_EcuM_02905]) and EcuM_AL_DriverInitOne (see [SWS_EcuM_02907]) are provided to define the init blocks 0 and I. These blocks contain the initialization activities associated with the StartPreOS sequence.

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 (see EcuM_-AL_DriverInitZero, [SWS_EcuM_02905]) or EcuM_AL_DriverInitOne callouts (see EcuM_AL_DriverInitOne, [SWS_EcuM_02907]). Details can be found in the Specification of MCU Driver [10].

[SWS_EcuM_02181] ⌈The ECU Manager module shall call EcuM_GetValidated-WakeupEvents with the configured default shutdown target (EcuMDefaultShut-downTarget).⌋*()*

See section 7.7 Shutdown Targets.

[SWS_EcuM_02603] ⌈The StartPreOS Sequence shall initialize all basic software modules that are needed to start the OS.⌋*()*

7.3.3 Activities in the StartPostOS Sequence

StartPostOS Sequence		
Initialization Activity	**Comment**	**Opt.**
Start BSW Scheduler		no
Init BSW Mode Manager		no
Init BSW Scheduler	Initialize the semaphores for critical sections used by BSW modules	no
Start Scheduler Timing	Start periodical events for BSW/SWCs	no

Table 7.2: StartPostOS Sequence

Note to column *Opt.* : Optional activities can be switched on or off by configuration. See section 10.1 Common Containers and configuration parameters for details.

[SWS_EcuM_02932] ⌈When activated through the EcuM_StartupTwo (see [SWS_EcuM_02838]) function, the ECU Manager module shall perform the actions in StartPostOS Sequence (see table 7.2).⌋*(SRS_ModeMgm_09113)*

[3]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.

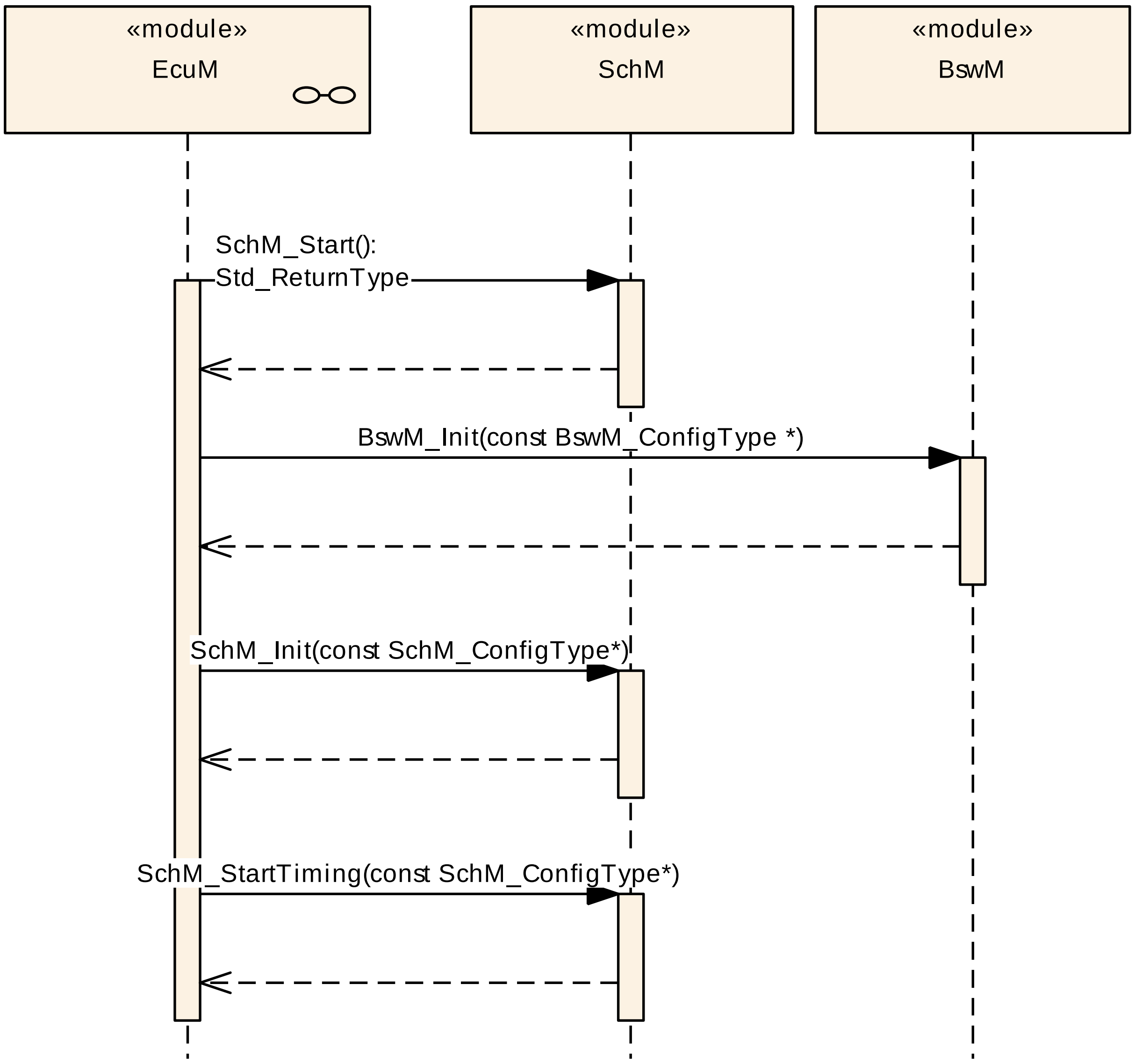

«module»
EcuM
«module»
SchM
«module»
BswM
SchM_Start():
Std_ReturnType
BswM_Init(const BswM_ConfigType *)
SchM_Init(const SchM_ConfigType*)
SchM_StartTiming(const SchM_ConfigType*)

Figure 7.5: StartPostOS Sequence

7.3.4 Checking Configuration Consistency

7.3.4.1 The Necessity for Checking Configuration Consistency in the ECU Manager

In an AUTOSAR ECU several configuration parameters are set and put into the ECU at different times. Pre-compile parameters are set, inserted 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.

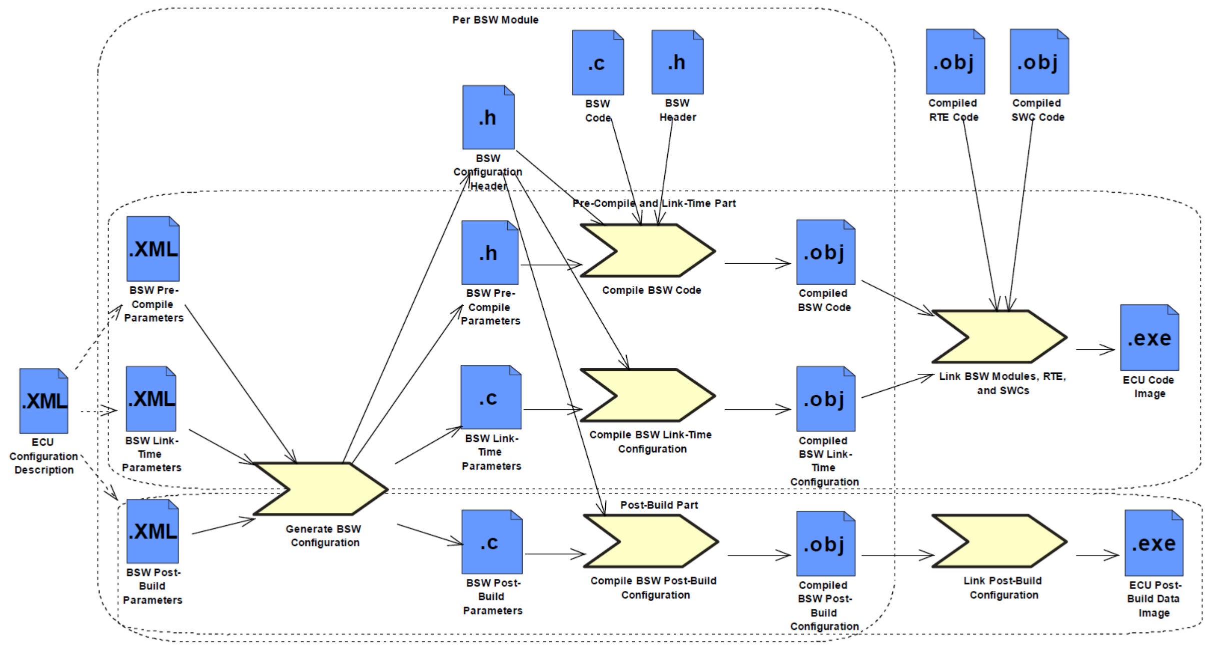

Per BSW Module
.c
BSW Code
.h
BSW Header
.obj
Compiled RTE Code
.obj
Compiled SWC Code
.h
BSW Configuration Header
Pre-Compile and Link-Time Part
.XML
BSW Pre-Compile Parameters
.h
BSW Pre-Compile Parameters
Compile BSW Code
.obj
Compiled BSW Code
Link BSW Modules, RTE, and SWCs
.exe
ECU Code Image
.XML
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 7.6: BSW Configuration Steps

The configuration tool can check the consistency of configuration time parameters itself. The compiler may detect parameter errors at compilation time and the linker may find additional errors at link time. Unfortunately, finding configuration errors in post-build parameters is very difficult. This can only be achieved 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.

Explanation for [SWS_EcuM_02796]: The ECU Manager module checks the consistency once before initializing the first BSW module to avoid multiple checks scattered over the different BSW modules.

This also implies that:

[SWS_EcuM_02796] ⌈The ECU Manager module shall not only check the consistency of its own parameters but of all post-build configurable BSW modules before initializing the first BSW module.⌋*()*

The ECU Manager Configuration Tool must compute a hash value over all pre-compile and link-time configuration parameters of all BSW modules and store the value in the link-time *ECUM_CONFIGCONSISTENCY_HASH* (see EcuMConfigConsistencyHash) configuration parameter. The hash value is necessary for two reasons. First, the pre-compile and link-time parameters are not accessible 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.

The ECU Manager module Configuration Tool must in turn put the computed *ECUM_CONFIGCONSISTENCY_HASH* value into the field in the EcuM_ConfigType structure which contains the root of all post-build configuration parameters.

[SWS_EcuM_02798] ⌈The ECU Manager module shall check in EcuM_Init (see [SWS_EcuM_02811]) that the field in the structure is equal to the value of *ECUM_CONFIGCONSISTENCY_HASH* .⌋*()*

By computing hash values at configuration time and comparing them at run-time the EcuM code can be very efficient and is furthermore independent of a particular hash computation algorithm. This allows 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.

[SWS_EcuM_02799] ⌈The hash computation algorithm used to compute a hash value over all pre-compile and link-time configuration parameters of all BSW modules 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.⌋*()*

7.3.4.2 Example Hash Computation Algorithm

Note: This chapter is not normative. It describes one possible way to compute 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.

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.

7.3.5 Driver Initialization

A driver's location in the initialization process depends strongly on its implementation and the target hardware design.

Drivers can be initialized by the ECU Manager module in Init Block 0 or Init Block 1 of the STARTUP phase or re-initialized in the `EcuM_AL_DriverRestart` callout of the WakeupRestart Sequence. Drivers can also be initialized or re-initialized by the BswM during the UP phase.

This chapter applies to those AUTOSAR Basic Software drivers, other than SchM and BswM, whose initialization and re-initialization is handled by the ECU Manager module and not the BswM.

[SWS_EcuM_02559] ⌈The configuration of the ECU Manager module shall specify the order of initialization calls inside init block 0 and init block 1. (see `EcuMDriverInitListZero` and `EcuMDriverInitListOne`).⌋*(SRS_BSW_00416, SRS_ModeMgm_09234)*

[SWS_EcuM_02730] ⌈The ECU Manager module shall call each driver's init function with the parameters derived from the driver's `EcuMModuleService` configuration container.⌋*(SRS_ModeMgm_09234)*

[SWS_EcuM_02947] ⌈For re-initialization during WakeupRestart, the integrator shall integrate a restart block into the integration code for `EcuM_AL_DriverRestart` (see [SWS_EcuM_02923]) using the `EcuMDriverRestartList`.⌋*(SRS_ModeMgm_09234)*

[SWS_EcuM_02562] ⌈`EcuMDriverRestartList` may contain drivers that serve as wakeup sources. `EcuM_AL_DriverRestart` shall re-arm the trigger mechanism of these drivers' 'wakeup detected' callback.⌋*()*

See Section 7.5.5 Activities in the WakeupRestart Sequence.

[SWS_EcuM_02561] ⌈The ECU Manager module shall initialize the drivers in Ecu MDriverRestartList in the same order as in the combined list of init block 0 and init block 1.⌋*()*

Hint for [SWS_EcuM_02561]: `EcuMDriverRestartList` will typically only contain a subset of the combined list of init block 0 and init block 1 drivers.

Table 7.3 shows one possible (and recommended) sequence of activities for the Init Blocks 0 and I. Depending on hardware and software configuration, BSW modules may be added or left out and other sequences may also be possible.

Recommended Init Block		
	Initialization Activity	**Comment**
Init Block 0[4]		
	Default Error Tracer	This should always be the first module to be initialized, so that other modules can report development errors.
	Diagnostic Event Manager	Pre-Initialization
	Any drivers needed to access post-build configuration data	These drivers shall not depend on the post-build configuration or on OS features.
Init Block I[5]		
	MCU Driver	
	Port Driver	
	General Purpose Timer	
	Watchdog Driver	Internal watchdogs only, external ones may need SPI
	Watchdog Manager	
	ADC Driver	
	ICU Driver	
	PWM Driver	
	OCU Driver	

Table 7.3: Driver Initialization Details, Sample Configuration

7.3.6 BSW Initialization

The remaining BSW modules are initialized by the BSW Mode Manager, using a configured function of the ECU Manager (EcuMDriverInitCalloutName ECUC_EcuM_00227) created from the configured list of init functions (`EcuMDriverInitListBswM`).

[SWS_EcuM_04142] ⌈The configuration of the ECU Manager module shall specify the order of initialization calls inside the BSW initialization (see `EcuMDriverInitListBswM`).⌋*()*

7.4 SHUTDOWN Phase

Refer to Section 7.1.3 SHUTDOWN Phase for an overview of the SHUTDOWN phase. `EcuM_GoDownHaltPoll` with shutdown target RESET or OFF initiates the SHUTDOWN Phase.

[SWS_EcuM_02756] ⌈When a wakeup event occurs during the shutdown phase, the ECU Manager module shall complete the shutdown and restart immediately thereafter.⌋*()*

[4]Drivers in Init Block 0 are listed in the EcuMDriverInitListZero configuration container.
[5]Drivers in Init Block I are listed in the EcuMDriverInitListOne configuration container.

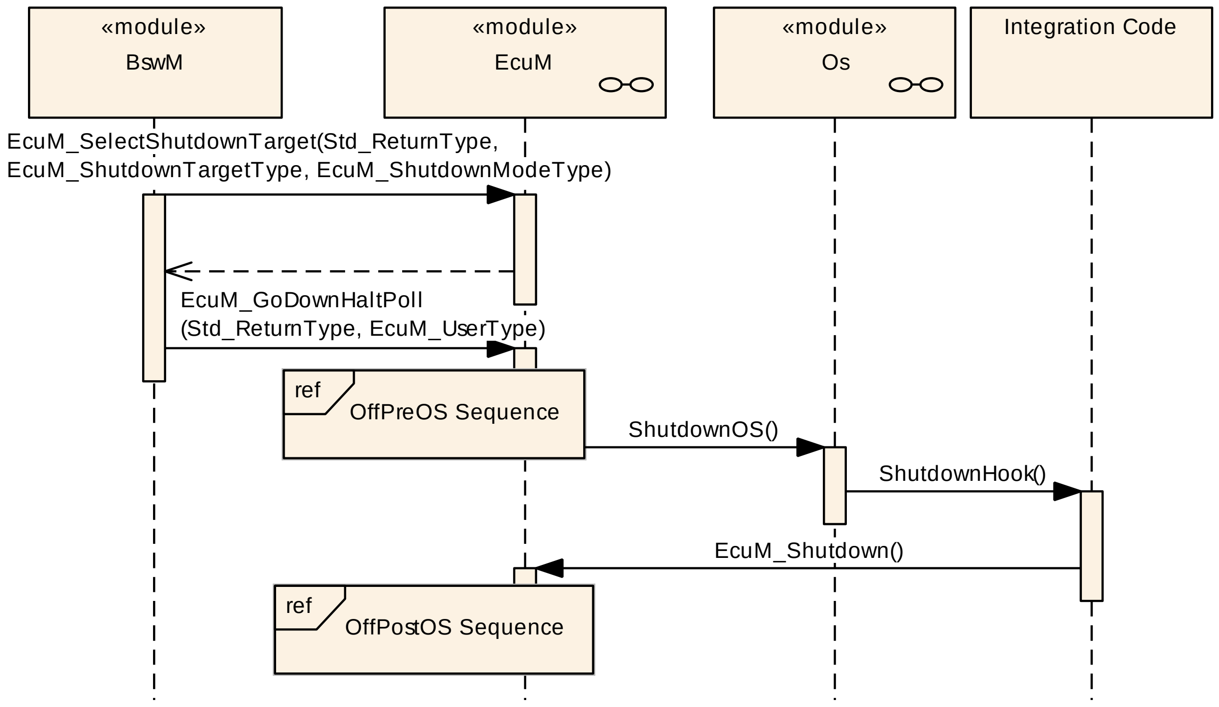

«module» BswM
«module» EcuM
«module» Os
Integration Code
EcuM_SelectShutdownTarget(Std_ReturnType, EcuM_ShutdownTargetType, EcuM_ShutdownModeType)
EcuM_GoDownHaltPoll (Std_ReturnType, EcuM_UserType)
ref OffPreOS Sequence
ShutdownOS()
ShutdownHook()
EcuM_Shutdown()
ref OffPostOS Sequence

Figure 7.7: SHUTDOWN Phase

7.4.1 Activities in the OffPreOS Sequence

[SWS_EcuM_03021] ⌈See 7.4⌋*(SRS_ModeMgm_09127)*

OffPreOS Sequence		
Shutdown Activity	**Comment**	**Opt.**
De-init BSW Mode Manager		no
De-init BSW Scheduler		no
Check for pending wakeup events	Purpose is to detect wakeup events that occurred during shutdown	no
Set RESET as shutdown target, if wakeup events are pending (default reset mode of `EcuMDefaultReset-ModeRef` will be used)	This action shall only be carried out when pending wakeup events were detected to allow an immediate startup	no
ShutdownOS	Last operation in this OS task	no

Table 7.4: OffPreOs Sequence

Note to column *Opt.* : Optional activities can be switched on or off by configuration. It shall be the system designers choice if a module is compiled in or not for an ECU design. See chapter 10.1 Common Containers and configuration parameters for details.

[SWS_EcuM_02952] ⌈As its last activity, the ECU Manager module shall call the ShutdownOS function.⌋*(SRS_ModeMgm_09104)*

The OS calls the shutdown hook at the end of its shutdown.

[SWS_EcuM_02953] ⌈The shutdown hook shall call EcuM_Shutdown (see [SWS_EcuM_02812]) to terminate the shutdown process. EcuM_Shutdown(see [SWS_EcuM_02812]) shall not return but switch off the ECU or issue a reset.⌋*(SRS_-ModeMgm_09104)*

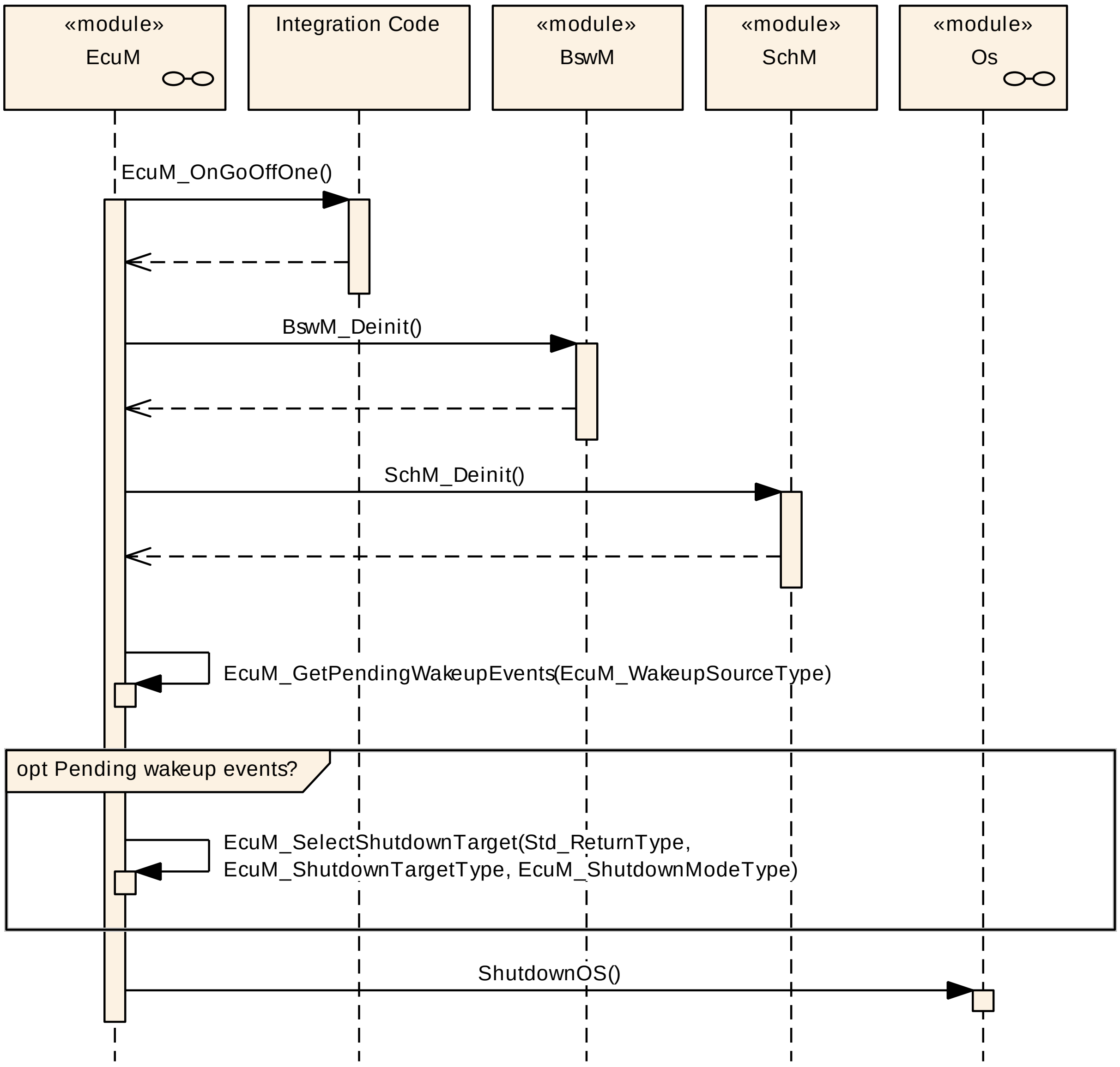

«module» EcuM
Integration Code
«module» BswM
«module» SchM
«module» Os
EcuM_OnGoOffOne()
BswM_Deinit()
SchM_Deinit()
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
opt Pending wakeup events?
EcuM_SelectShutdownTarget(Std_ReturnType, EcuM_ShutdownTargetType, EcuM_ShutdownModeType)
ShutdownOS()

Figure 7.8: OffPreOS Sequence

7.4.2 Activities in the OffPostOS Sequence

The OffPostOS sequence implements the final steps to reach the shutdown target after the OS has been shut down. EcuM_Shutdown (see [SWS_EcuM_02812]) initiates the sequence.

The shutdown target can be either ECUM_SHUTDOWN_TARGET_RESET or ECUM_SHUTDOWN_TARGET_OFF, whereby the specific reset modality is determined by the reset mode. See section 7.7 Shutdown Targets for details.

OffPostOS Sequence		
Shutdown Activity	**Comment**	**Opt.**
Callout `EcuM_OnGoOffTwo`		
Callout `EcuM_AL_Reset` or Callout `EcuM_AL_SwitchOff`	Depends on the selected shutdown target (RESET or OFF)	no

Table 7.5: OffPostOs Sequence

Note to column *Opt.* : Optional activities can be switched on or off by configuration. It shall be the system designers choice if a module is compiled in or not for an ECU design. See chapter 10.1 Common Containers and configuration parameters for details.

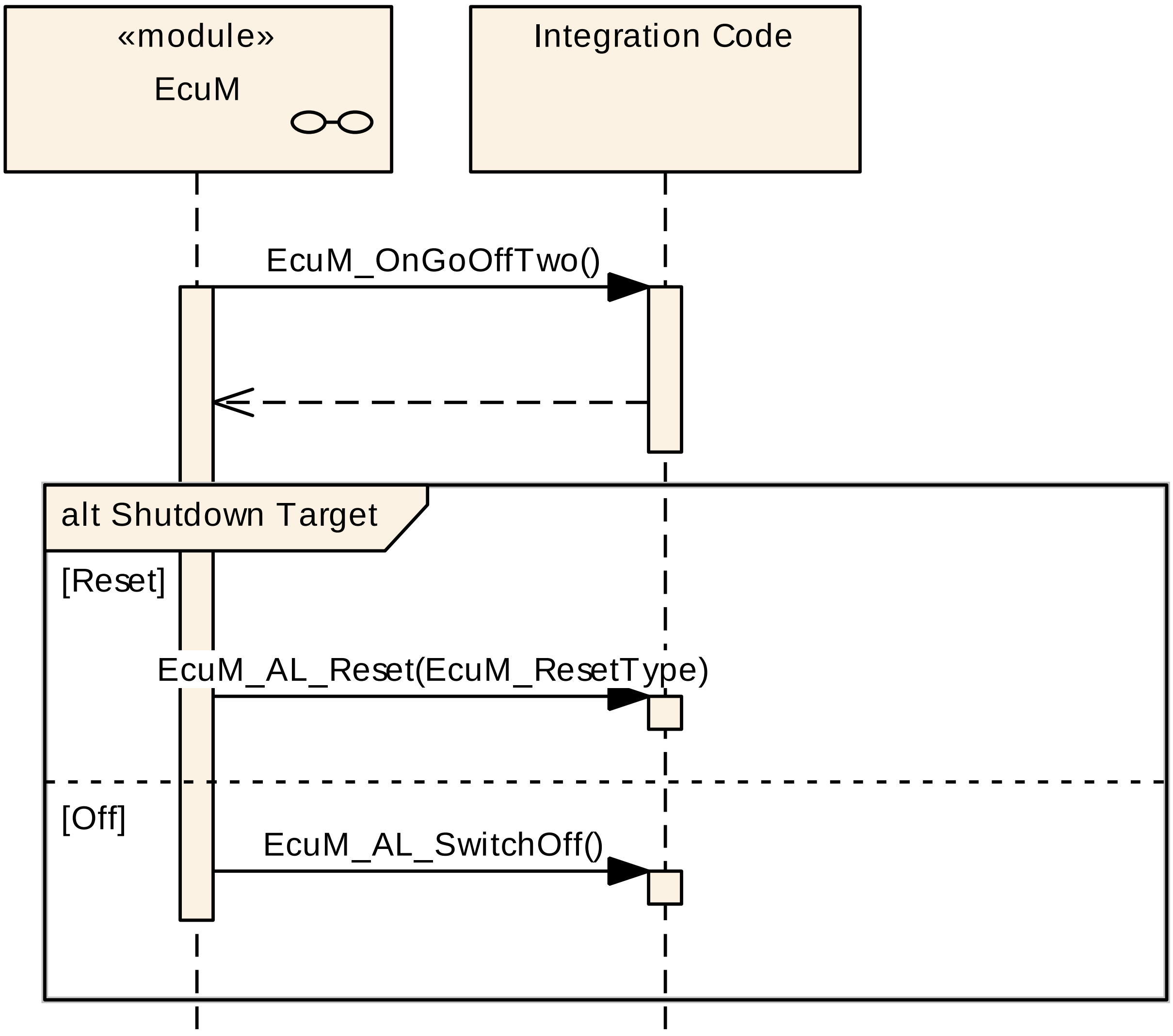


Figure 7.9: OffPostOS Sequence

[SWS_EcuM_04074] ⌈When the shutdown target is RESET, the ECU Manager module shall call the `EcuM_AL_Reset` callout.⌋*()*

See section 8.5.3.4 `EcuM_AL_Reset` ([SWS_EcuM_04065]) for details.

[SWS_EcuM_04075] ⌈When the shutdown target is OFF, the ECU Manager module shall call the `EcuM_AL_SwitchOff` callout.⌋*()*

See section 8.5.3.3 `EcuM_AL_SwitchOff` ([SWS_EcuM_02920]) for details.

7.5 SLEEP Phase

Refer to Section 7.1.4 SLEEP Phase for an overview of the SLEEP phase. `EcuM_GoDownHaltPoll` with shutdown target SLEEP initiate the SLEEP phase.

`EcuM_GoDownHaltPoll` with shutdown target SLEEP initiate two control streams, depending on the sleep mode selected (EcuMSleepModeSuspend parameter), that differ structurally in the mechanisms used to realize sleep. They share the sequences for preparing for and recovering from sleep, however.

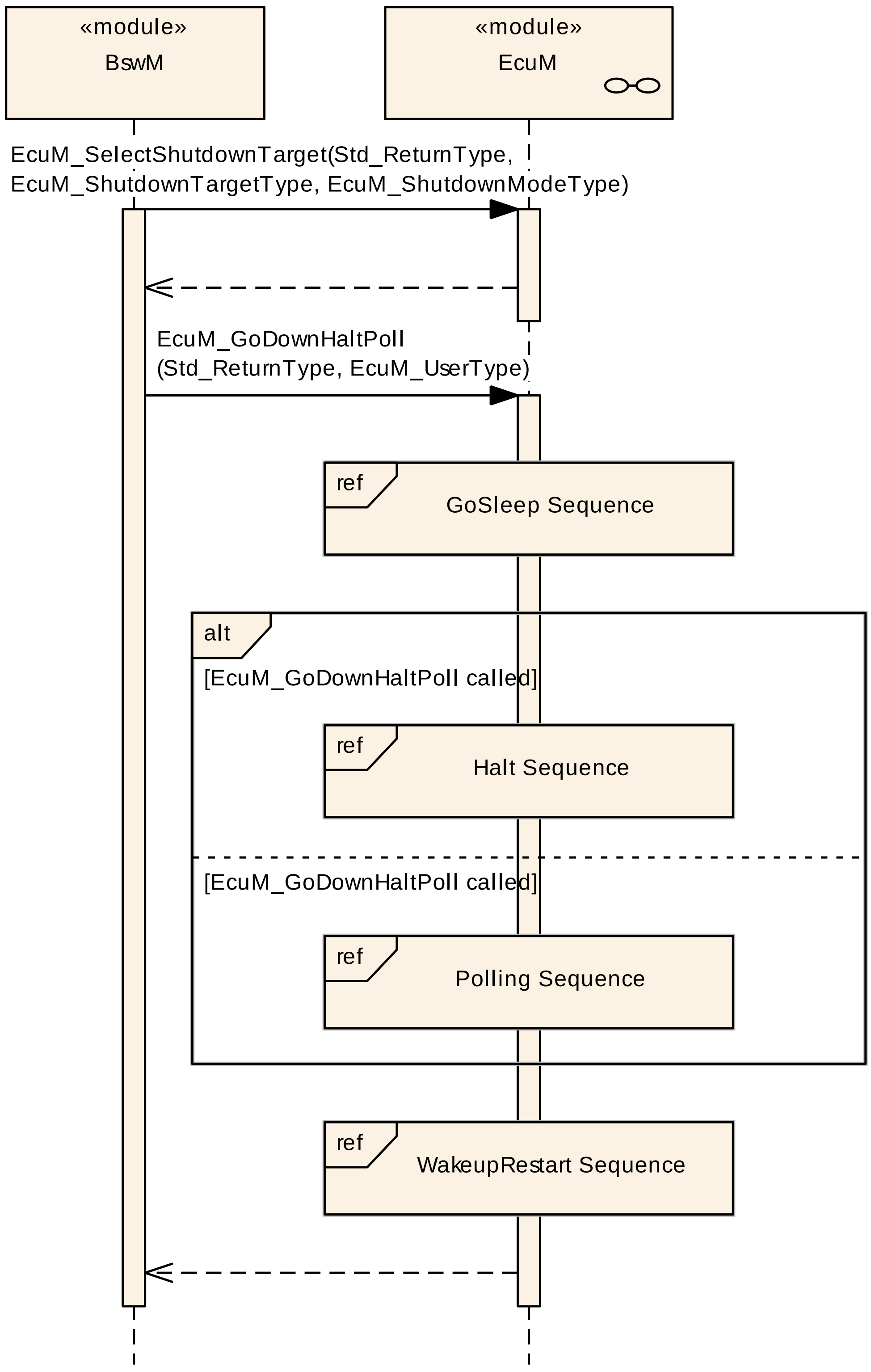

«module»
BswM
«module»
EcuM
EcuM_SelectShutdownTarget(Std_ReturnType, EcuM_ShutdownTargetType, EcuM_ShutdownModeType)
EcuM_GoDownHaltPoll (Std_ReturnType, EcuM_UserType)
ref
GoSleep Sequence
alt
[EcuM_GoDownHaltPoll called]
ref
Halt Sequence
[EcuM_GoDownHaltPoll called]
ref
Polling Sequence
ref
WakeupRestart Sequence

Figure 7.10: SLEEP Phase

Another module, presumably the BswM, although it could be an SW-C as well, must ensure that an appropriate ECUM_STATE_SLEEP shutdown target has been selected before calling EcuM_GoDownHaltPoll.

7.5.1 Activities in the GoSleep Sequence

In the GoSleep sequence the ECU Manager module configures hardware for the upcoming sleep phase and sets the ECU up for the next wakeup event.

[SWS_EcuM_02389] ⌈To set the wakeup sources up for the next sleep mode, the ECU Manager module shall execute the `EcuM_EnableWakeupSources` callout (see [SWS_EcuM_02546]) for each wakeup source that is configured in `EcuMWakeupSourceMask` for the target sleep mode.⌋*(SRS_ModeMgm_09100)*

[SWS_EcuM_02951] ⌈In contrast to the SHUTDOWN phase, the ECU Manager module shall not shut down the OS when entering the SLEEP phase. The sleep mode, i.e. combination of the EcuM SLEEP phase and the Mcu Mode, shall be transparent to the OS.⌋*()*

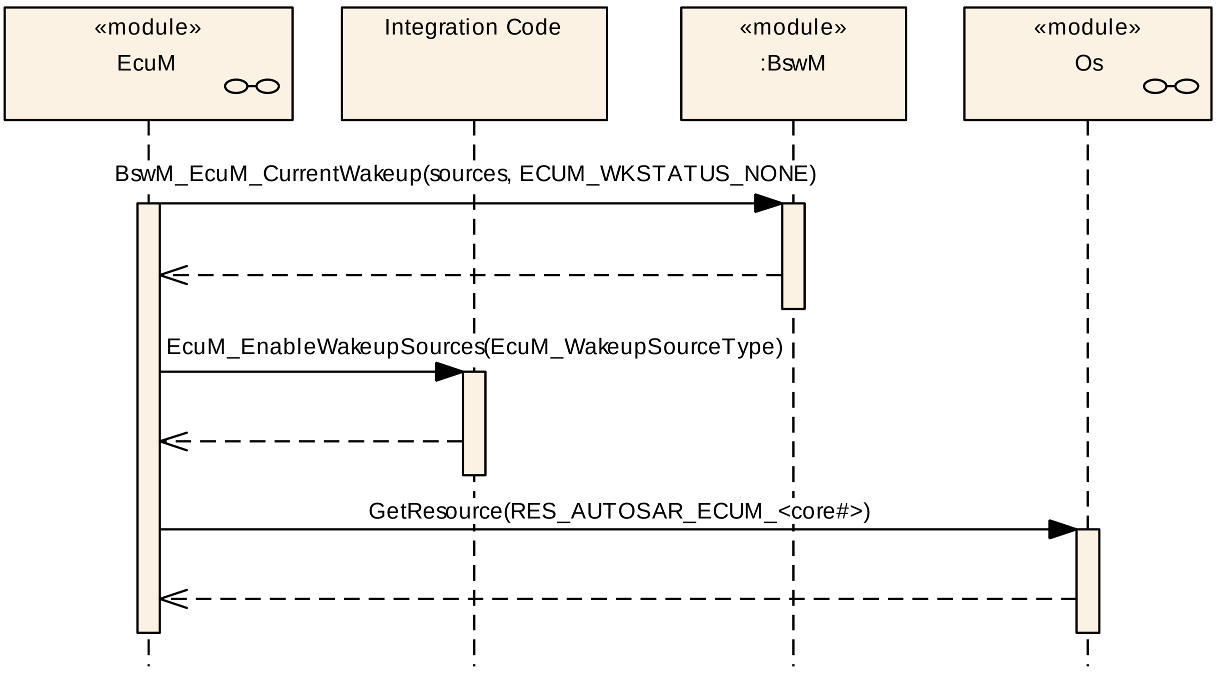

«module»
EcuM
Integration Code
«module»
:BswM
«module»
Os
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_NONE)
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(RES_AUTOSAR_ECUM_<core#>)

Figure 7.11: GoSleep Sequence

[SWS_EcuM_03010] ⌈When operating on a multicore ECU ECUM shall reserve a dedicated resource (RES_AUTOSAR_ECUM) for each core, which is allocated during Go Sleep.⌋*()*

7.5.2 Activities in the Halt Sequence

[SWS_EcuM_02960] ⌈The ECU Manager module shall execute the Halt Sequence in sleep modes that halt the microcontroller. In these sleep modes the ECU Manager module does not execute any code.⌋*()*

[SWS_EcuM_02863] ⌈The ECU Manager module shall invoke the `EcuM_GenerateRamHash` (see [SWS_EcuM_02919]) callout before halting the microcontroller the

EcuM_CheckRamHash (see [SWS_EcuM_02921]) callout after the processor returns from halt.

In case of applied multi core and existence of "slave" EcuM(s) this check should be executed on the "master" EcuM only. The "master" EcuM generates the hash out of all data that lie within its reach. Private data of "slave" EcuMs are out of scope.⌋*()*

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.

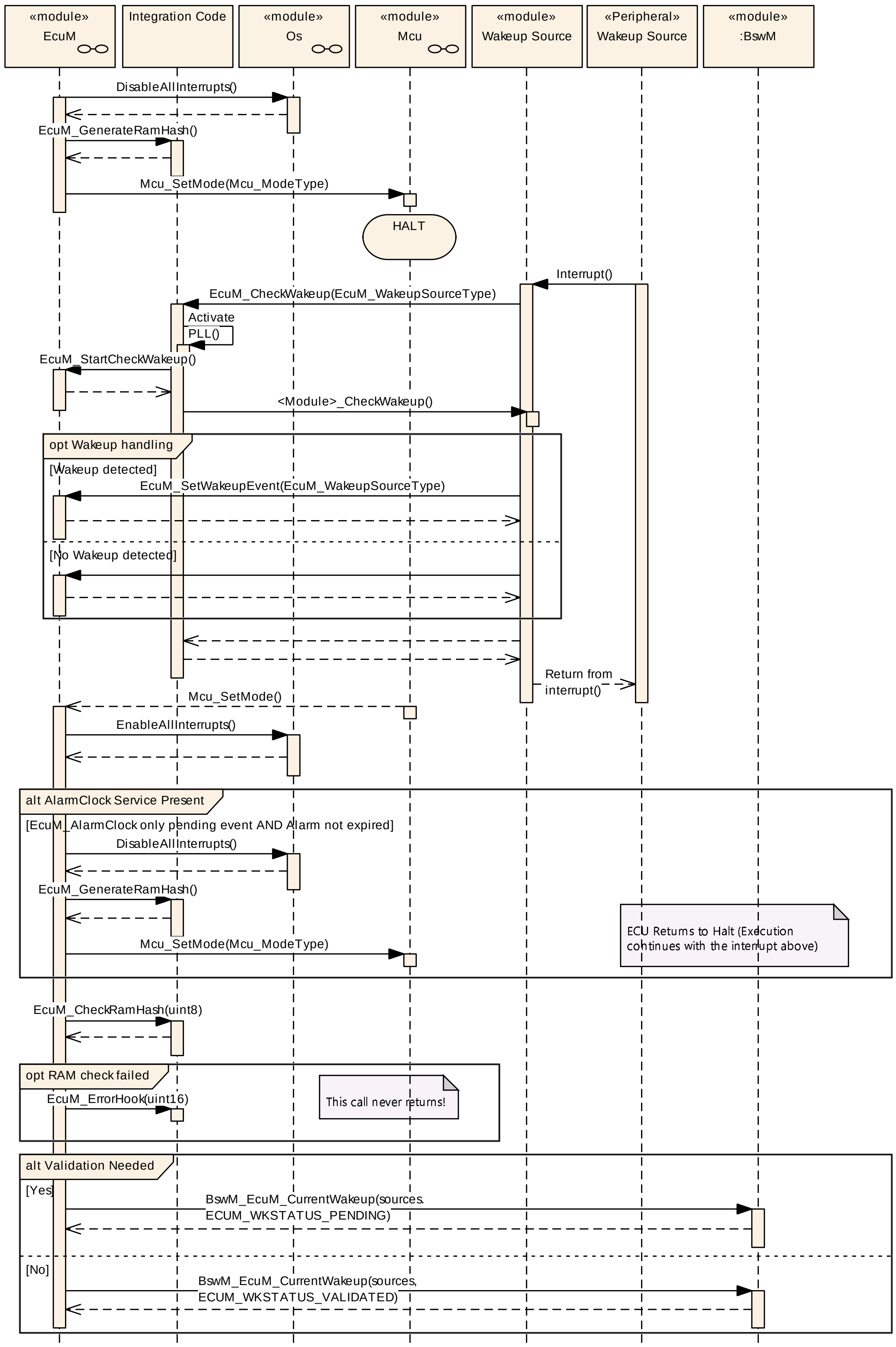


Figure 7.12: Halt Sequence

[SWS_EcuM_02961] ⌈The ECU Manager module shall invoke the EcuM_GenerateRamHash (see [SWS_EcuM_02919]) where the system designer can place a RAM integrity check.⌋*()*

7.5.3 Activities in the Poll Sequence

[SWS_EcuM_02962] ⌈The ECU Manager 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 EcuM shall call the callouts EcuM_SleepActivity and EcuM_CheckWakeup() in a blocking loop until a pending/validated wakeup event is reported.⌋*()*

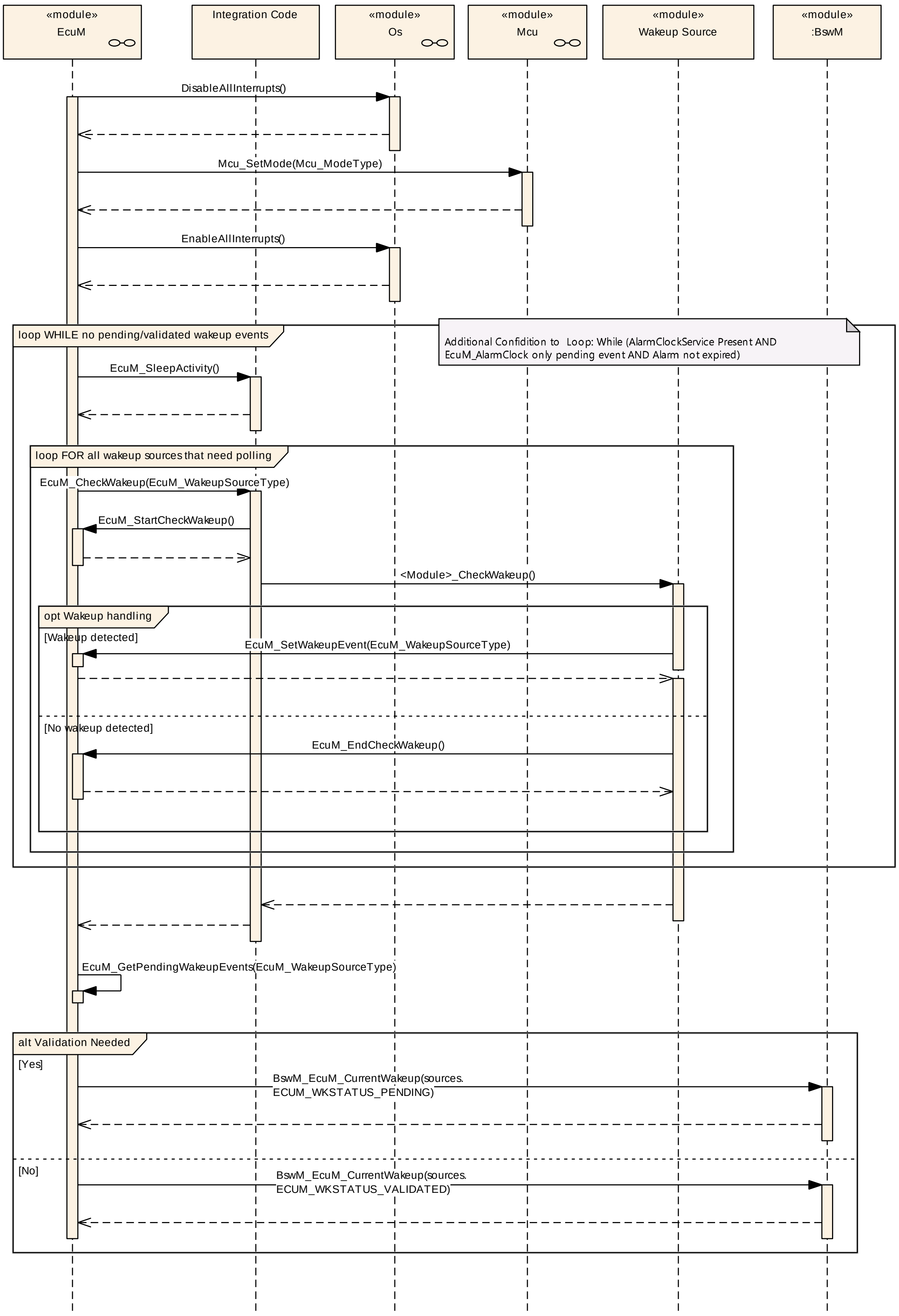

«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Wakeup Source
«module»
:BswM
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
loop WHILE no pending/validated wakeup events
Additional Confidition to Loop: While (AlarmClockService Present AND EcuM_AlarmClock only pending event AND Alarm not expired)
EcuM_SleepActivity()
loop FOR all wakeup sources that need polling
EcuM_CheckWakeup(EcuM_WakeupSourceType)
EcuM_StartCheckWakeup()
<Module>_CheckWakeup()
opt Wakeup handling
[Wakeup detected]
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
[No wakeup detected]
EcuM_EndCheckWakeup()
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
alt Validation Needed
[Yes]
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_PENDING)
[No]
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_VALIDATED)

Figure 7.13: Poll Sequence

7.5.4 Leaving Halt or Poll

[SWS_EcuM_02963] ⌈If a wakeup event (e.g. toggling a wakeup line, communication on a CAN bus etc.) occurs while the ECU is in Halt or Poll, then the ECU Manager module shall regain control and exit the SLEEP phase by executing the WakeupRestart sequence.

An ISR may be invoked to handle the wakeup event, but this depends on the hardware and the driver implementation.⌋*()*

See section 7.5.5 Activities in the WakeupRestart Sequence.

[SWS_EcuM_04001] ⌈If irregular events (a hardware reset or a power cycle) occur while the ECU is in Halt or Poll, the ECU Manager module shall restart the ECU in the STARTUP phase.⌋*()*

7.5.5 Activities in the WakeupRestart Sequence

WakeupRestart[6]		
Wakeup Activity	**Comment**	**Opt.**
Restore MCU normal mode	Selected MCU mode is configured in the configuration parameter EcuMNormalMcuModeRef	
Get the pending wakeup sources		
Callout `EcuM_DisableWakeupSources`	Disable currently pending wakeup source but leave the others armed so that later wakeups are possible.	
Callout `EcuM_AL_DriverRestart`	Initialize drivers that need restarting	
Unlock Scheduler	From this point on, all other tasks may run again.	

Table 7.6: Wakeup Restart activities

The ECU Manager module invokes the `EcuM_AL_DriverRestart` (see [SWS_EcuM_02923]) callout which is intended for re-initializing drivers. Among others, drivers with wakeup sources typically require re-initialization. For more details on driver initialization refer to section 7.3.5 Driver Initialization.

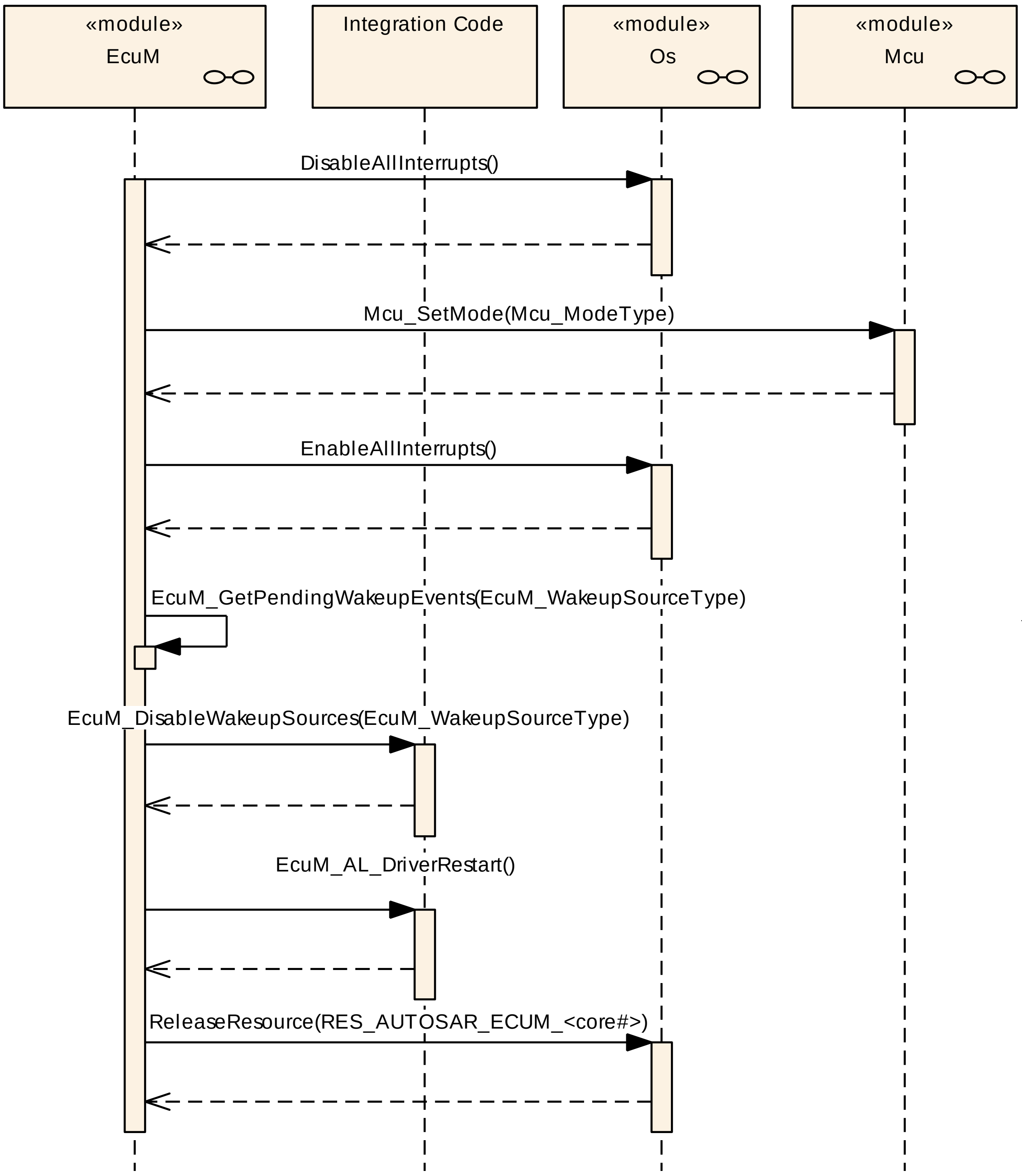
During re-initialization, a driver must check if one of its assigned wakeup sources was the reason for the previous wakeup. If this test is true, the driver must invoke its 'wakeup detected' callback (see the Specification of CAN Transceiver Driver [11] for example), which in turn must call the `EcuM_SetWakeupEvent` (see [SWS_EcuM_02826]) function.

The driver implementation should only invoke the wakeup callback once. Thereafter it should not invoke the wakeup callback again until it has been re-armed by an explicit function call. The driver must thus be re-armed to fire the callback again.

[SWS_EcuM_02539] ⌈If the ECU Manager module has a list of wakeup source candidates when the WakeupRestart Sequence has finished, the ECU Manager module shall validate these wakeup source candidates in `EcuM_MainFunction`.⌋*()*

See section 7.6.4 Activities in the WakeupValidation Sequence.

[SWS_EcuM_04066] ⌈


«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
EcuM_AL_DriverRestart()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)

⌋

Figure 7.14: WakeupRestart Sequence

()

7.6 UP Phase

In the UP Phase, the EcuM_MainFunction is executed regularly and it has three major functions:

- To check if wakeup sources have woken up and to initiate wakeup validation, if necessary (see 7.6.4 Activities in the WakeupValidation Sequence)
- To update the Alarm Clock timer
- Arbitrate RUN and POST_RUN requests and releases.

7.6.1 Alarm Clock Handling

See section 7.8.2 EcuM Clock Time in the UP Phase for implementation details.

[SWS_EcuM_04002] ⌈When the Alarm Clock service is present (see EcuMAlarmClockPresent) the EcuM_MainFunction shall update the Alarm Clock Timer⌋*()*

7.6.2 Wakeup Source State Handling

Wakeup source are not only handled during wakeup but continuously, in parallel to all other EcuM activities. This functionality runs in the EcuM_MainFunction fully decoupled from the rest of ECU management via mode requests.

[SWS_EcuM_04091] ⌈The wakeup sources can be in the following states:

State	Description
NONE	No wakeup event was detected or has been cleared.
PENDING	A wakeup event was detected but not yet validated.
VALIDATED	A wakeup event was detected and succesfully validated.
EXPIRED	A wakeup event was detected but validation failed.

Table 7.7: Wakeup sources

⌋*(SRS_ModeMgm_09136)*

Figure 7.15 illustrates the relationship between the wakeup source states and the conditions functions that evoke state changes. The two super-states Disabled and Validation are only shown here for clarification and better understandability.

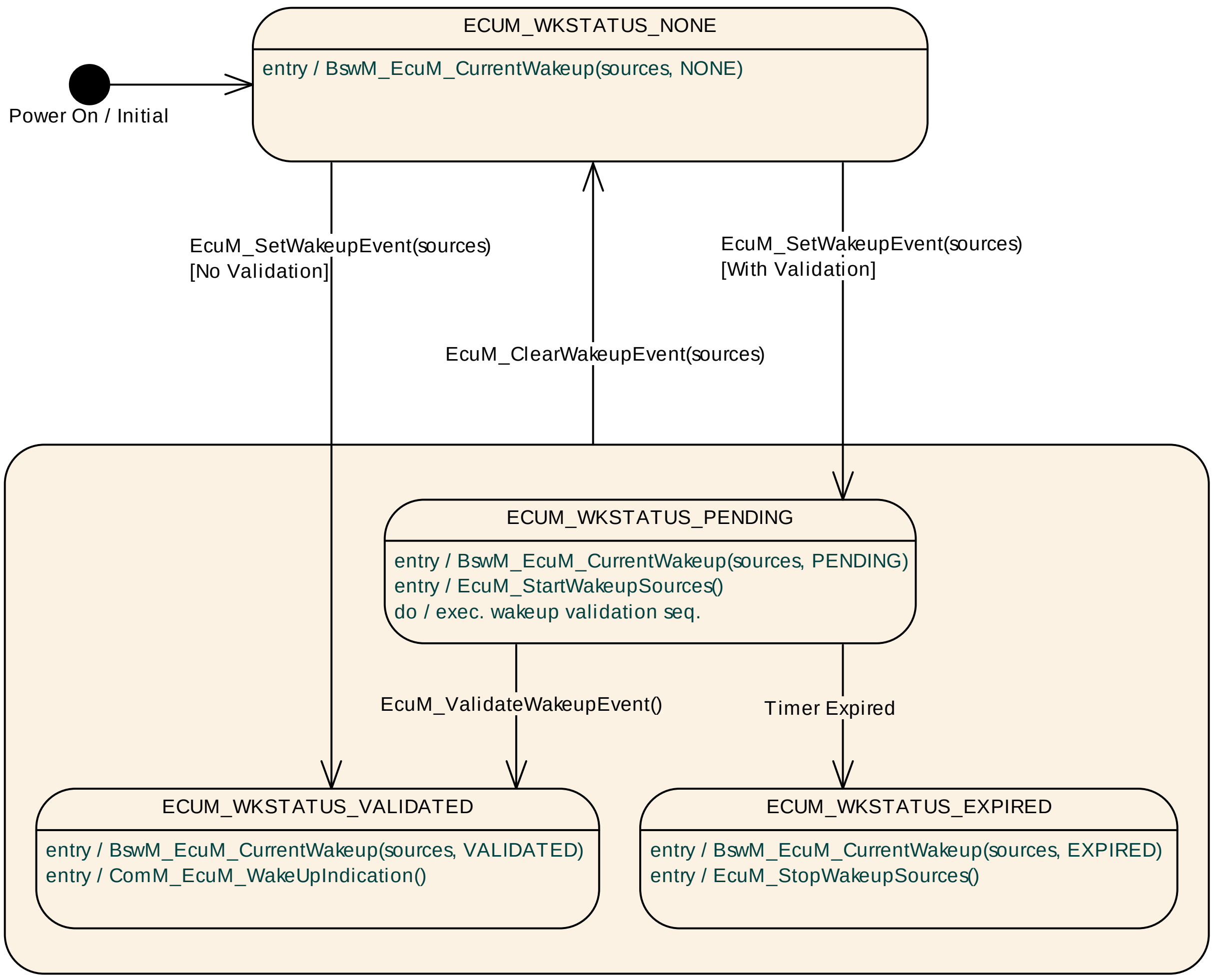

ECUM_WKSTATUS_NONE
entry / BswM_EcuM_CurrentWakeup(sources, NONE)
Power On / Initial
EcuM_SetWakeupEvent(sources)
[No Validation]
EcuM_SetWakeupEvent(sources)
[With Validation]
EcuM_ClearWakeupEvent(sources)
ECUM_WKSTATUS_PENDING
entry / BswM_EcuM_CurrentWakeup(sources, PENDING)
entry / EcuM_StartWakeupSources()
do / exec. wakeup validation seq.
EcuM_ValidateWakeupEvent()
Timer Expired
ECUM_WKSTATUS_VALIDATED
entry / BswM_EcuM_CurrentWakeup(sources, VALIDATED)
entry / ComM_EcuM_WakeUpIndication()
ECUM_WKSTATUS_EXPIRED
entry / BswM_EcuM_CurrentWakeup(sources, EXPIRED)
entry / EcuM_StopWakeupSources()

Figure 7.15: Wakeup Source States

[SWS_EcuM_04003] ⌈When an ECU Manager action causes the state of a wakeup source to change, the ECU Manager module shall issue a mode request to the BswM to change the wakeup source's mode to the new the wakeup source state.⌋*()*

For the communication of these wakeup source states the type `EcuM_WakeupStatusType` (see SWS_ECUM_04041) is used.

When the ECU Manager module is in the UP phase, wakeup events do not usually trigger state changes. They trigger the end of the Halt and Poll Sub-Phases, however. The ECU Manager module then executes the WakeupRestart Sequence automatically and returns thereafter to the UP phase.

It is up to the integrator to configure rules in the BswM so that the ECU reacts correctly to the wakeup events, as the reaction depends fully on the current ECU (not ECU Management) state.

If the wakeup source is valid, the BswM returns the ECU to its RUN state. If all wakeup events have gone back to NONE or EXPIRED, the BswM prepares the BSW for SLEEP or OFF again and invokes `EcuM_GoDownHaltPoll`.

Summarizing: every pending event is validated independently (if configured) and the EcuM publishes the result as a mode request to the BswM, which in turn can trigger state changes in the EcuM.

7.6.3 Internal Representation of Wakeup States

The EcuM manager module offers the following interfaces to ascertain the state of those wakeup sources:

- EcuM_GetPendingWakeupEvents
- EcuM_GetValidatedWakeupEvents
- EcuM_GetExpiredWakeupEvents

and manipulates the state of the wakeup sources through the following interfaces

- EcuM_ClearWakeupEvent
- EcuM_SetWakeupEvent
- EcuM_ValidateWakeupEvent
- EcuM_CheckWakeup
- EcuM_DisableWakeupSources
- EcuM_EnableWakeupSources
- EcuM_StartWakeupSources
- EcuM_StopWakeupSources

The ECU Manager module can manage up to 32 wakeup sources. The state of the wakeup sources is typically represented at the EcuM interfaces named above by means of an EcuM_WakeupSourceType bitmask where the individual wakeup sources correspond to a fixed bit position. There are 5 predefined bit positions and the rest can be assigned by configuration. See section 8.2.3 EcuM_WakeupSourceType for details.

On the one hand, the ECU Manager module manages the modes of each wakeup source. On the other hand, the ECU Manager module presupposes that there are "internal variables" (i.e. EcuM_WakeupSourceType instances) that track which wakeup sources are in a particular state (especially NONE (i.e. cleared), PENDING, VALIDATED and EXPIRED). The ECU Manager module uses these "internal variables" in the respective interface definitions to define the semantics of the interface.

Whether these "internal variables" are indeed implemented is therefore of secondary importance. They are simply used to explain the semantics of the interfaces.

7.6.4 Activities in the WakeupValidation Sequence

Since wakeup events can be generated unintentionally (e.g. EVM spike on CAN line), it is necessary to validate wakeups before the ECU resumes full operation.

The validation mechanism is the same for all wakeup sources. When a wakeup event occurs, the ECU is woken up from its SLEEP state and execution resumes within the MCU_SetMode service of the MCU driver [7] . When the WakeupRestart Sequence has finished, the ECU Manager module will have a list of pending wakeup events to be validated (see [SWS_EcuM_02539]). The ECU Manager module then releases the BSW Scheduler and all BSW MainFunctions; most notably in this case, the EcuM Main Function can resume processing.

Implementation hint: Since SchM will be running at the end of the StartPostOS and WakeupRestart sequences, there is the possibility that the `EcuM_MainFunction` will initiate validation for a source whose stack has not yet been initialized. The integrator should configure appropriate modes which indicate that the stack is not available and disable the `EcuM_MainFunction` accordingly (see [2]).

[7] 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.

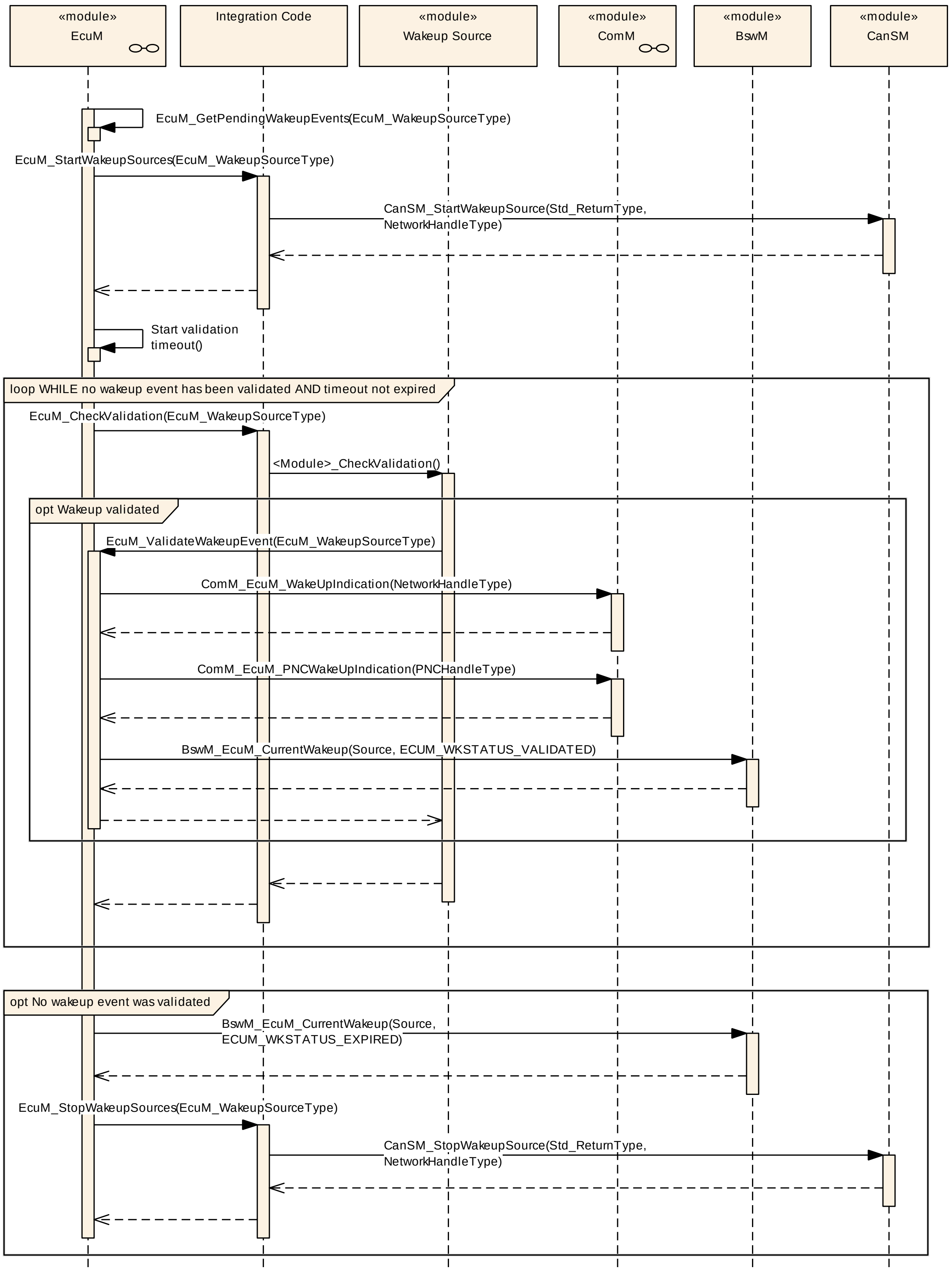

«module» EcuM
Integration Code
«module» Wakeup Source
«module» ComM
«module» BswM
«module» CanSM
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
EcuM_StartWakeupSources(EcuM_WakeupSourceType)
CanSM_StartWakeupSource(Std_ReturnType, NetworkHandleType)
Start validation timeout()
loop WHILE no wakeup event has been validated AND timeout not expired
EcuM_CheckValidation(EcuM_WakeupSourceType)
<Module>_CheckValidation()
opt Wakeup validated
EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType)
ComM_EcuM_WakeUpIndication(NetworkHandleType)
ComM_EcuM_PNCWakeUpIndication(PNCHandleType)
BswM_EcuM_CurrentWakeup(Source, ECUM_WKSTATUS_VALIDATED)
opt No wakeup event was validated
BswM_EcuM_CurrentWakeup(Source, ECUM_WKSTATUS_EXPIRED)
EcuM_StopWakeupSources(EcuM_WakeupSourceType)
CanSM_StopWakeupSource(Std_ReturnType, NetworkHandleType)

Figure 7.16: The WakeupValidation Sequence

[SWS_EcuM_02566] ⌈The ECU Manager module shall only invoke wakeup validation on those wakeup sources where it is required by configuration. If the validation protocol is not configured (see EcuMValidationTimeout), then a call to EcuM_SetWakeupEvent shall also imply a call to EcuM_ValidateWakeupEvent .⌋()

[SWS_EcuM_02565] ⌈The ECU Manager module shall start a validation timeout for each pending wakeup event that should be validated. The timeout shall be event-specific (see EcuMValidationTimeout).⌋()

Implementation hint for [SWS_EcuM_02565]: It is sufficient for an implementation to provide only one timer, which is prolonged to the largest timeout when new wakeup events are reported.

[SWS_EcuM_04081] ⌈When the validation timeout expires for a pending wakeup event, the EcuM_MainFunction sets (OR-operation) set the bit in the internal expired wakeup events variable.⌋()

See also section 7.6.3 Internal Representation of Wakeup States.

[SWS_EcuM_04082] ⌈When the validation timeout expires for a pending wakeup event, the EcuM_MainFunction shall invoke BswM_EcuM_Current_Wakeup with an EcuM_WakeupSourceType bitmask parameter with the bit corresponding to the wakeup event set and state value parameter set to ECUM_WKSTATUS_EXPIRED.⌋()

The BswM will be configured to monitor the wakeup validation through mode switch requests coming from the EcuM as the wakeup sources are validated or the timers expire. If the last validation timeout (see [SWS_EcuM_02565]) expires without validation then the BswM shall consider wakeup validation to have failed. If at least one of the pending events is validated then the entire validation shall have passed.

Pending events are validated with a call of EcuM_ValidateWakeupEvent (see [SWS_EcuM_02829]). 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 section 7.6.4.4 Requirements for Drivers with Wakeup Sources .

7.6.4.1 Wakeup of Communication Channels

If a wakeup occurs on a communication channel, the corresponding bus transceiver driver must notify the ECU Manager module by invoking EcuM_SetWakeupEvent (see [SWS_EcuM_02826]) function. Requirements for this notification are described in section 5.2 Peripherals with Wakeup Capability.

[SWS_EcuM_02479] ⌈The ECU Manager module shall execute the Wakeup Validation Protocol upon the EcuM_SetWakeupEvent (see [SWS_EcuM_02826]) function call according to *Interaction of Wakeup Sources and the ECU Manager* later in this chapter.⌋()

See also 7.6.4.2 Interaction of Wakeup Sources and the ECU Manager.

7.6.4.2 Interaction of Wakeup Sources and the ECU Manager

The ECU Manager module shall treat all wakeup sources in the same way. The procedure shall be as follows:

When a wakeup event occurs, the corresponding driver shall notify the ECU Manager module of the wakeup. The most likely modalities for this notification are:

- After exiting the Halt or Poll sequences. In this scenario, the ECU Manager module invokes `EcuM_AL_DriverRestart` (see [SWS_EcuM_02923]) to re-initialize of the relevant drivers, which in turn get a chance to scan their hardware e.g. for pending wakeup interrupts.
- If the wakeup source is actually in sleep mode, the driver must scan autonomously for wakeup events; either by polling or by waiting for an interrupt.

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

[SWS_EcuM_02976] ⌈If a wakeup 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 wakeup event is validated (either immediately or by the wakeup validation protocol), the ECU Manager module shall make the information that it is a source of the current ECU wakeup through the `EcuM_GetValidatedWakeupEvents` (see [SWS_EcuM_02830]) function.⌋*()*

7.6.4.3 Wakeup Validation Timeout

[SWS_EcuM_04004] ⌈The ECU Manager Module shall either provide a single wakeup validation timeout timer or one timer per wakeup source.⌋*()*

The following requirements apply:

[SWS_EcuM_02709] ⌈The ECU Manager module shall start the wakeup validation timeout timer when `EcuM_SetWakeupEvent` (see [SWS_EcuM_02826]) is called.⌋*()*

[SWS_EcuM_02710] ⌈`EcuM_ValidateWakeupEvent` shall stop the wakeup validation timeout timer (see [SWS_EcuM_02829]).⌋*()*

[SWS_EcuM_02712] ⌈If `EcuM_SetWakeupEvent` (see [SWS_EcuM_02826]) is called subsequently for the same wakeup source, the ECU Manager module shall not restart the wakeup validation timeout.⌋*()*

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

If `EcuM_SetWakeupEvent` (see [SWS_EcuM_02826]) is called for a wakeup source that did not yet fire during the same wakeup cycle then the ECU Manager module should prolong the validation timeout of that wakeup source.

Wakeup timeouts are defined by configuration (see `EcuMValidationTimeout`).

7.6.4.4 Requirements for Drivers with Wakeup Sources

The driver must invoke `EcuM_SetWakeupEvent` (see [SWS_EcuM_02826]) once when the wakeup event is detected and supply a `EcuM_WakeupSourceType` parameter identifying the source of the wakeup (see [SWS_EcuM_02165], [SWS_EcuM_02166]) as specified in the configuration (see `EcuMWakeupSourceId`).

[SWS_EcuM_02572] ⌈The ECU Manager module shall detect wakeups that occurr prior to driver initialization, both from Halt/Poll or from OFF.⌋*()*

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

The driver should enable the callback invocation in its initialization function.

7.6.5 Requirements for Wakeup Validation

If the wakeup 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` (see [SWS_EcuM_02829]) function.

[SWS_EcuM_02601] ⌈If the EcuM cannot determine the reset reason returned by the Mcu driver, then the EcuM set a wakeup event for default wakeup source ECUM_WKSOURCE_RESET instead.⌋*()*

7.6.6 Wakeup Sources and Reset Reason

The ECU Manager module API only provides one type (`EcuM_WakeupSourceType` , see 8.2.3 `EcuM_WakeupSourceType`), which can describe all reasons why the ECU starts or wakes up.

[SWS_EcuM_02625] ⌈The ECU Manager module shall never invoke validation for the following wakeup sources:

- ECUM_WKSOURCE_POWER
- ECUM_WKSOURCE_RESET
- ECUM_WKSOURCE_INTERNAL_RESET

- ECUM_WKSOURCE_INTERNAL_WDG
- ECUM_WKSOURCE_EXTERNAL_WDG.

⌋()

7.6.7 Wakeup Sources with Integrated Power Control

SLEEP can be realized by a system chip which controls the MCU's power supply. Typical examples are CAN transceivers with integrated power supplies which switch power off at application request and switch power on upon CAN activity.

The consequence is that SLEEP looks like OFF to the ECU Manager module on this type of hardware. This distinction is rather philosophical and not of practical importance.

The practical impact is that a passive wakeup on CAN looks like a power on reset to the ECU. Hence, the ECU will continue with the STARTUP sequence after a wakeup event. Wakeup validation is required nonetheless and the system designer must consider the following topics:

- The CAN transceiver is initialized during one of the driver initialization blocks (under BswM control 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 functions to find out if it was the CAN transceiver which started the ECU due to a passive wakeup. It is the system designer's responsibility to check the CAN transceiver for wakeup reasons and pass this information on to the ECU Manager module by using the `EcuM_SetWakeupEvent` (see [SWS_EcuM_02826]) and `EcuM_ClearWakeupEvents` (see [SWS_EcuM_02828]) functions.

These principles can be applied to all wakeup sources with integrated power control. The CAN transceiver only serves as an example.

7.7 Shutdown Targets

"Shutdown Targets" is a descriptive term for all states ECU where no code is executed. They are called shutdown targets because they are the destination states where the state machine will drive to when the UP phase is left. The following states are shutdown targets:

- Off[8]
- Sleep

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

- Reset

Note that the time at which a shutdown target is or can be determined is not necessarily the start of the shutdown. Since the BswM now controls most ECU resources, it will determine the time at which the shutdown target should be set and will set it, either directly or indirectly. The BswM must therefore ensure that, for example, the shutdown target must be changed from its default to ECUM_STATE_SLEEP before calling `EcuM_GoDownHaltPoll`.

In previous versions of the ECU Manager module, sleep targets were treated specially, as the sleep modes realized in the ECU depended on the capabilities of the ECU. These sleep modes depend on hardware 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 [10]). There are also various modalities for performing a reset which are controlled, or triggered, by different modules:

- Mcu_PerformReset
- WdgM_PerformReset
- Toggle I/O Pin via DIO / SPI

The ECU Manager module offers a facility to manage these reset modalities by to tracking the time and cause of previous resets. The various reset modalities will be treated as reset modes, using the same mode facitlities as sleep.

Refer to section 8.3.4 Shutdown Management for the shutdown management facility's interface definitions.

7.7.1 Sleep

[SWS_EcuM_02188] ⌈No wakeup event shall be missed in the SLEEP phase. The Halt or Poll Sequences shall not be entered if a wakeup event has occurred in the Go Sleep sequence.⌋*()*

[SWS_EcuM_02957] ⌈The ECU Manager module may define a configurable set of sleep modes (see `EcuMSleepMode`) where each mode itself is a shutdown target.⌋*()*

[SWS_EcuM_02958] ⌈The ECU Manager module shall allow mapping the MCU sleep modes to ECU sleep modes and hence allow them to be addressed as shutdown targets.⌋*()*

[SWS_EcuM_04092] ⌈The ShutdownTarget Sleep shall put the all cores into sleep.⌋*()*

7.7.2 Reset

[SWS_EcuM_04005] ⌈The ECU Manager module shall define a configurable set of reset modes (see EcuMResetMode and EcuM_ResetType), where each mode itself is a shutdown target. The set will minimally contain targets for

- Mcu_PerformReset
- WdgM_PerformReset
- Toggle I/O Pin via DIO / SPI

⌋*()*

[SWS_EcuM_04006] ⌈The ECU Manager module shall allow defining aliases for reset targets (See EcuM180_Conf).⌋*()*

[SWS_EcuM_04007] ⌈The ECU Manager module shall define a configurable set of reset causes (see EcuMShutdownCause and EcuM_ShutdownCauseType). The set shall minimally contain targets for

- ECU state machine entered a shutdown state
- WdgM detected a failure
- DCM requests shutdownl

and the time of the reset.⌋*()*

[SWS_EcuM_04008] ⌈The ECU Manager Module shall offer facilities to BSW modules and SW-Cs to

- Record a shutdown cause
- Get a set of recent shutdown causes

⌋*()*

See also section 8.3.4 Shutdown Management.

7.8 Alarm Clock

The ECU Manager module provides an optional persistent clock service which remains "active" even during sleep. It thus guarantees that an ECU will be woken up at a certain time in the future (assuming that the hardware does not fail) and provides clock services for long-term activities (i.e. measured in hours to days, even years).

Generally, this service will be realized with timers in the ECU that can induce wakeups. In some cases, external devices can also use a regular interrupt line to periodically wake the ECU up, however. Whatever the mechanism used, the service uses one wakeup source privately.

The ECU Manager module maintains a master alarm clock whose value determines the time at which the ECU will be woken up. Moreover the ECU manager manages an internal clock, the EcuM clock, which is used to compare with the master alarm.

Note that the alarm wakeup mechanisms are only relevant to the SLEEP phase. SW-Cs and BSW modules can set and retrieve alarm values during the UP phase (and only during the UP phase), which will be respected during the SLEEP phase, however.

Compared to other timing/wakeup mechanisms that could be implemented using general ECU Manager module facilities, the Alarm Clock service will not initiate the WakeupRestart Sequence until the timer expires. When the ECU Module detects that its timer has caused a wakeup event, it increments its timer and returns immediately to sleep unless the clock time has exceeded the alarm time.

[SWS_EcuM_04069] ⌈When the Alarm Clock service is present (see `EcuMAlarmClockPresent`) the EcuM Manager module shall maintain an EcuM clock whose time shall be the time in seconds since battery connect.⌋*()*

[SWS_EcuM_04086] ⌈The EcuM clock shall track time in the UP and SLEEP phases.⌋*()*

[SWS_EcuM_04087] ⌈Hardware permitting, the EcuM clock time shall not be reset by an ECU reset.⌋*()*

[SWS_EcuM_04088] ⌈There shall be one and only one wakeup source assigned to the EcuM Clock (see `EcuMAlarmWakeupSource`).⌋*()*

7.8.1 Alarm Clocks and Users

SW-Cs and BSW modules can each maintain an alarm clock (user alarm clock). Each user alarm clock (see `EcuMAlarmClock`) is associated with an `EcuMAlarmClockUser` which identifies the respective SW-C or BSW module.

[SWS_EcuM_04070] ⌈Each EcuM User shall have at most one user alarm clock.⌋*()*

[SWS_EcuM_04071] ⌈An EcuM User shall not be able to set the value of another user's alarm clock.⌋*()*

[SWS_EcuM_04072] ⌈The ECU Manager module shall set always the master alarm clock value to the value of the earliest user alarm clock value.⌋*()*

This means as well that when an EcuM User issues an abort on its alarm clock and that user alarm clock determines the current master alarm clock value, the ECU Manager module shall set the master alarm clock value to the next earliest user alarm clock value.

[SWS_EcuM_04073] ⌈Only authorized EcuM Users can set the EcuM clock time (see `EcuMSetClockAllowedUsers`).⌋*()*

Rationale for [SWS_EcuM_04073]: Generally EcuM Users shall not be able to set the EcuM clock time. The EcuM clock time can be set to an arbitrary time to allow testing alarms that take days to expire.

7.8.2 EcuM Clock Time

[SWS_EcuM_04089] ⌈If the underlying hardware mechanism is tick based, the ECUM shall "correct" the time accordingly⌋*()*

7.8.2.1 EcuM Clock Time in the UP Phase

The `EcuM_MainFunction` increments the EcuM clock during the UP Phase. It uses standard OS mechanisms (alarms / counters) to derive its time. Note the difference in granularity between the counters and EcuM time, which is measured in seconds ([SWS_EcuM_04069]).

7.8.2.2 EcuM Clock Time in the Sleep Phase

There are two alternatives to increment the EcuM clock during sleep depending on which sleep mode was selected (EcuMSleepModeSuspend parameter)

Within the Halt Sequence (see 7.5.2 Activities in the Halt Sequence) the GPT Driver must be put in to a GPT_MODE_SLEEP to only configure those timer channels required for the time base. It also requires the GPT to enable the timer based wakeup channel using the Gpt_EnableWakeup API. Preferably the Gpt_StartTimer API will be set to 1 sec but if this value is not reachable the EcuM will need to be woken up more often to accumulate several timer wakeups until 1 sec has been accumulated to increment the clock value.

Within the Poll Sequence (see 7.5.3 Activities in the Poll Sequence) the EcuM clock can be periodically updated during the `EcuM_SleepActivity` function using the `EcuM_SetClock` function, assuming a notion of time is still available. The clock must only be incremented when 1 sec of time has been accumulated.

In both situations after the clock has been incremented during Sleep the ECU Manager module must evaluate if the master alarm has expired. If so the BswM will initiate a full startup or set the ECU in Sleep again.

[SWS_EcuM_04009] ⌈When leaving the Sleep state the ECU Manager Module will abort any active user alarm clock and the master alarm clock. This means that both clock induced and wakeups due to other events will result in clearing all alarms.⌋*(SRS_ModeMgm_09187)*

[SWS_EcuM_04010] ⌈User alarms and the master alarm shall be cancelled during the StartPreOS Sequence, in the WakeupRestart Sequence and the OffPreOS Sequence.⌋*(SRS_ModeMgm_09188)*

7.9 MultiCore

The distribution of BSW modules onto different partitions was introduced.

A partition can be seen as an independent section that is mapped on one core. So every core (both in single and in multi core architectures) contains at least one but also can contain arbitrary numbers of partitions. But no partition can span over more than one core.

The BSW modules can be distributed over different partitions and therefore over different cores. Some BSW modules as the BswM have to be included into every partition. Other modules like the OS or the EcuM have be included into one partition per core.

An example is shown in Figure 7.17.

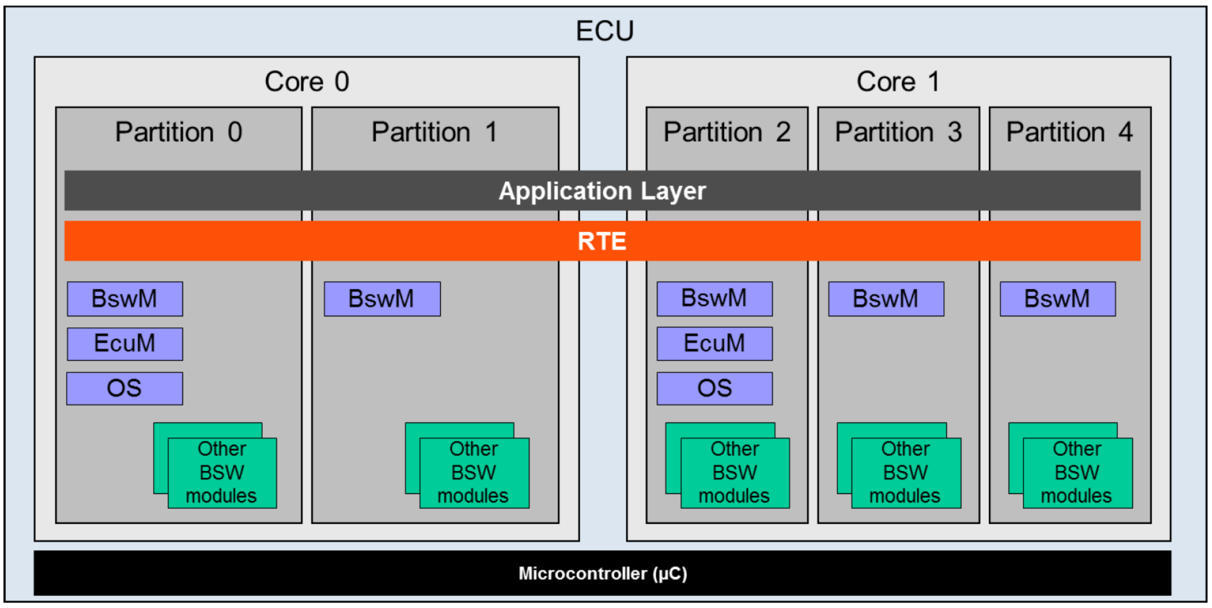


Figure 7.17: Partitions inside an ECU

In a multi core architecture the EcuM has to be distributed in a way, that one instance per core exists.

There is one designated master core in which the boot loader starts the master EcuM via `EcuM_Init`. The master EcuM starts some drivers, determines the Post Build configuration and starts all remaining cores with all their satellite EcuMs.

Each EcuM now starts the core local OS and all core local BswMs (in every partition resides exactly one BswM).

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.

The ECU Manager module supports the same phases on a MultiCore ECU as are available on conventional ECUs (i.e. STARTUP, UP, SHUTDOWN and SLEEP).

If safety mechanisms are used, The ECU State Manager has to run with full trust level.

This section uses previous ECU Manager terms for various ECU states, notably Run/PostRun. With flexible ECU management, the system integrator determines the ECU's states' names and semantics. Methods to ensure a de-initialization phase must be upheld, however. The names used here are therefore not normative.

7.9.1 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 is 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.

7.9.2 Slave Core

On every slave core, one satellite EcuM has to run. If a core contains more than one partition, only on EcuM per core has to exist.

7.9.3 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.

7.9.3.1 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.

Refer to the Guide to Mode Management [23] for a more complete description of the solution approaches and for a discussion of the considerations in choosing between them.

7.9.3.2 Example for Shutdown Synchronization

Before calling `ShutdownAllCores`, the "master" ECU Manager Module 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. The EcuM activates afterwards tasks for every configured slave core. The slave modules read the flag inside the main routine and shutdown if requested. The task name is "EcuM_SlaveCore<X>_Task", where X is a number. The task need to be configured by the integrator. The number of tasks which need to be activated can be calculated by counting the instances of EcuMPartitionRef minus one, because one EcuMFlexPartionRef is used for the master.

Example: Three instances of EcuMPartitionRef are configured. Then during call of `EcuM_GoDownHaltPoll`() "EcuM_SlaveCore1_Task" and "EcuM_SlaveCore2_Task" would be started. 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 18 illustrates how this applies to the problem of synchronizing the cores before calling `ShutdownAllCores` (whereby the de-initialization 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.

The WaitEvent() function can be replaced by a GetEvent() loop if the caller already has taken a resource or spinlock.

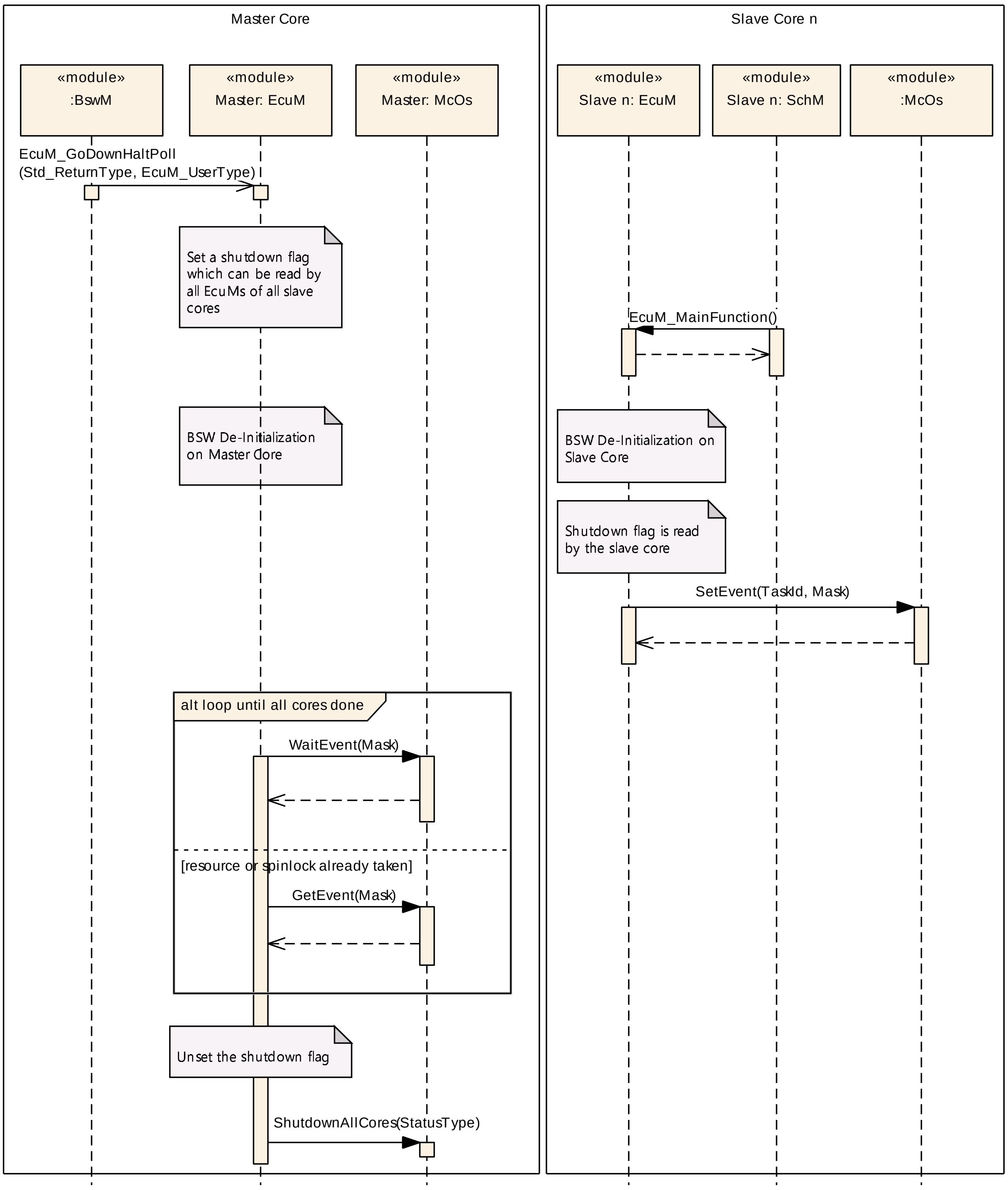

Master Core
Slave Core n
«module» :BswM
«module» Master: EcuM
«module» Master: McOs
«module» Slave n: EcuM
«module» Slave n: SchM
«module» :McOs
EcuM_GoDownHaltPoll (Std_ReturnType, EcuM_UserType)
Set a shutdown flag which can be read by all EcuMs of all slave cores
BSW De-Initialization on Master Core
alt loop until all cores done
WaitEvent(Mask)
[resource or spinlock already taken]
GetEvent(Mask)
Unset the shutdown flag
ShutdownAllCores(StatusType)
EcuM_MainFunction()
BSW De-Initialization on Slave Core
Shutdown flag is read by the slave core
SetEvent(TaskId, Mask)

Figure 7.18: Master / Slave Core Shutdown Synchronization (this is an example)

Note: Figure 7.18 is an example of the logical control flow on the master core. The API `EcuM_GoDownHaltPoll` needs to be offered on every core managed by the EcuM. The behavior of this function on slave cores is implementation specific.

Integration note: If synchronization between master and slave cores is achieved by means `SetEvent`/`WaitEvent`, then `EcuM_GoDownHaltPoll` will be called by the

BswM in the context of its main function task (deferred processing of mode arbitration). This additionally requires that the main function task is an extended task.

7.9.4 UP Phase

From the hardware perspective, it is possible that wakeup interrupts could occur on all cores. Then the whole ECU gets woken up and the EcuM running on that processes the wakeup event.

[SWS_EcuM_04011] ⌈The `EcuM_MainFunction` shall run in all EcuM instances.⌋*()*

[SWS_EcuM_04012] ⌈Each instance of the ECU Manager module shall process the wakeup events of its core.⌋*()*

As in the single-core case, the BswM (as configured by the integrator) has the responsibility for controlling ECU resources, establishing that the local core can be powered down or halted as well as de-initializing the appropriate applications and BSW before handing control over to the EcuM of its core.

7.9.5 STARTUP Phase

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.

[SWS_EcuM_04146] ⌈If `EcuMEcucCoreDefinitionRef` is missing then the initialization call shall only be performed on the master core.⌋*()*

Note: If you need to initialize a module on multiple cores you have to add the module for each core to the specific initialization list. Please be aware that in such cases the init() function might be called in parallel from different cores and init() functions are normally defined to be non-reentrant.

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.

StartPostOS is executed on each core and the SchM is initialized on each core. All core local BswMs are initialized by each EcuM.

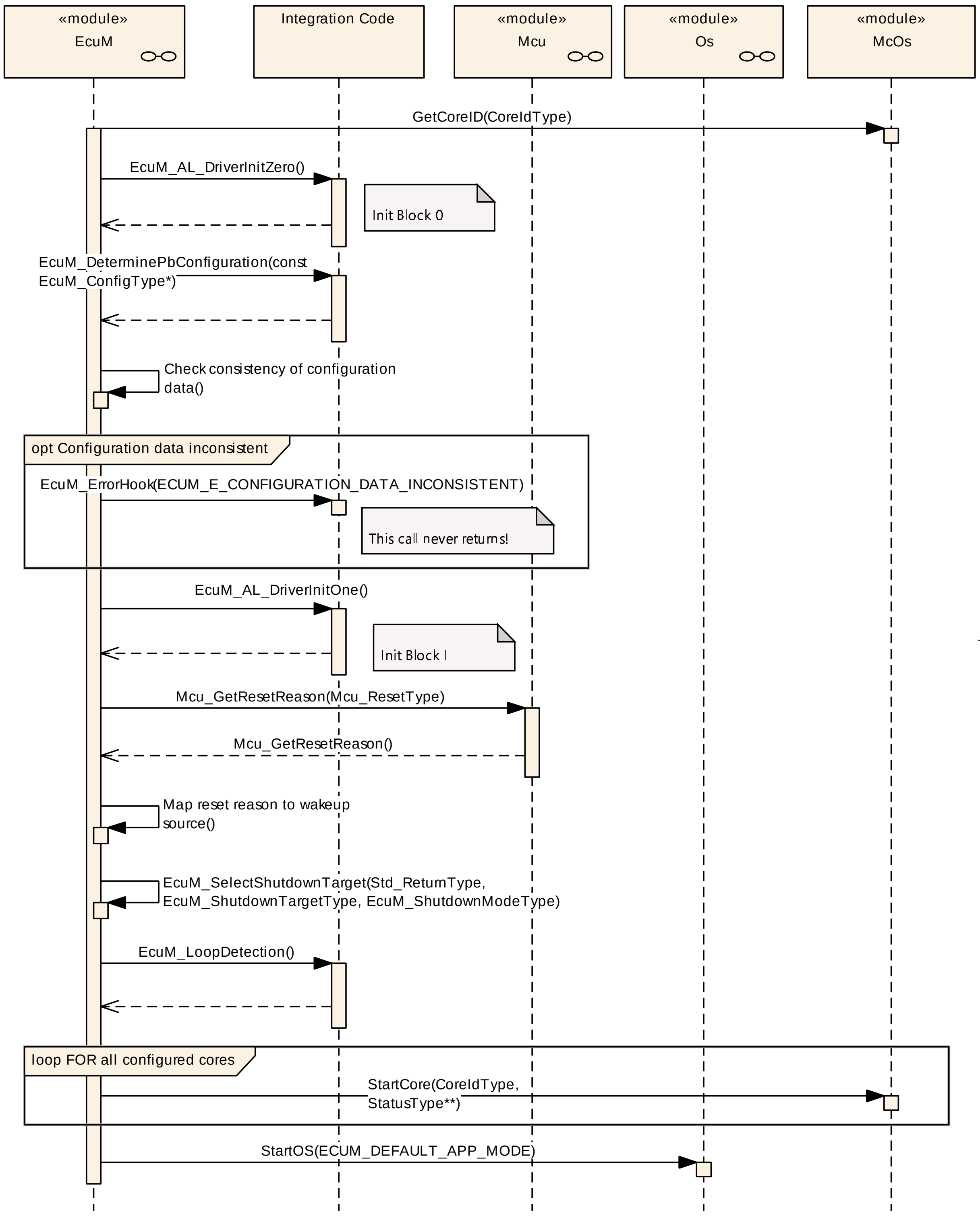
One BswM on every partition has to start the RTE for that core.

[SWS_EcuM_04093] ⌈The ECU Manager module shall start the SchM and the OS on every core.⌋*()*

[SWS_EcuM_04014] ⌈The ECU Manager module shall call `BswM_Init` for all core local BswMs on the master and all slave cores.⌋*()*

7.9.5.1 Master Core STARTUP

[SWS_EcuM_04015] ⌈

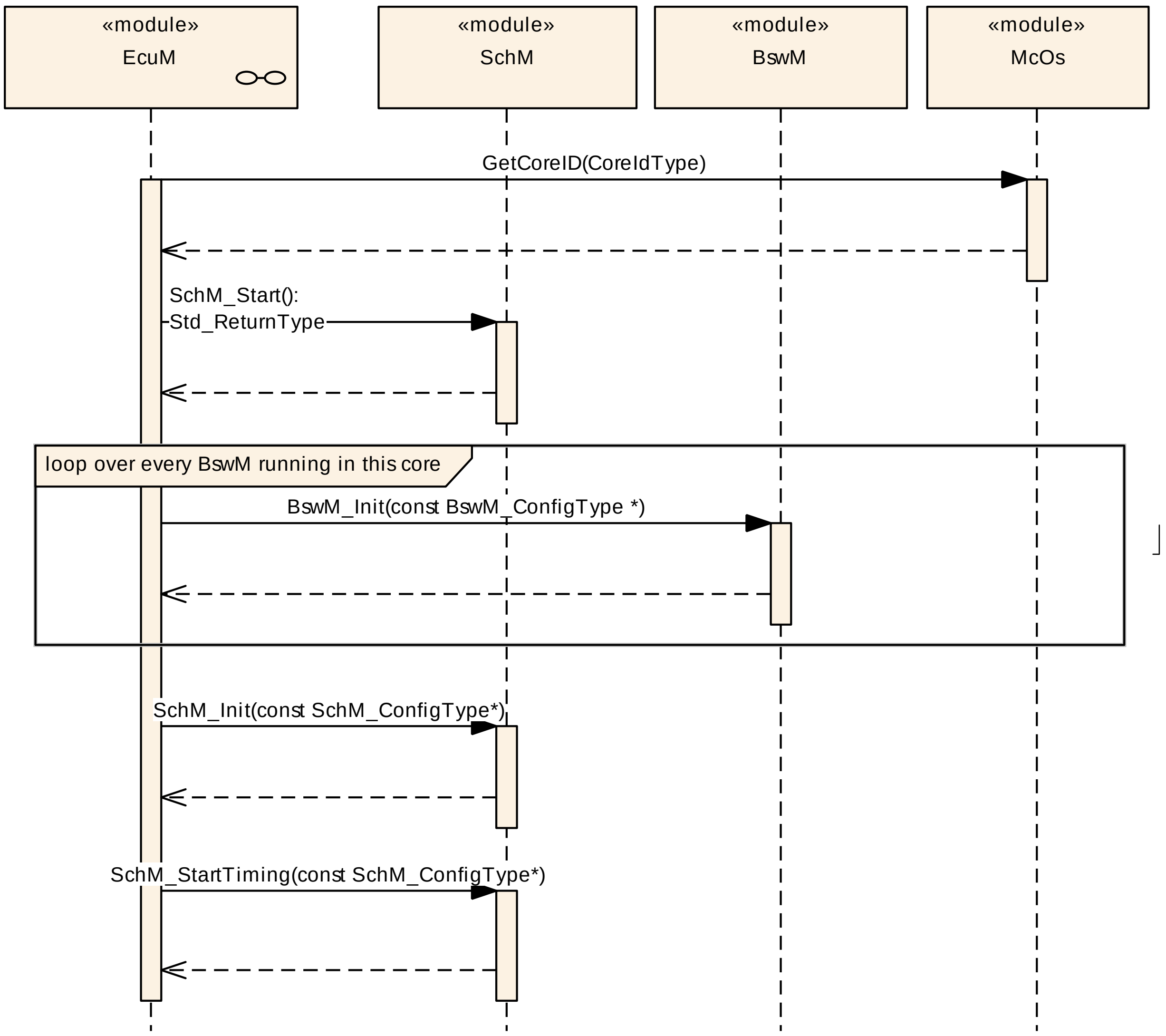

«module»
EcuM
Integration Code
«module»
Mcu
«module»
Os
«module»
McOs
GetCoreID(CoreIdType)
EcuM_AL_DriverInitZero()
Init Block 0
EcuM_DeterminePbConfiguration(const EcuM_ConfigType*)
Check consistency of configuration data()
opt Configuration data inconsistent
EcuM_ErrorHook(ECUM_E_CONFIGURATION_DATA_INCONSISTENT)
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_ShutdownTargetType, EcuM_ShutdownModeType)
EcuM_LoopDetection()
loop FOR all configured cores
StartCore(CoreIdType, StatusType**)
StartOS(ECUM_DEFAULT_APP_MODE)

⌋

Figure 7.19: Master Core StartPreOS Sequence

()

[SWS_EcuM_04016] ⌈


«module»
EcuM
«module»
SchM
«module»
BswM
«module»
McOs
GetCoreID(CoreIdType)
SchM_Start():
Std_ReturnType
loop over every BswM running in this core
BswM_Init(const BswM_ConfigType *)
SchM_Init(const SchM_ConfigType*)
SchM_StartTiming(const SchM_ConfigType*)

⌋

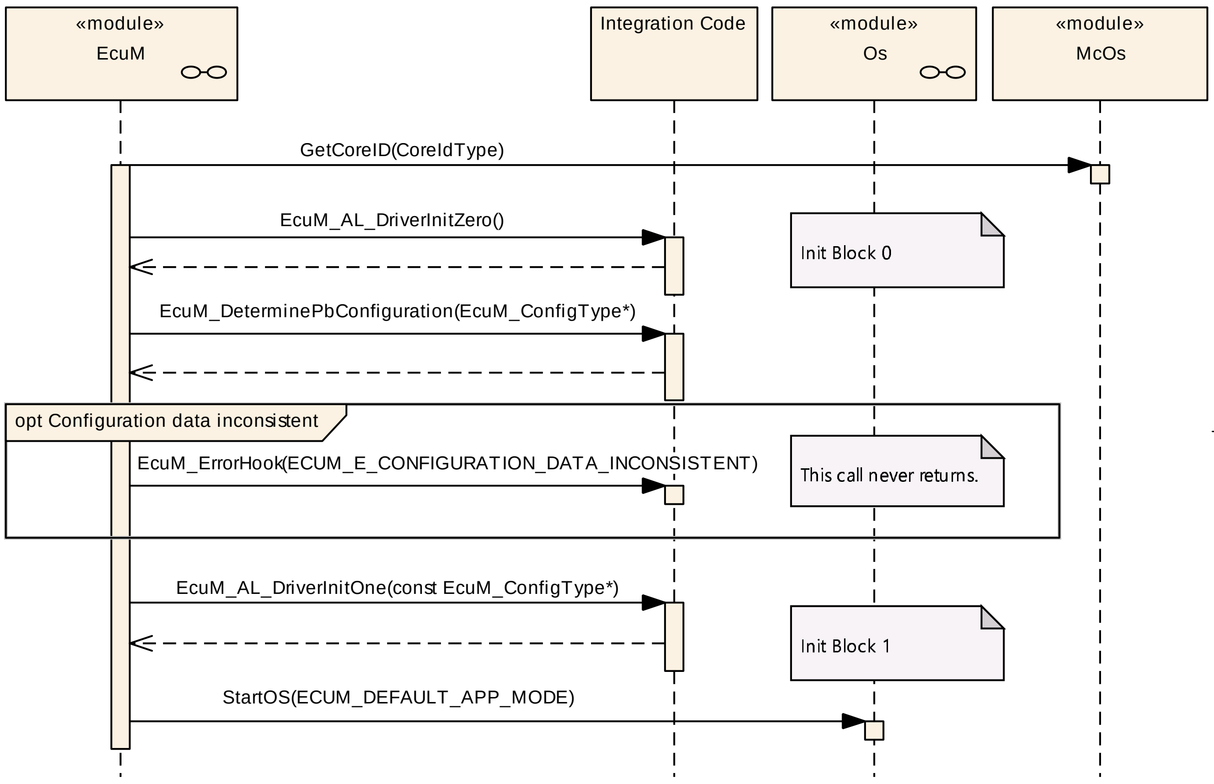
Figure 7.20: Master Core StartPostOS Sequence

()

7.9.5.2 Slave Core STARTUP

[SWS_EcuM_04145] ⌈The EcuM `EcuM_AL_DriverInitZero` and `EcuM_AL_DriverInitOne` functions shall be called by the `EcuM_Init` function on each core. The implementation of these callout functions shall ensure that only those MCAL modules are initialized that run on the currently active core.⌋*()*

[SWS_EcuM_04017] ⌈


«module»
EcuM
Integration Code
«module»
Os
«module»
McOs
GetCoreID(CoreIdType)
EcuM_AL_DriverInitZero()
Init Block 0
EcuM_DeterminePbConfiguration(EcuM_ConfigType*)
opt Configuration data inconsistent
EcuM_ErrorHook(ECUM_E_CONFIGURATION_DATA_INCONSISTENT)
This call never returns.
EcuM_AL_DriverInitOne(const EcuM_ConfigType*)
Init Block 1
StartOS(ECUM_DEFAULT_APP_MODE)

⌋

Figure 7.21: Slave Core StartPreOS Sequence

()

[SWS_EcuM_04018] ⌈

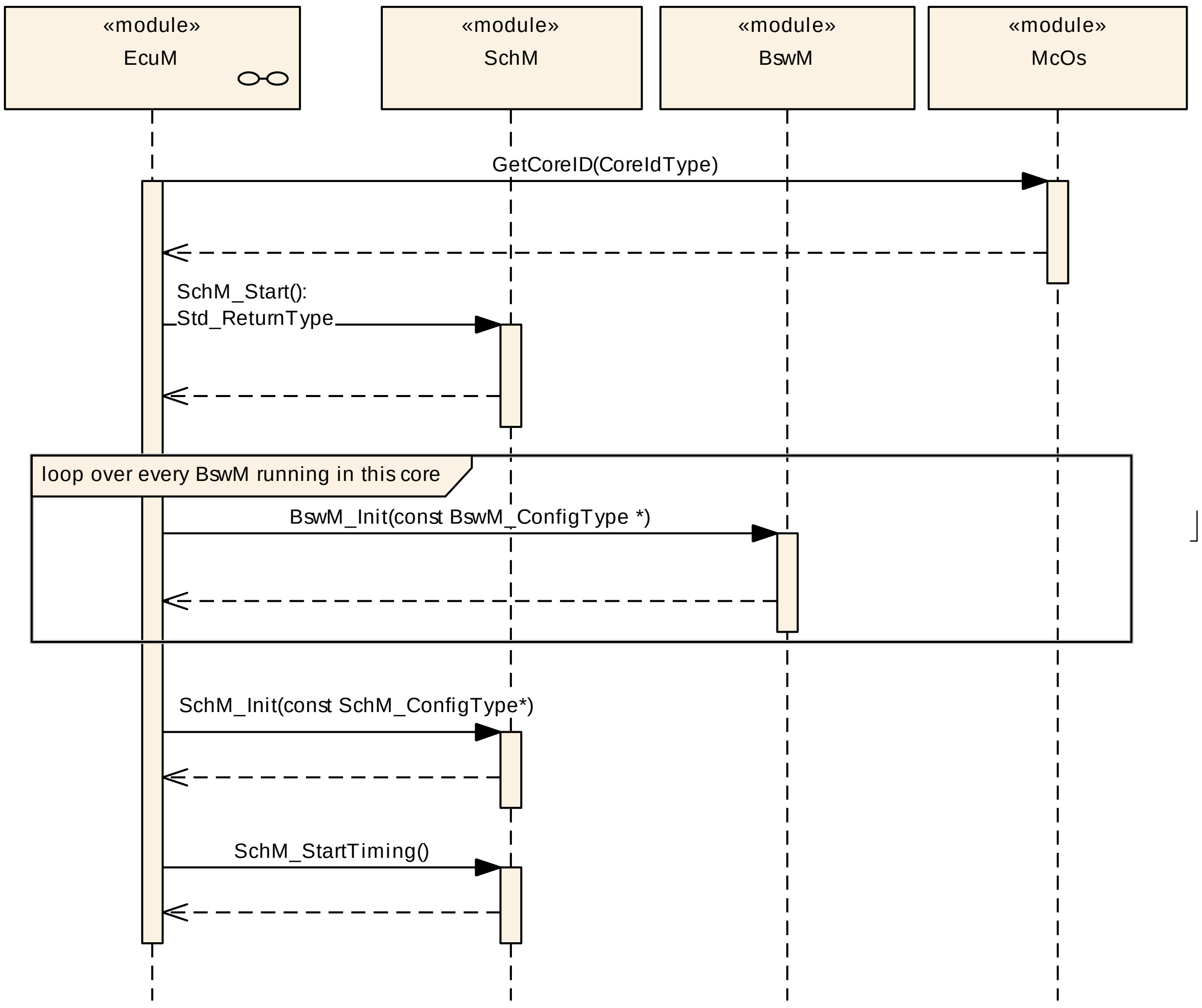


Figure 7.22: Slave Core StartPostOS Sequence

()

7.9.6 SHUTDOWN Phase

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.

The BswM (which is distributed over all partitions) ascertains that the ECU should be shut down and synchronizes with each BwsM in the ECU. All BswMs induce de-initialization of all the partition's BSWs, SWCs and CDDs and send appropriate signals to the other BswMs to indicate their readiness to shut down.

For a shutdown of the ECU, the BswM (which lies in the same partition of the master EcuM) ultimately calls GoOff on the master core which distributes that request to all slave cores. The "master" EcuM de-initializes the BswM, and the SchM. The EcuMs on the slave cores de-initialize their SchM and BswM and then send a signal to indicate that the core is ready for ShutdownOS (again, see section section 7.9.3 Master Core - Slave Core Signalling for details).

The master EcuM waits for the signal from each slave core EcuM and then initiates shutdown as usual on the master core (the master EcuM calls `ShutdownAllCores`, and the ECU is put to bed with the global shutdown hook)

7.9.6.1 Master Core SHUTDOWN

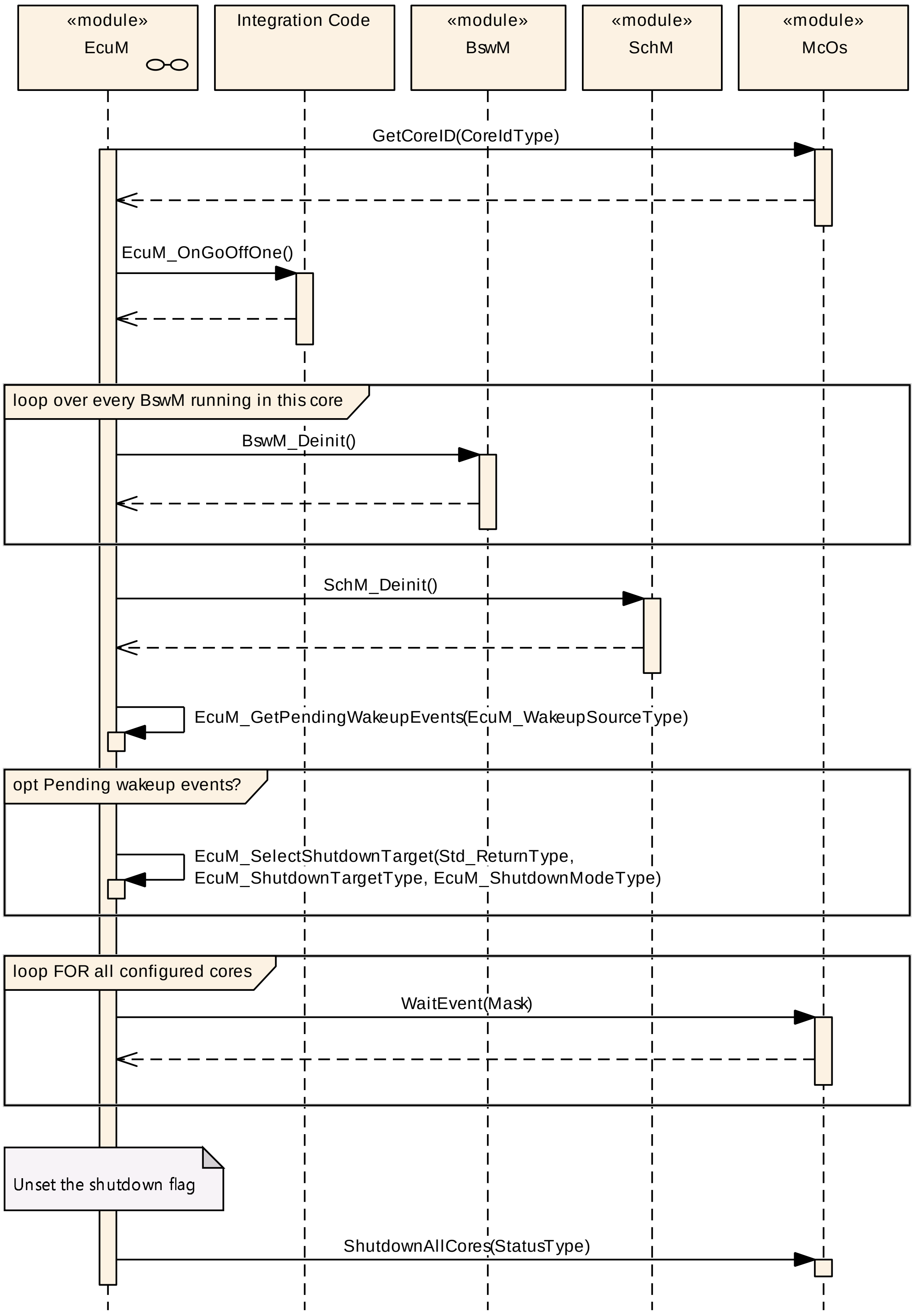
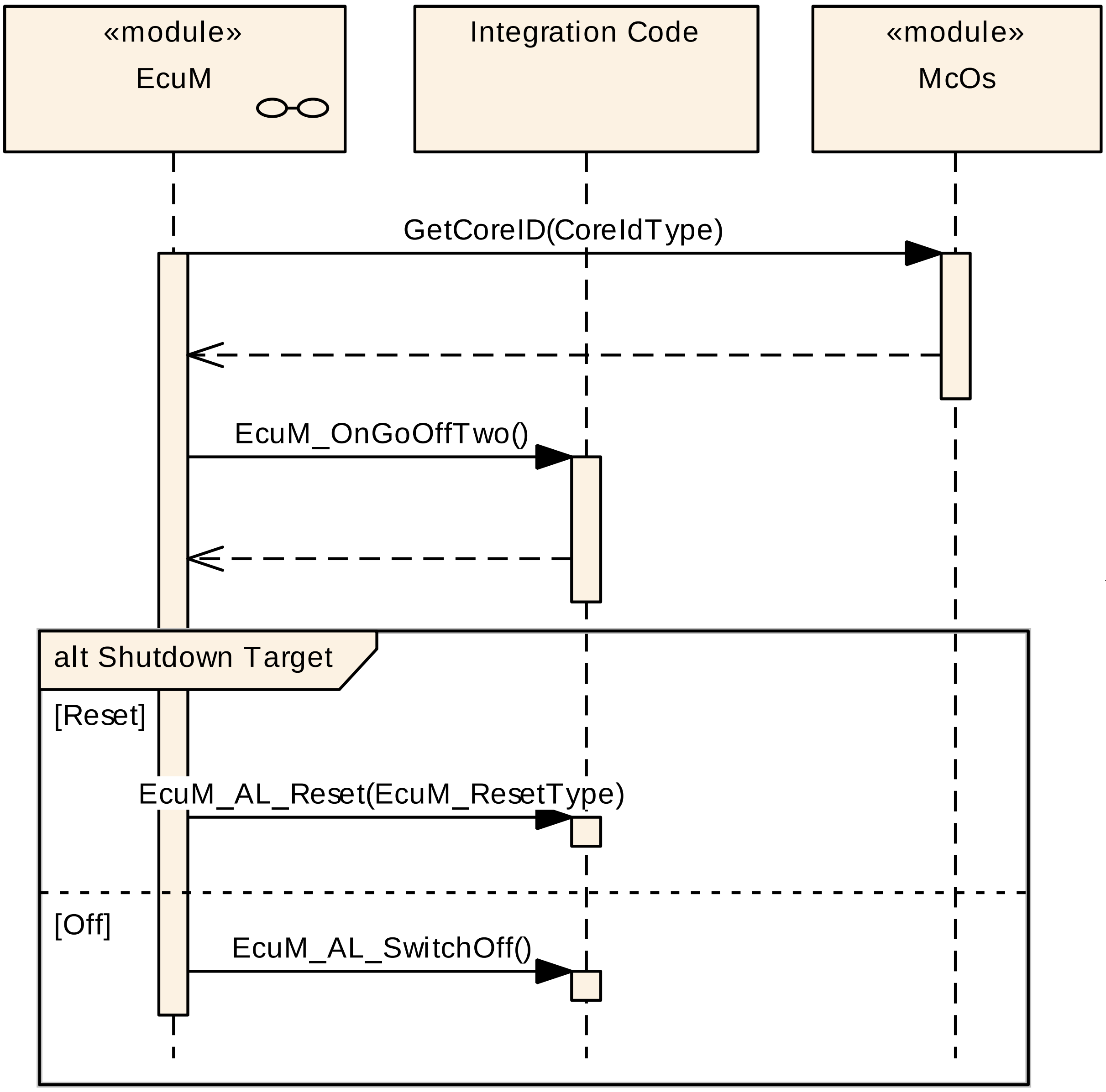

«module»
EcuM
Integration Code
«module»
BswM
«module»
SchM
«module»
McOs
GetCoreID(CoreIdType)
EcuM_OnGoOffOne()
loop over every BswM running in this core
BswM_Deinit()
SchM_Deinit()
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
opt Pending wakeup events?
EcuM_SelectShutdownTarget(Std_ReturnType,
EcuM_ShutdownTargetType, EcuM_ShutdownModeType)
loop FOR all configured cores
WaitEvent(Mask)
Unset the shutdown flag
ShutdownAllCores(StatusType)

Figure 7.23: Master Core OffPreOS Sequence

[SWS_EcuM_04020] ⌈



⌋

Figure 7.24: Master Core OffPostOS Sequence

()

7.9.6.2 Slave Core SHUTDOWN

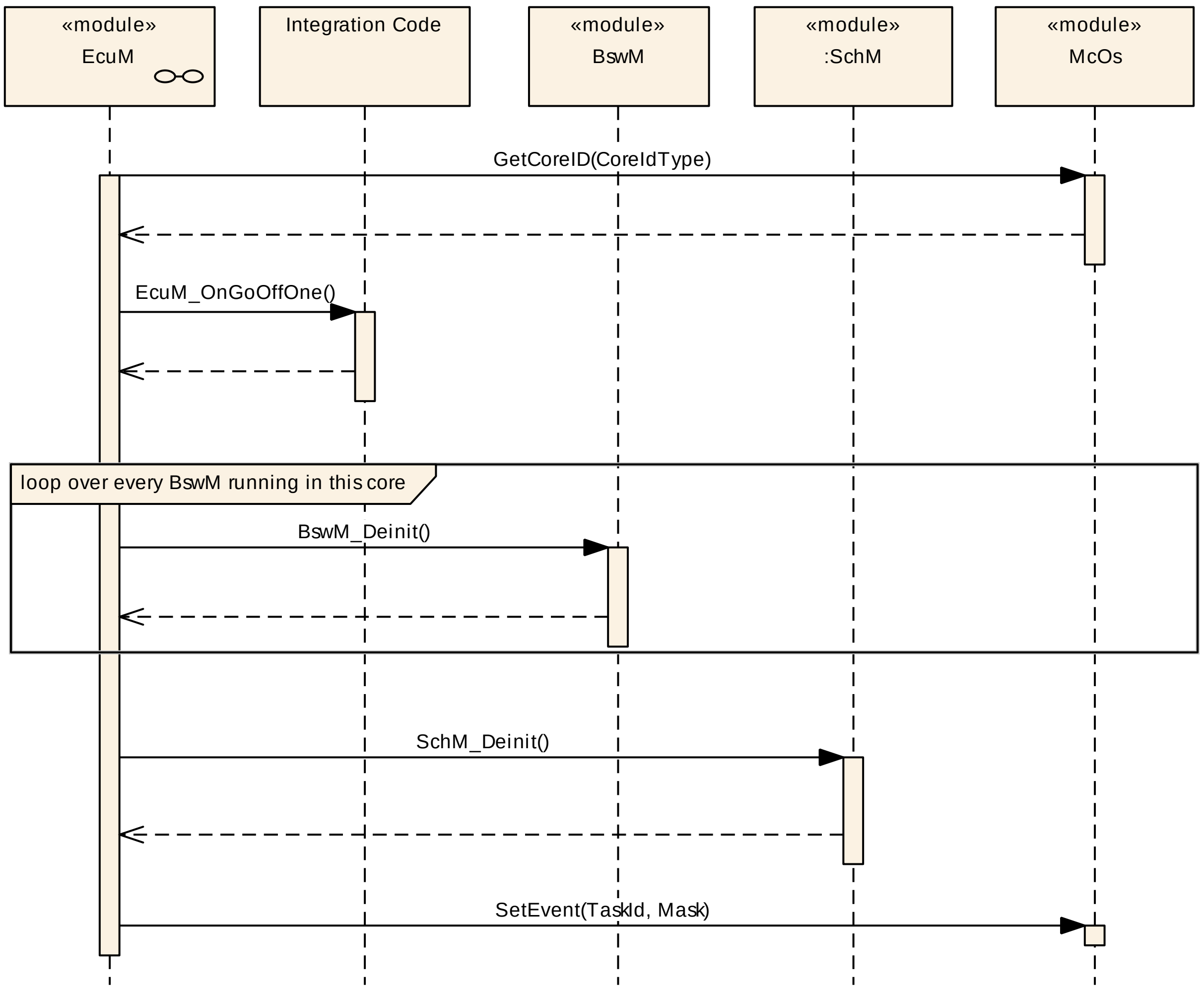


Figure 7.25: Slave Core OffPreOS Sequence

[SWS_EcuM_04022] ⌈

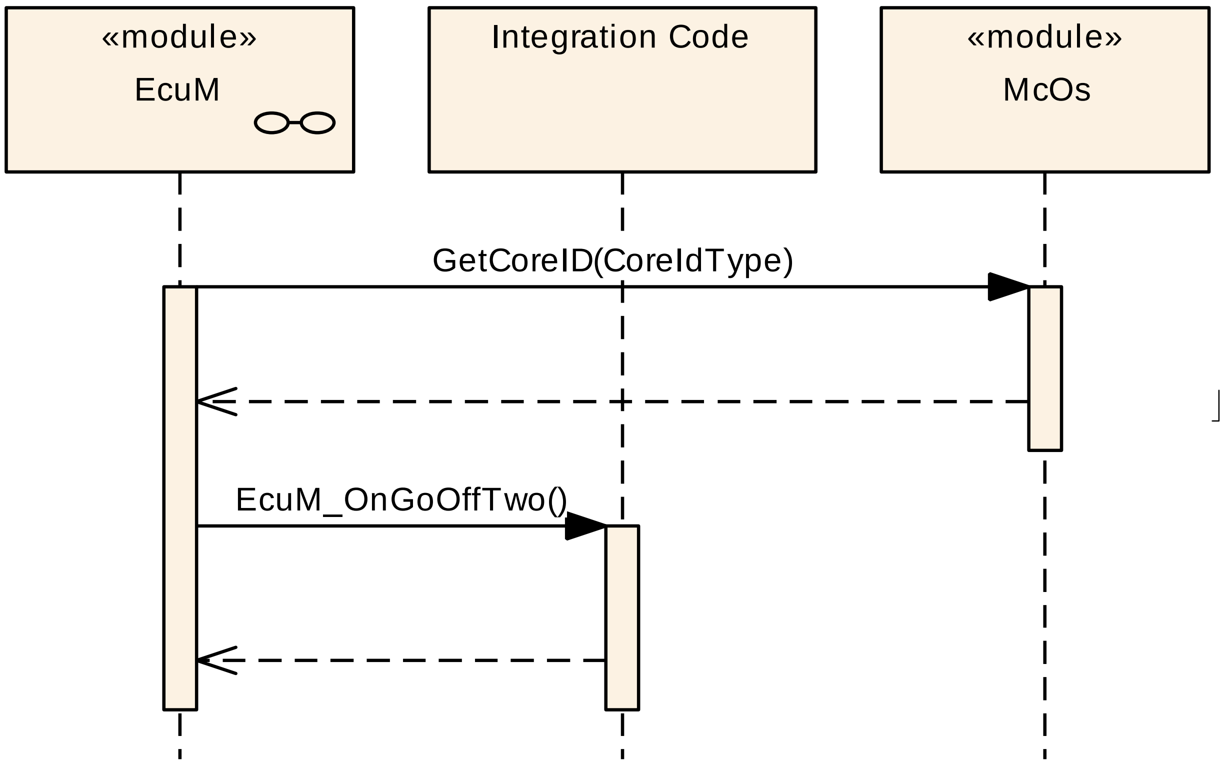


Figure 7.26: Slave Core OffPostOS Sequence

()

7.9.7 SLEEP Phase

When the shutdown target Sleep is requested, all cores are put to sleep simultaneously. The MCU must issue a halt for each core. As task timing and priority are local to a core in the OS, neither the scheduler nor the RTE must be synchronized after a halt. Because the master core could issue the MCU halt before all slave cores are finished processing, the cores must be synchronized before entering GoHalt.

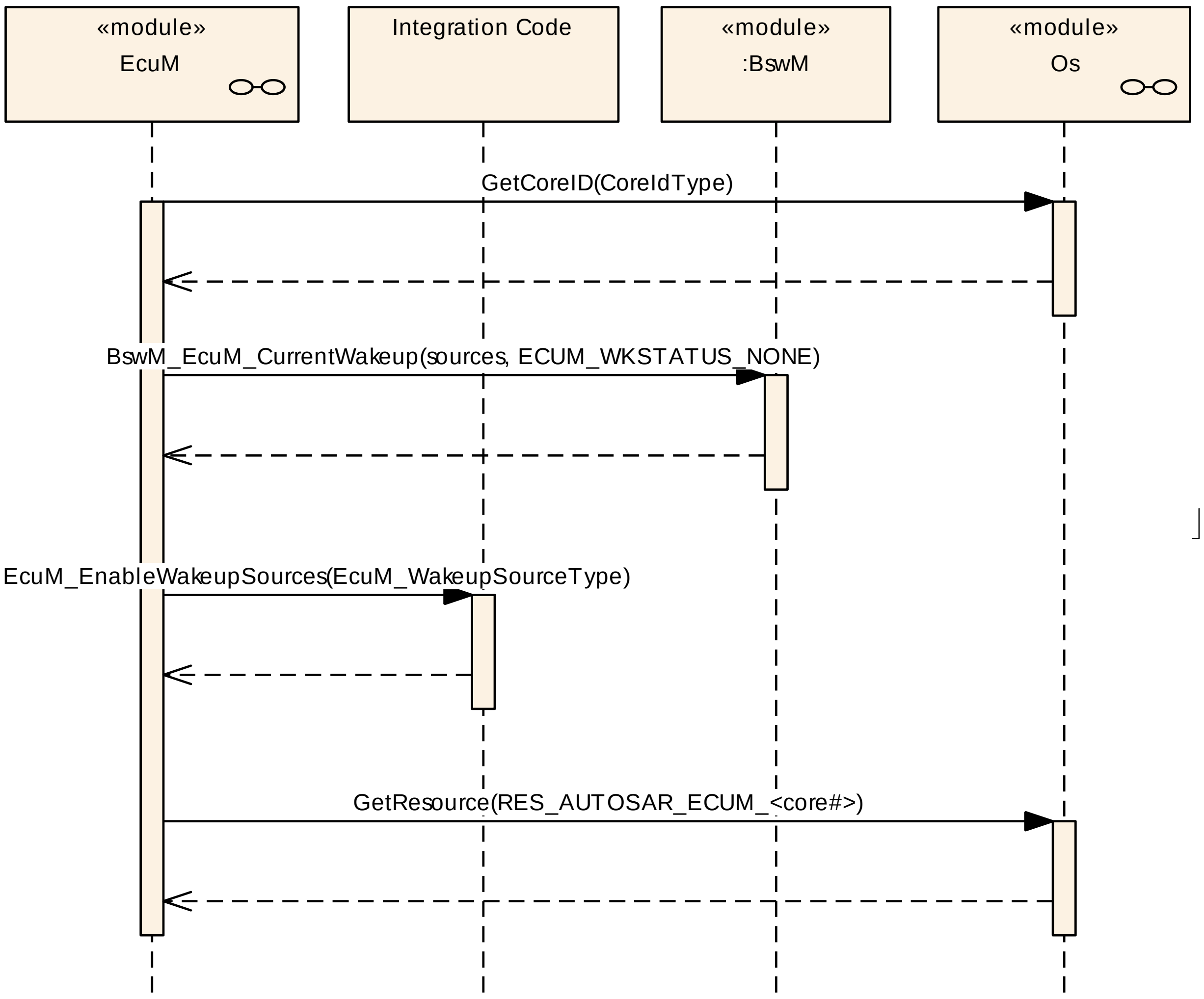
The BswMs ascertain that sleep should be initiated and distribute an appropriate ECU mode to each core. The BSWs, SWCs and CDDs on the slave cores must be informed by their partition local BswM, de-initialize appropriately and send appropriate mode requests to the BswM to indicate their readiness.

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 Go Off.

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

7.9.7.1 Master Core SLEEP

[SWS_EcuM_04023] ⌈


«module»
EcuM
Integration Code
«module»
:BswM
«module»
Os
GetCoreID(CoreIdType)
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_NONE)
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(RES_AUTOSAR_ECUM_<core#>)

⌋

Figure 7.27: Master Core GoSleep Sequence

()

[SWS_EcuM_04024] ⌈

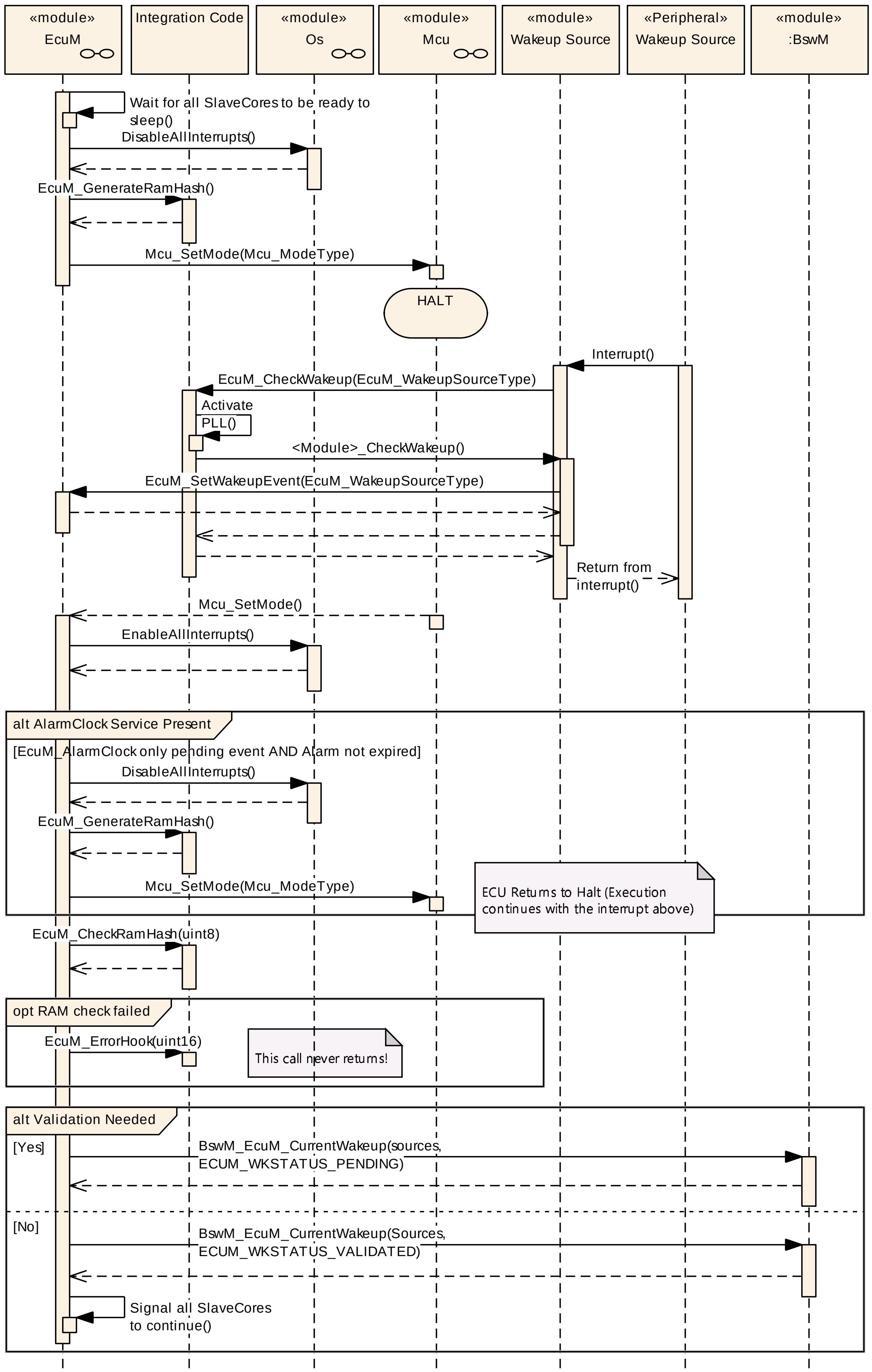
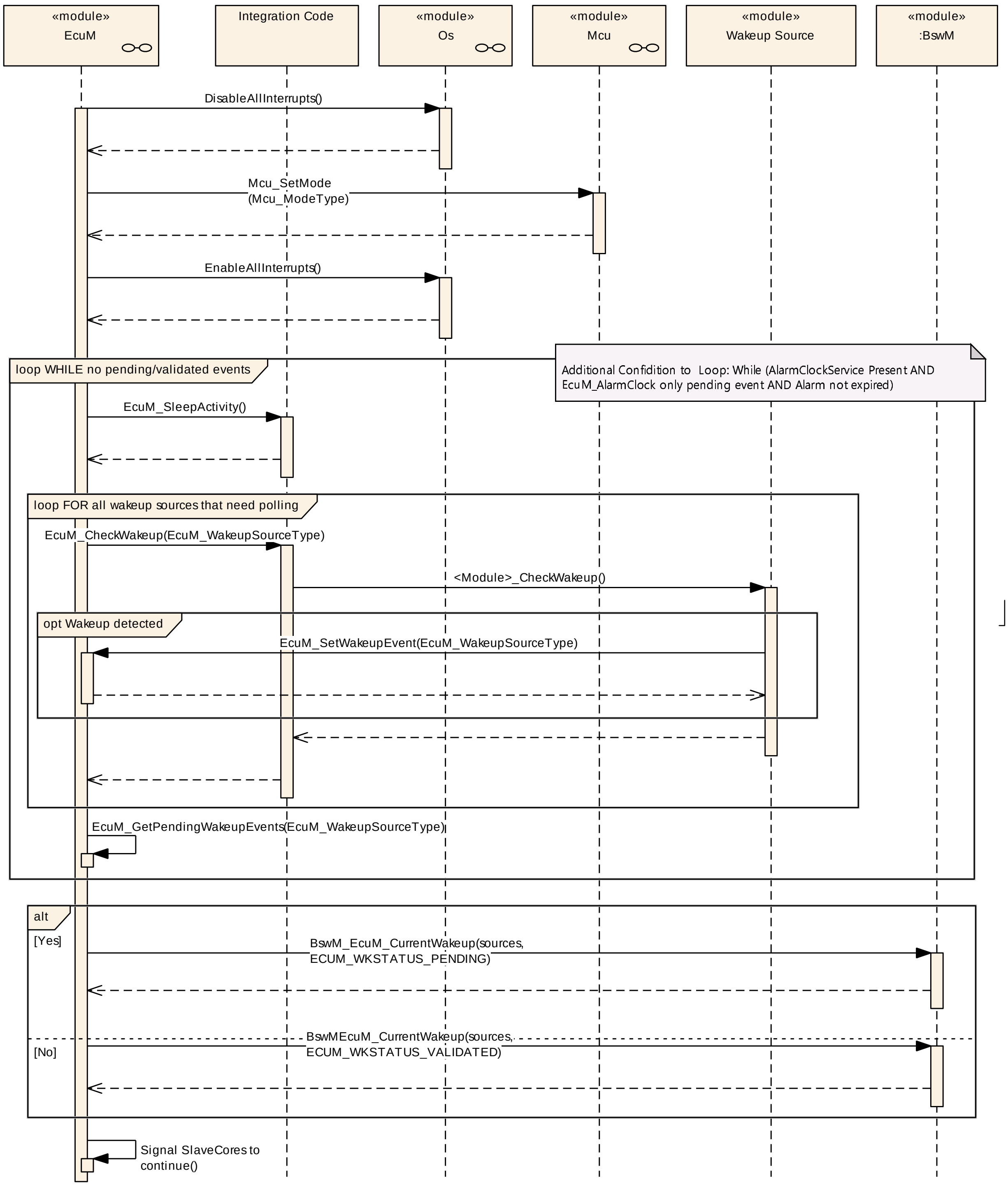

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Wakeup Source
«Peripheral» Wakeup Source
«module» :BswM
Wait for all SlaveCores to be ready to sleep()
DisableAllInterrupts()
EcuM_GenerateRamHash()
Mcu_SetMode(Mcu_ModeType)
HALT
Interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
Activate PLL()
<Module>_CheckWakeup()
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Mcu_SetMode()
EnableAllInterrupts()
alt AlarmClock Service Present
[EcuM_AlarmClock only pending event AND Alarm not expired]
DisableAllInterrupts()
EcuM_GenerateRamHash()
Mcu_SetMode(Mcu_ModeType)
ECU Returns to Halt (Execution continues with the interrupt above)
EcuM_CheckRamHash(uint8)
opt RAM check failed
EcuM_ErrorHook(uint16)
This call never returns!
alt Validation Needed
[Yes]
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_PENDING)
[No]
BswM_EcuM_CurrentWakeup(Sources, ECUM_WKSTATUS_VALIDATED)
Signal all SlaveCores to continue()

Figure 7.28: Master Core Halt Sequence

⌋ (*SRS_ModeMgm_09239*)

[SWS_EcuM_04025] ⌈

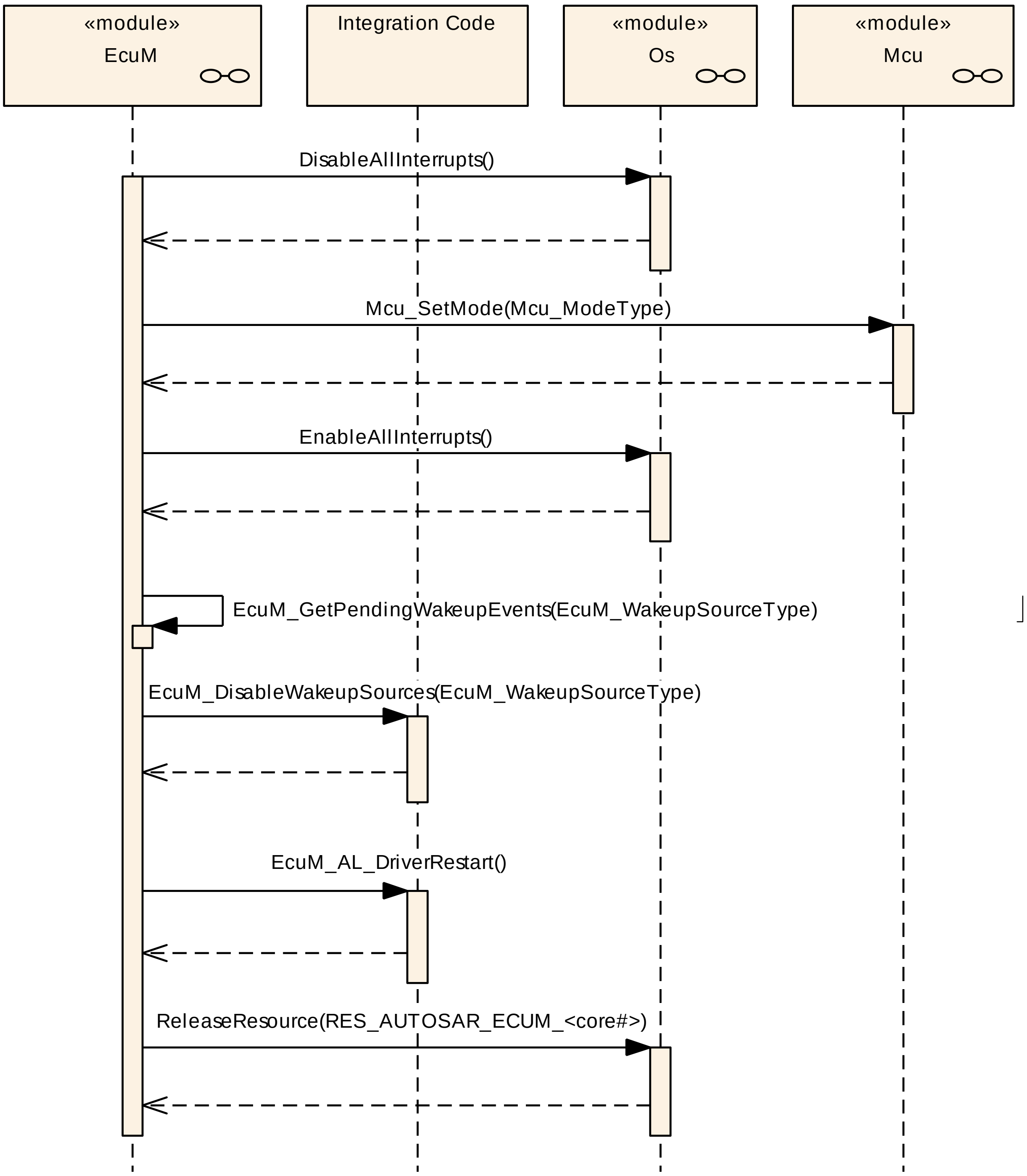

«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Wakeup Source
«module»
:BswM
DisableAllInterrupts()
Mcu_SetMode
(Mcu_ModeType)
EnableAllInterrupts()
loop WHILE no pending/validated events
Additional Condidition to Loop: While (AlarmClockService Present AND EcuM_AlarmClock only pending event AND Alarm not expired)
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)
alt
[Yes]
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_PENDING)
[No]
BswMEcuM_CurrentWakeup(sources, ECUM_WKSTATUS_VALIDATED)
Signal SlaveCores to continue()

⌋

Figure 7.29: Master Core Poll Sequence

()

[SWS_EcuM_04026] ⌈


«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_GetPendingWakeupEvents(EcuM_WakeupSourceType)
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
EcuM_AL_DriverRestart()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)

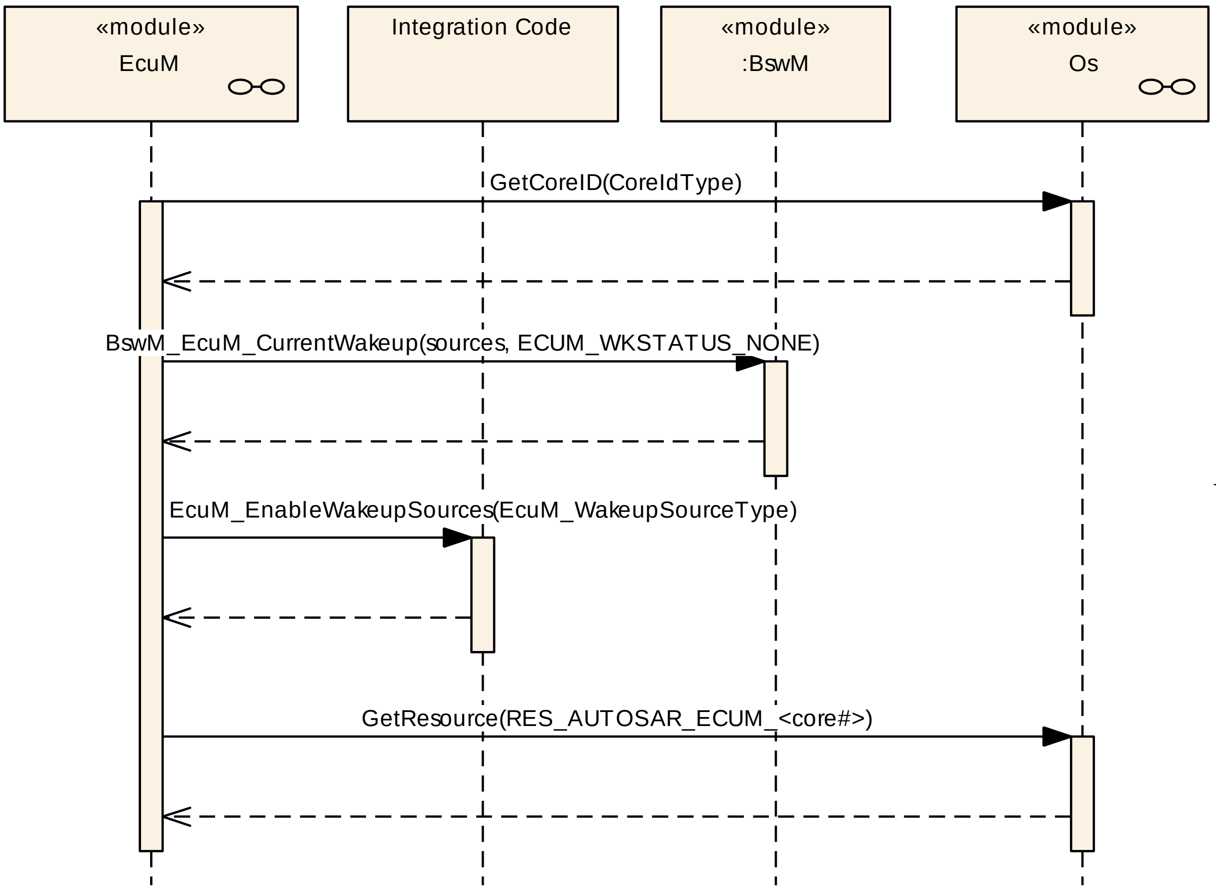
⌋

Figure 7.30: Master Core WakeupRestart Sequence

()

7.9.7.2 Slave Core SLEEP

[SWS_EcuM_04027] ⌈

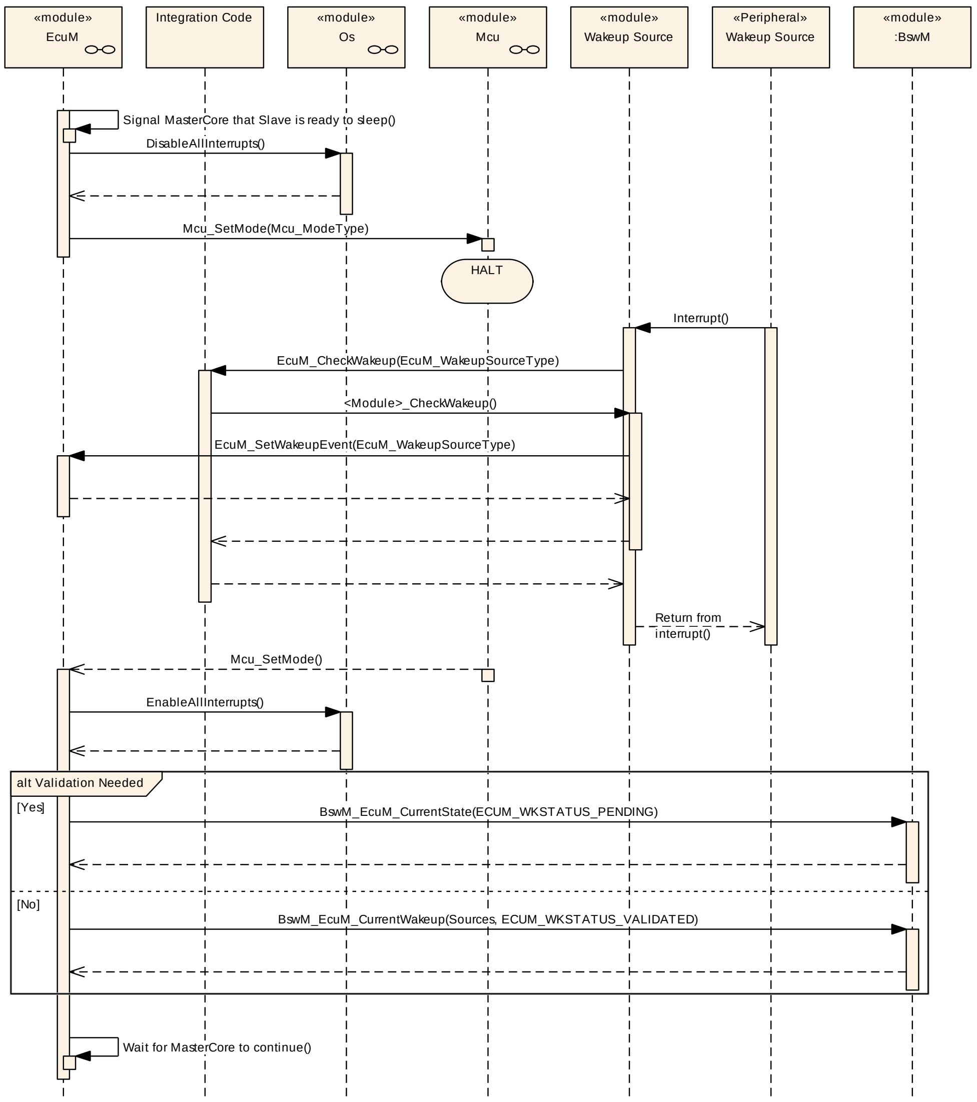

«module»
EcuM
Integration Code
«module»
:BswM
«module»
Os
GetCoreID(CoreIdType)
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_NONE)
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(RES_AUTOSAR_ECUM_<core#>)

⌋

Figure 7.31: Slave Core GoSleep Sequence

()

[SWS_EcuM_04028] ⌈

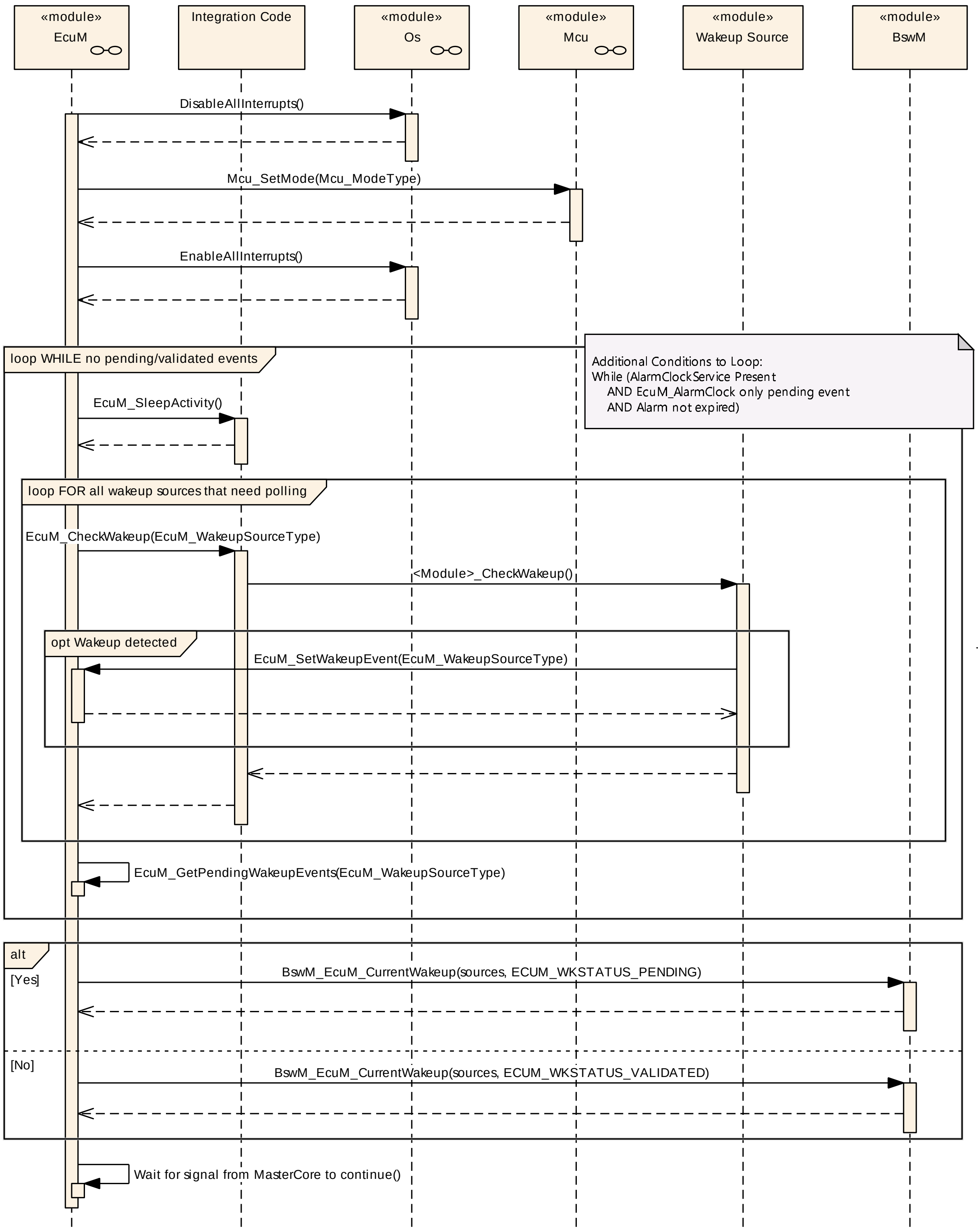

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Wakeup Source
«Peripheral» Wakeup Source
«module» :BswM
Signal MasterCore that Slave is ready to sleep()
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
<Module>_CheckWakeup()
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Mcu_SetMode()
EnableAllInterrupts()
alt Validation Needed
[Yes]
BswM_EcuM_CurrentState(ECUM_WKSTATUS_PENDING)
[No]
BswM_EcuM_CurrentWakeup(Sources, ECUM_WKSTATUS_VALIDATED)
Wait for MasterCore to continue()

⌋

Figure 7.32: Slave Core Halt Sequence

()

[SWS_EcuM_04029] ⌈

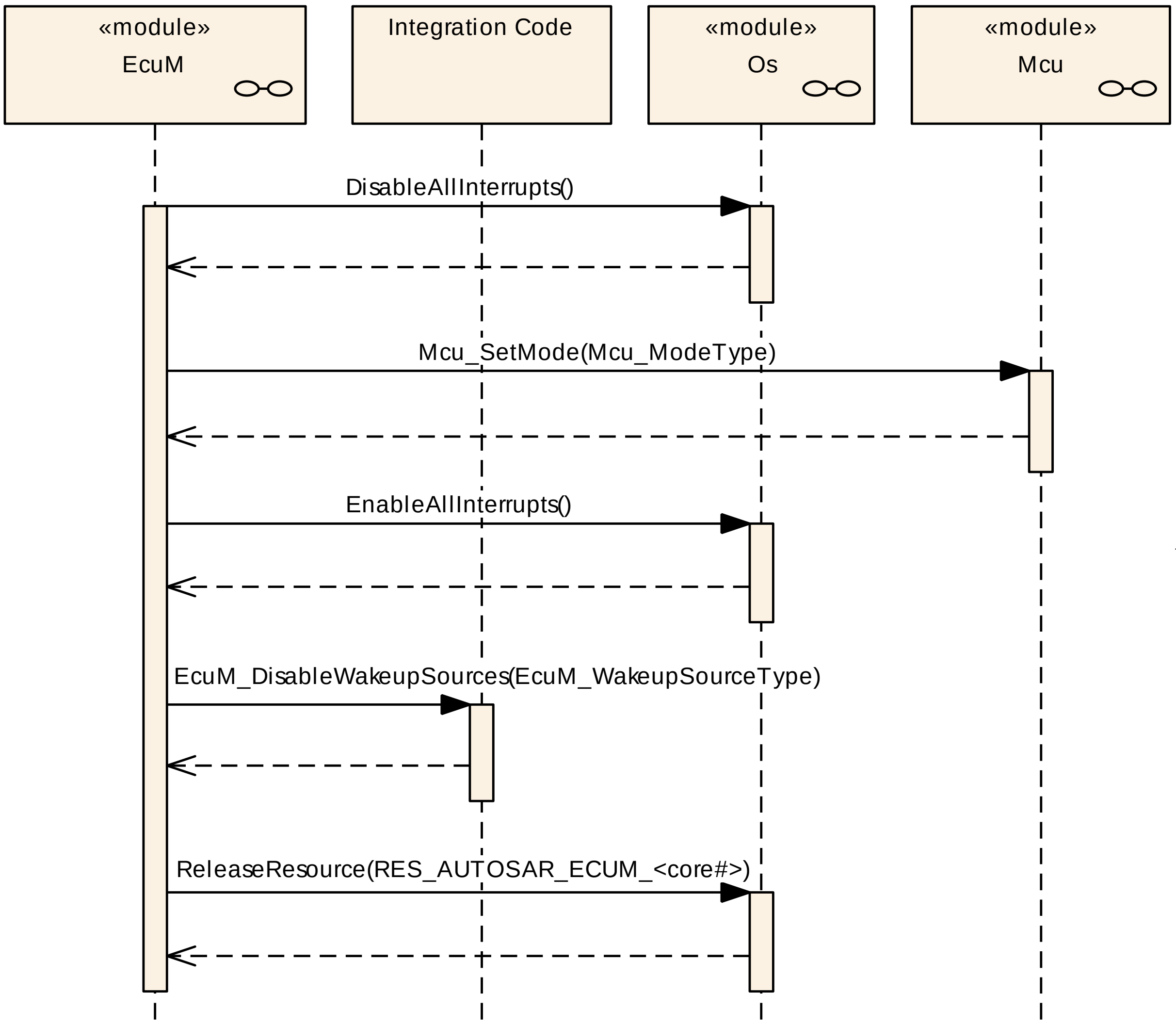

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Wakeup Source
«module» BswM
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
loop WHILE no pending/validated events
Additional Conditions to Loop:
While (AlarmClockService Present
AND EcuM_AlarmClock only pending event
AND Alarm not expired)
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)
alt
[Yes]
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_PENDING)
[No]
BswM_EcuM_CurrentWakeup(sources, ECUM_WKSTATUS_VALIDATED)
Wait for signal from MasterCore to continue()

⌋

Figure 7.33: Slave Core Poll Sequence

()

[SWS_EcuM_04030] ⌈


«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)

⌋

Figure 7.34: Slave Core WakeupRestart Sequence

()

7.9.8 Runnables and Entry points

7.9.8.1 Internal behavior

[SWS_EcuM_03018] ⌈The definition of the internal behavior of the the ECU Manager module shall be as follows. This detailed description is only needed for the configuration of the local RTE.

```
InternalBehavior EcuStateManager {

  // Runnable entities of the EcuStateManager
  RunnableEntity SelectShutdownTarget
    symbol "EcuM_SelectShutdownTarget"
    canbeInvokedConcurrently = TRUE
  RunnableEntity GetShutdownTarget
```

```
    symbol "EcuM_GetShutdownTarget"
    canbeInvokedConcurrently = TRUE
  RunnableEntity GetLastShutdownTarget
    symbol "EcuM_GetLastShutdownTarget"
    canbeInvokedConcurrently = TRUE
  RunnableEntity SelectShutdownCause
    symbol "EcuM_SelectShutdownCause"
    canbeInvokedConcurrently = TRUE
  RunnableEntity GetShutdownCause
    symbol "EcuM_GetShutdownCause"
    canbeInvokedConcurrently = TRUE
  RunnableEntity SelectBootTarget
    symbol "EcuM_SelectBootTarget"
    canbeInvokedConcurrently = TRUE
  RunnableEntity GetBootTarget
    symbol "EcuM_GetBootTarget"
    canbeInvokedConcurrently = TRUE
  RunnableEntity SetRelWakeupAlarm
    symbol "EcuM_SetRelWakeupAlarm"
    canbeInvokedConcurrently = TRUE
  RunnableEntity SetAbsWakeupAlarm
    symbol "EcuM_SetAbsWakeupAlarm"
    canbeInvokedConcurrently = TRUE
  RunnableEntity AbortWakeupAlarm
    symbol "EcuM_AbortWakeupAlarm"
    canbeInvokedConcurrently = TRUE
  RunnableEntity GetCurrentTime
    symbol "EcuM_GetCurrentTime"
    canbeInvokedConcurrently = TRUE
  RunnableEntity GetWakeupTime
    symbol "EcuM_GetWakeupTime"
    canbeInvokedConcurrently = TRUE
  RunnableEntity SetClock
    symbol "EcuM_SetClock"
    canbeInvokedConcurrently = TRUE
  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

  // 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=EcuMUser[0].User }
  (...)
  SRnnn.RequestRUN -> RequestRUN
  SRnnn.ReleaseRUN -> ReleaseRUN
  SRnnn.RequestPOSTRUN -> RequestPOSTRUN
  SRnnn.ReleasePOSTRUN -> RequestPOSTRUN
  PortArgument  {port=SRnnn, value.type=EcuM_UserType,
               value.value=EcuMUser[nnn].User }

  shutDownTarget.SelectShutdownTarget -> SelectShutdownTarget
  shutDownTarget.GetShutdownTarget -> GetShutdownTarget
  shutDownTarget.GetLastShutdownTarget -> GetLastShutdownTarget
  shutDownTarget.SelectShutdownCause -> SelectShutdownCause
  shutDownTarget.GetShutdownCause -> GetShutdownCause
  bootTarget.SelectBootTarget -> SelectBootTarget
  bootTarget.GetBootTarget -> GetBootTarget
  alarmClock.SetRelWakeupAlarm-> SetRelWakeupAlarm
  alarmClock.SetAbsWakeupAlarm -> SetAbsWakeupAlarm
  alarmClock.AbortWakeupAlarm -> AbortWakeupAlarm
  alarmClock.GetCurrentTime -> GetCurrentTime
  alarmClock.GetWakeupTime -> GetWakeupTime
  alarmClock.SetClock -> SetClock
};
```

⌋*()*

7.10 EcuM Mode Handling

The ECU State Manager provides interfaces for SW-Cs to request and release the modes RUN and POST_RUN optionally.

EcuMFlex arbitrates the requests and releases made by SW-Cs and propagates the result to BswM. The cooperation between EcuM and BswM is necessary as only the BswM can decide when a transition to a different mode can be made. Due to the fact that the EcuM does not have an own state machine, the EcuM relies on the state transitions made by BswM. Therefore the EcuM does not request a state. Furthermore it notifies the BswM about the current arbitration of all requests. And the BswM is notified when the RTE has executed all Runnables belonging to a certain mode.

ArchitecturalComponentsofECUModeHandling

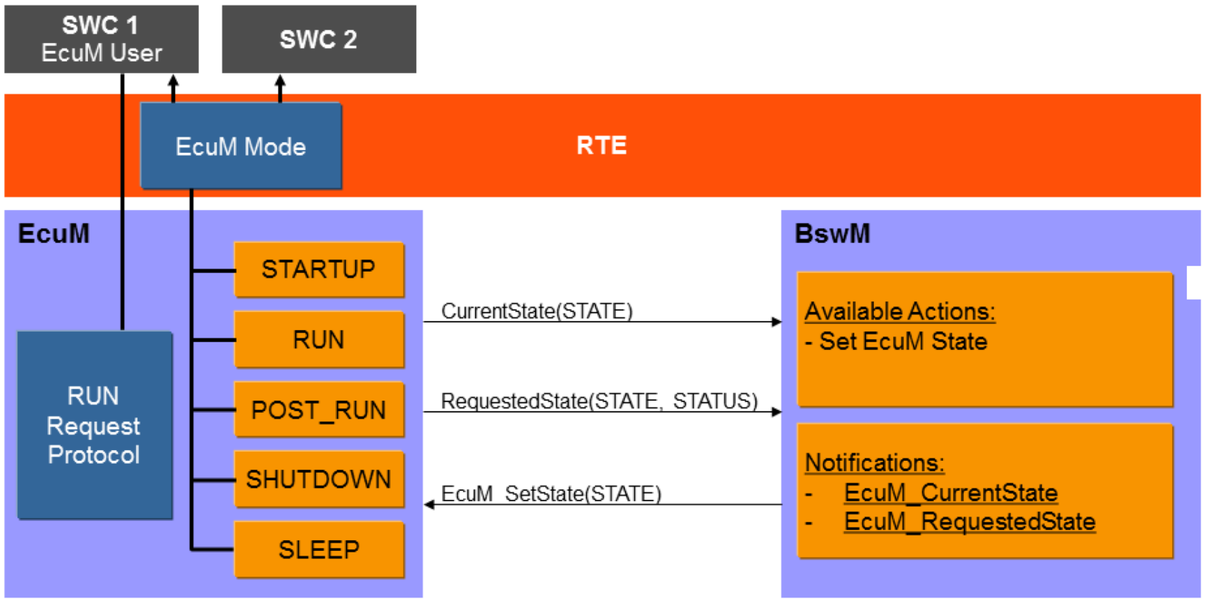


Figure 7.35: Architectural Components of ECU Mode Handling

Figure 7.35 illustrates the architectural components of ECU Mode Handling.

[SWS_EcuM_04115] ⌈ECU Mode Handling shall be applied when EcuMModeHandling is configured to true.⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_04116] ⌈When the BswM sets a state of the EcuM by EcuM_SetState, the EcuM shall indicate the corresponding mode to the RTE.⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_04117] ⌈When the last RUN request has been released, ECU State Manager module shall indicate this to BswM using the API BswM_EcuM_RequestedState(ECUM_STATE_APP_RUN, ECUM_RUNSTATUS_RELEASED).⌋*(SRS_ModeMgm_09116)*

If a SW-C needs post run activity during POST_RUN (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.

[SWS_EcuM_04118] ⌈When the ECU State Manager is not in the state which is requested by a SWC, it shall inform BswM about requested states using the BswM_EcuM_RequestedState API.⌋*(SRS_ModeMgm_09116)*

POST_RUN state provides a post run phase for SW-C's and allows them to save important data or switch off peripherals.

[SWS_EcuM_04144] ⌈When the first RUN or POST_RUN request has been received, ECU State Manager module shall indicate this to BswM using BswM_EcuM_RequestedState(ECUM_STATE_APP_RUN, ECUM_RUNSTATUS_REQUESTED).⌋*()*

[SWS_EcuM_04119] ⌈When the last POST_RUN request has been released, ECU State Manager module shall indicate this to BswM using the API BswM_EcuM_RequestedState(ECUM_STATE_APP_POST_RUN, ECUM_RUNSTATUS_RELEASED).⌋*(SRS_ModeMgm_09116)*

Hint: To prevent, that the mode machine instance of ECU Mode lags behind and the states EcuM and the RTE get out of phase, the EcuM can use acknowledgement feedback for the mode switch notification.

Note that EcuM only requests Modes from and to RUN and POST_RUN, the SLEEP Mode has to be set by BswM, as the EcuM has no information about when this Mode can be entered.

State	Description
STARTUP	Initial value. Set by Rte when Rte_Start() has been called.
RUN	As soon as all necesseray BSW modules are inistialized, BswM switches to this Mode.
POST_RUN	EcuM requests POST_RUN, when no RUN requests are available.
SLEEP	EcuM requests SLEEP Mode when no RUN and POST_RUN requests are available and Shutdown Target is set to SLEEP.
SHUTDOWN	EcuM requests SHUTDOWN Mode when no RUN and POST_RUN requests are available and Shutdown Target is set to SHUTDOWN.

Table 7.8: EcuM Modes

[SWS_EcuM_04143] ⌈EcuM shall notify BswM about the current State by calling the interface `BswM_EcuM_CurrentState(EcuM_StateType State)`. A new state shall be set by EcuM when RTE has given its feedback via the acknowledgement port.⌋*()*

7.11 Advanced Topics

7.11.1 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` .

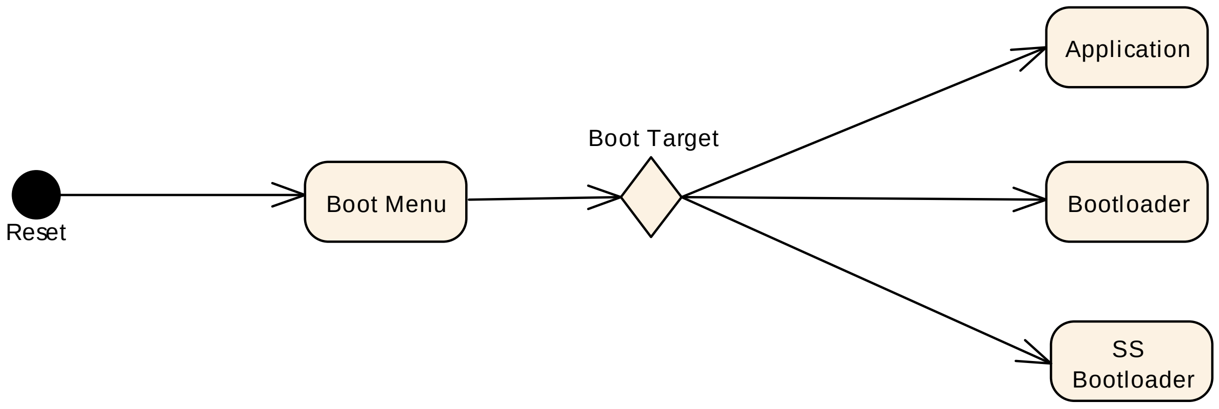


Figure 7.36: Selection of Boot Targets

Bootloader, system supplier bootloader and application are separate program images, which in many cases even can be flashed separately. The only way to get from one image to another is through reset. The boot menu will branch into the one or other image depending on the selected boot target.

7.11.2 Relation to Complex Drivers

If a complex driver handles a wakeup source, it must follow the protocol for handling wakeup events specified in this document.

7.11.3 Handling Errors during Startup and Shutdown

[SWS_EcuM_02980] ⌈The ECU Manager module shall ignore all types of errors that occur during initialization, e.g. values returned by init functions⌋*()*

Initialization is a configuration issue (see `EcuMDriverInitListZero` , `EcuM-DriverInitListOne` and `EcuMDriverRestartList`) and therefore cannot be standardized.

BSW modules are responsible themselves for reporting errors occurring during their initialization directly to the DEM module or the DET module, as specified in their SWSs. The ECU Manager module does not report the errors. The BSW module is also responsible for taking any special measures to react to errors occurring during their initialization.

7.12 ErrorHook

[SWS_EcuM_04033] ⌈In the unrecoverable error situations defined in the first column of Table in [SWS_EcuM_04032], the ECU Manager module shall call the `EcuM_ErrorHook` callout with the parameter value set to the corresponding related error code.⌋*()*

Clarification to [SWS_EcuM_04033]: EcuM shall assume that the EcuM_ErrorHook will not return (integrator's code).

Clarification to [SWS_EcuM_04033]: In case a Dem error is needed, it is integrator's responsibility to define a strategy to handle it (e.g.: As EcuM does not directly call Dem, set the Dem error after a reset recovery).

[SWS_EcuM_04139] ⌈If an OS function call fails and no other fault reaction is defined, the EcuM shall not change the requested state. In such cases an error reporting via EcuM_ErrorHook shall be performed.⌋*()*

Note: The exact error code used when calling EcuM_ErrorHook depends on the OS function and their return value and is not standardized.

7.13 Error classification

Section "Error Handling" of the document [6] describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

AUTOSAR BSW modules normally 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 subchapters contains all error codes which might be reported from the EcuM (besides those individual error codes defined by the integrator).

7.13.1 Development Errors

[SWS_EcuM_04032] ⌈

Type of error	*Related error code*	*Error value*
Multiple requests by the same user were detected	ECUM_E_MULTIPLE_RUN_REQUESTS	Assigned by Implementation
A function was called which was disabled by configuration	ECUM_E_SERVICE_DISABLED	Assigned by Implementation
A service was called prior to initialization	ECUM_E_UNINIT	Assigned by Implementation

▽

△

Type of error	*Related error code*	*Error value*
An unknown wakeup source was passed as a parameter to an API	ECUM_E_UNKNOWN_WAKEUP_SOURCE	Assigned by Implementation
The initialization failed	ECUM_E_INIT_FAILED	Assigned by Implementation
A state, passed as an argument to a service, was out of range (specific parameter test)	ECUM_E_STATE_PAR_OUT_OF_RANGE	Assigned by Implementation
A parameter was invalid (unspecific)	ECUM_E_INVALID_PAR	Assigned by Implementation
A invalid pointer was passed as an argument	ECUM_E_PARAM_POINTER	Assigned by Implementation
A previous matching request for the provided user was not found	ECUM_E_MISMATCHED_RUN_RELEASE	Assinged by Implementation

⌋*(SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00350, SRS_BSW_00385)*

7.13.2 Runtime Errors

[SWS_EcuM_91003] ⌈

Type of error	*Related error code*	*Error value*
Postbuild configuration data is inconsistent	ECUM_E_CONFIGURATION_DATA_ INCONSISTENT	Assigned by Implementation
The RAM check during wakeup failed	ECUM_E_RAM_CHECK_FAILED	Assigned by Implementation

⌋*()*

7.13.3 Transient Faults

There are no transient faults.

7.13.4 Production Errors

There are no production errors.

7.13.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported Types

This section lists all types imported by the ECU Manager module from the corresponding AUTOSAR modules.

[SWS_EcuM_02810] ⌈

Module	***Header File***	***Imported Type***
Adc	Adc.h	Adc_ConfigType
BswM	BswM.h	BswM_ConfigType
Can	Can.h	Can_ConfigType
CanTrcv	CanTrcv.h	CanTrcv_ConfigType
ComStack_Types	ComStack_Types.h	NetworkHandleType
	ComStack_Types.h	PNCHandleType
Dem	Dem.h	Dem_ConfigType
Det	Det.h	Det_ConfigType
Eth	Eth.h	Eth_ConfigType
EthSwt	EthSwt.h	EthSwt_ConfigType
EthTrcv	EthTrcv.h	EthTrcv_ConfigType
Fls	Fls.h	Fls_ConfigType
Fr	Fr.h	Fr_ConfigType
FrTrcv	FrTrcv.h	FrTrcv_ConfigType
Gpt	Gpt.h	Gpt_ConfigType
Icu	Icu.h	Icu_ConfigType
IoHwAb	IoHwAb.h	IoHwAb<Init_Id>_ConfigType
Lin	Lin.h	Lin_ConfigType
LinTrcv	LinTrcv.h	LinTrcv_ConfigType
McOs	Os.h	AppModeType
	Os.h	CoreIdType
Mcu	Mcu.h	Mcu_ConfigType
	Mcu.h	Mcu_ModeType
	Mcu.h	Mcu_ResetType
Ocu	Ocu.h	Ocu_ConfigType
Os	Os.h	StatusType
Port	Port.h	Port_ConfigType
Pwm	Pwm.h	Pwm_ConfigType
SchM	SchM.h	SchM_ConfigType
Spi	Spi.h	Spi_ConfigType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType
Wdg	Wdg.h	Wdg_ConfigType

⌋*()*

[SWS_EcuM_03019] ⌈ECUM_E_EARLIER_ACTIVE and ECUM_E_PAST shall be of type Std_ReturnType and represent the following values

- ECUM_E_EARLIER_ACTIVE = 3
- ECUM_E_PAST = 4

⌋*()*

8.2 Type definitions

8.2.1 EcuM_ConfigType

[SWS_EcuM_04038] ⌈

Name	EcuM_ConfigType	
Kind	Structure	
Elements	-	
	Type	–
	Comment	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.	
Available via	EcuM.h	

⌋*()*

[SWS_EcuM_02801] ⌈The structure defined by type EcuM_ConfigType shall hold the post-build configuration parameters for the ECU Manager module as well as pointers to all ConfigType structures of modules that are initialized by the ECU Manager module.⌋*()*

The ECU Manager module Configuration Tool must generate the structure defined by the EcuM_ConfigType type specifically for a given set of basic software modules that comprise the ECU configuration. The set of basic software modules is derived from the corresponding EcuM parameters

[SWS_EcuM_02794] ⌈The structure defined in the EcuM_ConfigType type shall contain an additional post-build configuration variant identifier (uint8/uint16/uint32 depending on algorithm to compute the identifier).⌋*()*

See also Chapter 7.3.4 Checking Configuration Consistency.

[SWS_EcuM_02795] ⌈The structure defined by the EcuM_ConfigType type shall contain an additional hash code that is tested against the configuration parameter `EcuM-ConfigConsistencyHash` for checking consistency of the configuration data.⌋*()*

See also section 7.3.4 Checking Configuration Consistency.

For each given ECU configuration, the ECU Manager module Configuration Tool must generate an instance of this structure that is filled with the post-build configuration parameters of the ECU Manager module as well as pointers to instances of configuration structures for the modules mentioned above. The pointers are derived from the corresponding EcuM parameters.

8.2.2 EcuM_RunStatusType

[SWS_EcuM_04120] ⌈

Name	EcuM_RunStatusType		
Kind	Type		
Derived from	uint8		
Range	ECUM_RUNSTATUS_UNKNOWN	0	Unknown status. Init Value.
	ECUM_RUNSTATUS_ REQUESTED	1	Status requested from EcuM
	ECUM_RUNSTATUS_ RELEASED	2	Status released from EcuM.
Description	Result of the Run Request Protocol sent to BswM		
Available via	EcuM.h		

⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_04121] ⌈The ECU Manager module shall inform BswM about the state of the Run Request Protocol as listed in the EcuM_RunStatusType.⌋*(SRS_ModeMgm_-09116)*

8.2.3 EcuM_WakeupSourceType

[SWS_EcuM_04040] ⌈

Name	EcuM_WakeupSourceType		
Kind	Type		
Derived from	uint32		
	ECUM_WKSOURCE_POWER	0x01	Power cycle (bit 0)
	ECUM_WKSOURCE_RESET (default)	0x02	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	0x04	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	0x08	Reset by internal watchdog (bit 3)
	ECUM_WKSOURCE_ EXTERNAL_WDG	0x10	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.		
Available via	EcuM.h		

⌋*()*

[SWS_EcuM_02165] ⌈Additional wakeup sources (to the pre-defined sources) shall be assigned individually to bitfield positions 5 to 31 by configuration. The bit assignment shall be done by the configuration tool.⌋*()*

[SWS_EcuM_02166] ⌈The EcuMWakeupSourceId (see ECUC_EcuM_00151) field in the EcuMWakeupSource container shall define the position corresponding to that wakeup source in all instances the EcuM_WakeupSourceType bitfield.⌋*()*

8.2.4 EcuM_WakeupStatusType

[SWS_EcuM_04041] ⌈

Name	EcuM_WakeupStatusType		
Kind	Type		
Derived from	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
Description	The type describes the possible states of a wakeup source.		
Available via	EcuM.h		

⌋*()* NOTE: This declaration has to be changed to a mode. The name has to be changed.

8.2.5 EcuM_ResetType

[SWS_EcuM_04044] ⌈

Name	EcuM_ResetType		
Kind	Type		
Derived from	uint8		
Range	ECUM_RESET_MCU	0	Microcontroller reset via Mcu_ PerformReset
	ECUM_RESET_WDG	1	Watchdog reset via WdgM_ PerformReset
	ECUM_RESET_IO	2	Reset by toggeling an I/O line.
Description	This type describes the reset mechanisms supported by the ECU State Manager. It can be extended by configuration.		
Available via	EcuM.h		

⌋*()*

8.2.6 EcuM_StateType

[SWS_EcuM_91005] ⌈

Name	EcuM_StateType		
Kind	Type		
Derived from	uint8		
Range	ECUM_SUBSTATE_MASK	0x0f	–
	ECUM_STATE_STARTUP	0x10	–
	ECUM_STATE_RUN	0x32	–
	ECUM_STATE_POST_RUN	0x33	–
	ECUM_STATE_SHUTDOWN	0x40	–
	ECUM_STATE_SLEEP	0x50	–
Description	ECU State Manager states.		
Available via	EcuM.h		

⌋*(SRS_BSW_00331)*

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

8.3 Function Definitions

This is a list of functions provided for upper layer modules.

8.3.1 General

8.3.1.1 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.	
Available via	EcuM.h	

⌋*(SRS_BSW_00407, SRS_BSW_00411)*

8.3.2 Initialization and Shutdown Sequences

8.3.2.1 EcuM_GoDownHaltPoll

[SWS_EcuM_91002] ⌈

Service Name	EcuM_GoDownHaltPoll	
Syntax	`Std_ReturnType EcuM_GoDownHaltPoll ( EcuM_UserType UserID )`	
Service ID [hex]	0x2c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	UserID	Id of the user calling this API. Only configured users are allowed to call this function.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_NOT_OK: The request was not accepted. E_OK: If the ShutdownTargetType is SLEEP the call successfully returns, the ECU has left the sleep again. If the ShutdownTargetType is RESET or OFF this call will not return.
Description	Instructs the ECU State Manager module to go into a sleep mode, Reset or OFF depending on the previously selected shutdown target.	

▽

△

Available via	EcuM.h

⌋()

8.3.2.2 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)
Available via	EcuM.h

⌋*(SRS_BSW_00358, SRS_BSW_00414, SRS_BSW_00101)*

8.3.2.3 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.

▽

△

Available via	EcuM.h

⌋()

[SWS_EcuM_02806] ⌈Caveats of EcuM_StartupTwo: This function must be called from a task, which is started directly as a consequence of StartOS. I.e. either the EcuM_StartupTwo function must be called from an autostart task or the EcuM_Startup Two function must be called from a task, which is explicitly started.⌋*()*

Clarification to [SWS_EcuM_02806] : The OS offers different mechanisms to activate a task on startup. Normally EcuM_StartupTwo would be configured as an autostart task in the default application mode.

The integrator can configure the OS to activate the EcuM_StartupTwo task by any mechanism, as long as it is started immediately after StartOS is called. The task can also be activated from within another task and this other task could be an autostart task.

Starting EcuM_StartupTwo as an autostart task is an implicit activation. The other mechanisms would be an explicit activation.

8.3.2.4 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.
Available via	EcuM.h

⌋*(SRS_ModeMgm_09114)*

8.3.3 State Management

8.3.3.1 EcuM_ SetState

[SWS_EcuM_04122] ⌈

Service Name	EcuM_SetState	
Syntax	`void EcuM_SetState ( EcuM_StateType state )`	
Service ID [hex]	0x2b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	state	State indicated by BswM.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Function called by BswM to notify about State Switch.	
Available via	EcuM.h	

⌋*()*

[SWS_EcuM_04123] ⌈The EcuM_SetState function shall set the EcuM State to the value of the State parameter.

If the State parameter is not a valid value, the EcuM_SetState function shall not change the State and if Development Error Reporting is turned on, the EcuM_SetState function shall additionally send an ECUM_E_STATE_PAR_OUT_OF_RANGE error message to the DET module.⌋*(SRS_ModeMgm_09116)*

8.3.3.2 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
Description	Places a request for the RUN state. Requests can be placed by every user made known to the state manager at configuration time.	
Available via	EcuM.h	

⌋()

[SWS_EcuM_04125] ⌈Requests of EcuM_RequestRUN cannot be nested, i.e. one user can only place one request but not more. Additional or duplicate user requests by the same user shall be reported to DET. Of course the DET will only be notified under development conditions.⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_04126] ⌈An implementation must track requests for each user known on the ECU. Run requests are specific to the user.⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_03024] ⌈If development error detection is enabled and there are multiple requests by the same user detected by `EcuM_RequestRUN` the function shall report `ECUM_E_MULTIPLE_RUN_REQUESTS` to Det.⌋*()*

8.3.3.3 EcuM_ReleaseRUN

[SWS_EcuM_04127] ⌈

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
Description	Releases a RUN request previously done with a call to EcuM_RequestRUN. The service is intended for implementing AUTOSAR ports.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_03023] ⌈If development error detection is enabled and `EcuM_ReleaseRUN` did not find a previous matching request for the provided user, then the function shall report `ECUM_E_MISMATCHED_RUN_RELEASE` to Det.⌋*()*

Configuration of EcuM_ReleaseRUN: Refer to EcuM_UserType for more information about user IDs and their generation.

8.3.3.4 EcuM_RequestPOST_RUN

[SWS_EcuM_04128] ⌈

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
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.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_03025] ⌈If development error detection is enabled and there are multiple requests by the same user detected by `EcuM_RequestPOST_RUN` the function shall report `ECUM_E_MULTIPLE_RUN_REQUESTS` to Det.⌋*()*

All requirements of 8.3.3.2 EcuM_RequestRUN apply accordingly to the function EcuM_RequestPOST_RUN.

Configuration of EcuM_RequestPOST_RUN: Refer to EcuM_UserType for more information about user IDs and their generation.

8.3.3.5 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
Description	Releases a POST RUN request previously done with a call to EcuM_RequestPOST_RUN. The service is intended for implementing AUTOSAR ports.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09116)*

[SWS_EcuM_03026] ⌈If development error detection is enabled and `EcuM_Release-POST_RUN` did not find a previous matching request for the provided user, then the function shall report `ECUM_E_MISMATCHED_RUN_RELEASE` to Det.⌋*()*

Configuration of EcuM_ReleasePOST_RUN: Refer to EcuM_UserType for more information about user IDs and their generation.

8.3.4 Shutdown Management

8.3.4.1 EcuM_SelectShutdownTarget

[SWS_EcuM_02822] ⌈

Service Name	EcuM_SelectShutdownTarget	

Syntax	`Std_ReturnType EcuM_SelectShutdownTarget (`
	`  EcuM_ShutdownTargetType shutdownTarget,`
	`  EcuM_ShutdownModeType shutdownMode`
	`)`	

Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	shutdownTarget	The selected shutdown target.
	shutdownMode	The identfier of a sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or a reset mechanism (if target is ECUM_SHUTDOWN_TARGET_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.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09114, SRS_ModeMgm_09128, SRS_ModeMgm_09235)*

[SWS_EcuM_00624] ⌈The EcuM_SelectShutdownTarget function shall set the shutdown target to the value of the shutdownTarget parameter.⌋*(SRS_ModeMgm_09114, SRS_ModeMgm_09235)*

[SWS_EcuM_02185] ⌈The parameter mode of the function EcuM_SelectShutdown Target shall be the identifier of a sleep or reset mode. The mode parameter shall only be used if the target parameter equals ECUM_SHUTDOWN_TARGET_SLEEP or ECUM_SHUTDOWN_TARGET_RESET. In all other cases, it shall be ignored. Only sleep or reset modes that are defined at configuration time and are stored in the Ecu MCommonConfiguration container (see ECUC_EcuM_00181) are allowed as parameters.⌋*(SRS_ModeMgm_09114)*

[SWS_EcuM_02585] ⌈EcuM_SelectShutdownTarget shall not initiate any setup activities but only store the value for later use in the SHUTDOWN or SLEEP phase.⌋*(SRS_ModeMgm_09114)*

Implementation hint: The ECU Manager module does not define any mechanism to resolve conflicts arising from requests from different sources. The shutdown target is always the last value set.

8.3.4.2 EcuM_GetShutdownTarget

[SWS_EcuM_02824] ⌈

Service Name	EcuM_GetShutdownTarget	
Syntax	`Std_ReturnType EcuM_GetShutdownTarget ( EcuM_ShutdownTargetType* shutdownTarget, EcuM_ShutdownModeType* shutdownMode )`	
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_SHUTDOWN_TARGET_SLEEP ECUM_SHUTDOWN_TARGET_RESET ECUM_SHUTDOWN_TARGET_OFF
	shutdownMode	If the out parameter "shutdownTarget" is ECUM_SHUTDOWN_TARGET_SLEEP, sleepMode tells which of the configured sleep modes was actually chosen. If "shutdownTarget" is ECUM_SHUTDOWN_TARGET_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.	

▽

△

Available via	EcuM.h

⌋*(SRS_ModeMgm_09128, SRS_ModeMgm_09235)*

[SWS_EcuM_02788] ⌈If the pointer to the shutdownMode parameter is NULL, EcuM_GetShutdownTarget shall simply ignore the shutdownMode parameter. If Development Error Detection is activated, `EcuM_GetShutdownTarget` shall send the ECUM_E_PARAM_POINTER development error to the DET module.⌋*()*

8.3.4.3 EcuM_GetLastShutdownTarget

[SWS_EcuM_02825] ⌈

Service Name	EcuM_GetLastShutdownTarget	
Syntax	`Std_ReturnType EcuM_GetLastShutdownTarget ( EcuM_ShutdownTargetType* shutdownTarget, EcuM_ShutdownModeType* shutdownMode )`	
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_SHUTDOWN_TARGET_SLEEP ECUM_SHUTDOWN_TARGET_RESET ECUM_SHUTDOWN_TARGET_OFF
	shutdownMode	If the out parameter "shutdownTarget" is ECUM_SHUTDOWN_TARGET_SLEEP, sleepMode tells which of the configured sleep modes was actually chosen. If "shutdownTarget" is ECUM_SHUTDOWN_TARGET_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.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09128, SRS_ModeMgm_09235)*

[SWS_EcuM_02156] ⌈EcuM_GetLastShutdownTarget shall return the ECU state from which the last wakeup or power up occurred in the shutdownTarget parameter. EcuM_GetLastShutdownTarget shall always return the same value until the next shutdown.⌋*(SRS_ModeMgm_09235)*

[SWS_EcuM_02336] ⌈If the call of GetLastShutdownTarget() passes ECU_STATE_SLEEP in the parameter shutdownTarget, in the parameter shutdownMode it returns which of the configured sleep modes was actually chosen.

If the call of GetLastShutdownTarget() passes ECU_STATE_RESET in the parameter shutdownTarget, in the parameter sleepMode it returns which of the configured reset modes was actually chosen.⌋*()*

[SWS_EcuM_02337] ⌈If the pointer to the shutdownMode parameter is NULL, EcuM_GetLastShutdownTarget shall simply ignore the shutdownMode parameter and return the last shutdown target regardless of whether it was SLEEP or not. If Development Error Detection is activated, EcuM_GetShutdownTarget shall send the ECUM_E_PARAM_POINTER development error to the DET module.⌋*()*

[SWS_EcuM_02157] ⌈EcuM_GetLastShutdownTarget may return a shutdown target in a STARTUP phase that set late in a previous SHUTDOWN phase. If so, implementation specific limitations shall be clearly documented.⌋*()*

Rationale for [SWS_EcuM_02157]

The `EcuM_GetLastShutdownTarget` function is intended primarily for use in the ECU STARTUP or RUN states. To simplify implementation, it is acceptable if the value is set in late shutdown phase for use during the next startup.

8.3.4.4 EcuM_SelectShutdownCause

[SWS_EcuM_04050] ⌈

Service Name	EcuM_SelectShutdownCause	
Syntax	`Std_ReturnType EcuM_SelectShutdownCause ( EcuM_ShutdownCauseType target )`	
Service ID [hex]	0x1b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	target	The selected shutdown cause.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The new shutdown cause was set E_NOT_OK: The new shutdown cause was not set
Description	EcuM_SelectShutdownCause elects the cause for a shutdown. EcuM_SelectShutdownCause is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋*()*

8.3.4.5 EcuM_GetShutdownCause

[SWS_EcuM_04051] ⌈

Service Name	EcuM_GetShutdownCause	

Syntax	`Std_ReturnType EcuM_GetShutdownCause (`
	`  EcuM_ShutdownCauseType* shutdownCause`
	`)`	

Service ID [hex]	0x1c	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	shutdownCause	The selected cause of the next shutdown.
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_GetShutdownCause returns the selected shutdown cause as set by EcuM_Select ShutdownCause. EcuM_GetShutdownCause is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋*()*

8.3.5 Wakeup Handling

8.3.5.1 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_WakeupSource Type	All wakeup events
Description	Gets pending wakeup events.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09126)*

[SWS_EcuM_01156] ⌈`EcuM_GetPendingWakeupEvents` shall return wakeup events which have been set to pending but not yet validated as bits set in a `EcuM_-WakeupSourceType` bitmask.⌋*()*

[SWS_EcuM_02172] ⌈EcuM_GetPendingWakeupEvents shall be callable from interrupt context, from OS context and an OS-free context.⌋*()*

[SWS_EcuM_03003] ⌈Caveat of EcuM_GetPendingWakeupEvents: This function only returns the wakeup events with status ECUM_WKSTATUS_PENDING.⌋*()*

8.3.5.2 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.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09126)*

[SWS_EcuM_02683] ⌈EcuM_ClearWakeupEvent clears all pending events passed as a bit set in the sources in parameter (EcuM_WakeupSourceType bitmask) from the internal pending wakeup events variable, the internal validated events variable and the internal expired events variable.⌋*()*

See also section 7.6.3 Internal Representation of Wakeup States.

[SWS_EcuM_02807] ⌈EcuM_ClearWakeupEvent shall be callable from interrupt context, from OS context and an OS-free context.⌋*()*

Integration note: The clearing of wakeup sources shall take place during ECU shutdown prior to the call of Dem_Shutdown() and NvM_WriteAll(). This can be achieved by configuring BswMRules in the BswM module containing BswMActions of type BswMUserCallout with their BswMUserCalloutFunction parameter set to "EcuM_ClearWakeupEvents(<sources>)". Hereby <sources> needs to be derived from the EcuMWakeupSourceIds in the EcuM configuration. These BswMRules must then be configured in a way that they get triggered during ECU shutdown prior to the call of Dem_Shutdown() and NvM_WriteAll().

8.3.5.3 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_WakeupSource Type	All wakeup events
Description	Gets validated wakeup events.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09126)*

[SWS_EcuM_02533] ⌈`EcuM_GetValidatedWakeupEvent` shall return wakeup events which have been set to validated in the internal validated events variable as bits set in a `EcuM_WakeupSourceType` bitmask.⌋*()*

See also section 7.6.3 Internal Representation of Wakeup States.

[SWS_EcuM_02532] ⌈`EcuM_GetValidatedWakeupEvent` shall be callable from interrupt context, from OS context and an OS-free context.⌋*()*

8.3.5.4 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_WakeupSource Type	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.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09126)*

[SWS_EcuM_04076] ⌈`EcuM_GetExpiredWakeupEvents` shall return wakeup events which have been set to validated in the internal expired events variable as bits set in a `EcuM_WakeupSourceType` bitmask.⌋*()*

See also section 7.6.3 Internal Representation of Wakeup States.

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

8.3.6 Alarm Clock

8.3.6.1 EcuM_SetRelWakeupAlarm

[SWS_EcuM_04054] ⌈

Service Name	EcuM_SetRelWakeupAlarm	

Syntax	`Std_ReturnType EcuM_SetRelWakeupAlarm (`
	`  EcuM_UserType user,`
	`  EcuM_TimeType time`
	`)`	

Service ID [hex]	0x22	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	user	The user that wants to set the wakeup alarm.
	time	Relative time from now in seconds.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed ECUM_E_EARLIER_ACTIVE: An earlier alarm is already set
Description	EcuM_SetRelWakeupAlarm sets a user's wakeup alarm relative to the current point in time. EcuM_SetRelWakeupAlarm is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09186, SRS_ModeMgm_09190)*

[SWS_EcuM_04055] ⌈If the relative time from now is earlier than the current wakeup time, `EcuM_SetRelWakeupAlarm` shall update the wakeup time.⌋*(SRS_ModeMgm_-09186)*

[SWS_EcuM_04056] ⌈If the relative time from now is later than the current wakeup time, `EcuM_SetRelWakeupAlarm` shall not update the wakeup time and shall return ECUM_E_EARLIER_ACTIVE.⌋*(SRS_ModeMgm_09186)*

8.3.6.2 EcuM_SetAbsWakeupAlarm

[SWS_EcuM_04057] ⌈

Service Name	EcuM_SetAbsWakeupAlarm	
Syntax	`Std_ReturnType EcuM_SetAbsWakeupAlarm ( EcuM_UserType user, EcuM_TimeType time )`	
Service ID [hex]	0x23	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	user	The user that wants to set the wakeup alarm.
	time	Absolute time in seconds. Note that, absolute alarms use knowledge of the current time.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed ECUM_E_EARLIER_ACTIVE: An earlier alarm is already set ECUM_E_PAST: The given point in time has already passed
Description	EcuM_SetAbsWakeupAlarm sets the user's wakeup alarm to an absolute point in time. EcuM_SetAbsWakeupAlarm is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09186, SRS_ModeMgm_09199)*

[SWS_EcuM_04058] ⌈If the time parameter is earlier than the current wakeup time, `EcuM_SetAbsWakeupAlarm` shall update the wakeup time.⌋*(SRS_ModeMgm_09186)*

[SWS_EcuM_04059] ⌈If the time parameter is later than the current wakeup time, `EcuM_SetAbsWakeupAlarm` shall not update the wakeup time and shall return ECUM_E_EARLIER_ACTIVE.⌋*(SRS_ModeMgm_09186)*

[SWS_EcuM_04060] ⌈If the time parameter is earlier than now, `EcuM_SetAbsWakeupAlarm` shall not update the wakeup time and shall return ECUM_E_PAST.⌋*(SRS_ModeMgm_09186)*

8.3.6.3 EcuM_AbortWakeupAlarm

[SWS_EcuM_04061] ⌈

Service Name	EcuM_AbortWakeupAlarm	

Syntax	`Std_ReturnType EcuM_AbortWakeupAlarm (`
	`  EcuM_UserType user`
	`)`	

Service ID [hex]	0x24	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	user	The user that wants to cancel the wakeup alarm.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed ECUM_E_NOT_ACTIVE: No owned alarm found
Description	Ecum_AbortWakeupAlarm aborts the wakeup alarm previously set by this user. EcuM_Abort WakeupAlarm is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋()

8.3.6.4 EcuM_GetCurrentTime

[SWS_EcuM_04062] ⌈

Service Name	EcuM_GetCurrentTime	

Syntax	`Std_ReturnType EcuM_GetCurrentTime (`
	`  EcuM_TimeType* time`
	`)`	

Service ID [hex]	0x25	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	time	Absolute time in seconds since battery connect.
Return value	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: time points to NULL or the module is not initialized
Description	EcuM_GetCurrentTime returns the current value of the EcuM clock (i.e. the time since battery connect). EcuM_GetCurrentTime is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋()

8.3.6.5 EcuM_GetWakeupTime

[SWS_EcuM_04063] ⌈

Service Name	EcuM_GetWakeupTime	
Syntax	`Std_ReturnType EcuM_GetWakeupTime ( EcuM_TimeType* time )`	
Service ID [hex]	0x26	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	time	Absolute time in seconds for next wakeup. 0xFFFFFFFF means no active alarm.
Return value	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: time points to NULL or the module is not initialized
Description	EcuM_GetWakeupTime returns the current value of the master alarm clock (the minimum absolute time of all user alarm clocks). EcuM_GetWakeupTime is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋*()*

8.3.6.6 EcuM_SetClock

[SWS_EcuM_04064] ⌈

Service Name	EcuM_SetClock	
Syntax	`Std_ReturnType EcuM_SetClock ( EcuM_UserType user, EcuM_TimeType time )`	
Service ID [hex]	0x27	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	user	User that wants to set the clock
	time	Absolute time in seconds since battery connect.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed
Description	EcuM_SetClock sets the EcuM clock time to the provided value. This API is useful for testing the alarm services; Alarms that take days to expire can be tested. EcuM_SetClock is part of the ECU Manager Module port interface.	
Available via	EcuM.h	

⌋*(SRS_ModeMgm_09194)*

8.3.7 Miscellaneous

8.3.7.1 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.	
Available via	EcuM.h	

⌋*()*

[SWS_EcuM_02247] ⌈The service `EcuM_SelectBootTarget` shall store the selected target in a way that is compatible with the boot loader.⌋*()*

Explanation for [SWS_EcuM_02247]: This may mean format AND location. The implementer must ensure that the boot target information is placed at a safe location which then can be evaluated by the boot manager after a reset.

[SWS_EcuM_03000] ⌈Caveat for the function `EcuM_SelectBootTarget`: This service may depend on the boot loader used. This service is only intended for use by SW-C's related to diagnostics (boot management).⌋*()*

8.3.7.2 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.	
Available via	EcuM.h	

⌋*(SRS_BSW_00172)*

8.4 Callback Definitions

8.4.1 Callbacks from Wakeup Sources

8.4.1.1 EcuM_CheckWakeup

See `EcuM_StartCheckWakeup` ([SWS_EcuM_02929]) for a description of the `EcuM_CheckWakeup` function.

This service `EcuM_CheckWakeup` is a Callout of the ECU Manager module as well as a Callback that wakeup sources invoke when they process wakeup interrupts.

8.4.1.2 EcuM_SetWakeupEvent

[SWS_EcuM_02826] ⌈

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 (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Sets the wakeup event.	
Available via	EcuM.h	

⌋*(SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00440, SRS_ModeMgm_09098)*

[SWS_EcuM_01117] ⌈EcuM_SetWakeupEvent sets (OR-operation) all events passed as a bit set in the sources in parameter (EcuM_WakeupSourceType bitmask) in the internal pending wakeup events variable.⌋*()*

See also section 7.6.3 Internal Representation of Wakeup States.

[SWS_EcuM_02707] ⌈EcuM_SetWakeupEvent shall start the wakeup validation timeout timer according to *Wakeup Validation Timeout*.⌋*()*

See section 7.6.4.3 Wakeup Validation Timeout.

[SWS_EcuM_02867] ⌈If Development Error Reporting is turned on and parameter "sources" contains an unknown (unconfigured) wakeup source, EcuM_SetWakeupEvent shall not update its internal variable and shall send the ECUM_E_UNKNOWN_WAKEUP_SOURCE error message to the DET module instead.⌋*()*

[SWS_EcuM_02171] ⌈EcuM_SetWakeupEvent must be callable from interrupt context, from OS context and an OS-free context.⌋*(SRS_BSW_00333)*

[SWS_EcuM_04138] ⌈EcuM_SetWakeupEvent shall ignore all events passed in the sources parameter that are not associated to the selected sleep mode.⌋*()*

8.4.1.3 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.	
Available via	EcuM.h	

⌋*(SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00440)*

[SWS_EcuM_04078] ⌈EcuM_ValidateWakeupEvent sets (OR-operation) all events passed as a bit set in the sources in parameter (EcuM_WakeupSourceType bitmask) in the internal validated wakeup events variable.⌋*()*

See also section 7.6.3 Internal Representation of Wakeup States.

[SWS_EcuM_04079] ⌈EcuMValidateWakeupEvent shall invoke BswM_EcuM_Current Wakeup with its sources parameter and state value ECUM_WKSTATUS_VALIDATED.⌋ *()*

[SWS_EcuM_02645] ⌈`EcuM_ValidateWakeupEvent` shall invoke ComM_EcuM_ WakeUpIndication for each wakeup event if the EcuMComMChannelRef parameter (see ECUC_EcuM_00101) in the EcuMWakeupSource configuration container for the corresponding wakeup source is configured.⌋ *()*

[SWS_EcuM_02868] ⌈If Development Error Reporting is turned on and the sources parameter contains an unknown (unconfigured) wakeup source, `EcuM_Validate-WakeupEvent` shall ignore the call and send the ECUM_E_UNKNOWN_WAKEUP_ SOURCE error message to the DET module.⌋ *()*

[SWS_EcuM_02345] ⌈`EcuM_ValidateWakeupEvent` shall be callable from interrupt context and task context.⌋ *(SRS_BSW_00333)*

[SWS_EcuM_02790] ⌈`EcuM_ValidateWakeupEvent` shall return without effect for all sources except communication channels when called while the ECU Manager module is in the RUN state.⌋ *()*

[SWS_EcuM_02791] ⌈`EcuM_ValidateWakeupEvent` shall have full effect in any ECU Phase for those sources that correspond to a communication channel (see [SWS_EcuM_02645]).⌋ *()*

[SWS_EcuM_04140] ⌈`EcuM_ValidateWakeupEvent` shall invoke ComM_EcuM_ PNCWakeUpIndication for each wakeup event and for every referenced PNC if at least one EcuMComMPNCRef parameter (see ECUC_EcuM_00228) in the EcuMWakeup Source configuration container for the corresponding wakeup source is configured.⌋ *()*

8.5 Callout Definitions

Callouts are code fragments that must be added to the ECU Manager module during ECU integration. The content of most callouts is hand-written code. The ECU Manager module configuration tool generates a default implementation for some callouts which is edited manually by the integrator. Conceptually, these callouts belong to the ECU integration code.

8.5.1 Generic Callouts

8.5.1.1 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.	
Available via	EcuM_Externals.h	

⌋*()* The ECU Manager module can invoke `EcuM_ErrorHook`: in all phases

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 codes shall not conflict with the development and production error codes as defined in Table 7.1 and Table 7.13.1 nor with the standard error codes, i.e. E_OK, E_NOT_OK, etc.

8.5.2 Callouts from the STARTUP Phase

8.5.2.1 EcuM_AL_SetProgrammableInterrupts

[SWS_EcuM_04085] ⌈

Service Name	EcuM_AL_SetProgrammableInterrupts
Syntax	`void EcuM_AL_SetProgrammableInterrupts ( void )`
Service ID [hex]	0x4A
Sync/Async	Asynchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None

▽

△

Description	If the configuration parameter EcuMSetProgrammableInterrupts is set to true, this callout EcuM_AL_SetProgrammableInterrupts is executed and shall set the interrupts on ECUs with programmable interrupts.
Available via	EcuM_Externals.h

⌋*()*

8.5.2.2 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.
Available via	EcuM_Externals.h

⌋*()* The ECU Manager module invokes `EcuM_AL_DriverInitZero` early in the PreOS Sequence (see section 7.3.2 Activities in StartPreOS Sequence)

The ECU Manager module configuration tool must generate a default implementation of the `EcuM_AL_DriverInitZero` callout ([SWS_EcuM_02905]) from the sequence of modules defined in the EcuMDriverInitListZero configuration container (see ECUC_EcuM_00114). See also [SWS_EcuM_02559] and [SWS_EcuM_02730].

8.5.2.3 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.	
Available via	EcuM_Externals.h	

⌋*()*

The ECU Manager module invokes `EcuM_DeterminePbConfiguration` early in the PreOS Sequence (see section 7.3.2 Activities in StartPreOS Sequence)

8.5.2.4 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.
Available via	EcuM_Externals.h

⌋*()*

The ECU Manager module invokes EcuM_AL_DriverInitOne in the PreOS Sequence (see section 7.3.2 Activities in StartPreOS Sequence)

The ECU Manager module configuration tool must generate a default implementation of the EcuM_AL_DriverInitOne callout from the sequence of modules defined in the EcuMDriverInitListOne configuration container (see ECUC_EcuM_00111). See also [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.

8.5.2.5 EcuM_LoopDetection

[SWS_EcuM_04137] ⌈

Service Name	EcuM_LoopDetection
Syntax	`void EcuM_LoopDetection ( void )`
Service ID [hex]	0x4B
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	If the configuration parameter EcuMResetLoopDetection is set to true, this callout EcuM_Loop Detection is called on every startup.
Available via	EcuM_Externals.h

⌋*()*

8.5.3 Callouts from the SHUTDOWN Phase

8.5.3.1 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.
Available via	EcuM_Externals.h

⌋*()*

The ECU Manager module invokes EcuM_OnGoOffOne on entry to the OffPreOS Sequence (see section 7.4.1 Activities in the OffPreOS Sequence).

8.5.3.2 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.
Available via	EcuM_Externals.h

⌋*()*

The ECU Manager module invokes EcuM_OnGoOffTwo on entry to the OffPostOS Sequence (see section 7.4.2 Activities in the OffPostOS Sequence).

8.5.3.3 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 (inout)	None
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.
Available via	EcuM_Externals.h

⌋*()*

The ECU Manager module invokes EcuM_AL_SwitchOff as the last activity in the Off PostOS Sequence (see section 7.4.2 Activities in the OffPostOS Sequence).

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().

8.5.3.4 EcuM_AL_Reset

[SWS_EcuM_04065] ⌈

Service Name	EcuM_AL_Reset	

Syntax	`void EcuM_AL_Reset (`
	`  EcuM_ResetType reset`
	`)`	

Service ID [hex]	0x4C	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	reset	Type of reset to be performed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	

▽

△

Description	This callout shall take the code for resetting the ECU.
Available via	EcuM_Externals.h

⌋*()*

8.5.4 Callouts from the SLEEP Phase

8.5.4.1 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.	
Available via	EcuM_Externals.h	

⌋*()*

The ECU Manager module invokes EcuM_EnableWakeupSources in the GoSleep Sequence (see section 7.5.1 Activities in the GoSleep Sequence)

[SWS_EcuM_02546] ⌈The ECU Manager module shall derive the wakeup sources to be enabled (and used as the wakeupSource parameter) from the EcuMWakeupSource (see ECUC_EcuM_00152) bitfield configured for the current sleep mode.⌋*()*

8.5.4.2 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
Available via	EcuM_Externals.h

⌋*()*

The ECU Manager module invokes EcuM_GenerateRamHash: in the Halt Sequence just before putting the ECU physically to sleep (see section 7.5.2 Activities in the Halt Sequence).

8.5.4.3 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.
Available via	EcuM_Externals.h

⌋*()*

The ECU Manager module invokes EcuM_SleepActivity periodically during the Poll Sequence (see section 7.5.3 Activities in the Poll Sequence) if the MCU is not halted (i.e. clock is reduced).

Note: If called from the Poll sequence the EcuMcalls 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.

8.5.4.4 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 Wakeup Source is started. If EcuMCheckWakeupTimeout <= 0 the API call is ignored by the EcuM.	
Available via	EcuM_Externals.h	

⌋*()*

8.5.4.5 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.
Available via	EcuM_Externals.h

⌋*()*

[SWS_EcuM_04098] ⌈If `EcuM_SetWakeupEvent` is called for the corresponding wakeup source the CheckWakeupTimer is cancelled.⌋*()*

8.5.4.6 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.	
Available via	EcuM_Externals.h	

⌋*()*

The ECU Manager module invokes EcuM_CheckWakeup periodically during the Poll Sequence (see section 7.5.3 Activities in the Poll Sequence) if the MCU is not halted, or when handling a wakeup interrupt.

Note: If called from the Poll sequence the EcuMcalls 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.

[SWS_EcuM_04080] ⌈The ECU Manager module shall derive the wakeup sources to be checked (and used as the wakeupSource parameter) from the EcuMWakeup Source (see ECUC_EcuM_00152) bitfield configured for the current sleep mode. The integration code used for this callout must determine which wakeup sources must be checked.⌋*()*

8.5.4.7 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-Ecu M(s): this API will be called by the Master-EcuM only.	
Available via	EcuM_Externals.h	

⌋*()*

The ECU Manager module invokes EcuM_CheckRamHash early in the WakeupRestart Sequence (see section 7.5.5 Activities in the WakeupRestart Sequence)

[SWS_EcuM_02987] ⌈When the RAM check fails on wakeup the ECU Manager module shall invoke `EcuM_ErrorHook` with the parameter `ECUM_E_RAM_CHECK_FAILED` . It is left integrator's discretion to allow `EcuM_ErrorHook` to relay the error to the DEM when he judges that the DEM will not write damaged NVRAM blocks.⌋*(SRS_BSW_-00339)*

See also section 7.5.2 Activities in the Halt Sequence.

8.5.4.8 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.	
Available via	EcuM_Externals.h	

⌋*()*

The ECU Manager module invokes EcuM_DisableWakeupSources in the Wakeup Restart Sequence (see section 7.5.5 Activities in the WakeupRestart Sequence)

[SWS_EcuM_04084] ⌈The ECU Manager module shall derive the wakeup sources to be disabled (and used as the wakeupSource parameter) from the internal pending events variable (NOT operation). The integration code used for this callout must determine which wakeup sources must be disabled.⌋*()*

8.5.4.9 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.
Available via	EcuM_Externals.h

⌋*()*

The ECU Manager module invokes EcuM_EcuM_AL_DriverRestart in the Wakeup Restart Sequence (see section 7.5.5 Activities in the WakeupRestart Sequence)

The ECU Manager 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 ECUC_EcuM_00115). See also [SWS_EcuM_02561], [SWS_EcuM_02559] and [SWS_EcuM_02730].

8.5.5 Callouts from the UP Phase

8.5.5.1 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.	
Available via	EcuM_Externals.h	

⌋*()*

The EcuM Manager module invokes EcuM_StartWakeupSources in the WakeupValidation Sequence (see section 7.6.4 Activities in the WakeupValidation Sequence).

8.5.5.2 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().	
Available via	EcuM_Externals.h	

⌋()

The EcuM Manager module invokes EcuM_CheckValidation in the WakeupValidation Sequence (see section 7.6.4 Activities in the WakeupValidation Sequence).

8.5.5.3 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.	
Available via	EcuM_Externals.h	

⌋()

The EcuM Manager module invokes EcuM_StopWakeupSources in the WakeupValidation Sequence (see section 7.6.4 Activities in the WakeupValidation Sequence).

8.6 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.

8.6.1 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.
Available via	SchM_EcuM.h

⌋*(SRS_BSW_00425, SRS_BSW_00373)* To determine the period, the system designer should consider:

- The function will perform wakeup validation (see 7.8 Wakeup Validation Protocol). The shortest validation timeout typically should limit the period.
- As a rule of thumb, the period of this function should be approximately half as long as the shortest validation timeout.

EcuM_MainFunction should not be called from tasks that may invoke runnable entities.

8.7 Expected Interfaces

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

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

[SWS_EcuM_02858] ⌈

API Function	***Header File***	***Description***
BswM_Deinit	BswM.h	Deinitializes the BSW Mode Manager.
BswM_EcuM_CurrentWakeup	BswM_EcuM.h	Function called by EcuM to indicate the current state of a wakeup source.
BswM_Init	BswM.h	Initializes the BSW Mode Manager.
CanSM_StartWakeupSource	CanSM.h	This function shall be called by EcuM when a wakeup source shall be started.
CanSM_StopWakeupSource	CanSM.h	This function shall be called by EcuM when a wakeup source shall be stopped.
ComM_EcuM_PNCWakeUpIndication	ComM_EcuM.h	Notification of a wake up on the corresponding partial network cluster.
ComM_EcuM_WakeUpIndication	ComM_EcuM.h	Notification of a wake up on the corresponding channel.

▽

△

API Function	***Header File***	***Description***
Dem_Init	Dem.h	Initializes or reinitializes this module.
Dem_PreInit	Dem.h	Initializes the internal states necessary to process events reported by BSW-modules.
Dem_Shutdown	Dem.h	Shuts down this module.
GetResource	Os.h	–
Mcu_GetResetReason	Mcu.h	The service reads the reset type from the hardware, if supported.
Mcu_Init	Mcu.h	This service initializes the MCU driver.
Mcu_PerformReset	Mcu.h	The service performs a microcontroller reset.
Mcu_SetMode	Mcu.h	This service activates the MCU power modes.
ReleaseResource	Os.h	–
SchM_Deinit	SchM.h	SchM_Deinit is used to finalize Basic Software Scheduler part of the RTE of the core on which it is called. This service releases all system resources allocated by the Basic Software Scheduler part on that core.
SchM_Init	SchM.h	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	Os.h	–
StartOS	Os.h	–

⌋*()*

8.7.1 Optional Interfaces

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

[SWS_EcuM_02859] ⌈

API Function	***Header File***	***Description***
Adc_Init	Adc.h	Initializes the ADC hardware units and driver.
Can_Init	Can.h	This function initializes the module.
CanTrcv_Init	CanTrcv.h	Initializes the CanTrcv module.
Det_Init	Det.h	Service to initialize the Default Error Tracer.
Det_ReportError	Det.h	Service to report development errors.
Eth_Init	Eth.h	Initializes the Ethernet Driver
EthSwt_Init	EthSwt.h	Initializes the Ethernet Switch Driver
EthTrcv_Init	EthTrcv.h	Initializes the Ethernet Transceiver Driver
Fls_Init	Fls.h	Initializes the Flash Driver.
Fr_Init	Fr.h	Initializes the Fr.
FrTrcv_Init	FrTrcv.h	This service initializes the FrTrcv.
GetCoreID	Os.h	The function returns a unique core identifier.

▽

△

API Function	***Header File***	***Description***
Gpt_Init	Gpt.h	Initializes the GPT driver.
Icu_Init	Icu.h	This function initializes the driver.
IoHwAb_Init<Init_Id>	IoHwAb.h	Initializes either all the IO Hardware Abstraction software or is a part of the IO Hardware Abstraction.
Lin_Init	Lin.h	Initializes the LIN module.
LinTrcv_Init	LinTrcv.h	Initializes the Lin Transceiver Driver module.
Ocu_Init	Ocu.h	Service for OCU initialization.
Port_Init	Port.h	Initializes the Port Driver module.
Pwm_Init	Pwm.h	Service for PWM initialization.
ShutdownAllCores	Os.h	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.
Spi_Init	Spi.h	Service for SPI initialization.
StartCore	Os.h	It is not supported to call this function after Start OS(). 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.
Wdg_Init	Wdg.h	Initializes the module.
WdgM_PerformReset	WdgM.h	Instructs the Watchdog Manager to cause a watchdog reset.

⌋*()*

8.7.2 Configurable interfaces

8.7.2.1 Callbacks from the STARTUP phase

[SWS_EcuM_91001] ⌈

Service Name	EcuM_AL_DriverInitBswM_<x>
Syntax	`void EcuM_AL_DriverInitBswM_<x> ( void )`
Service ID [hex]	0x28
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	This callback shall provide BSW module initializations to be called by the BSW Mode Manager.

▽

△

Available via	EcuM.h

⌋*()*

The EcuM_AL_DriverInitBswM_<x> callbacks are called by the BSW Mode Manager during initialization. The ECU Manager module configuration tool must generate a default implementation of the EcuM_AL_DriverInitBswM_<x> callbacks from the sequence of modules defined in the EcuMDriverInitListBswM configuration container (see ECUC_EcuM_00226). See also [SWS_EcuM_04142].

[SWS_EcuM_04114] ⌈EcuM_AL_DriverInitBswM_<x> is generated for every configured EcuMDriverInitListBswM. The name of the generated functions shall be EcuM_AL_DriverInitBswM_<x>, where <x> represents the short name of the EcuMDriverInitListBswM container.⌋*()*

8.8 Specification of the Port Interfaces

This chapter specifies the port interfaces and ports needed to access the ECU Manager module over the VFB.

8.8.1 Ports and Port Interface for EcuM_ShutdownTarget Interface

8.8.1.1 General Approach

The EcuM_ShutdownTarget client-server interface allows an SW-C to select a shutdown target which will be respected during the next shutdown phase. Note that the ECU Manager module does not offer a port interface to allow a SW-C to initiate shutdown, however.

8.8.1.2 Service Interfaces

[SWS_EcuM_03011] ⌈

Name	EcuM_ShutdownTarget		
Comment	A SW-C can select a shutdown target using this interface		
IsService	true		
Variation	–		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	GetLastShutdownTarget	
Comment	Returns the shutdown target of the previous shutdown	
Variation	–	
Parameters	shutdownTarget	
	Type	EcuM_ShutdownTargetType
	Direction	OUT
	Comment	The shutdown target of the previous shutdown
	Variation	–
	shutdownMode	
	Type	EcuM_ShutdownModeType
	Direction	OUT
	Comment	The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the shutdown
	Variation	–
Possible Errors	E_OK E_NOT_OK	

Operation	GetShutdownCause	
Comment	Returns the selected shutdown cause as set by the operation SelectShutdownCause.	
Variation	–	
Parameters	shutdownCause	
	Type	EcuM_ShutdownCauseType
	Direction	OUT
	Comment	The selected cause of the next shutdown
	Variation	–
Possible Errors	E_OK E_NOT_OK	

Operation	GetShutdownTarget	
Comment	Returns the currently selected shutdown target for the next shutdown as set by the operation SelectShutdownTarget.	
Variation	–	
Parameters	shutdownTarget	
	Type	EcuM_ShutdownTargetType
	Direction	OUT
	Comment	The shutdown target of the next shutdown
	Variation	–
	shutdownMode	
	Type	EcuM_ShutdownModeType
	Direction	OUT
	Comment	The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the shutdown
	Variation	–

▽

△

Possible Errors	E_OK E_NOT_OK

Operation	SelectShutdownCause	
Comment	–	
Variation	–	
Parameters	shutdownCause	
	Type	EcuM_ShutdownCauseType
	Direction	IN
	Comment	The selected shutdown cause
	Variation	–
Possible Errors	E_OK E_NOT_OK	

Operation	SelectShutdownTarget	
Comment	The SW-C selects the cause corresponding to the next shutdown target	
Variation	–	
Parameters	shutdownTarget	
	Type	EcuM_ShutdownTargetType
	Direction	IN
	Comment	The selected shutdown cause
	Variation	–
	shutdownMode	
	Type	EcuM_ShutdownModeType
	Direction	IN
	Comment	The identfier of a sleep mode (if shutdownTarget is ECUM_SHUTDOWN_TARGET_SLEEP) or a reset mechanism (if shutdownTarget is ECUM_SHUTDOWN_TARGET_RESET) as defined by configuration.
	Variation	–
Possible Errors	E_OK E_NOT_OK	

⌋*()*

[SWS_EcuM_02979] ⌈The shutdownMode parameter shall determine the specific sleep or reset mode (see ECUC_EcuM_00132) relevant to SelectShutdownTarget, GetShutdownTarget and GetLastShutdownTarget. The ECU Manager module shall only use the shutdownMode parameter is if the shutdownTarget parameter is equal to ECUM_SHUTDOWN_TARGET_SLEEP or ECUM_SHUTDOWN_TARGET_RESET, otherwise it shall be ignored.⌋*()*

8.8.2 Port Interface for EcuM_BootTarget Interface

8.8.2.1 General Approach

A SW-C that wants to select a boot target must require the client-server interface EcuM_BootTarget.

8.8.2.2 Service Interfaces

[SWS_EcuM_03012] ⌈

Name	EcuM_BootTarget		
Comment	A SW-C that wants to select a boot target must use the client-server interface EcuM_BootTarget.		
IsService	true		
Variation	–		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	GetBootTarget	
Comment	Returns the current boot target	
Variation	–	
Parameters	target	
	Type	EcuM_BootTargetType
	Direction	OUT
	Comment	The currently selected boot target
	Variation	–
Possible Errors	E_OK	

Operation	SelectBootTarget	
Comment	Selects a boot target	
Variation	–	
Parameters	target	
	Type	EcuM_BootTargetType
	Direction	IN
	Comment	The selected boot target
	Variation	–
Possible Errors	E_OK E_NOT_OK	

⌋*()*

8.8.3 Port Interface for EcuM_AlarmClock Interface

8.8.3.1 General Approach

A SW-C that wants to use an alarm clock must require the client-server interface Ecu M_AlarmClock. The EcuM_AlarmClock interface uses port-defined argument values to identify the user that manages its alarm clock. See [SWS_Rte_1350] in the Specification of RTE [2] for a description of port-defined argument values.

8.8.3.2 Service Interfaces

[SWS_EcuM_04105] ⌈

Name	EcuM_AlarmClock		
Comment	A SW-C that wants to use an alarm clock must use the client-server interface EcuM_Alarm Clock.		
IsService	true		
Variation	{ecuc(EcuM/EcuMFlexGeneral/EcuMAlarmClockPresent)} == True		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed
	3	ECUM_E_EARLIER_ ACTIVE	An earlier alarm is already set
	4	ECUM_E_PAST	The desired point in time has already passed
	5	ECUM_E_NOT_ACTIVE	No active alarm found

Operation	AbortWakeupAlarm
Comment	Aborts the wakeup alarm previously set by this user
Variation	–
Possible Errors	E_OK E_NOT_OK ECUM_E_NOT_ACTIVE

Operation	SetAbsWakeupAlarm	
Comment	Sets the user's wakeup alarm to an absolute point in time	
Variation	–	
Parameters	time	
	Type	EcuM_TimeType
	Direction	IN
	Comment	Absolute time in seconds. Note that, absolute alarms use knowledge of the current time
	Variation	–
Possible Errors	E_OK E_NOT_OK ECUM_E_EARLIER_ACTIVE ECUM_E_PAST	

Operation	SetClock	
Comment	Sets the EcuM clock time to the provided value	
Variation	–	
Parameters	time	
	Type	EcuM_TimeType
	Direction	IN
	Comment	Absolute time in seconds since battery connect
	Variation	–
Possible Errors	E_OK E_NOT_OK	

Operation	SetRelWakeupAlarm	
Comment	Sets a user's wakeup alarm relative to the current point in time	
Variation	–	
Parameters	time	
	Type	EcuM_TimeType
	Direction	IN
	Comment	Relative time from now in seconds
	Variation	–
Possible Errors	E_OK E_NOT_OK ECUM_E_EARLIER_ACTIVE	

⌋*()*

8.8.4 Port Interface for EcuM_Time Interface

8.8.4.1 General Approach

A SW-C that wants to use the time functionality of the EucM must require the client-server interface EcuM_Time.

8.8.4.2 Data Types

The EcuM_Time service does not have any specific data types.

8.8.4.3 Service Interfaces

[SWS_EcuM_04109] ⌈

Name	EcuM_Time		
Comment	–		
IsService	true		
Variation	–		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	GetCurrentTime	
Comment	Returns the current value of the EcuM clock (i.e. the time in seconds since battery connect)	
Variation	–	
Parameters	time	
	Type	EcuM_TimeType
	Direction	OUT
	Comment	Absolute time in seconds since battery connect
	Variation	–
Possible Errors	E_OK E_NOT_OK	

Operation	GetWakeupTime	
Comment	Returns the current value of the master alarm clock (the minimum absolute time of all user alarm clocks)	
Variation	–	
Parameters	time	
	Type	EcuM_TimeType
	Direction	OUT
	Comment	Absolute time in seconds for next wakeup. 0xFFFFFFFF means no active alarm.
	Variation	–
Possible Errors	E_OK E_NOT_OK	

⌋*()*

8.8.5 Port Interface for EcuM_StateRequest Interface

[SWS_EcuM_04130] ⌈The ECU State Manager module shall provide System Services for the following functionalities when the container EcuMModeHandling (see 10.2.1) is available:

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

⌋*(SRS_ModeMgm_09116)*

8.8.5.1 General Approach

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_1350] for a description of port-defined argument values.

8.8.5.2 Data Types

No data types are needed for this interface.

8.8.5.3 Service Interfaces

[SWS_EcuM_04131] ⌈

Name	EcuM_StateRequest		
Comment	Interface to request a specific ECU state		
IsService	true		
Variation	–		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	ReleasePOSTRUN
Comment	–
Variation	–
Possible Errors	E_OK E_NOT_OK

Operation	ReleaseRUN
Comment	–
Variation	–
Possible Errors	E_OK E_NOT_OK

Operation	RequestPOSTRUN
Comment	–
Variation	–
Possible Errors	E_OK E_NOT_OK

Operation	RequestRUN
Comment	–
Variation	–
Possible Errors	E_OK E_NOT_OK

⌋*()*

8.8.6 Port Interface for EcuM_CurrentMode Interface

8.8.6.1 General Approach

[SWS_EcuM_04132] ⌈The mode port of the ECU State Manager module shall declare the following modes:

- STARTUP
- RUN
- POST_RUN
- SLEEP
- SHUTDOWN

⌋*(SRS_ModeMgm_09116)*

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_04133] ⌈Mode changes shall be notified to SW-Cs through the RTE mode ports when the mode change occurs.

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.⌋*()*

8.8.6.2 Data Types

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

ModeDeclarationGroup EcuM_Mode {

{ STARTUP, RUN, POST_RUN, SLEEP, SHUTDOWN }

initialMode = STARTUP

};

[SWS_EcuM_04107] ⌈

Name	EcuM_Mode	
Kind	ModeDeclarationGroup	
Category	ALPHABETIC_ORDER	
Initial mode	STARTUP	
On transition value	–	
Modes	POST_RUN	–
	RUN	–
	SHUTDOWN	–
	SLEEP	–
	STARTUP	–
Description	–	

⌋*()*

8.8.6.3 Service Interfaces

[SWS_EcuM_04108] ⌈

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

⌋*()*

8.8.7 Definition of the ECU Manager Service

This section provides guidance on the definition of the ECU Manager module Service. Note that these definitions can only be completed during ECU configuration (since certain ECU Manager module configuration parameters determine the number of ports provided by the ECU Manager module service). Also note a SW-C's implementation does not depend on these definitions.

In an AUTOSAR system, there are ports both above and below the RTE. The ECU Manager module service description defines ports provided to the RTE and the descriptions of every SW-C that uses this service must contain "service ports" which required these ECU Manager module ports from the RTE.

The EcuM provides the following ports:

[SWS_EcuM_04111] ⌈

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/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.SHORT-NAME)}		

⌋*()*

[SWS_EcuM_04110] ⌈

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/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.SHORT-NAME)}		

⌋*()*

[SWS_EcuM_03017] ⌈

Name	AlarmClock_{UserName}		
Kind	ProvidedPort	***Interface***	EcuM_AlarmClock
Description	Provides to SW-Cs an alarm clock. The EcuM_AlarmClock port uses port-defined argument values to identify the user that manages its alarm clock.		
Port Defined Argument Value(s)	***Type***	EcuM_UserType	
	Value	{ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.value)}	
Variation	{ecuc(EcuM/EcuMFlexGeneral/EcuMAlarmClockPresent)} == true UserName = {ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMAlarmClock.SHORT-NAME)}		

⌋*()*

[SWS_EcuM_04113] ⌈

Name	time		
Kind	ProvidedPort	***Interface***	EcuM_Time
Description	Provides the EcuM's time service to SWCs		
Variation	–		

⌋()

[SWS_EcuM_04135] ⌈

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/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.value)}
Variation	UserName = {ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.SHORT-NAME)}	

⌋()

[SWS_EcuM_04112] ⌈

Name	currentMode		
Kind	ProvidedPort	***Interface***	EcuM_CurrentMode
Description	–		
Variation	–		

⌋()

The EcuM provides the following types:

[SWS_EcuM_91004] ⌈

Name	EcuM_UserType
Kind	Type
Derived from	uint8
Description	Unique value for each user.
Variation	–
Available via	Rte_EcuM_Type.h

⌋()

[SWS_EcuM_04102] ⌈

Name	EcuM_TimeType		
Kind	Type		
Derived from	uint32		
Description	This data type represents the time of the ECU Manager module.		
Variation	–		
Available via	Rte_EcuM_Type.h		

⌋*()*

[] ⌈

Name	EcuM_BootTargetType		
Kind	Type		
Derived from	uint8		
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
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.		
Variation	–		
Available via	Rte_EcuM_Type.h		

⌋*()*

[SWS_EcuM_04045] ⌈

Name	EcuM_ShutdownCauseType		
Kind	Type		
Derived from	uint8		
Range	ECUM_CAUSE_UNKNOWN	0	No cause was set.
	ECUM_CAUSE_ECU_STATE	1	ECU state machine entered a state for shutdown
	ECUM_CAUSE_WDGM	2	Watchdog Manager detected a failure
	ECUM_CAUSE_DCM	3	Diagnostic Communication Manager requests a shutdown due to a service request
Description	This type describes the cause for a shutdown by the ECU State Manager. It can be extended by configuration.		
Variation	–		
Available via	Rte_EcuM_Type.h		

⌋*()*

[SWS_EcuM_04101] ⌈

Name	EcuM_ShutdownModeType		
Kind	Type		
Derived from	uint16		
Range	{ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMResetMode.SHORT-NAME)}	{256 + ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMResetMode.EcuMResetModeId)}	Configured Reset Modes
	{ecuc(EcuM/EcuMConfiguration/EcuMCommonConfiguration/EcuMSleepMode.SHORT-NAME)}	{ecuc(EcuM/ EcuMConfiguration/EcuMCommon Configuration/EcuMSleepMode.EcuMSleepModeId)}	Configured Sleep Modes
Description	This data type represents the modes of the ECU Manager module.		
Variation	–		
Available via	Rte_EcuM_Type.h		

⌋*()*

[SWS_EcuM_04136] ⌈

Name	EcuM_ShutdownTargetType		
Kind	Type		
Derived from	uint8		
Range	ECUM_SHUTDOWN_TARGET_SLEEP	0x0	–
	ECUM_SHUTDOWN_TARGET_RESET	0x1	–
	ECUM_SHUTDOWN_TARGET_OFF	0x2	–
Description	–		
Variation	–		
Available via	Rte_EcuM_Type.h		

⌋*()*

[SWS_EcuM_04094] ⌈In the case of a MultiCore ECU, the EcuM AUTOSAR service (Standardized AUTOSAR Interfaces) may be offered on one or more cores.⌋*()*

Although the EcuM service interfaces are available on every core (see section 7.9 Multi Core for details), the EcuC allows the provided ports to be bound to the interface on a particular partition, and therefore to a particular core (see the Specification of ECU Configuration [5]) and only that port will be visible to the VFB. In the case of Multi-Core, this should be bound to the master core. SW-Cs and CDDs on the ECU that need to access EcuM Services can access the master core via the IOC as generated by the RTE.

[SWS_EcuM_04095] ⌈In the case of a MultiCore ECU, the EcuM C-API Interfaces (Standardized Interfaces) which are used by other BSW modules shall be offered in every partition a EcuM runs in.⌋*()*

The C-API interfaces which are used by other BSW module to communicate with the EcuM are offered by every EcuM instance because every EcuM instance can do some independent actions. If BSW modules want to use the EcuM but are inside partitions that contain no own EcuM instance. These modules can use the SchM functions to cross partition boundaries.

8.9 API Parameter Checking

[SWS_EcuM_03009] ⌈If Development Error Detection is enabled for this module, then all functions shall test input parameters and running conditions and use the following error codes in an adequate way:

- ECUM_E_UNINIT
- ECUM_E_SERVICE_DISABLED
- ECUM_E_PARAM_POINTER
- ECUM_E_INVALID_PAR

Specific development errors are listed in the functions, where they apply.⌋*(SRS_BSW_-00323)*

9 Sequence Charts

9.1 State Sequences

Sequence charts showing the behavior of the ECU Manager 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 7.3 - STARTUP Phase
- Figure 7.4 - StartPreOS Sequence
- Figure 7.5 - StartPostOS Sequence
- Figure 7.7 - SHUTDOWN Phase
- Figure 7.8 - OffPreOS Sequence
- Figure 7.9 - OffPostOS Sequence
- Figure 7.10 - SLEEP Phase
- Figure 7.11 - GoSleep Sequence
- Figure 7.12 - Halt Sequence
- Figure 7.13 - Poll Sequence
- Figure 7.14 - WakeupRestart Sequence
- Figure 7.16 - The WakeupValidation Sequence

9.2 Wakeup 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 Wakeup 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 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 9.1 shows the corresponding sequence of calls.

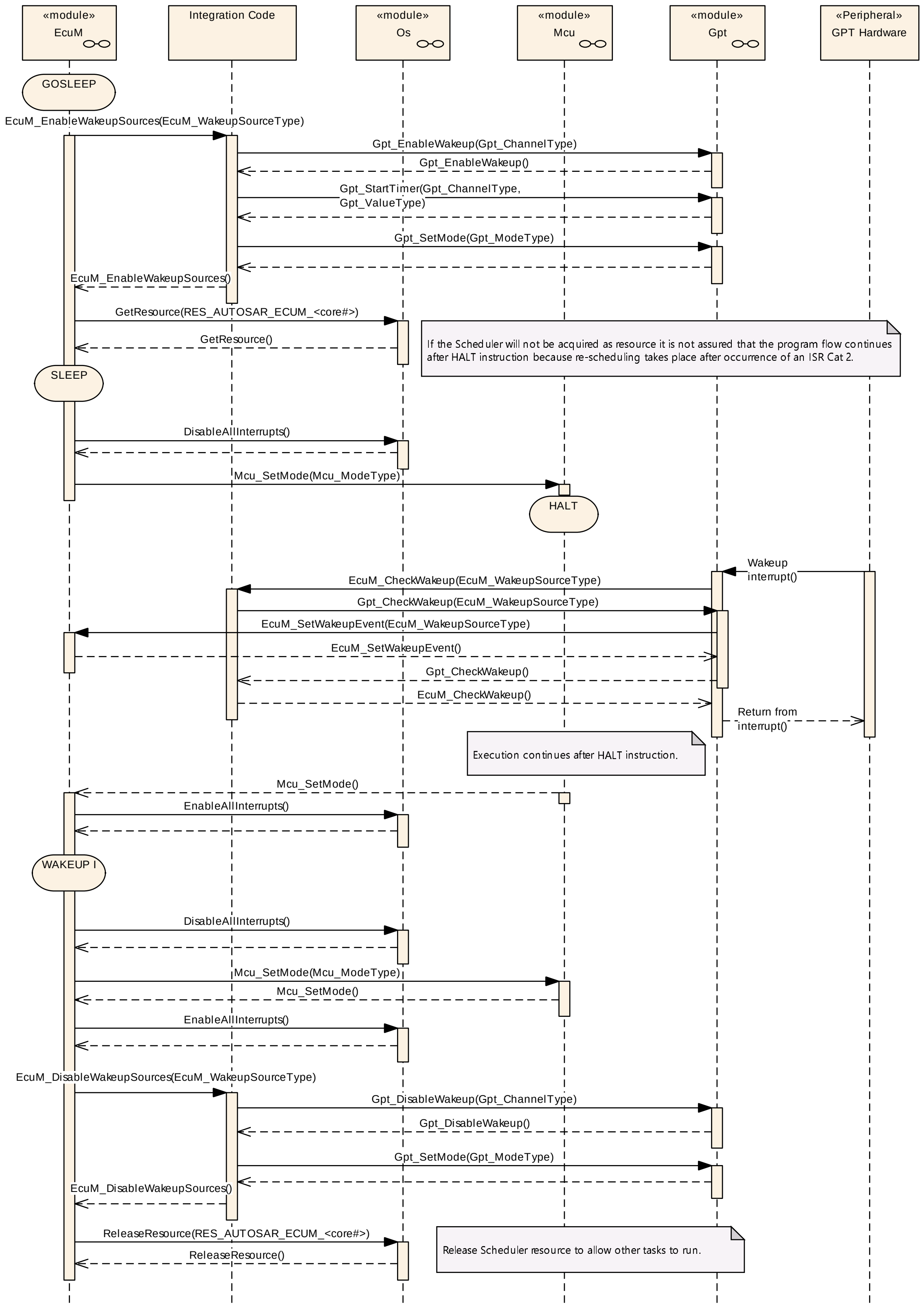

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Gpt
«Peripheral» GPT Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Gpt_EnableWakeup(Gpt_ChannelType)
Gpt_EnableWakeup()
Gpt_StartTimer(Gpt_ChannelType, Gpt_ValueType)
Gpt_SetMode(Gpt_ModeType)
EcuM_EnableWakeupSources()
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 9.1: 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 9.2 .

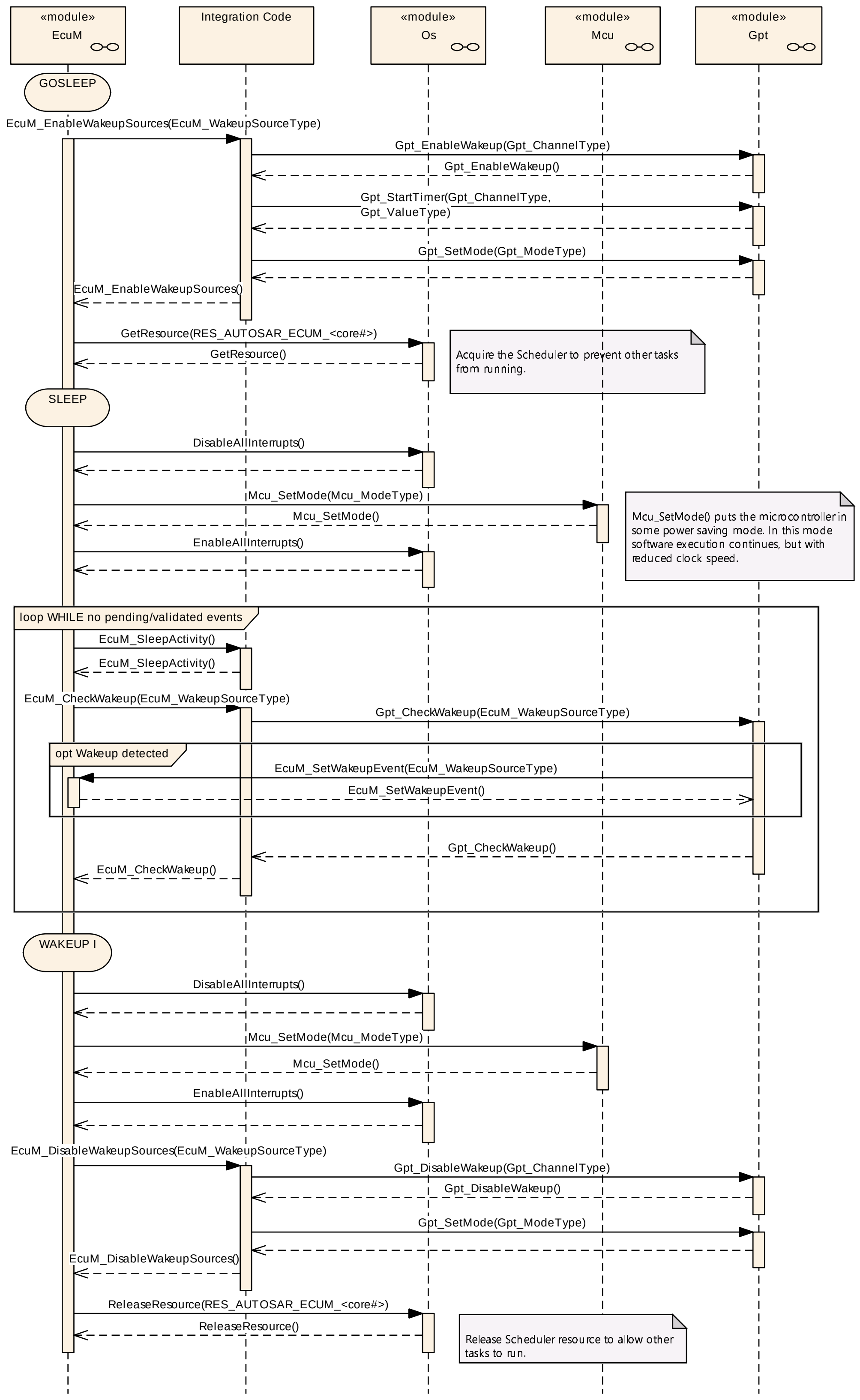

«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Gpt
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Gpt_EnableWakeup(Gpt_ChannelType)
Gpt_EnableWakeup()
Gpt_StartTimer(Gpt_ChannelType, Gpt_ValueType)
Gpt_SetMode(Gpt_ModeType)
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_<core#>)
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/validated 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#>)
ReleaseResource()
Release Scheduler resource to allow other tasks to run.

Figure 9.2: GPT wake up by polling

9.2.2 ICU Wakeup 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 (see [SWS_EcuM_02929]) 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 (see [SWS_EcuM_02929]). 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 (see 8.2.3 EcuM_WakeupSourceType) contains one bit for each wake up source, so that multiple wake up sources can be passed in one call. Figure 9.3 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 38.

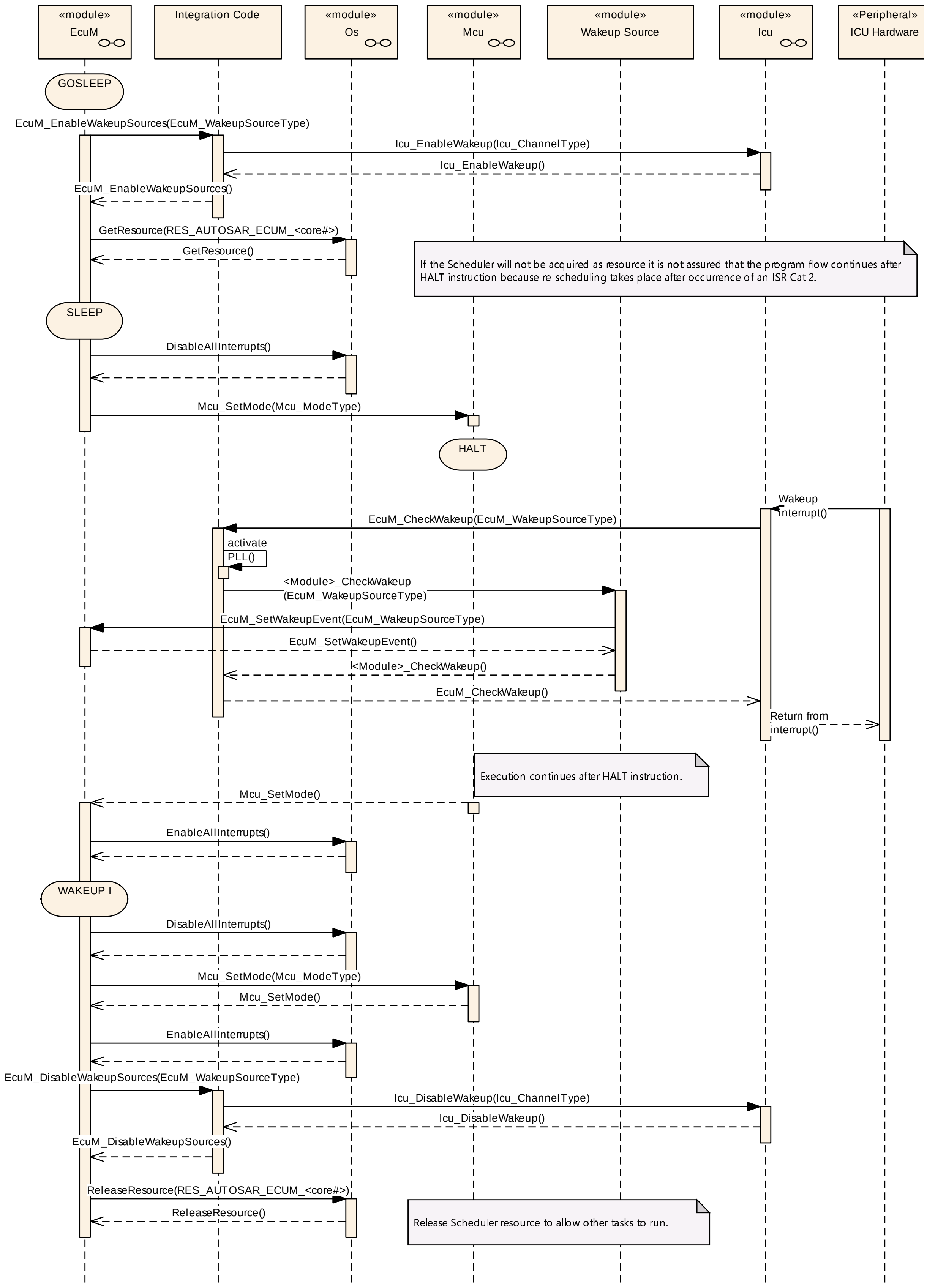

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Wakeup Source
«module» Icu
«Peripheral» ICU Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Icu_EnableWakeup(Icu_ChannelType)
Icu_EnableWakeup()
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_<core#>)
GetResource()
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()
<Module>_CheckWakeup (EcuM_WakeupSourceType)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
<Module>_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)
Icu_DisableWakeup(Icu_ChannelType)
Icu_DisableWakeup()
EcuM_DisableWakeupSources()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
ReleaseResource()
Release Scheduler resource to allow other tasks to run.

Figure 9.3: ICU wake up by interrupt

9.2.3 CAN Wakeup 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 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 an 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.

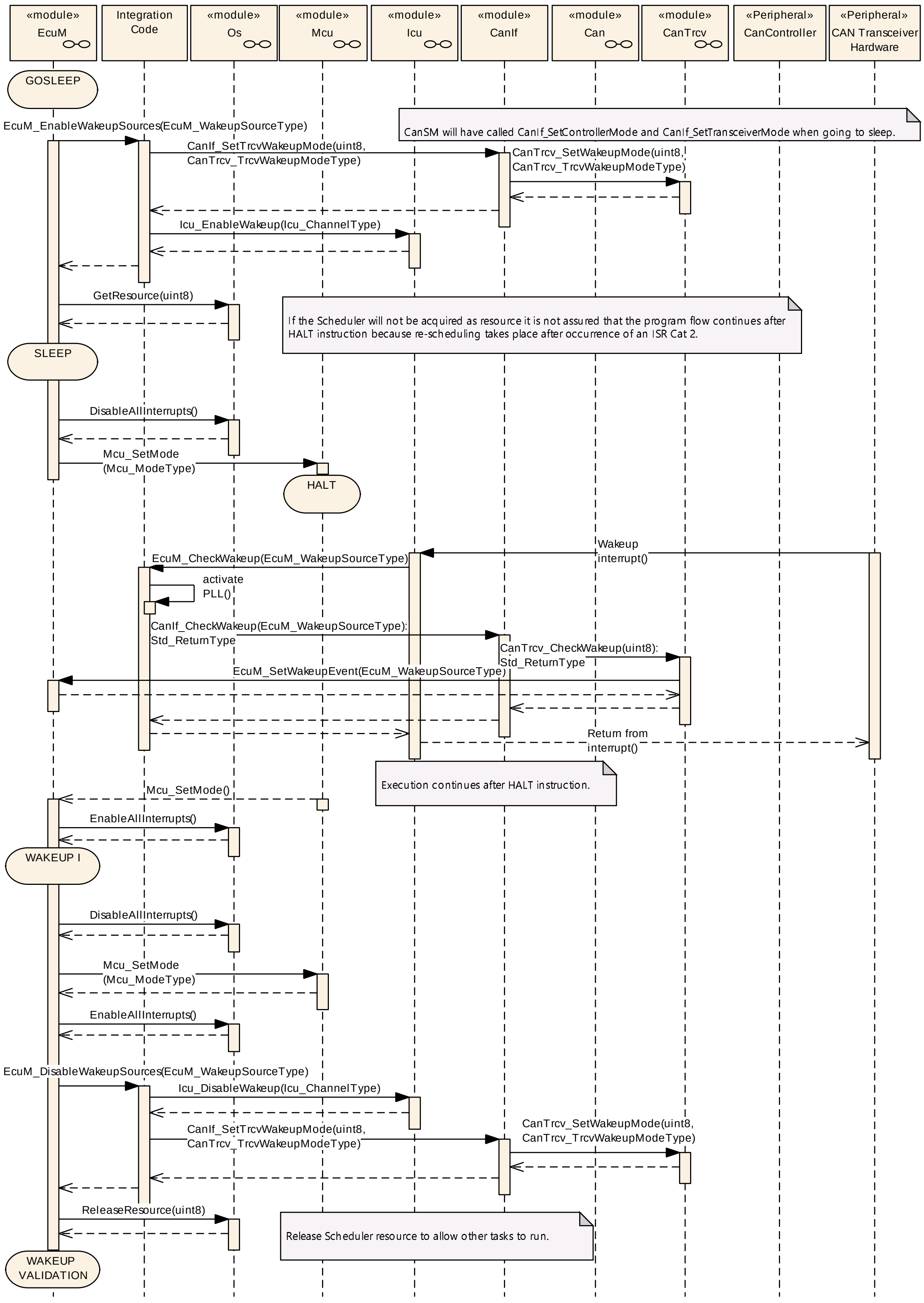

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode 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 9.4: CAN transceiver wake up by interrupt

Figure 9.4 shows the CAN transceiver wakeup via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.

A CAN controller wakeup by interrupt works similar to the GPT wakeup. Here the interrupt handler and the CheckWakeup functionality are both encapsulated in the CAN Driver module, as shown in Figure 9.5 .

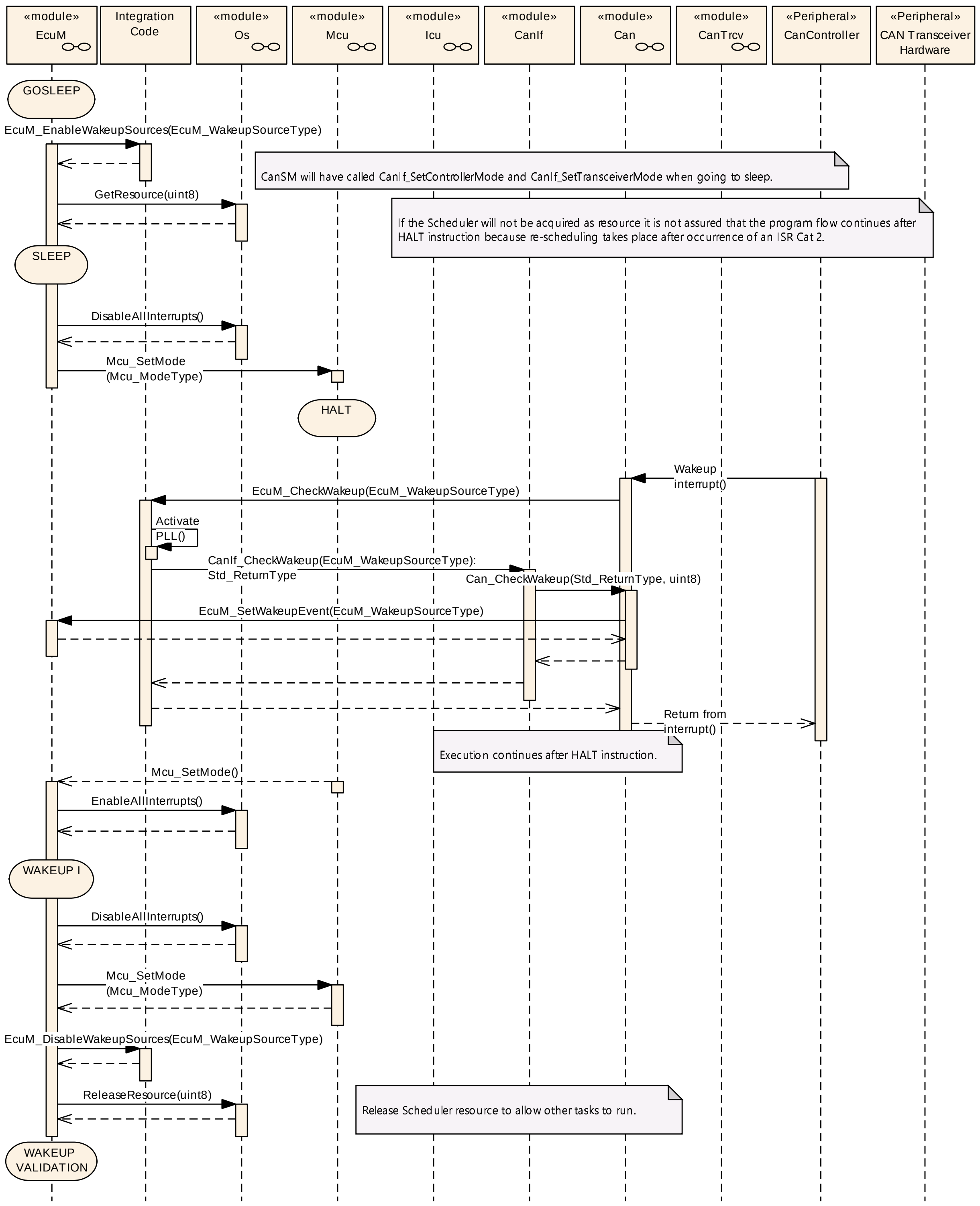

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode when going to sleep.
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
Can_CheckWakeup(Std_ReturnType, uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Execution continues after HALT instruction.
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode (Mcu_ModeType)
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(uint8)
Release Scheduler resource to allow other tasks to run.
WAKEUP VALIDATION

Figure 9.5: CAN controller wake up by interrupt

Wake up by polling is possible both for CAN transceiver and controller. The ECU State Manager 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 9.6 .

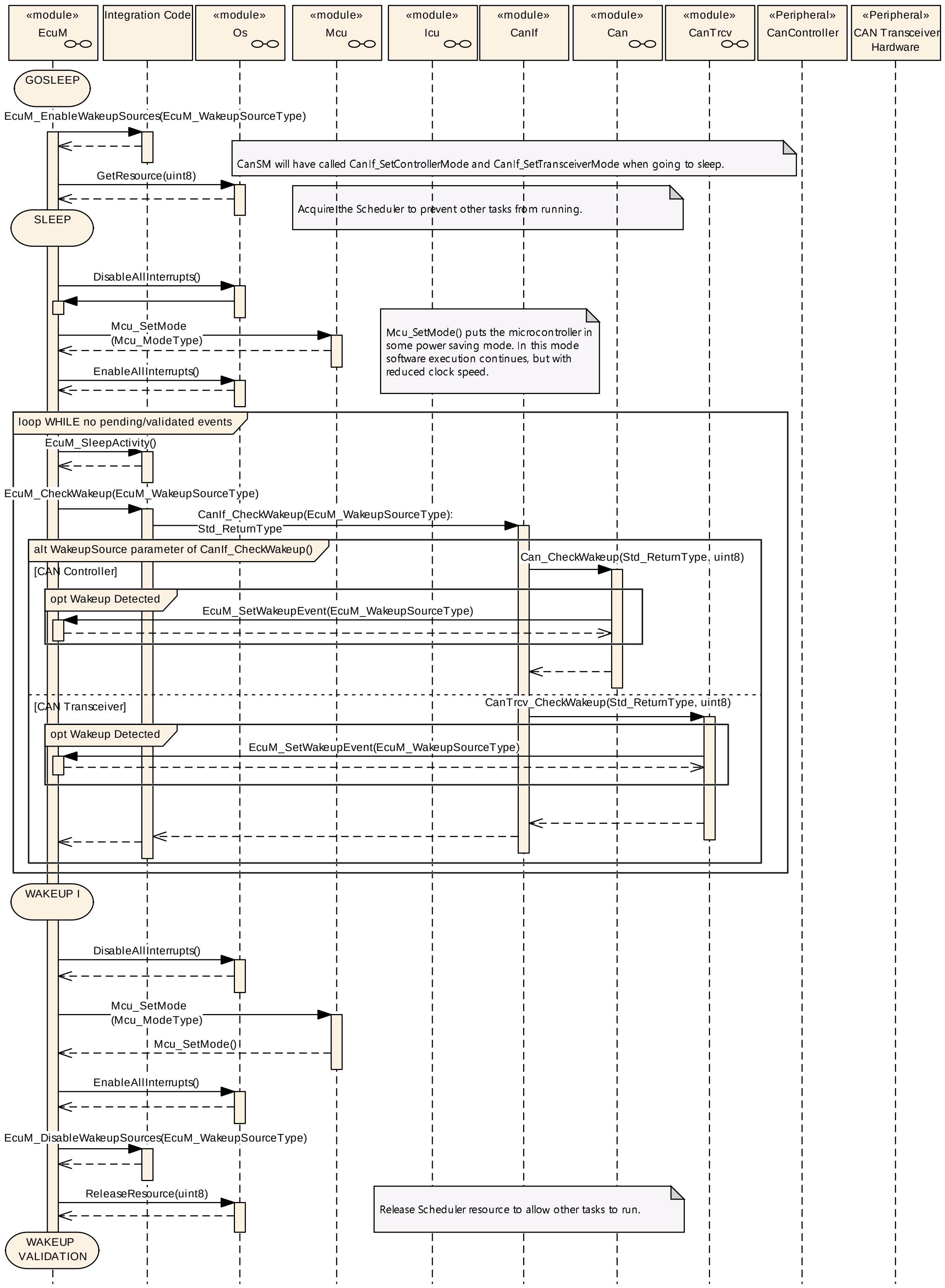

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode when going to sleep.
GetResource(uint8)
Acquire the Scheduler to prevent other tasks from running.
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.
EnableAllInterrupts()
loop WHILE no pending/validated 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)
Release Scheduler resource to allow other tasks to run.
WAKEUP VALIDATION

Figure 9.6: CAN controller or transceiver wake up by polling

After the detection of a wake up event from the CAN transceiver or controller by either interrupt or polling, the wake up event can be validated (see [SWS_EcuM_02566]). This is done by switching on the corresponding CAN transceiver and controller in EcuM_StartWakeupSources (see [SWS_EcuM_02924]). It depends on the used CAN transceivers and controllers, which function calls in Integrator Code EcuM_Start WakeupSource are necessary. In Figure 9.7 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 (CanIf) to any upper layer module.

Only when the corresponding PDU channel modes of the CanIf are set to "Online", it will forward CAN messages.

The CanIf recognizes the successful reception of at least one message and records it as a successful validation. During validation the ECU State Manager module regularly checks the CanIf in Integrator Code EcuM_CheckValidation (see [SWS_EcuM_02925]).

The ECU State Manager 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 transceiver in EcuM_StopWakeup Sources (see [SWS_EcuM_02926]) and go back to sleep.

The resulting sequence is shown in Figure 9.7 .

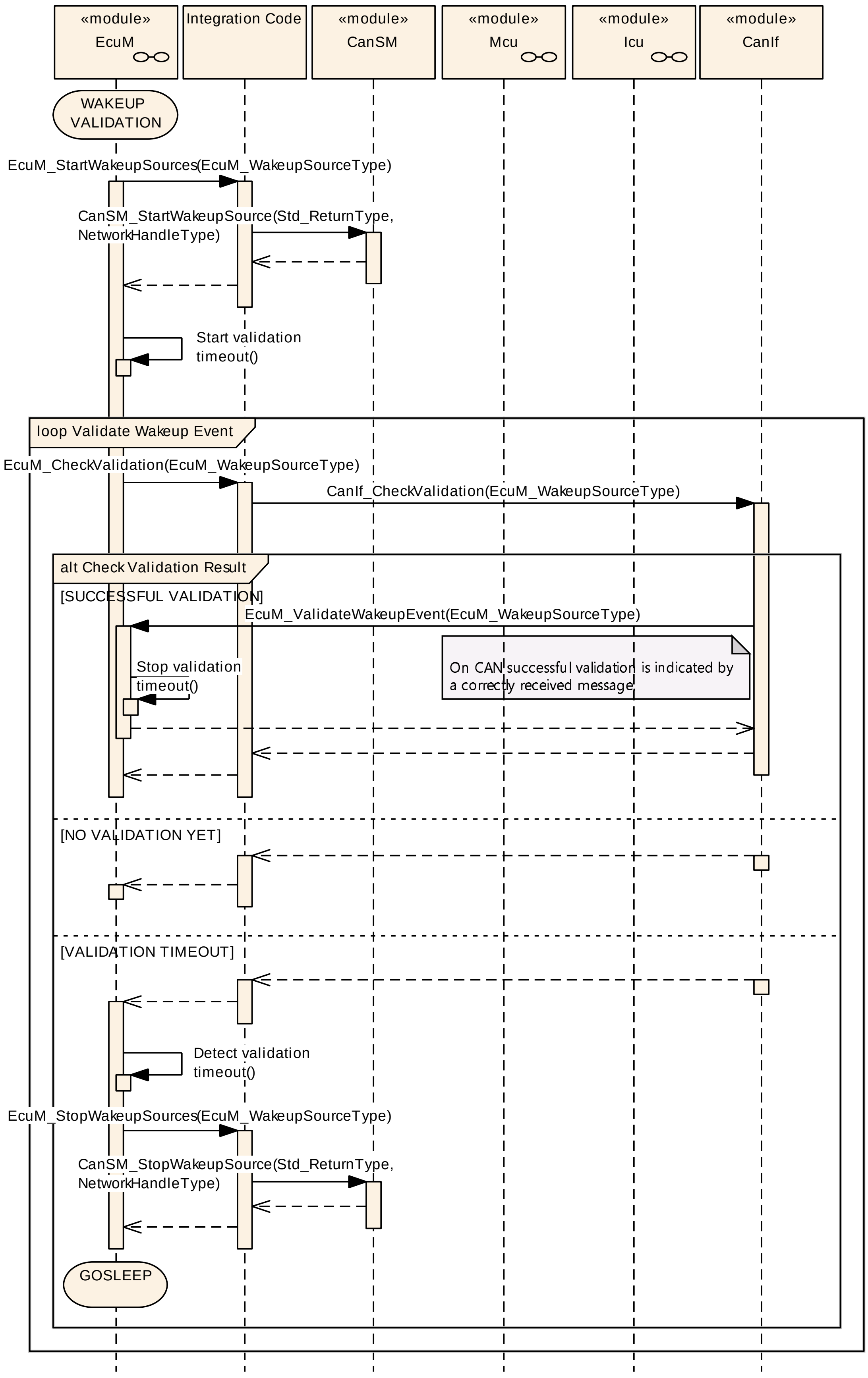

«module» EcuM
Integration Code
«module» CanSM
«module» Mcu
«module» Icu
«module» CanIf
WAKEUP VALIDATION
EcuM_StartWakeupSources(EcuM_WakeupSourceType)
CanSM_StartWakeupSource(Std_ReturnType, NetworkHandleType)
Start validation timeout()
loop Validate Wakeup Event
EcuM_CheckValidation(EcuM_WakeupSourceType)
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 9.7: CAN wake up validation

9.2.4 LIN Wakeup Sequences

Figure 9.8 shows the LIN transceiver wakeup via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2 .

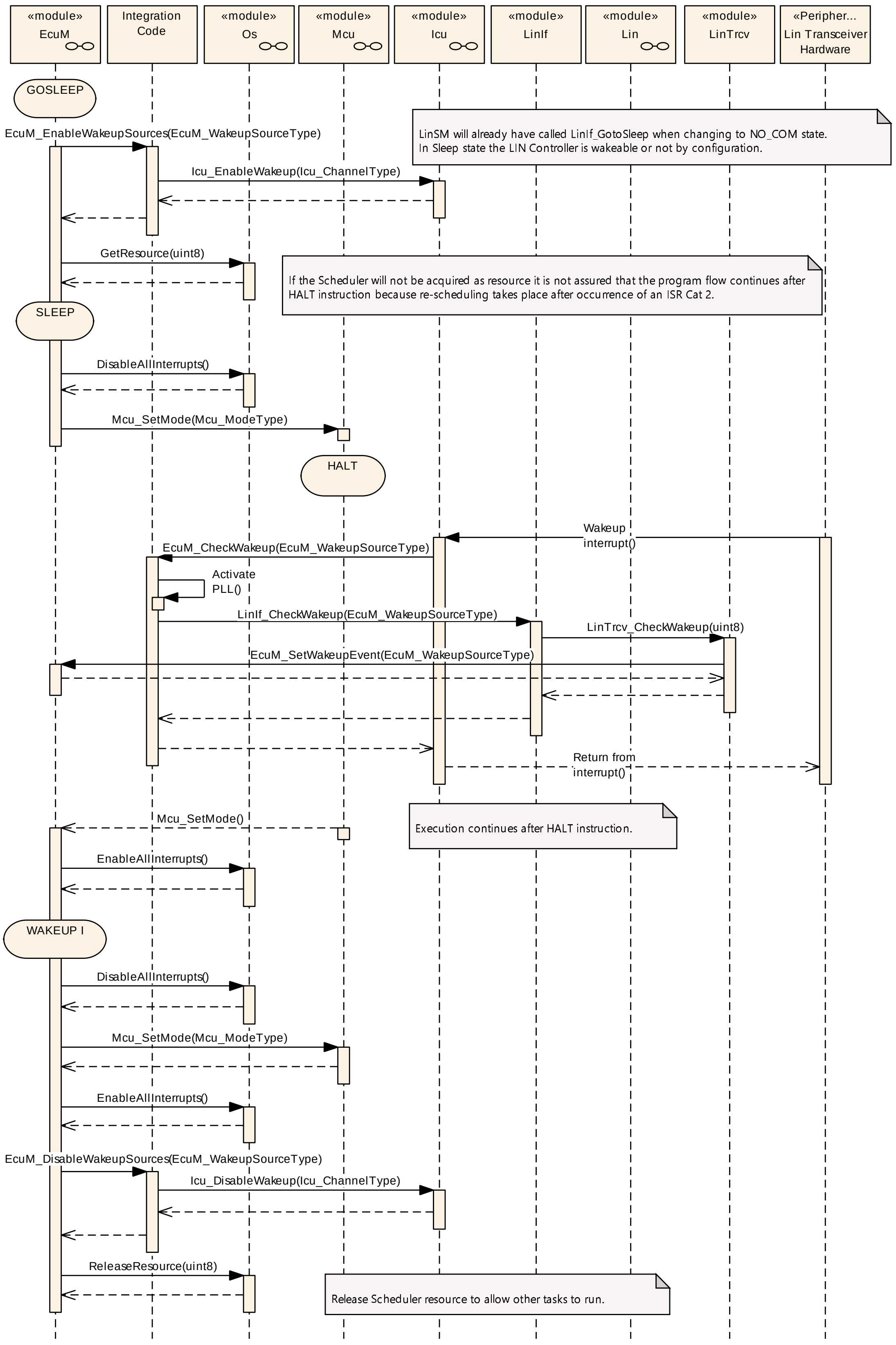

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» LinIf
«module» Lin
«module» LinTrcv
«Peripher... Lin Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
LinSM will already have called LinIf_GotoSleep when changing to NO_COM state. In Sleep state the LIN Controller is wakeable or not by configuration.
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()
LinIf_CheckWakeup(EcuM_WakeupSourceType)
LinTrcv_CheckWakeup(uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Mcu_SetMode()
Execution continues after HALT instruction.
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Icu_DisableWakeup(Icu_ChannelType)
ReleaseResource(uint8)
Release Scheduler resource to allow other tasks to run.

Figure 9.8: LIN transceiver wake up by interrupt

As shown in Figure 9.9 , 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.

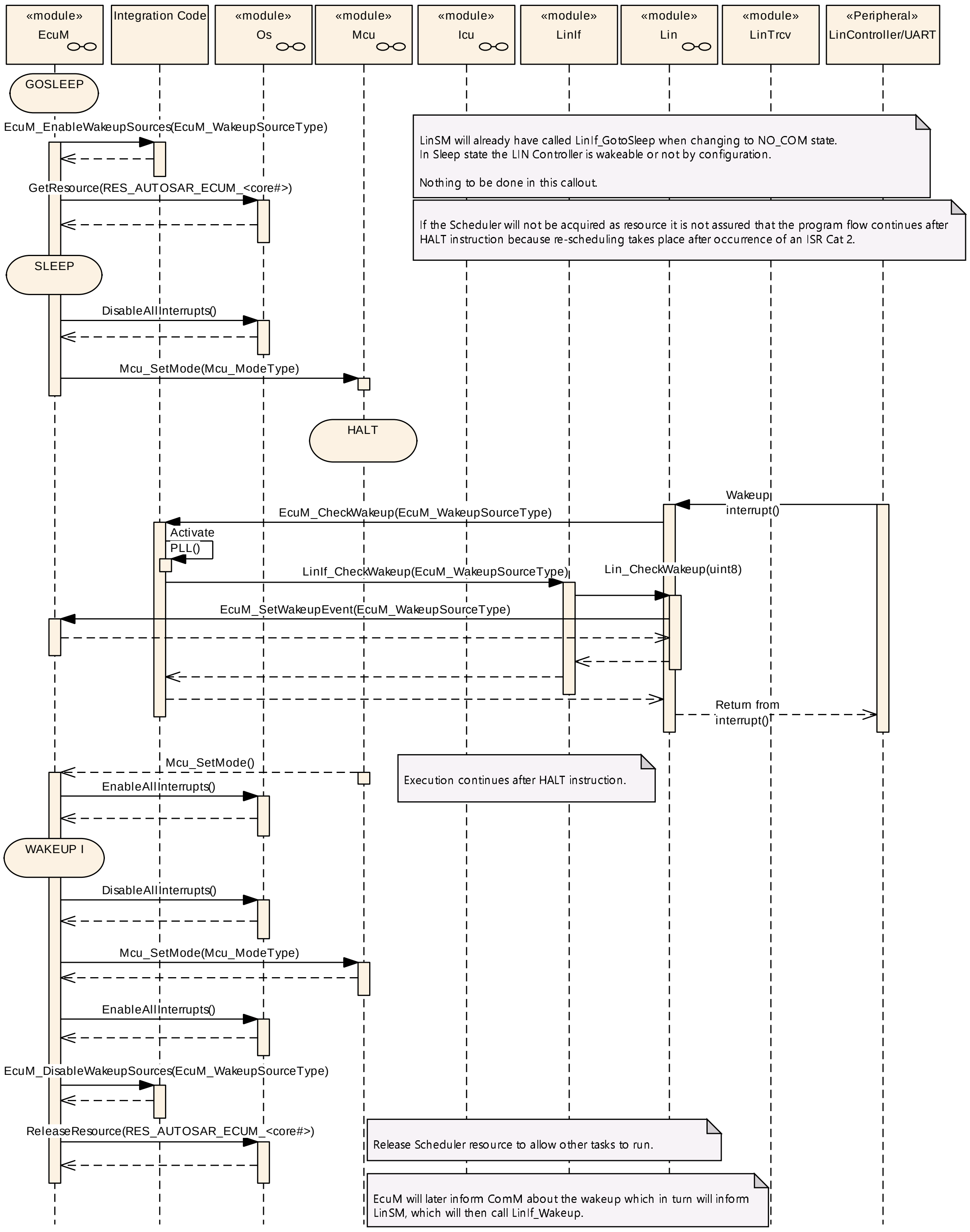

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» LinIf
«module» Lin
«module» LinTrcv
«Peripheral» LinController/UART
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
LinSM will already have called LinIf_GotoSleep when changing to NO_COM state.
In Sleep state the LIN Controller is wakeable or not by configuration.
Nothing to be done in this callout.
GetResource(RES_AUTOSAR_ECUM_<core#>)
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()
LinIf_CheckWakeup(EcuM_WakeupSourceType)
Lin_CheckWakeup(uint8)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
Mcu_SetMode()
EnableAllInterrupts()
Execution continues after HALT instruction.
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
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 9.9: LIN controller wake up by interrupt

Wake up by polling is possible for LIN transceiver and controller. The ECU State Manager 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 9.10 .

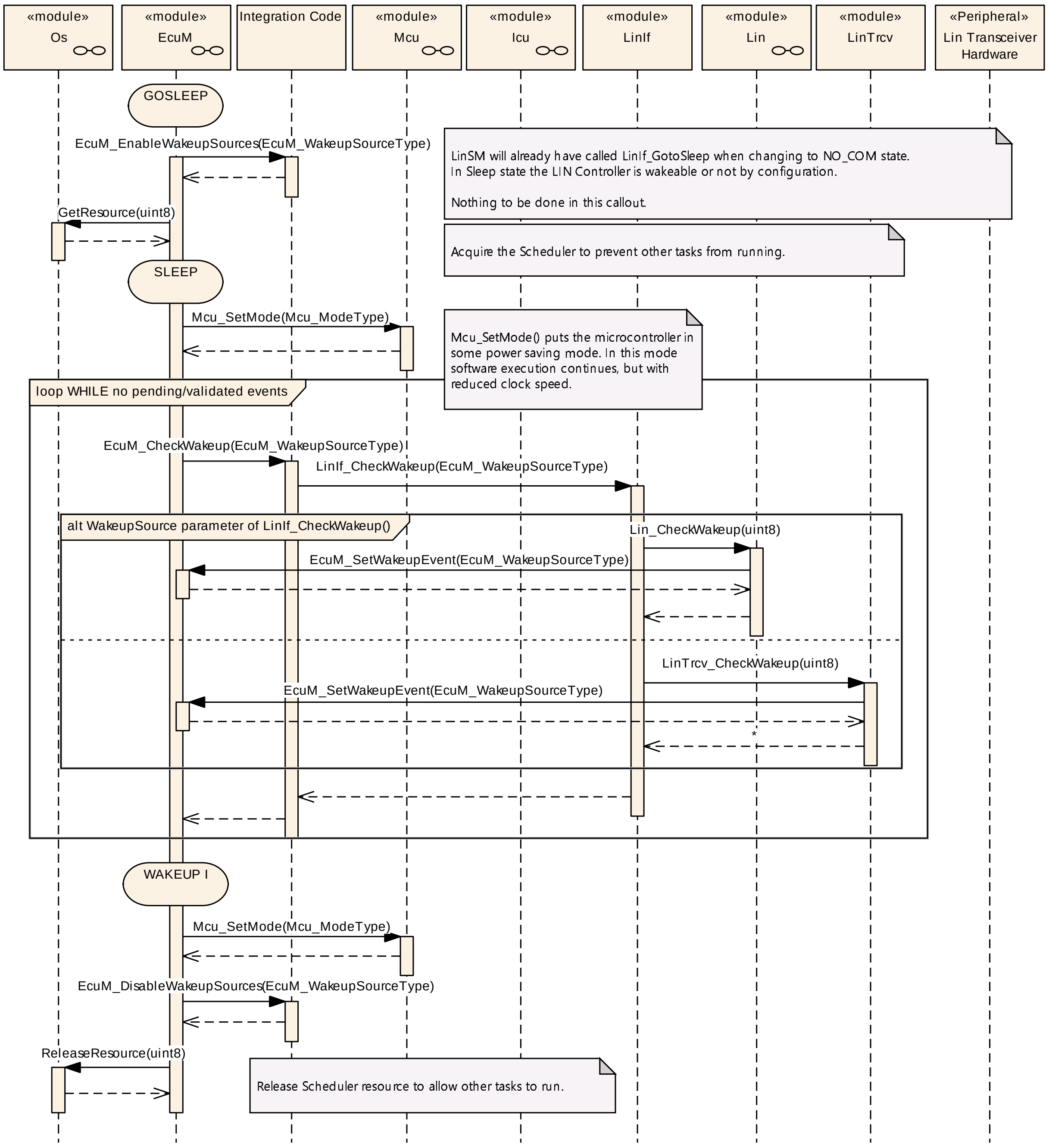

«module» Os
«module» EcuM
Integration Code
«module» Mcu
«module» Icu
«module» LinIf
«module» Lin
«module» LinTrcv
«Peripheral» Lin Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
LinSM will already have called LinIf_GotoSleep when changing to NO_COM state. In Sleep state the LIN Controller is wakeable or not by configuration.
Nothing to be done in this callout.
GetResource(uint8)
Acquire the Scheduler to prevent other tasks from running.
SLEEP
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.
loop WHILE no pending/validated 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)
Release Scheduler resource to allow other tasks to run.

Figure 9.10: LIN controller or transceiver wake up by polling

Note that LIN does not require wakeup validation.

9.2.5 FlexRay Wakeup 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 9.11 shows the FlexRay transceiver wakeup via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.

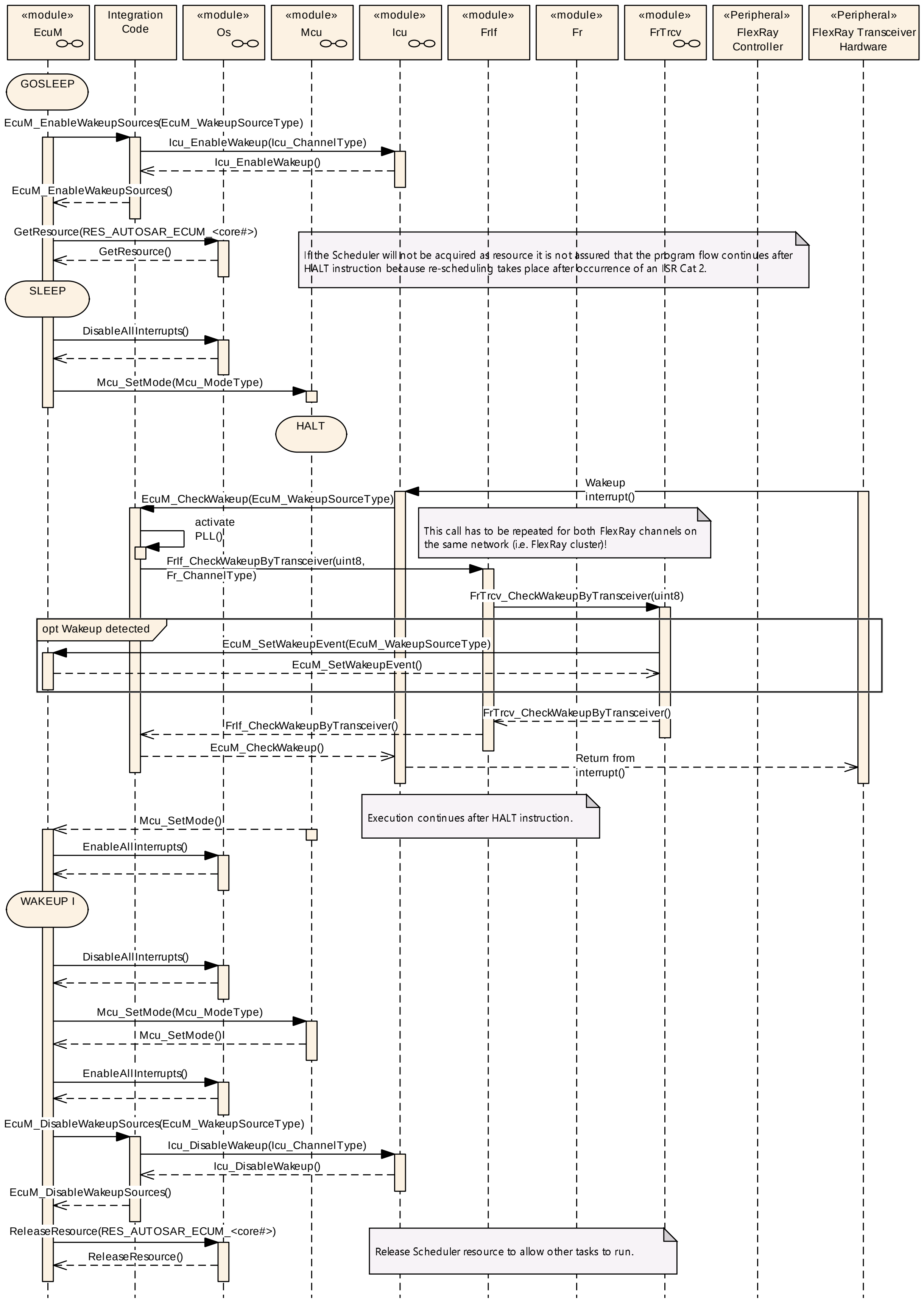

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» FrIf
«module» Fr
«module» FrTrcv
«Peripheral» FlexRay Controller
«Peripheral» FlexRay Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Icu_EnableWakeup(Icu_ChannelType)
Icu_EnableWakeup()
EcuM_EnableWakeupSources()
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 9.11: FlexRay transceiver wake up by interrupt

Note that in EcuM_CheckWakeup (see [SWS_EcuM_02929]) there need to be two separate calls to FrIf_WakeupByTransceiver, one for each FlexRay channel.

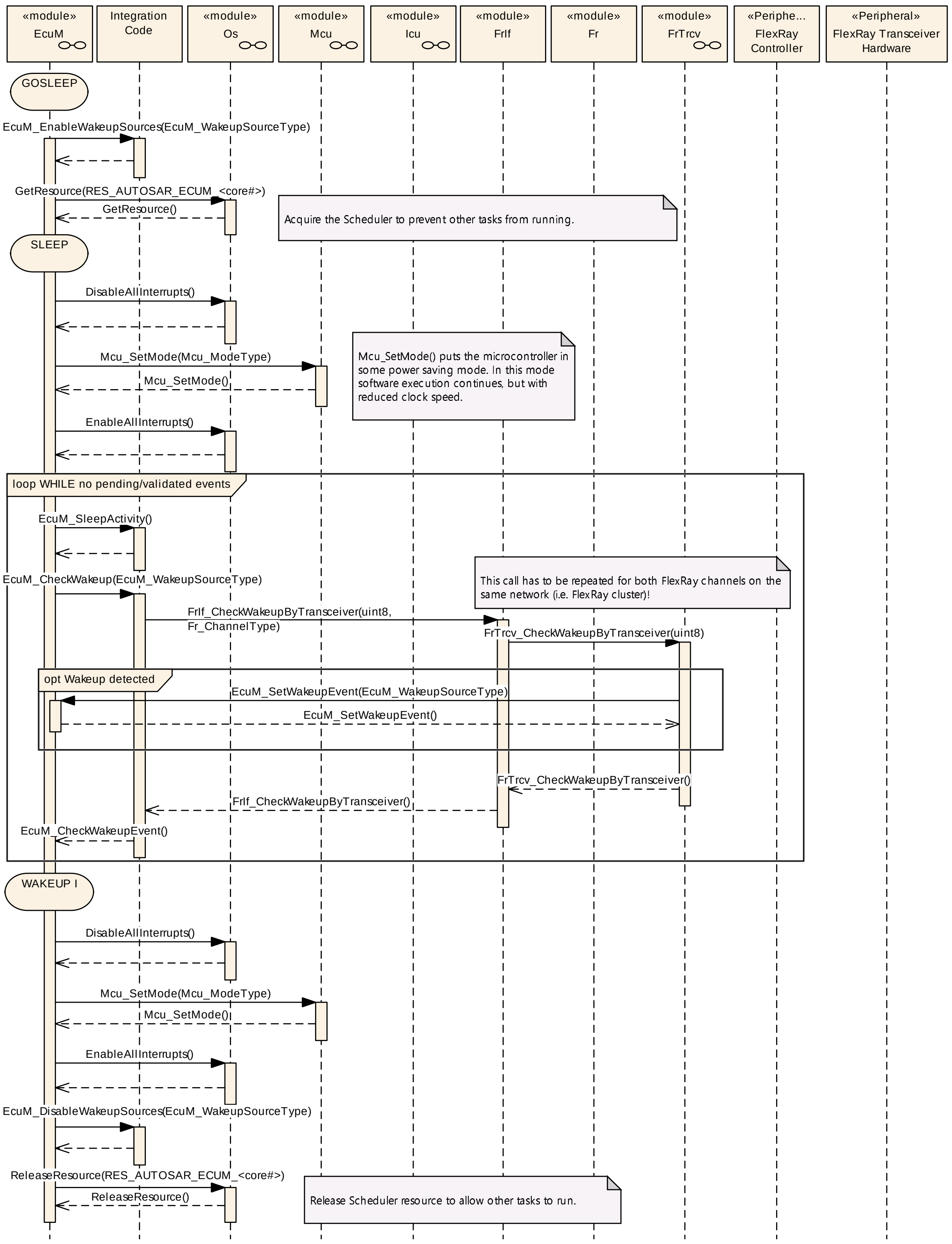

«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» FrIf
«module» Fr
«module» FrTrcv
«Periphe... FlexRay Controller
«Peripheral» FlexRay Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
GetResource(RES_AUTOSAR_ECUM_<core#>)
GetResource()
Acquire the 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/validated 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 9.12: FlexRay transceiver wake up by polling

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers.

Chapters 10.1 and 10.2 specify the structure (containers) and the parameters of the module ECU Manager.

Chapter 10.3 specifies published information of the module ECU State Manager.

10.1 Common Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapters 7 and Chapter 8.

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.

10.1.1 EcuM

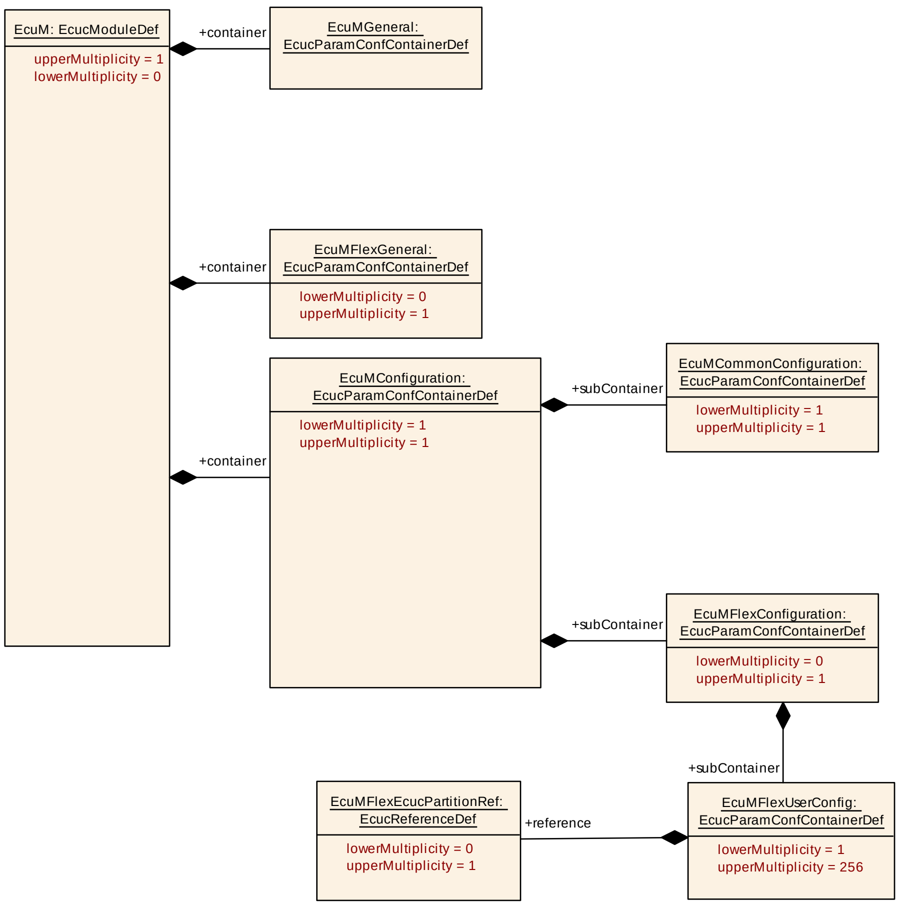


Figure 10.1: EcuM configuration overview

Module 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, VARIANT-PRE-COMPILE	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
EcuMConfiguration	1	This container contains the configuration (parameters) of the ECU State Manager.
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.

10.1.2 EcuMGeneral

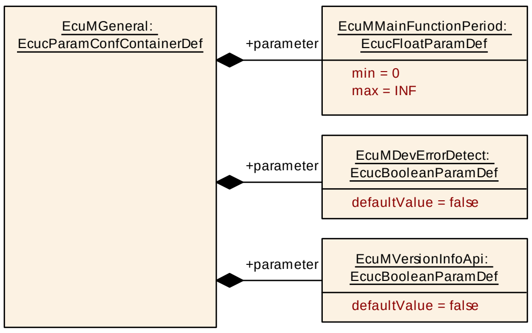

EcuMGeneral: EcucParamConfContainerDef
+parameter
EcuMMainFunctionPeriod: EcucFloatParamDef
min = 0
max = INF
+parameter
EcuMDevErrorDetect: EcucBooleanParamDef
defaultValue = false
+parameter
EcuMVersionInfoApi: EcucBooleanParamDef
defaultValue = false

Figure 10.2: EcuMGeneral configuration overview

SWS Item	[ECUC_EcuM_00116]
Container Name	EcuMGeneral
Parent Container	EcuM
Description	This container holds the general, pre-compile configuration parameters.
Configuration Parameters	

Name	EcuMDevErrorDetect [ECUC_EcuM_00108]		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMMainFunctionPeriod [ECUC_EcuM_00121]	
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	EcuMVersionInfoApi [ECUC_EcuM_00149]		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.1.3 EcuMConfiguration

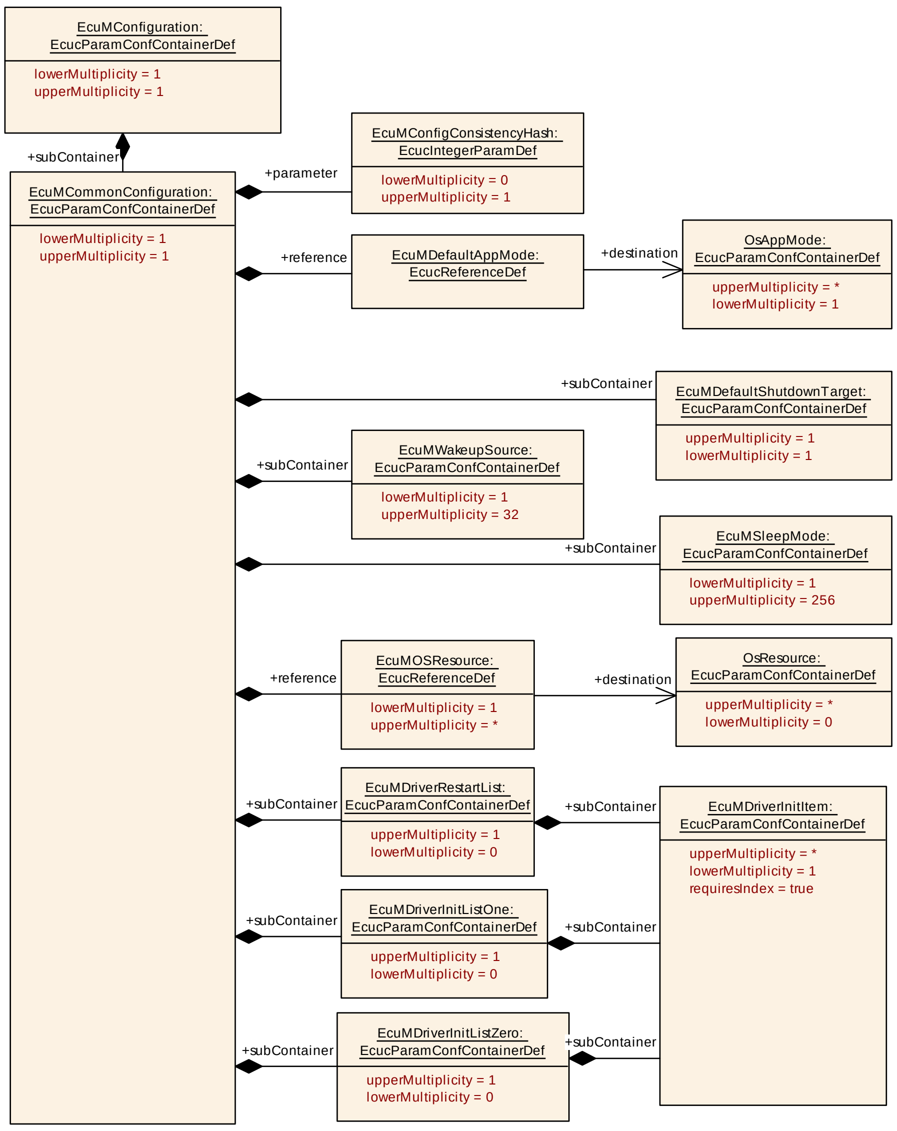


Figure 10.3: EcuMConfiguration configuration overview

SWS Item	[ECUC_EcuM_00103]
Container Name	EcuMConfiguration
Parent Container	EcuM
Description	This container contains the configuration (parameters) of the ECU State Manager.
Configuration Parameters	

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
EcuMCommon Configuration	1	This container contains the common configuration (parameters) of the ECU State Manager.

EcuMFlexConfiguration	0..1	This container contains the configuration (parameters) of the EcuMFlex. Only applicable if EcuMFlex is implemented.

10.1.4 EcuMCommonConfiguration

SWS Item	[ECUC_EcuM_00181]
Container Name	EcuMCommonConfiguration
Parent Container	EcuMConfiguration
Description	This container contains the common configuration (parameters) of the ECU State Manager.
Configuration Parameters	

Name	EcuMConfigConsistencyHash [ECUC_EcuM_00102]		
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. Note: In systems which do not make use of post-build configurations this parameter can be omitted.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default Value			
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	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	EcuMDefaultAppMode [ECUC_EcuM_00104]		
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**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	EcuMOSResource [ECUC_EcuM_00183]		
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	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		

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.

10.1.5 EcuMDefaultShutdownTarget

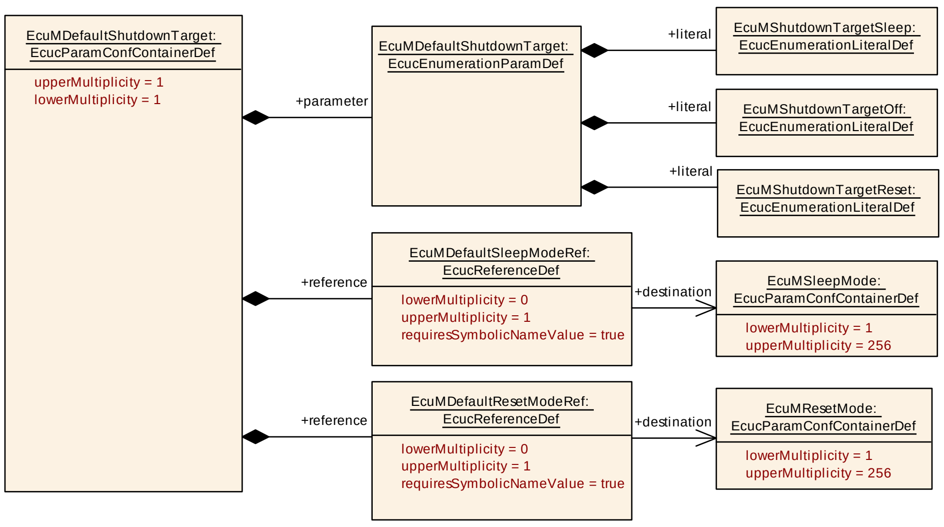

EcuMDefaultShutdownTarget: EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 1
+parameter
EcuMDefaultShutdownTarget: EcucEnumerationParamDef
+literal
EcuMShutdownTargetSleep: EcucEnumerationLiteralDef
+literal
EcuMShutdownTargetOff: EcucEnumerationLiteralDef
+literal
EcuMShutdownTargetReset: EcucEnumerationLiteralDef
+reference
EcuMDefaultSleepModeRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
EcuMSleepMode: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 256
+reference
EcuMDefaultResetModeRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
EcuMResetMode: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 256

Figure 10.4: EcuMDefaultShutdownTarget configuration overview

SWS Item	[ECUC_EcuM_00105]
Container Name	EcuMDefaultShutdownTarget
Parent Container	EcuMCommonConfiguration
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	

Name	EcuMDefaultShutdownTarget [ECUC_EcuM_00107]		
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.	
	EcuMShutdownTarget Reset	Corresponds to ECUM_SHUTDOWN_TARGET_RESET in EcuM_ShutdownTargetType. This literal is only be applicable for EcuMFlex.	
	EcuMShutdownTarget Sleep	Corresponds to ECUM_SHUTDOWN_TARGET_SLEEP in EcuM_ShutdownTargetType.	
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	EcuMDefaultResetModeRef [ECUC_EcuM_00205]		
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**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	EcuMDefaultSleepModeRef [ECUC_EcuM_00106]		
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**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.1.6 EcuMDriverInitListOne

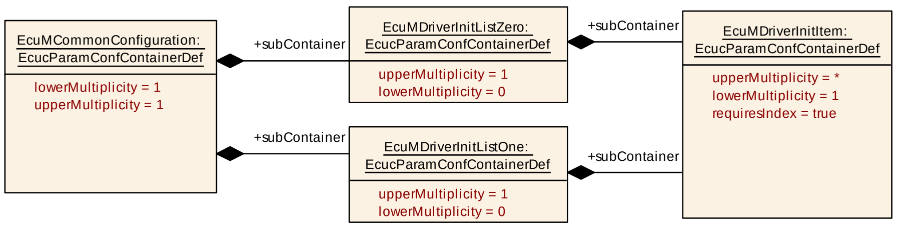


Figure 10.5: EcuMInitLists configuration overview

SWS Item	[ECUC_EcuM_00111]
Container Name	EcuMDriverInitListOne
Parent Container	EcuMCommonConfiguration
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.
Post-Build Variant Multiplicity	false

Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

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

10.1.7 EcuMDriverInitListZero

SWS Item	[ECUC_EcuM_00114]		
Container Name	EcuMDriverInitListZero		
Parent Container	EcuMCommonConfiguration		
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.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

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

10.1.8 EcuMDriverRestartList

SWS Item	[ECUC_EcuM_00115]		
Container Name	EcuMDriverRestartList		
Parent Container	EcuMCommonConfiguration		
Description	List of modules to be initialized.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

Configuration Parameters		

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

10.1.9 EcuMDriverInitItem

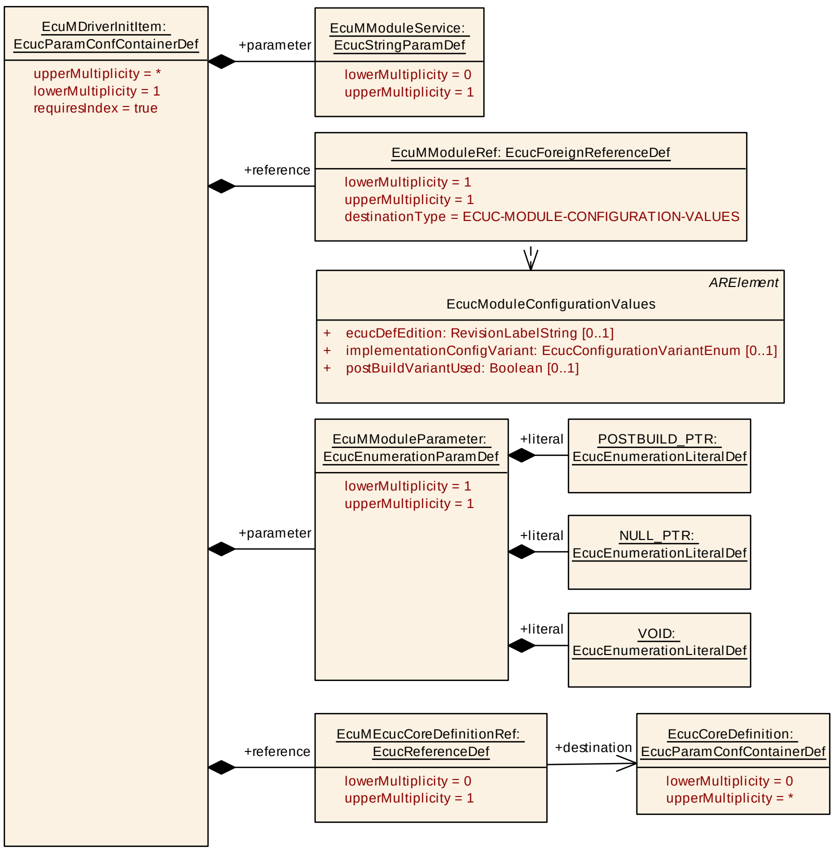

EcuMDriverInitItem: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity = 1
requiresIndex = true
+parameter
EcuMModuleService: EcucStringParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+reference
EcuMModuleRef: EcucForeignReferenceDef
lowerMultiplicity = 1
upperMultiplicity = 1
destinationType = ECUC-MODULE-CONFIGURATION-VALUES
ARElement
EcucModuleConfigurationValues
+ ecucDefEdition: RevisionLabelString [0..1]
+ implementationConfigVariant: EcucConfigurationVariantEnum [0..1]
+ postBuildVariantUsed: Boolean [0..1]
+parameter
EcuMModuleParameter: EcucEnumerationParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
+literal
POSTBUILD_PTR: EcucEnumerationLiteralDef
+literal
NULL_PTR: EcucEnumerationLiteralDef
+literal
VOID: EcucEnumerationLiteralDef
+reference
EcuMEcucCoreDefinitionRef: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
+destination
EcucCoreDefinition: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *

Figure 10.6: EcuMDriverInitItem configuration overview

SWS Item	[ECUC_EcuM_00110]		
Container Name	EcuMDriverInitItem		
Parent Container	EcuMDriverInitListBswM, EcuMDriverInitListOne, EcuMDriverInitListZero, EcuMDriverRestartList		
Description	These containers describe the entries in a driver init list. **Attributes:** requiresIndex=true		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

Configuration Parameters

Name	EcuMModuleParameter [ECUC_EcuM_00224]		
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		

Name	EcuMModuleService [ECUC_EcuM_00124]
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	
Regular Expression	
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		

Name	EcuMEcucCoreDefinitionRef [ECUC_EcuM_00229]		
Parent Container	EcuMDriverInitItem		
Description	Reference denotes the core the EcuM AUTOSAR services shall be offered on.		
Multiplicity	0..1		
Type	Reference to EcucCoreDefinition		
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		

Name	EcuMModuleRef [ECUC_EcuM_00223]		
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

10.1.10 EcuMSleepMode

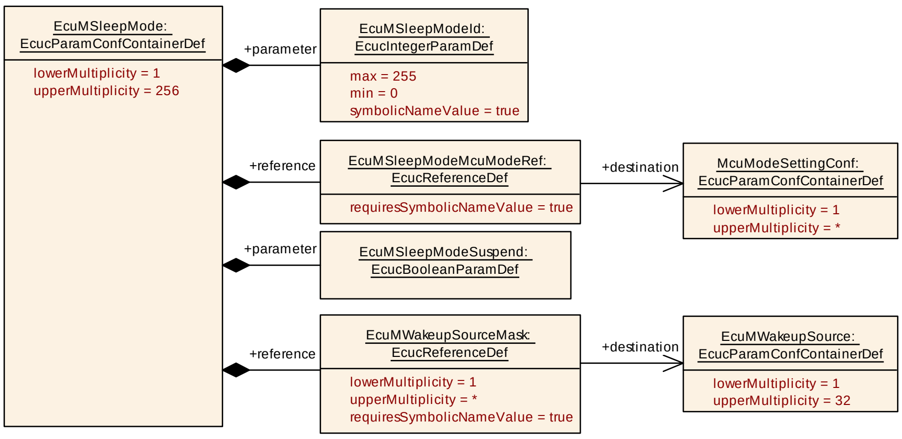

EcuMSleepMode: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 256
+parameter
EcuMSleepModeId: EcucIntegerParamDef
max = 255
min = 0
symbolicNameValue = true
+reference
EcuMSleepModeMcuModeRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
McuModeSettingConf: EcucParamConfContainerDef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

Figure 10.7: EcuMSleepMode configuration overview

SWS Item	[ECUC_EcuM_00131]		
Container Name	EcuMSleepMode		
Parent Container	EcuMCommonConfiguration		
Description	These containers describe the configured sleep modes. The names of these containers specify the symbolic names of the different sleep modes.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMSleepModeId [ECUC_EcuM_00132]		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	EcuMSleepModeSuspend [ECUC_EcuM_00136]		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMSleepModeMcuModeRef [ECUC_EcuM_00133]		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMWakeupSourceMask [ECUC_EcuM_00152]		
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	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		

No Included Containers

10.1.11 EcuMWakeupSource

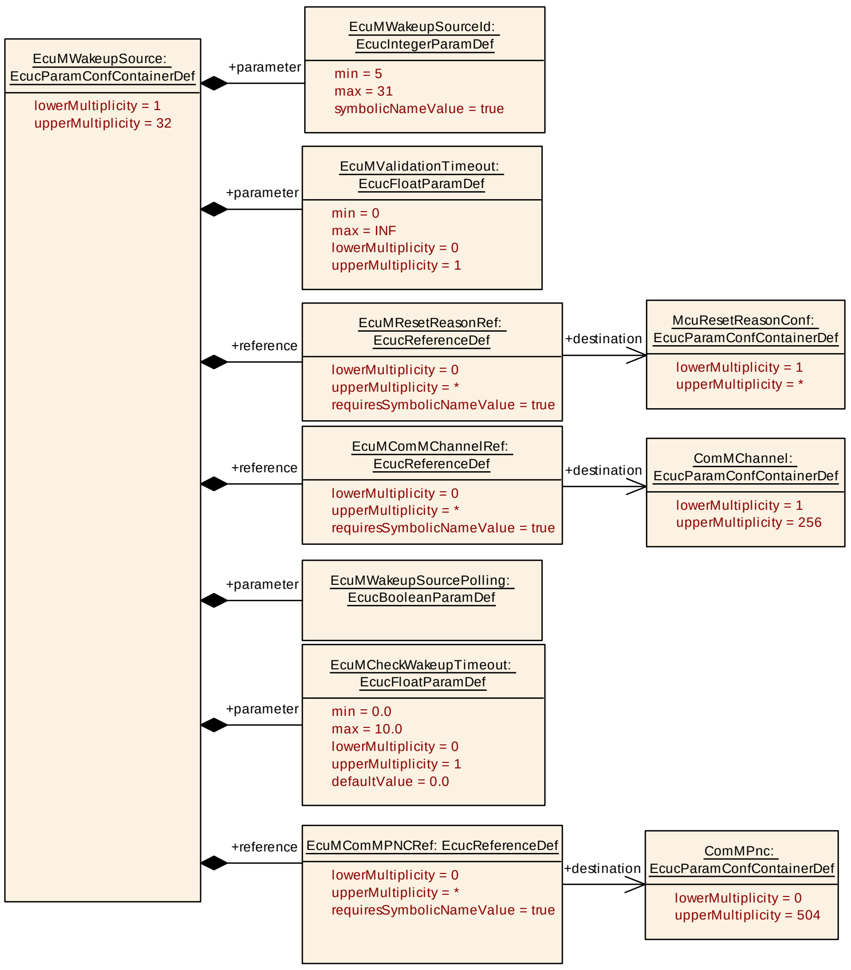


Figure 10.8: EcuMWakeupSource configuration overview

SWS Item	[ECUC_EcuM_00150]		
Container Name	EcuMWakeupSource		
Parent Container	EcuMCommonConfiguration		
Description	These containers describe the configured wakeup sources.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	EcuMCheckWakeupTimeout [ECUC_EcuM_00208]		
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		

Name	EcuMValidationTimeout [ECUC_EcuM_00148]		
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	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		

Name	EcuMWakeupSourceId [ECUC_EcuM_00151]		
Parent Container	EcuMWakeupSource		
Description	This parameter defines the identifier of this wakeup source. The first five bits are reserved values from the EcuM_WakeupSourceType.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	5 .. 31		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	EcuMWakeupSourcePolling [ECUC_EcuM_00153]		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMComMChannelRef [ECUC_EcuM_00101]		
Parent Container	EcuMWakeupSource		
Description	This parameter could reference multiple Networks (channels) defined in the Communication Manager. No reference indicates that the wakeup source is not a communication channel.		
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	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		

Name	EcuMComMPNCRef [ECUC_EcuM_00228]		
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		

Name	EcuMResetReasonRef [ECUC_EcuM_00128]		
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	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		

No Included Containers

10.2 EcuM-Flex Containers and configuration parameters

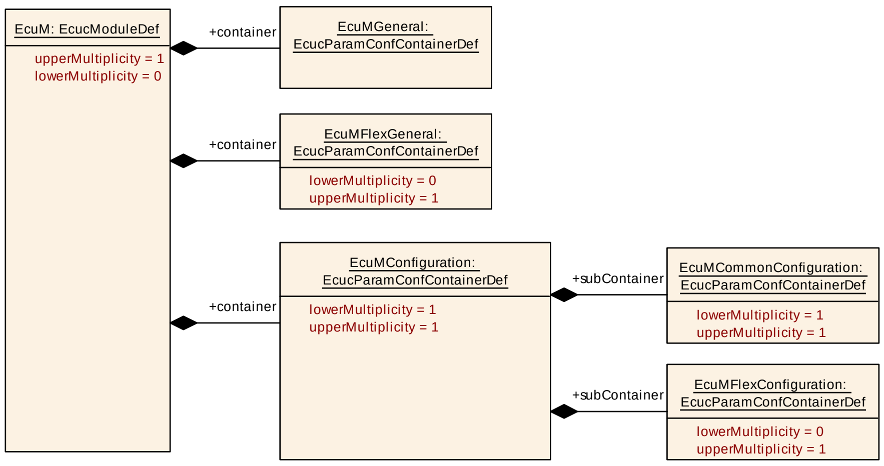


Figure 10.9: EcuMFlex configuration overview

10.2.1 EcuMFlexGeneral

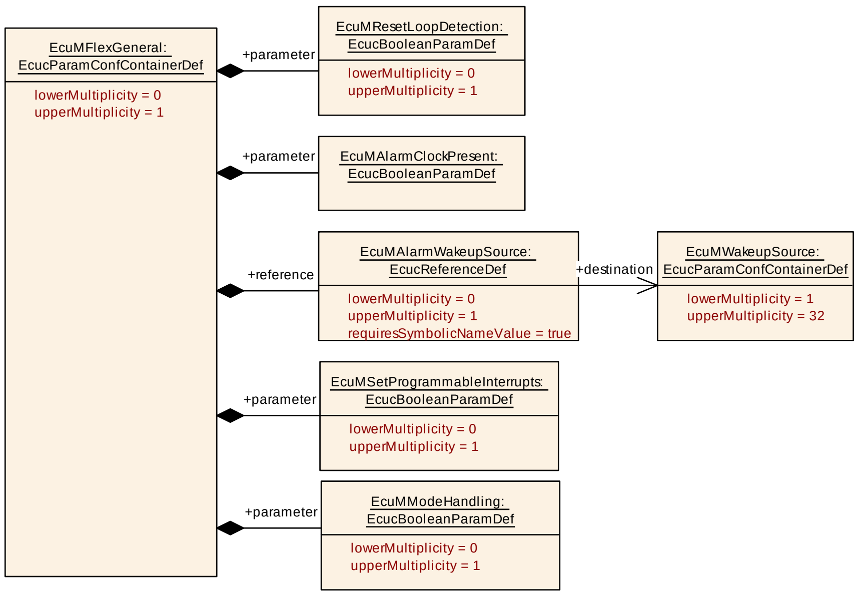

EcuMFlexGeneral: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
EcuMResetLoopDetection: EcucBooleanParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
EcuMAlarmClockPresent: EcucBooleanParamDef
+reference
EcuMAlarmWakeupSource: EcucReferenceDef
lowerMultiplicity = 0
upperMultiplicity = 1
requiresSymbolicNameValue = true
+destination
EcuMWakeupSource: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 32
+parameter
EcuMSetProgrammableInterrupts: EcucBooleanParamDef
lowerMultiplicity = 0
upperMultiplicity = 1
+parameter
EcuMModeHandling: EcucBooleanParamDef
lowerMultiplicity = 0
upperMultiplicity = 1

Figure 10.10: EcuMFlexGeneral configuration overview

SWS Item	[ECUC_EcuM_00168]
Container Name	EcuMFlexGeneral
Parent Container	EcuM
Description	This container holds the general, pre-compile configuration parameters for the EcuMFlex. Only applicable if EcuMFlex is implemented.
Post-Build Variant Multiplicity	false

Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMAlarmClockPresent [ECUC_EcuM_00199]		
Parent Container	EcuMFlexGeneral		
Description	This flag indicates whether the optional AlarmClock feature is present.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMModeHandling [ECUC_EcuM_00221]		
Parent Container	EcuMFlexGeneral		
Description	If false, Run Request Protocol is not performed.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value			
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		

Name	EcuMResetLoopDetection [ECUC_EcuM_00171]
Parent Container	EcuMFlexGeneral
Description	If false, no reset loop detection is performed. If this configuration parameter exists and is set to true, the callout "EcuM_LoopDetection" is called during startup of EcuM (during StartPreOS).
Multiplicity	0..1
Type	EcucBooleanParamDef
Default Value	

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		

Name	EcuMSetProgrammableInterrupts [ECUC_EcuM_00210]		
Parent Container	EcuMFlexGeneral		
Description	If this configuration parameter exists and is to true, the callout "EcuM_AL_SetProgrammableInterrupts" is called during startup of EcuM (during StartPreOS).		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value			
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		

Name	EcuMAlarmWakeupSource [ECUC_EcuM_00200]
Parent Container	EcuMFlexGeneral
Description	This parameter describes the reference to the EcuMWakeupSource being used for the EcuM AlarmClock.
Multiplicity	0..1
Type	Symbolic name reference to EcuMWakeupSource
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		

No Included Containers

10.2.2 EcuMFlexConfiguration

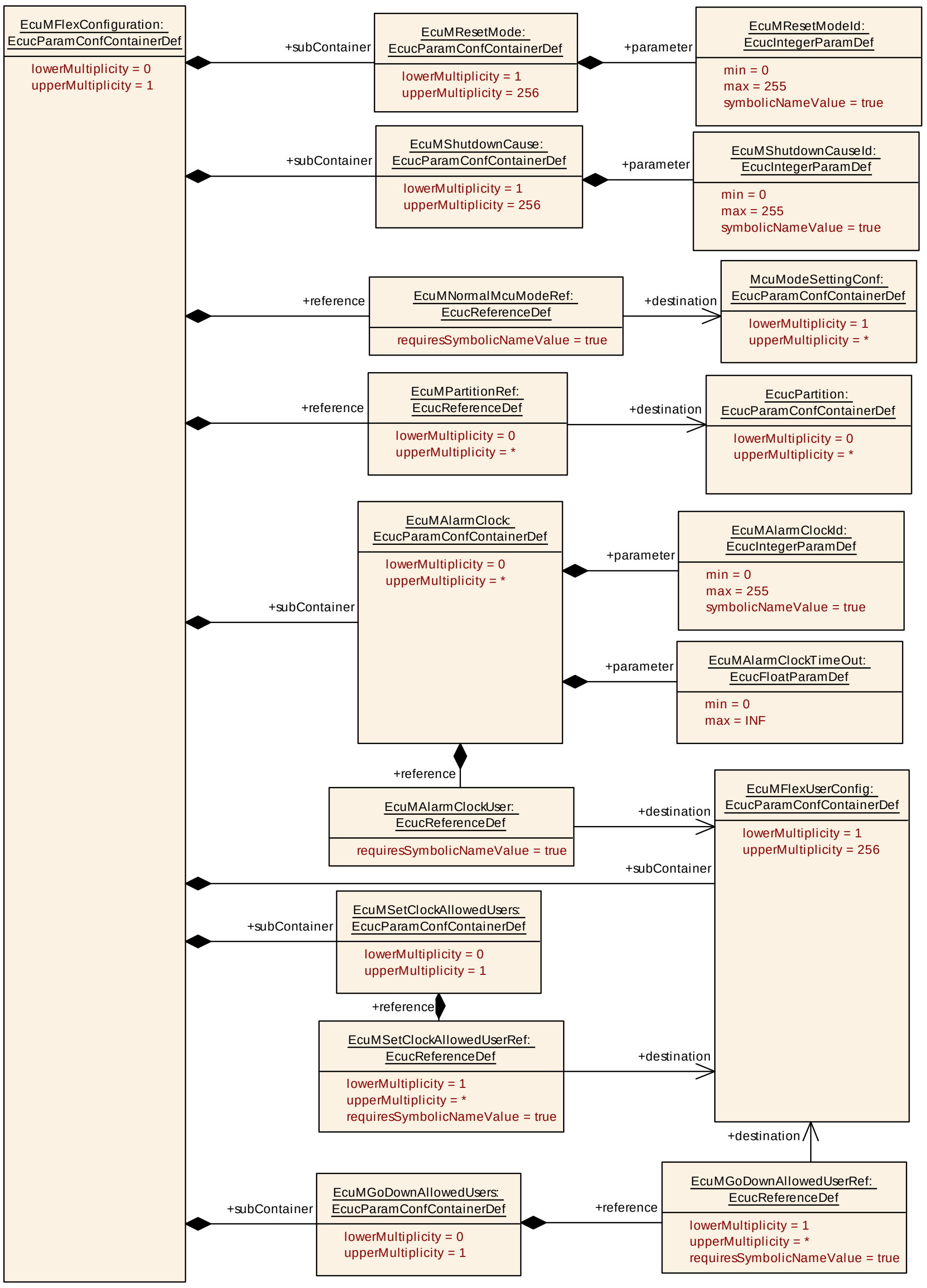

EcuMFlexConfiguration: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
EcuMResetMode: EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 256
+parameter
EcuMResetModeId: EcucIntegerParamDef
min = 0
max = 255
symbolicNameValue = true
EcuMShutdownCause: EcucParamConfContainerDef
EcuMShutdownCauseId: EcucIntegerParamDef
+reference
EcuMNormalMcuModeRef: EcucReferenceDef
requiresSymbolicNameValue = true
+destination
McuModeSettingConf: EcucParamConfContainerDef
upperMultiplicity = *
EcuMPartitionRef: EcucReferenceDef
EcucPartition: EcucParamConfContainerDef
EcuMAlarmClock: EcucParamConfContainerDef
EcuMAlarmClockId: EcucIntegerParamDef
EcuMAlarmClockTimeOut: EcucFloatParamDef
max = INF
EcuMAlarmClockUser: EcucReferenceDef
EcuMFlexUserConfig: EcucParamConfContainerDef
EcuMSetClockAllowedUsers: EcucParamConfContainerDef
EcuMSetClockAllowedUserRef: EcucReferenceDef
EcuMGoDownAllowedUsers: EcucParamConfContainerDef
EcuMGoDownAllowedUserRef: EcucReferenceDef

Figure 10.11: EcuMFlexConfiguration configuration overview

SWS Item	[ECUC_EcuM_00167]
Container Name	EcuMFlexConfiguration
Parent Container	EcuMConfiguration
Description	This container contains the configuration (parameters) of the EcuMFlex. Only applicable if EcuMFlex is implemented.

Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMNormalMcuModeRef [ECUC_EcuM_00204]		
Parent Container	EcuMFlexConfiguration		
Description	This parameter is a reference to the normal MCU mode to be restored after a sleep.		
Multiplicity	1		
Type	Symbolic name reference to McuModeSettingConf		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMPartitionRef [ECUC_EcuM_00217]		
Parent Container	EcuMFlexConfiguration		
Description	Reference denotes the partition a EcuM shall run inside. Please note that in case of a multicore ECU this reference is mandatory.		
Multiplicity	0..*		
Type	Reference to EcucPartition		
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		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
EcuMAlarmClock	0..*	These containers describe the configured alarm clocks. The name of these conatiners allows giving a symbolic name to one alarm clock.

EcuMDriverInitListBswM	0..*	This container holds a list of modules to be initialized by the BswM.
EcuMFlexUserConfig	1..256	These containers describe the identifiers that are needed to refer to a software component or another appropriate entity in the system which uses the EcuMFlex Interfaces.
EcuMGoDownAllowedUsers	0..1	This container describes the collection of allowed users which are allowed to call the EcuM_GoDownHaltPoll API (only applies in the case that the previously set shutdown target is TARGET_RESET or TARGET_OFF).
EcuMResetMode	1..256	These containers describe the configured reset modes. The name of these containers allows one of the following symbolic names to be given to the different reset modes: • ECUM_RESET_MCU • ECUM_RESET_WDG • ECUM_RESET_IO.
EcuMSetClockAllowedUsers	0..1	This container describes the collection of allowed users which are allowed to call the EcuM_SetClock API.
EcuMShutdownCause	1..256	These containers describe the configured shut down or reset causes. The name of these containers allows to give one of the following symbolic names to the different shut down causes: • ECUM_CAUSE_ECU_STATE - ECU state machine entered a state for shutdown, • ECUM_CAUSE_WDGM - WdgM detected failure, • ECUM_CAUSE_DCM - Dcm requests shutdown (split into UDS services?), • and values from configuration.

10.2.3 EcuMAlarmClock

SWS Item	[ECUC_EcuM_00184]		
Container Name	EcuMAlarmClock		
Parent Container	EcuMFlexConfiguration		
Description	These containers describe the configured alarm clocks. The name of these conatiners allows giving a symbolic name to one alarm clock.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMAlarmClockId [ECUC_EcuM_00186]		
Parent Container	EcuMAlarmClock		
Description	This ID identifies this alarmclock.		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMAlarmClockTimeOut [ECUC_EcuM_00188]		
Parent Container	EcuMAlarmClock		
Description	This parameter allows to define a timeout for this alarm clock.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMAlarmClockUser [ECUC_EcuM_00195]		
Parent Container	EcuMAlarmClock		
Description	This parameter allows an alarm to be assigned to a user.		
Multiplicity	1		
Type	Symbolic name reference to EcuMFlexUserConfig		
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

10.2.4 EcuMDriverInitListBswM

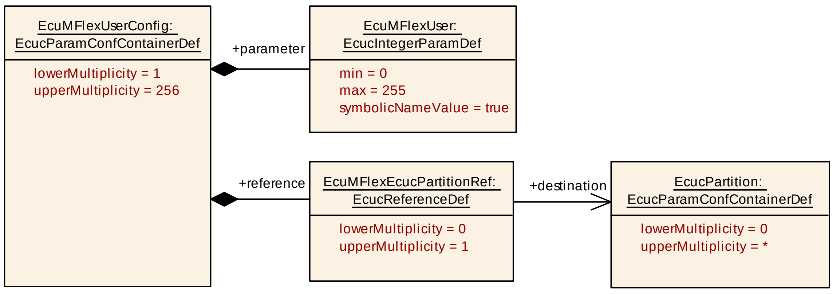


Figure 10.12: EcuMFlexUserConfig configuration overview

SWS Item	[ECUC_EcuM_00201]		
Container Name	EcuMFlexUserConfig		
Parent Container	EcuMFlexConfiguration		
Description	These containers describe the identifiers that are needed to refer to a software component or another appropriate entity in the system which uses the EcuMFlex Interfaces.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMFlexUser [ECUC_EcuM_00146]		
Parent Container	EcuMFlexUserConfig		
Description	Parameter used to identify one user.		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	EcuMFlexEcucPartitionRef [ECUC_EcuM_00203]
Parent Container	EcuMFlexUserConfig
Description	Denotes in which "EcucPartition" the user of the EcuM is executed.
Multiplicity	0..1
Type	Reference to EcucPartition
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		

No Included Containers

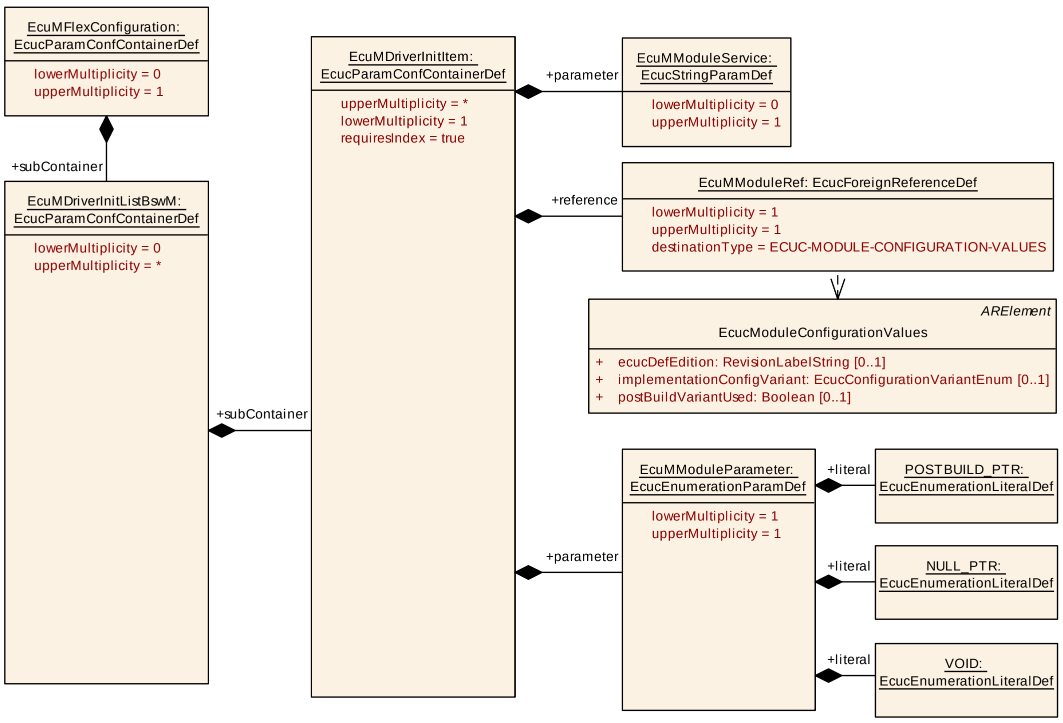

EcuMFlexConfiguration: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = 1
+subContainer
EcuMDriverInitListBswM: EcucParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+subContainer
EcuMDriverInitItem: EcucParamConfContainerDef
upperMultiplicity = *
lowerMultiplicity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

Figure 10.13: EcuMFlexDriverInitListBswM configuration overview

SWS Item	[ECUC_EcuM_00226]		
Container Name	EcuMDriverInitListBswM		
Parent Container	EcuMFlexConfiguration		
Description	This container holds a list of modules to be initialized by the BswM.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

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

10.2.5 EcuMGoDownAllowedUsers

SWS Item	[ECUC_EcuM_00206]		
Container Name	EcuMGoDownAllowedUsers		
Parent Container	EcuMFlexConfiguration		
Description	This container describes the collection of allowed users which are allowed to call the EcuM_GoDownHaltPoll API (only applies in the case that the previously set shutdown target is TARGET_RESET or TARGET_OFF).		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMGoDownAllowedUserRef [ECUC_EcuM_00207]		
Parent Container	EcuMGoDownAllowedUsers		
Description	This references an allowed user.		
Multiplicity	1..*		
Type	Symbolic name reference to EcuMFlexUserConfig		
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		

No Included Containers

10.2.6 EcuMResetMode

SWS Item	[ECUC_EcuM_00172]

Container Name	EcuMResetMode		
Parent Container	EcuMFlexConfiguration		
Description	These containers describe the configured reset modes. The name of these containers allows one of the following symbolic names to be given to the different reset modes: • ECUM_RESET_MCU • ECUM_RESET_WDG • ECUM_RESET_IO.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMResetModeId [ECUC_EcuM_00173]		
Parent Container	EcuMResetMode		
Description	This ID identifies this reset 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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.7 EcuMSetClockAllowedUsers

SWS Item	[ECUC_EcuM_00175]
Container Name	EcuMShutdownCause
Parent Container	EcuMFlexConfiguration

Description	These containers describe the configured shut down or reset causes. The name of these containers allows to give one of the following symbolic names to the different shut down causes: • ECUM_CAUSE_ECU_STATE - ECU state machine entered a state for shutdown, • ECUM_CAUSE_WDGM - WdgM detected failure, • ECUM_CAUSE_DCM - Dcm requests shutdown (split into UDS services?), • and values from configuration.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMShutdownCauseId [ECUC_EcuM_00176]		
Parent Container	EcuMShutdownCause		
Description	This ID identifies this shut down cause.		
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	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

SWS Item	[ECUC_EcuM_00197]		
Container Name	EcuMSetClockAllowedUsers		
Parent Container	EcuMFlexConfiguration		
Description	This container describes the collection of allowed users which are allowed to call the EcuM_SetClock API.		
Post-Build Variant Multiplicity	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Configuration Parameters			

Name	EcuMSetClockAllowedUserRef [ECUC_EcuM_00198]		
Parent Container	EcuMSetClockAllowedUsers		
Description	These parameters describe the references to the users which are allowed to call the EcuM_SetClock API.		
Multiplicity	1..*		
Type	Symbolic name reference to EcuMFlexUserConfig		
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		

No Included Containers

10.3 Published Information

Currently there exists no published information except the ones specified in SWS BSW General.

A Not applicable requirements

[SWS_EcuM_NA_00000] ⌈These requirements are not applicable to this specification.⌋*(SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00406, SRS_BSW_00437, SRS_BSW_00168, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00432, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00325, SRS_BSW_00164, SRS_BSW_00160, SRS_BSW_00453, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00450, SRS_BSW_00410, SRS_BSW_00314, SRS_BSW_00348, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00439, SRS_BSW_00449, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00330, SRS_BSW_00010, SRS_BSW_00341, SRS_BSW_00334)*