Document Title	Specification of ECU State Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	078
Document Classification	Standard

Document Version	2.1.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Ver.	Changed by	Change Description
28.02.2014	2.1.0	AUTOSAR Release Management	• CanSM is now involved in WakeupValidation and is called by EcuM in EcuM_StartWakeupSource and EcuM_StopWakeupSource • Introduced EcuM_KillAllPostRUNRequests to kill PostRUN requests • Editorial changes • Removed chapter(s) on change documentation
26.05.2012	2.0.0	AUTOSAR Administration	• Improved asynchronous transceiver handling for CAN Wake up (added EcuM_StartCheckWakeup EcuM_EndCheckWakeup) • Update of Wake-up Sequence II (initialization of modules from Init List III on wake up by wake up source with integrated power control) • Post-build parameters for multi purpose ECUs: fixed configuration parameter tables (EcuMConfiguration) • Clarification of shutdown target in case wake up validation fails. • Clarification about NvM_CancelWriteAll in shutdown paths

Document Change History

Date	Ver.	Changed by	Change Description
07.04.2011	1.4.0	AUTOSAR Administration	• Fixed possible loss of CAN wake up and improved wake up handling. • Changed behavor of EcuM_KillAllRUNRequests so that this API does only influence requests to RUN, but not to POST_RUN. • A SW-C which needs to be able to keep track of the current mode shall use the mode switch interface EcuM_CurrentMode. • Re-assign call of EcuM_SetWakeupEvent from integrator code to transceiver driver. • Legal disclaimer revised
16.09.2010	1.3.0	AUTOSAR Administration	• Added EcuM3020 • Fixed description in EcuM2904 • Update description ErrorHook • Change of AppMode • Update ErrorHook with note • Added note for exit from GO SLEEP • Reformulated EcuM2863 and added rationale • Added a note to EcuM_AL_SwitchOff • Legal disclaimer revised
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised
11.12.2007	1.2.0	AUTOSAR Administration	• Fixed wake up 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

Document Change History			
Date	**Ver.**	**Changed by**	**Change Description**
31.01.2007	1.1.0	AUTOSAR Administration	• Corrected startup flow and wake up 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
28.06.2006	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Functional Overview

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

- is responsible for the initialization and de-initialization of all basic software modules including OS and RTE,
- cooperates with the Communication Manager, and hence indirectly with network management, to shut down the ECU when needed,
- manages all wake up events and configures the ECU for SLEEP when requested.

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

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

These protocols were specified with the following underlying constraints:

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

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

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

1.2 Conventions Used in this Specification

1.2.1 Font Faces

EcuM123: Requirements are tagged with an ID in bold font.

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

`Source code` is printed in a Courier font.

`Configuration Parameters` are printed in Courier Italic.

STATE names are written in capital letters.

1.2.2 Figures

Figure X - Title (diagram type)

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

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

In the present version of this documentation, there is only one top level diagram: The main state machine.
The next level is covered by high level diagrams. There are five high level sequence diagrams:

Figure 3 – Startup Sequence (high level diagram)
Figure 7 – RUN State Sequence (high level diagram)
Figure 11 – Shutdown Sequence (high level diagram)
Figure 16 – Sleep Sequence (high level diagram)
Figure 19 – Wake up Sequence (high level diagram)

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

2 Definitions and Acronyms

Term	**Description**
Inoperation	An artificial word to describe the ECU when it is not operational, i.e. not running. Comprises all meanings of *off, sleeping, frozen*, etc. Using this definition is beneficial since it has no predefined meaning.
Shutdown Target	The shutdown of an ECU may end up in different states, depending on what application requires or desires for the next shutdown. By selecting a shutdown target, the application can communicate its wishes to the ECU State Manager module. SLEEP, OFF, and RESET are shutdown targets.
Callback	Within this document, the term 'callback' is used for API services which are intended for notifications to other BSW modules.
Callout	Within this document, the term 'callout' is used for function stubs which can be filled by the system designer, usually at configuration time, with the purpose to add functionality to the ECU State Manager module. Callouts are separated into two classes, where one class is optional to be filled. The other class is mandatory and serves as a hardware abstraction layer.
ECU Firmware	In this specification ECU Firmware does not refer to any AUTOSAR module. It is a placeholder for pieces of code that have to be added at configuration and integration time. The ECU firmware implements all the Callouts of the ECU State Manager module. Some of the callouts are also used by other BSW modules.
Passive Wake up	A wake up caused from an attached bus rather than an internal event like a timer or sensor activity.
Post run	Post run is the period from when the application detects a reason to start the shutdown until the shutdown actually occurs. Typically this period starts when all network communication is put to sleep and lasts until the ECU is put to sleep.
Vital Data	Any kind of data (RAM or NVRAM) that must stay consistent to ensure correct operation of the ECU. E.g. stacks, important state variables, etc.
Wake up Event	A physical event which causes a wake up. A CAN message or a toggling IO line can be wake up events. Similarly, the internal SW representation, e.g. an interrupt, may also be called a wake up event.
Wake up Reason	The wake up reason is the wake up event being the actual cause of the last wake up.
Wake up Source	The peripheral or ECU component which deals with wake up events is called a wake up source.

Acronym	**Description**
BSW	Basic Software
BSWM	Basic Software Module
ISR	Interrupt Service Routine
RTE	Runtime Environment
SW-C	Software Component (above the RTE)
TTII	Time-Triggered Increased Inoperation

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

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

[4] Requirements on Mode Management
AUTOSAR_SRS_ModeManagement.pdf

3.2 Related standards and norms

None

3.3 Related AUTOSAR Software Specifications

[5] Glossary
AUTOSAR_Glossary.pdf

[6] Specification of Communication Manager
AUTOSAR_SWS_ComManager.pdf

[7] Specification of Watchdog Manager
AUTOSAR_SWS_WatchdogManager.pdf

[8] Specification of CAN Interface
AUTOSAR_SWS_CAN_Interface.pdf

[9] Specification of LIN Interface
AUTOSAR_SWS_LIN_Interface.pdf

[10] Specification of FlexRay Interface
AUTOSAR_SWS_FlexRayInterface.pdf

[11] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAM_Manager.pdf

[12] Specification of MCU Driver
AUTOSAR_SWS_MCU_Driver.pdf

[13] Specification of SPI Handler/Driver

AUTOSAR_SWS_SPIHandlerDriver.pdf

[14] Specification of EEPROM Interface
AUTOSAR_SWS_EEPROM_Driver.pdf

[15] Specification of Flash Interface
AUTOSAR_SWS_Flash_Driver.pdf

[16] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[17] Specification of RTE
AUTOSAR_SWS_RTE.pdf

[18] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DEM.pdf

[19] Specification of Development Error Tracer
AUTOSAR_SWS_DET.pdf

[20] Specification of CAN Transceiver Driver
AUTOSAR_SWS_CAN_TransceiverDriver.pdf

[21] Specification of C Implementation Rules
AUTOSAR_SWS_C_ImplementationRules.pdf

[22] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

4 Constraints and Assumptions

4.1 Limitations

The shutdown target OFF requires special hardware on the ECU so that it can actually be reached (e.g. a power hold circuit). If this hardware is not available, this specification proposes to issue a reset instead but other default behaviors can be defined.

EcuM2522: Applications (SW-C's) shall not assume that it is actually possible to switch off ECUs (i.e. power consumption is zero).

4.2 Hardware Requirements

The following requirements are needed to support switching the application mode (see *7.11.1* Application Modes). Other basic software modules also need this requirement.

EcuM2261: ECU RAM must keep contents of vital data while ECU clock is switched off. This requirement is needed to implement sleep states as required in *7.6* SLEEP State.

EcuM2262: ECU RAM must provide a no init area which keeps contents over a reset cycle. A no init area shall only be initialized on a power on event (clamp 30). The system designer is responsible for establishing an initialization strategy.

4.3 Applicability to car domains

The ECU State 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 have to fulfill to collaborate correctly with the ECU State Manager module.

5.1 Mode Management Modules

5.1.1 Communication Manager

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

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

5.1.2 Watchdog Manager

The Watchdog Manager is initialized by the ECU State Manager module.

The ECU State Manager module also switches Watchdog Manager modes when it changes its states.

Furthermore, the ECU State Manager module is one of the Supervised Entities of the Watchdog Manager.

5.2 SPAL Modules

5.2.1 MCU Driver

The MCU Driver is the first basic software module initialized by the ECU State Manager module. However, returning `MCU_Init`, the MCU and the MCU driver are not necessarily fully initialized. Additional, MCU specific steps may be needed. The ECU State Manager module provides a callout where this additional code can be placed. For details on how this code should look like refer to [12].

5.2.2 Driver Dependencies and Initialization Order

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

---

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

EcuM2502: The system designer is responsible for defining the initialization order at configuration time.

5.3 Peripherals with Wake up Capability

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

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

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

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

5.4 Operating System

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

5.5 Runtime Environment (RTE)

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

The ECU State Manager module shall use the mode port feature of RTE to notify about state changes. See chapter *7.10* AUTOSAR Ports for more information.

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

5.6 BSW Scheduler

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

5.7 NVRAM Manager

The following operations of the NVRAM Manager [11] are executed by the ECU State Manager module.

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

NVRAM is not read during the wake up sequence since RAM contents is assumed to be still valid from the previous cycle. To verify this, RAM integrity is checked[4]. NVRAM is only read during the STARTUP.

The NVRAM Manager shall call the callbacks defined in chapter *8.5.1* Callbacks from NVRAM Manager to notify the ECU State Manager module about job status.

5.8 Diagnostic Event Manager

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

5.9 Software Components

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

- Application modes[5]
- Setting of shutdown targets

It is assumed in this specification that these properties are set by the application (through AUTOSAR ports), typically by some ECU specific part of the application. The ECU State Manager module does not prohibit two application overriding each other’s settings. The policy must be defined at a higher level.
The following two requirements formulate an attempt to resolve this issue.

---

[4] See *8.6.4.6* EcuM_GenerateRamHash and *8.6.5.1* EcuM_CheckRamHash for details.
[5] In this context, ‘application mode’ is a technical term which is defined by the AUTOSAR OS specification.

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

The generation tool may only allow configuration which have only one SW-C accessing application mode or shutdown target.

5.10 File Structure

EcuM2675: The file structure shall be as follows:

- One or more C file `EcuM_xxx.c` containing the entire or parts of ECU State Manager code
- One C file `EcuM_PBcfg.c` containing post build time configuration.
- One file `EcuM_Callout_Stubs.c` containing the stubs of the defined callouts. Whether this file shall be modified directly or includes other generated files is specific to the implementation.
- An API interface `EcuM.h` providing the fix type declarations, forward declaration to generated types, and function prototypes
- A type header `EcuM_Generated_Types.h` providing generated types and fulfills the forward declarations from `EcuM.h`.
- A type header `EcuM_Cfg.h` providing the configuration parameters
- A callback/callout interface `EcuM_Cbk.h` providing the callback/callout function prototype

EcuM2676: It shall only be necessary to include `EcuM.h` to use all services of the ECU State Manager module.

EcuM2677: It shall only be necessary to include `EcuM_Cbk.h` to interact with the callbacks and callouts of the ECU State Manager module.

EcuM2862: The ECU State Manager module implementation shall include `SchM_EcuM.h` and `MemMap.h`.

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

6 Requirements traceability

Document: General Requirements on Basic Software Modules [3]

Requirement		Satisfied by
[BSW00344]	Reference to link-time configuration	EcuM2500 EcuM does not define configuration sets but references the init configuration, e.g. for driver initialization
[BSW00404]	Reference to post build time configuration	
[BSW00405]	Reference to multiple configuration sets	
[BSW00345]	Pre-compile-time configuration	*10.2* Configurable Parameters *5.10* File Structure
[BSW159]	Tool-based configuration	not applicable (EcuM does not specify the configuration tool)
[BSW167]	Static configuration checking	*10.2* Configurable Parameters
[BSW171]	Configurability of optional functionality	*10.2* Configurable Parameters
[BSW00380]	Separate C-files for configuration parameters	*5.10* File Structure
[BSW00419]	Separate C-files for pre-compile-time configuration parameters	
[BSW00381]	Separate configuration header files for pre-compile-time parameters	
[BSW00412]	Separate H-file for configuration parameters	
[BSW00383]	List dependencies to other configuration files	*10.2* Configurable Parameters
[BSW00384]	List dependencies to other modules	*5* Dependencies to other modules *8.7*Expected Interfaces
[BSW00387]	Specify the configuration class of a callback function	*8.5* Callback Definitions
[BSW00388] - [BSW00400]		*10.2* Configurable Parameters
[BSW00402]	Published information	*10.3* Published Parameters
[BSW00375]	Notification of wake up reason	*8.3.4* Wake up
[BSW101]	Initialization interface	*8.3.2.1* EcuM_Init
[BSW00416]	Sequence of initialization	EcuM2559
[BSW00406]	Check module initialization	not applicable (EcuM initializes the BSW, hence EcuM is always initialized from the point of view of any other BSW module.)
[BSW00435]	Header File Structure for the Basic Software Scheduler	EcuM2862
[BSW00436]	Module Header File Structure for the Basic Software Memory Mapping	EcuM2862
[BSW168]	Diagnostic Interface of SW components	not applicable (EcuM has no testing requirements)
[BSW00407]	Function to read out published parameters	*8.3.1.1* EcuM_GetVersionInfo
[BSW00423]	Usage of SW-C template to describe BSW modules with AUTOSAR interfaces	*7.10* AUTOSAR Ports
[BSW00424]	BSW main processing function task	Implementation of EcuM_MainFunction

	allocation	according to this specification does not require extended task mechanisms.
[BSW00425]	Trigger conditions for schedulable objects	*8.4.1* EcuM_MainFunction
[BSW00426]	Exclusive areas in BSW modules	not applicable (EcuM does not specify directly accessible global data.)
[BSW00427]	ISR description for BSW modules	not applicable (EcuM does not specify ISRs.)
[BSW00428]	Execution order dependencies of main processing functions	There are no requirements of this sort.
[BSW00429]	Restricted BSW OS functionality access	EcuM does not use any other than the allowed OS services.
[BSW00431]	The BSW Scheduler module implements task bodies	EcuM does not define any task body.
[BSW00432]	Modules should have separated main processing functions for a read/receive and write/transmit data path	not applicable (EcuM does not specify RxTx functionality.)
[BSW00433]	Calling of main processing functions	EcuM does not call any main processing function.
[BSW00434]	The Schedule Module shall provide an API for exclusive areas	not applicable (This is not an EcuM requirement)
[BSW00336]	Shutdown interface	*8.3.2.3* EcuM_Shutdown
Fault Operation and Error Detection		
[BSW00337]	Classification of errors	Table 5 - Error Classification
[BSW00338]	Detection and reporting of development errors	Table 5 - Error Classification
[BSW00369]	Do not return development error codes via API	*8* API specification
[BSW00339]	Reporting of production relevant error statuses	EcuM2759
[BSW00417]	Reporting of Error Events by Non-Basic Software	not applicable
[BSW00323]	API parameter checking	8.8 API Parameter Checking
[BSW004]	Version check	*10.3* Published Parameters
[BSW00409]	Header files for production code error IDs	*5.10* File Structure
[BSW00385]	List possible error notifications	Table 5 - Error Classification
[BSW00386]	Configuration for detecting errors	*7.12* Error Classification
[BSW161]	Microcontroller abstraction	not applicable (Requirements related to layered software architecture are reflected by the EcuM SRS)
[BSW162]	ECU layout abstraction	
[BSW005]	No hard coded horizontal interfaces within MCAL	
[BSW00415]	User dependent include files	not applicable (EcuM does not define user specific functionality)
[BSW164]	Implementation of ISRs	not applicable (EcuM does not specify ISRs.)
[BSW00325]	Runtime of ISRs	
[BSW00326]	Transition from ISRs to OS task	
[BSW00342]	Usage of source code and object code.	*5.10* File Structure
[BSW00343]	Specification and configuration of time	*10.2* Configurable Parameters
[BSW160]	Human-readable configuration data	not applicable (This specification does not define the configuration file)
[BSW007]	HIS MISRA C	The API definition complies with MISRA C.

		8 API specification
[BSW00300]	Module naming conventions.	*5.10* File Structure
[BSW00413]	Accessing instances of BSW modules	not applicable (EcuM defines only one instance.)
[BSW00347]	Naming separation of different instances of BSW drivers	
[BSW00305]	Self-defined data types naming conventions	*8.2* Type definitions
[BSW00307]	Global variables naming convention	not applicable (EcuM does not specify global variables.)
[BSW00310]	API naming conventions	*8* API specification
[BSW00373]	Main processing function naming convention	*8.4.1* EcuM_MainFunction
[BSW00327]	Error values naming convention	Table 5 - Error Classification
[BSW00335]	Status values naming convention	*8.2* Type definitions
[BSW00350]	Development error detection keyword	*10.2* Configurable Parameters
[BSW00408]	Configuration parameter naming convention	*10.2* Configurable Parameters
[BSW00410]	Compiler switches shall have defined values	not applicable (This specification does not define compiler switchers)
[BSW00411]	Get version info keyword	*10.3* Published Parameters
[BSW00346]	Basic set of module files	*5.10* File Structure
[BSW158]	Separation of configuration from implementation	*5.10* File Structure
[BSW00314]	Separation of interrupt frames from service routines	not applicable (EcuM does not specify ISRs.)
[BSW00370]	Separation of callback interface from API	*8* API specification
Standard Header Files		
[BSW00348]	Standard header type	not applicable (EcuM does not define standard types)
[BSW00353]	Platform specific type header	not applicable (EcuM is specified platform independent)
[BSW00361]	Compiler specific language extension header	not applicable (EcuM does not define language extensions)
[BSW00301]	Limited import information	*8.1* Imported Types
[BSW00302]	Limited export information	*8* API specification
[BSW00328]	Avoid duplication of code	Not applicable (Requirement to implementation)
[BSW00312]	Shared code shall be re-entrant	*8* API specification
[BSW006]	Platform independency	*8* API specification
[BSW00357]	Standard API return type	*8* API specification
[BSW00377]	Module specific API return types	*8* API specification
[BSW00304]	AUTOSAR integer data types	*8* API specification
[BSW00355]	Do not redefine AUTOSAR integer data types	*8* API specification
[BSW00378]	AUTOSAR boolean type	*8* API specification
[BSW00306]	Avoid direct use of compiler and platform specific keywords	*8* API specification
[BSW00308]	Defintion of global data	Not applicable (EcuM does not specify global data.)
[BSW00309]	Global data with read-only constraints	
[BSW00371]	Do not pass function pointers via API	*8* API specification

[BSW00358]	Return type of init() functions	EcuM2811
[BSW00414]	Parameter of init function	
[BSW00376]	Return type and parameters of main processing functions	*8.4.1* EcuM_MainFunction
[BSW00359]	Return type of callback functions	*8.5* Callback Definitions
[BSW00360]	Parameters of callback functions	*8.5* Callback Definitions
[BSW00329]	Avoidance of generic interfaces	*8* API specification
[BSW00330]	Usage of macros/inline functions instead of functions	not applicable (Requirement to implementation)
[BSW00331]	Separation of error and status values	*8.2* Type definitions
[BSW009]	Module user documentation	Fulfilled by usage of template/formal review
[BSW00401]	Documentation of multiple instances of configuration parameters	*10.2* Configurable Parameters
[BSW172]	Compatibility and documentation of scheduling strategy	EcuM2836
[BSW010]	Memory resource documentation	not applicable (requirement to implementation)
[BSW00333]	Documentation of callback function context	*8.5* Callback Definitions
[BSW00374]	Module vendor identification	*10.3* Published Parameters
[BSW00379]	Module identification	*10.3* Published Parameters
[BSW003]	Version identification	*10.3* Published Parameters
[BSW00318]	Format of module version numbers	*10.3* Published Parameters
[BSW00321]	Enumeration of module version numbers	*10.3* Published Parameters
[BSW00341]	Microcontroller compatibility documentation	not applicable (requirement to implementation)
[BSW00334]	Provision of XML file	not applicable (provided by system team)

Document: Requirements on Mode Management [4]

Requirement		***Satisfied by***
[BSW09120]	Configuration of initialization process of basic software	EcuM2559, EcuM2520, EcuM2521, *8.6.2* Callouts from STARTUP
[BSW09147]	Configuration of de-initialization process of basic software	
[BSW09122]	Configuration of users of the ECU State Manager module	EcuM487, *10.2* Configurable Parameters
[BSW09100]	Selection of wake up sources shall be configurable	EcuM2389, *10.2* Configurable Parameters
[BSW09146]	Configuration of time triggered increased inoperation	EcuM2654, EcuM2223, *10.2* Configurable Parameters
[BSW09001]	Standardization of state relations	EcuM2664
[BSW09116]	Requesting and releasing the RUN state	EcuM2814, EcuM2815
[BSW09114]	Starting/invoking the shutdown process	EcuM2311
[BSW09104]	ECU State Manager module shall take over control after OS shutdown	EcuM2328

[BSW09113]	Initialization of Basic Software modules	Table 1 - Initialization Activities
[BSW09127]	De-initialization of BSW	Table 3 - Shutdown Activities
[BSW09128]	Support of several shutdown targets	*7.6.2.1* Shutdown Targets
[BSW09119]	Support of several sleep modes	EcuM2363
[BSW09102]	API for selecting the sleep mode	EcuM2822
[BSW09072]	Force ECU shutdown	EcuM2821
[BSW09009]	Activation of software when entering/leaving ECU states	*8.6* Callout Definitions
[BSW09017]	Provide ECU state information	*8.3.3.1* EcuM_GetState
[BSW09138]	Selection of application modes of OS	EcuM2141, *7.11.1* Application Modes
[BSW09136]	Centralized wake up Management	*7.8* Wake up Validation Protocol
[BSW09098]	Registration of wake up reasons	*8.3.4* Wake up
[BSW09097]	Validation of physical channel wake up	*7.8* Wake up Validation Protocol
[BSW09118]	Time Triggered Increased Inoperation	*7.9* Time Triggered Increased Inoperation
[BSW09145]	Support of wake-sleep operation	*7.11.5* Configuration Alternative for Providing Wake-Sleep Operation
[BSW09126]	Provide an API for querying of wake up reason	*8.3.4* Wake up
[BSW09145]	Evaluate condition to stay in the RUN state	EcuM2311
[BSW09164]	Shutdown synchronization support for SW-Components	*7.4.3.4* RUN
[BSW09165]	Requesting and releasing the POST_RUN state	EcuM2819, EcuM2820
[BSW09166]	Evaludate condition to stay in POST_RUN state	EcuM2761
[BSW09170]	Triggering Watchdog Manager during Startup / Shutdown and Sleep	EcuM2861
[BSW09173]	Minimum duration of Run State	EcuM2310

7 Functional Specification

7.1 Main States of the ECU State Manager module

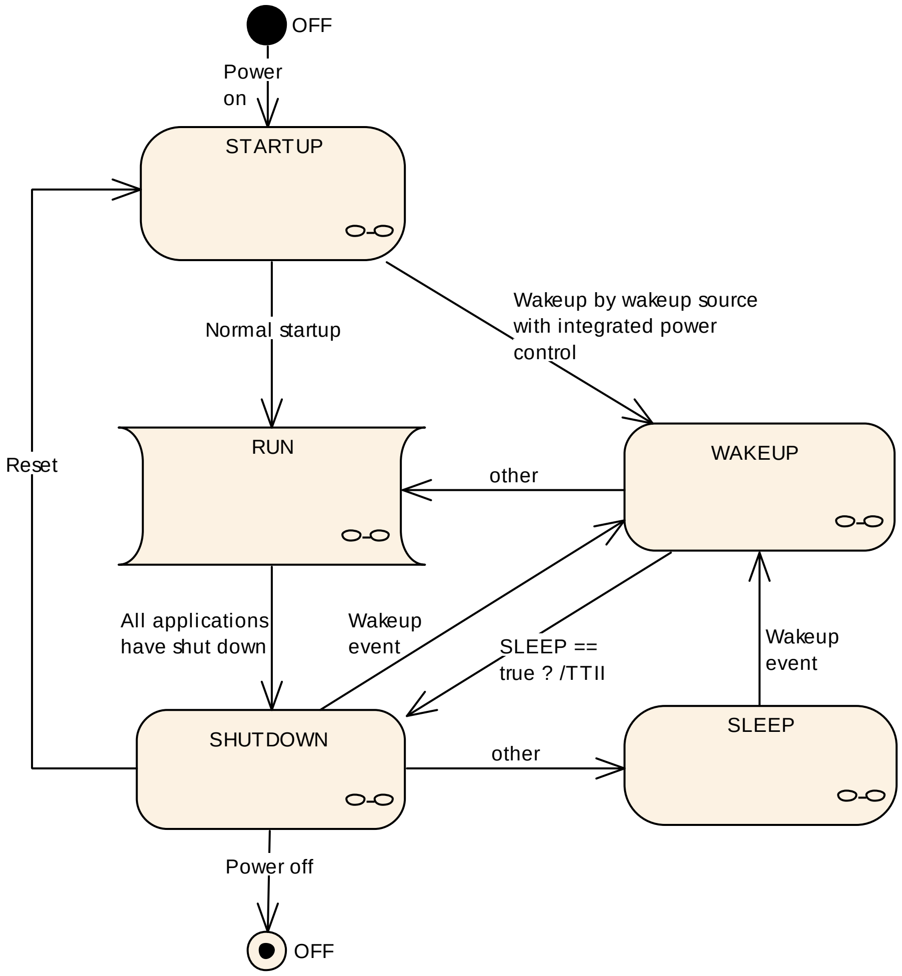

Figure 1 – ECU Main States (top level diagram)

Figure 1 shows the main state machine provided by the ECU State Manager module. This state machine manages the 'life cycle' of an ECU from OFF through STARTUP and RUN to SLEEP or OFF.

7.1.1 STARTUP State

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

7.1.2 RUN State

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

7.1.3 SHUTDOWN State

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

7.1.4 SLEEP State

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

[6] RUN state is requested implicitly if a non-idle state is requested from a Resource Manager. E.g. requesting any state but 'no communication' from the Communication Manager will have the Communication Manager requesting RUN state from the ECU State Manager in turn. This is a request for communication which implicitly results in a request for RUN state. See also [5].
[7] In this specification RUN and POST_RUN sub-states are called RUN II and RUN III.
[8] Some ECU designs actually do require code execution to implement a SLEEP state (and the wakeup capability). For these ECUs, the clock speed is typically dramatically reduced. These could be implemented with a small loop inside the SLEEP state.

7.1.5 WAKEUP State

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

7.1.6 OFF State

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

7.2 Structural Description of the ECU State Manager module

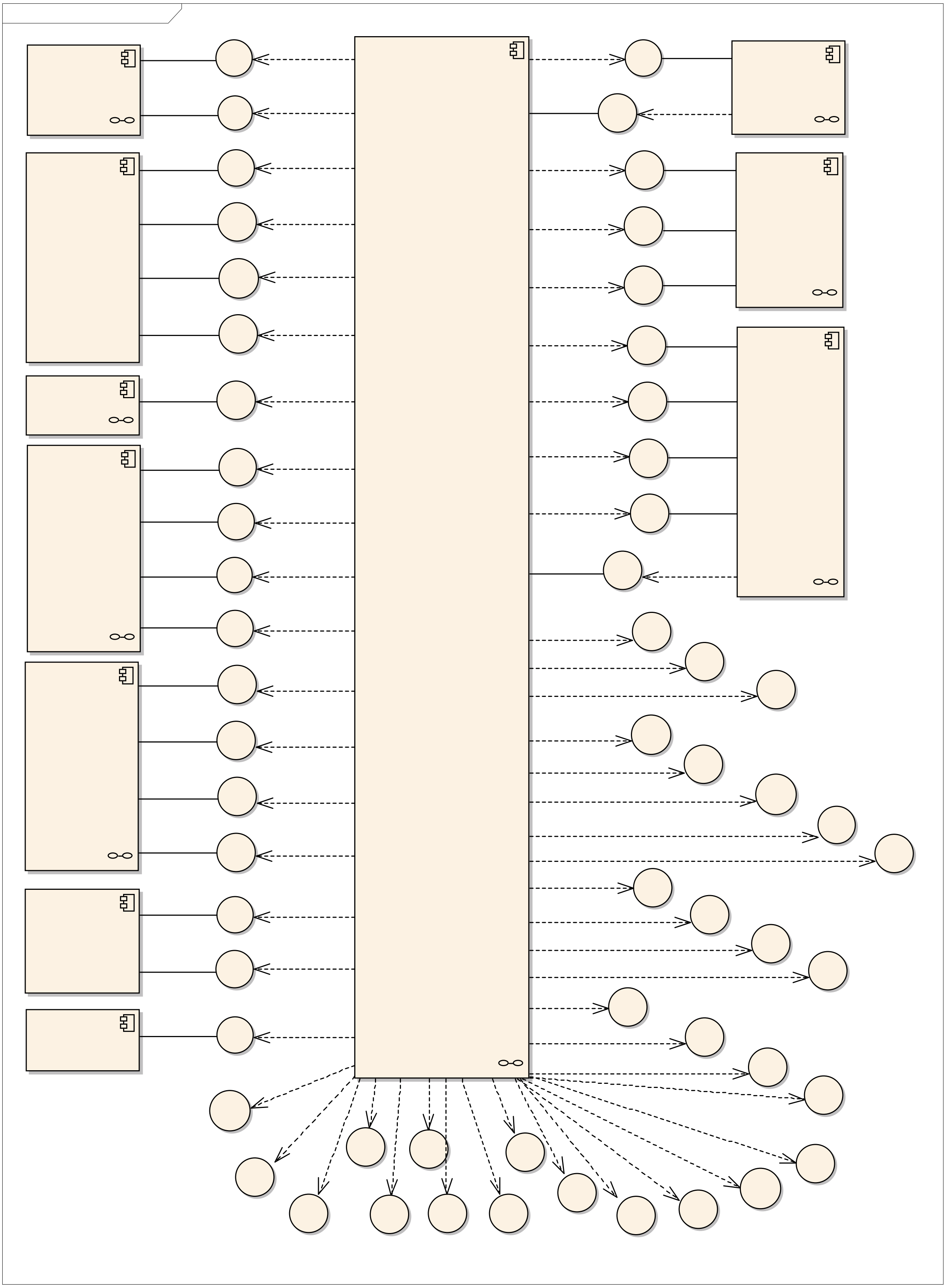

Figure 2 – Module Relationship (top level diagram)

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

7.2.1 Standardized AUTOSAR Software Modules

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

7.2.2 Software Components

SW Components contain the application code of an AUTOSAR ECU. Software components shall request the RUN state from the ECU State Manager module when they have the need to keep the ECU alive.
If the intent of the SW-C is primarily to communicate then it should request a communication state from the Communication Manager (see [5]). This will implicitly keep the ECU alive. A SW-C should clearly separate between the need to communicate and the need to keep an ECU alive. Mixing up these two ideas may result in an instable shutdown algorithm.
A SW-C interacts with the ECU State Manager module using AUTOSAR ports.

7.2.3 Resource Managers

The concept of resource managers allows adding new state machines to the BSW (as a part of new BSW modules) which behave like sub-state machines of the RUN state.
In order to collaborate correctly with the ECU State Manager module only very few requirements must be met:

EcuM2153: A Resource Manager has to define exactly one idle state which signifies the state where the Resource Manager isn't doing anything but waiting.

EcuM2154: A Resource Manager shall transit into its idle state after initialization. It shall request the RUN state from the ECU State Manager module whenever it leaves its idle state and it shall release the RUN state when it returns back to its idle state.

The Communication Manager is one such resource manager.

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

7.3 STARTUP State

See *7.1.1* STARTUP State for an overview description.

7.3.1 High Level Sequence Diagram

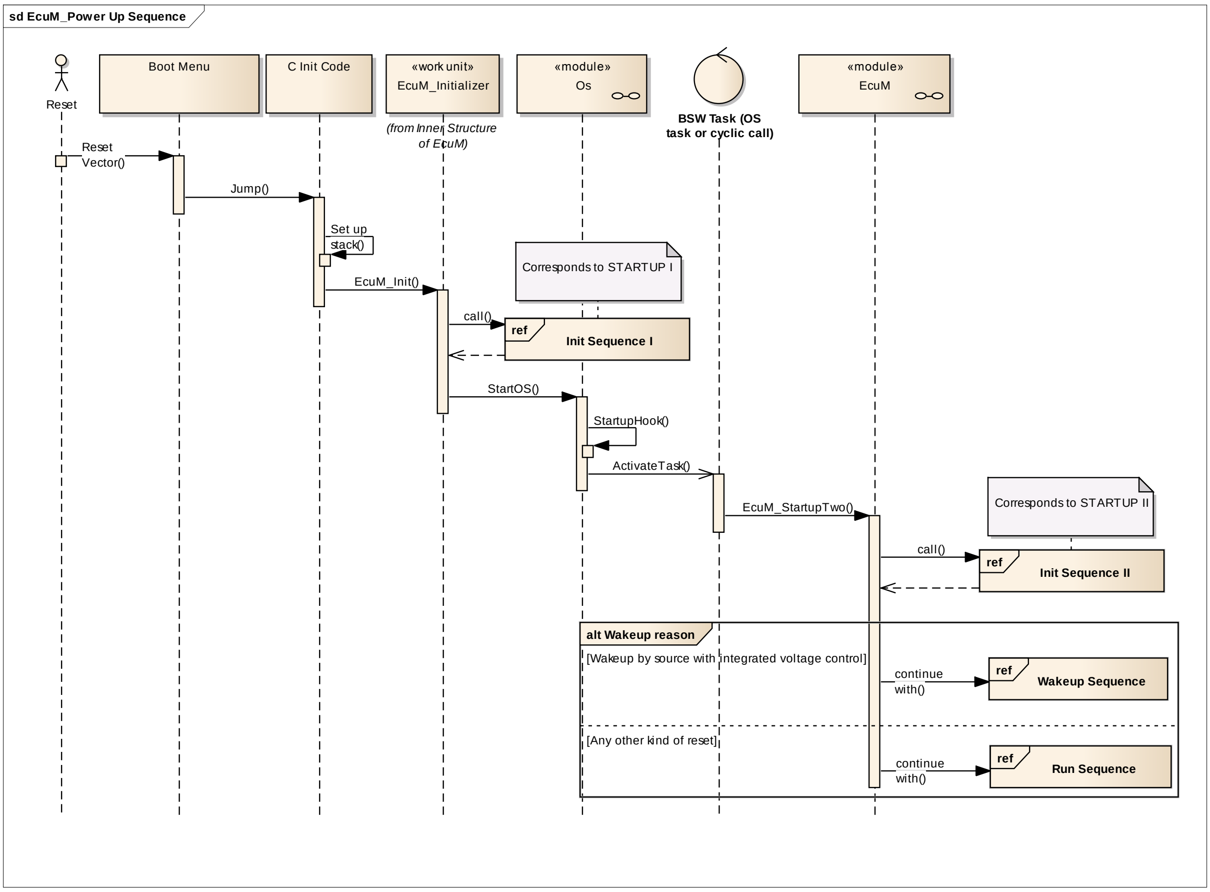

Figure 3 – Startup Sequence (high level diagram)

To see adjacent diagrams refer to
Figure 7 – RUN State Sequence
Figure 1 – ECU Main States (top level diagram)

The startup sequence shows the startup behavior of the ECU. With the invocation of EcuM_Init the ECU State Manager module takes control of the startup procedure. The startup falls into two parts. The first part, init sequence I or STARTUP I is finished when the AUTOSAR OS is started. The second part, init sequence II or STARTUP II is finished when RTE is started.

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

7.3.2 Activities before EcuM_Init

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

7.3.3 STARTUP Activity Overview

EcuM2411: The following table shows the activities and the order in which they shall be executed.

Sub-state ***Initialization Activity***[10]	***Comment***	***Opt.***[11]
STARTUP I		
Callout EcuM_AL_DriverInitZero	Init block 0 This callout may only initialize BSW modules that do not use post-build configuration parameters. The callout may not only contain driver initialization but any kind of pre-OS, low level initialization code. See *7.3.5* Driver Initialization	yes
Callout EcuM_DeterminePbConfiguration	This callout is expected to return a pointer to a fully initialized EcuM_ConfigType structure containing the post-build configuration data for EcuM and all other BSW modules.	no
Check consistency of configuration data	If check fails the EcuM_ErrorHook is called. See *10.4* Checking Configuration Consistency for details on the consistency check.	no
Callout EcuM_AL_DriverInitOne	Init block I The callout may not only contain driver initialization but any kind of pre-OS, low level initialization code. See *7.3.5* Driver Initialization	yes
Get reset reason	The reset reason is derived from a call to `Mcu_GetResetReason` and the mapping defined via the `EcuMWakeupSource` configuration containers. **EcuM2623**: The wake up source resulting from the reset reason translation shall be remembered by the ECU State Manager module. See *8.5.2.2* EcuM_SetWakeupEvent and *8.3.4.3* EcuM_GetValidatedWakeupEvents.	no
Select default shutdown target	See EcuM2181	no
Select application mode	See EcuM2242	no
Start OS	Start the AUTOSAR OS, see EcuM2603 and EcuM2141	no
STARTUP II		
Init BSW Scheduler	Initialize the semaphores for critical sections used by BSW modules	no

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

Sub-state ***Initialization Activity***[10]	***Comment***	***Opt.***[11]
Callout EcuM_AL_DriverInitTwo	Init block II The callout may only initialize BSW modules that need OS support but don't need access to private NvRam data (other that post-build configuration data in their <Module>_ConfigType) or manage that data on their own. See *7.3.5* Driver Initialization	yes
Callout EcuM_OnRTEStartup		no
Start RTE	From now on SW-Cs are running. RTE will signal the (initial) mode STARTUP during start.	no
Callout EcuM_AL_DriverInitThree	Init block III The callout may initialize BSW modules that need OS support and rely on their private NvRam data (other that post-build configuration data in their <Module>`_ConfigType`) to be restored. See *7.3.5* Driver Initialization	yes
Indicate mode change to RTE	Indicated mode is `SLEEP` if next state is WAKEUP VALIDATION, indicated mode is `RUN` if next state is RUN.	no

Table 1 - Initialization Activities

7.3.4 Sub-State Descriptions

7.3.4.1 STARTUP I

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

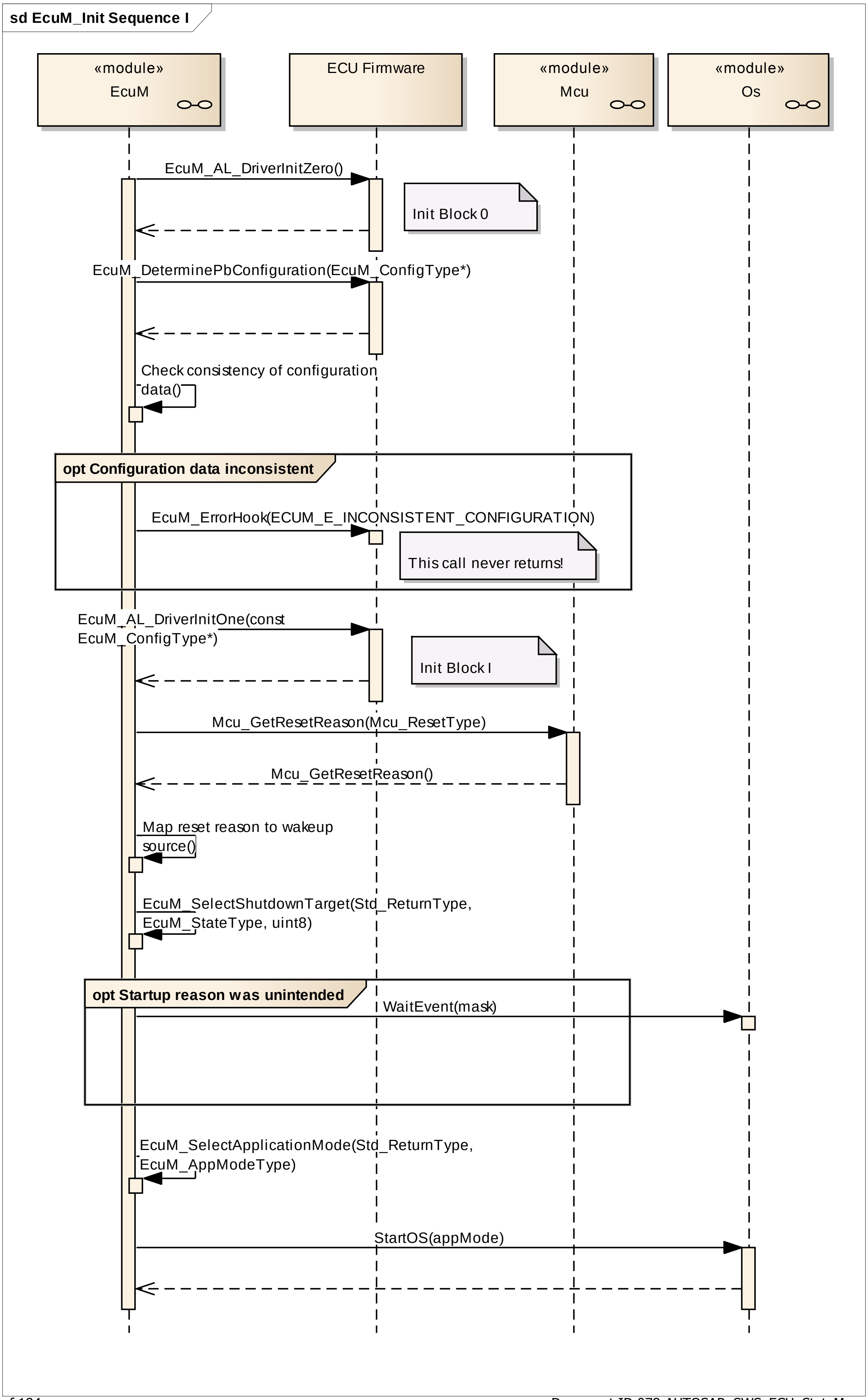
sd EcuM_Init Sequence I
«module»
EcuM
ECU Firmware
«module»
Mcu
«module»
Os
EcuM_AL_DriverInitZero()
Init Block 0
EcuM_DeterminePbConfiguration(EcuM_ConfigType*)
Check consistency of configuration data()
opt Configuration data inconsistent
EcuM_ErrorHook(ECUM_E_INCONSISTENT_CONFIGURATION)
This call never returns!
EcuM_AL_DriverInitOne(const EcuM_ConfigType*)
Init Block I
Mcu_GetResetReason(Mcu_ResetType)
Mcu_GetResetReason()
Map reset reason to wakeup source()
EcuM_SelectShutdownTarget(Std_ReturnType, EcuM_StateType, uint8)
opt Startup reason was unintended
WaitEvent(mask)
EcuM_SelectApplicationMode(Std_ReturnType, EcuM_AppModeType)
StartOS(appMode)

Figure 4 – Init Sequence I (STARTUP I)

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

EcuM2271: `MCU_Init` does not provide complete MCU initialization. Additionally, hardware dependent steps have to be executed and must be defined at system design time. These steps are supposed to be taken within the EcuM_AL_DriverInitZero or EcuM_AL_DriverInitOne callouts. Details can be found in [12].

EcuM2181: The ECU State Manager module must call EcuM_SelectShutdownTarget with the configured default shutdown target (see *7.6.2.1* Shutdown Targets, *7.9* Time Triggered Increased Inoperation and *10.2* Configurable Parameters).

EcuM2242: If the restart was unintended the ECU State Manager module must select the default application mode with a call to EcuM_SelectApplicationMode. Examples for unintended restarts are power hazards and restarts due to fault conditions like watchdog resets. In all other cases, the application mode shall not be changed. See *7.11.1* Application Modes for details how to change the application mode.

EcuM2603: At the end of the STARTUP I state, it must be possible to start the OS. All basic software modules which are needed by the OS must be initialized by this time. Modules left out so far may be initialized later in STARTUP II.

EcuM2141: The application mode parameter of the `StartOS` service shall be retrieved with the API call EcuM_GetApplicationMode. For more details about application modes see also *7.11.1* Application Modes.

EcuM2861: If a Watchdog Manager is configured and initialized in any of the Init Blocks, the ECU State Manager module shall insert calls to `WdgM_UpdateAliveCounter` from that point on after every operation it has executed. The configuration parameter `EcuMWdgMSupervisedEntityRef` defines the supervised entity identifier used as a parameter to that call.

---

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

The above requirement will allow supervising the ECU State Manager module and triggering watchdogs during STARTUP, RUN, SHUTDOWN and WAKEUP. For the handling of the functionalities during SLEEP see [7] Chapter 7.4.

7.3.4.2 STARTUP II

STARTUP II is carried out by the EcuM_StartupTwo API function.

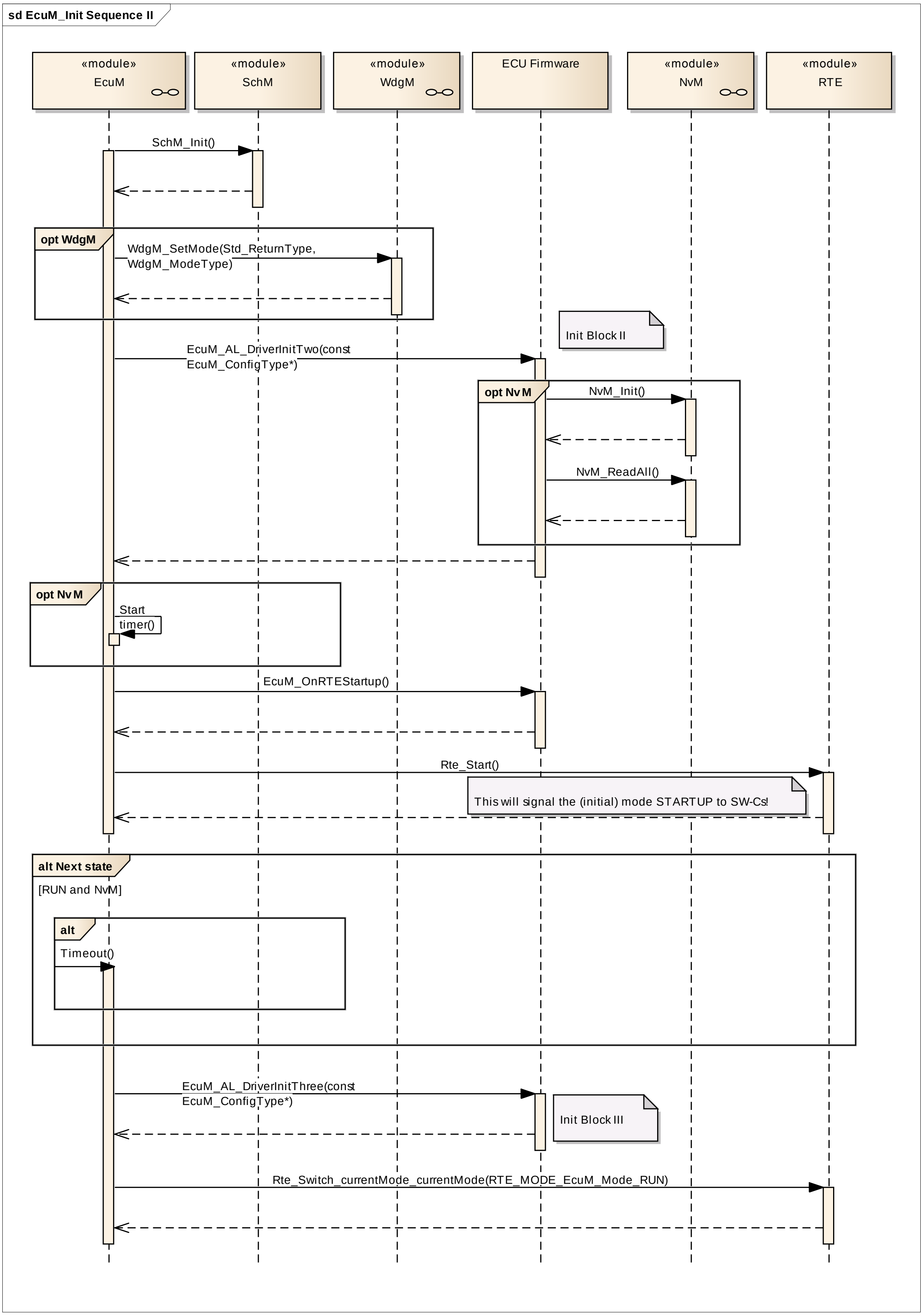

Figure 5 – Init Sequence II (STARTUP II)

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

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

EcuM2632: If one of the wake up sources listed in *7.8.7* Wake up Sources and Reset Reason is set, then exection shall continue with RUN state. In all other cases, execution shall continue with WAKEUP VALIDATION state.

7.3.5 Driver Initialization

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

EcuM2559: The order inside of the blocks shall be generated from configuration information (see *10.2* Configurable Parameters *EcuMDriverInitListZero*, *EcuMDriverInitListOne*, *EcuMDriverInitListTwo*, *EcuMDriverInitListThree*, and *EcuMDriverRestartList*).

EcuM2730: For each driver, its init function with the configured init configuration shall be called. The init parameter for the init function shall be derived from driver's configuration (see *10.2* Configurable Parameters *EcuMModuleConfigurationRef*).

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

EcuM2561: The restart list will typically only contain a subset of drivers. But drivers shall appear in the same order as in the combined list of init block I and init block II (see *10.2* Configurable Parameters, *EcuMDriverRestartList*).

EcuM2562: Drivers which serve as wake up sources may need to be re-initialized in the restart block. The driver restart shall re-arm the trigger mechanism of the 'wake up detected' callback (see *7.7.4.1* WAKEUP I).

EcuM2563: If hardware is put into a sleep mode during SHUTDOWN then this hardware must be restarted by its driver.
The restart list will be invoked in state WAKEUP I (see *7.1.5* WAKEUP State).

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

Recommended Init Block / ***Init Activity***	***Comment***
Init Block 0[13]	
Development Error Tracer	This always needs to be the first module to be initialized, so that other modules can report development errors.
Any drivers needed to access post-build configuration data	These drivers may themselves not need post-build configuration or OS features.
Init Block I[14]	
MCU Driver	
PORT	
DIO	
Diagnostic Event Manager	Pre-Initialization
General Purpose Timer	
Watchdog Driver	Internal watchdogs only, external ones may need SPI
Watchdog Manager	
SchM	
BswM	
ADC Driver	
ICU Driver	
PWM Driver	
Init Block II[15]	
SPI Driver	
EEPROM Driver	
Flash Driver	
NVRAM Manager	Initialization and start NvM_ReadAll job
CAN Transceiver	
CAN Driver	
CAN Interface	
CAN State Manager	
CAN TP	
LIN Driver	
LIN Interface	
LIN State Manager	
LIN TP	
FlexRay Transceiver	
FlexRay Driver	
FlexRay Interface	
FlexRay State Manager	
FlexRay TP	
PDU Router	
CAN NM	
FlexRay NM	
NM Interface	
I-PDU Multiplexer	
COM	
Diagnostic Communication Manager	
Init Block III[16]	
Communication Manager	
Diagnostic Event Manager	Full initialization

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

Recommended Init Block	
Init Activity	***Comment***
Function Inhibition Manager	

Table 2 - Driver Initialization Details, Sample Configuration

7.3.6 DET Initialization

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

EcuM2783: DET shall be initialized early during STARTUP I by the ECU State Manager module.

EcuM2634: DET is not *started* by default but the system designer has to configure the point where DET is started, preferably into one of the callouts EcuM_AL_DriverInitOne or EcuM_AL_DriverInitTwo. The best point for starting DET depends on its implementation and behavior. DET is started by invoking `Det_Start`.

7.4 RUN State

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

7.4.1 State Breakdown Structure

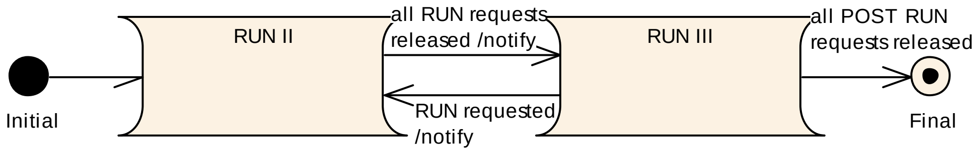

Figure 6 – RUN State Breakdown

7.4.2 High Level Sequence Diagram

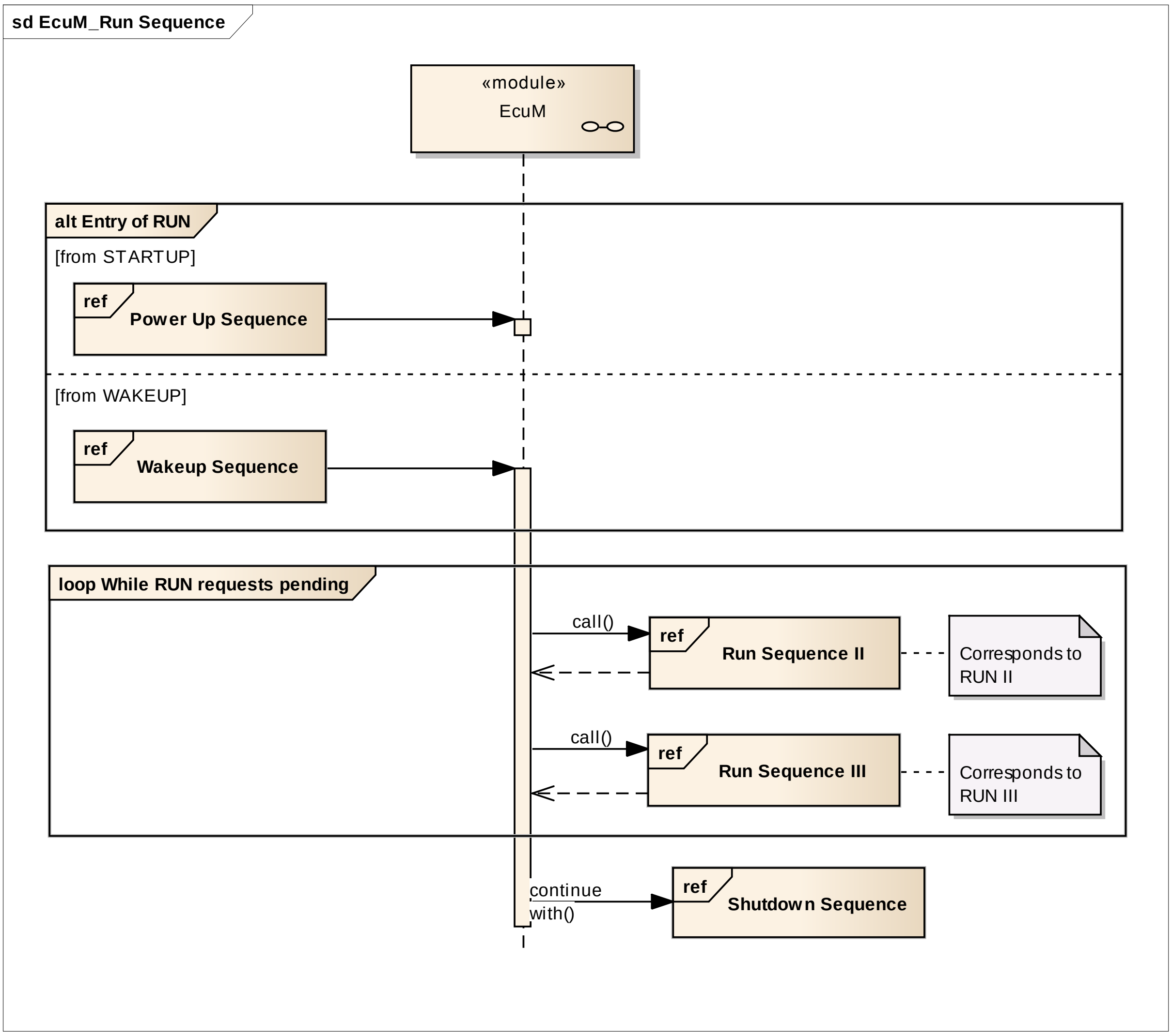

Figure 7 – RUN State Sequence (high level diagram)

To see adjacent diagrams refer to

- Figure 3 – Startup Sequence (high level diagram)
- Figure 19 – Wake up Sequence (high level diagram)
- Figure 11 – Shutdown Sequence (high level diagram)
- Figure 1 – ECU Main States (top level diagram)

7.4.3 Sub-State Description

7.4.3.1 RUN II

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

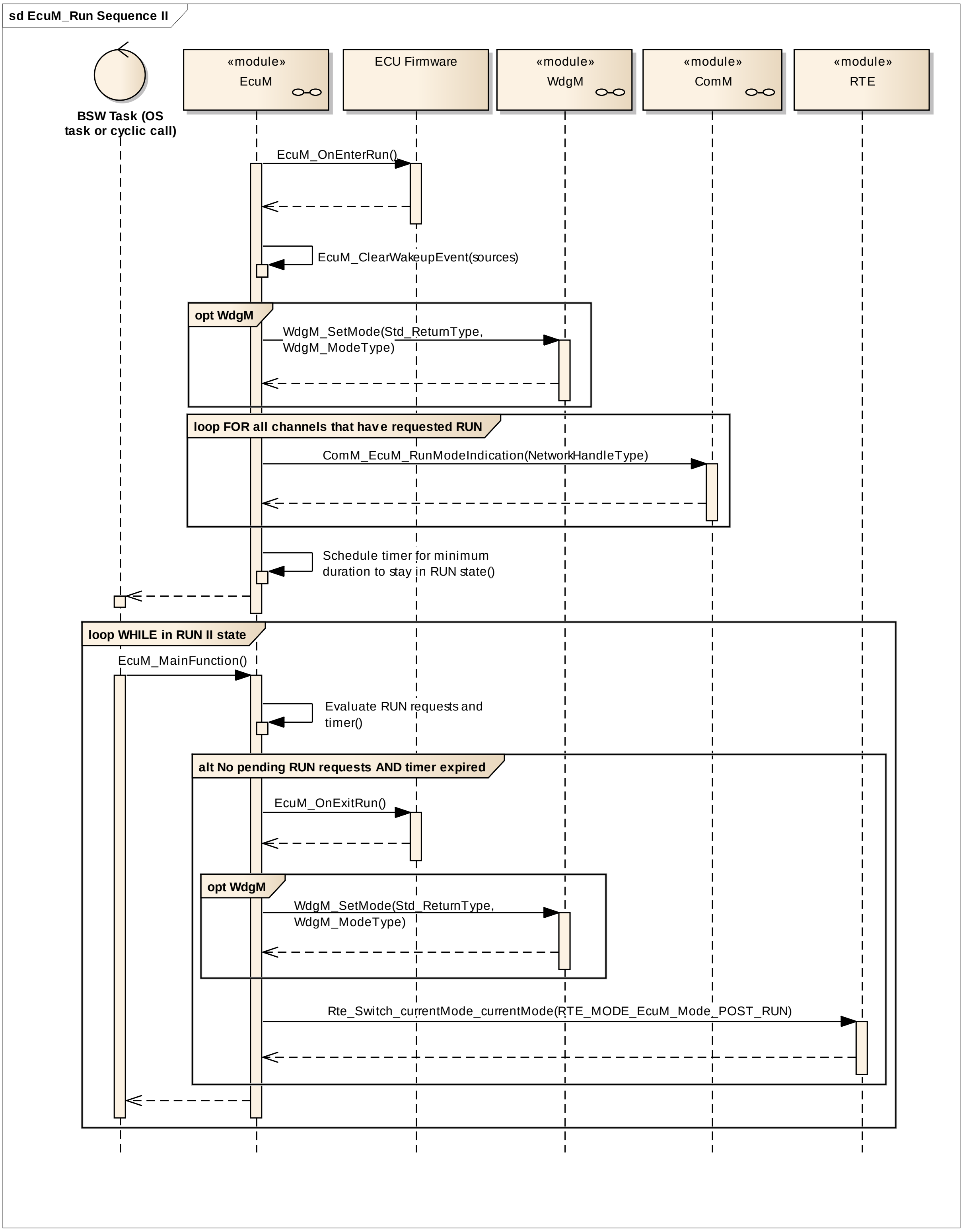

Figure 8 – RUN II State Sequence

7.4.3.2 Entering RUN II State

On entering RUN II state, the following steps must be done in the presented order:

EcuM2308: When entering RUN II state, the callout EcuM_OnEnterRun shall be invoked and RUN mode shall be indicated.

EcuM2384: RUN shall be indicated to the Communication Manager by invoking `ComM_EcuM_RunModeIndication`, see [5]. EcuM shall indicate RUN mode only to those channels for which ComM requested a RUN (via EcuM_ComM_RequestRUN(channel)).

EcuM2310: The ECU State Manager module shall remain in RUN state for a configurable minimum duration (see *10.2* Configurable Parameters parameter EcuMRunMinimumDuration).

The minimum duration of RUN state is needed to give the SW-Cs a chance to request RUN. Otherwise EcuM will immediately leave RUN again.

7.4.3.3 Leaving RUN II State

EcuM2311: When the last RUN request has been released, ECU State Manager module shall advance to the RUN III state. The evaluation is done with the next cyclic invocation of EcuM_MainFunction.

EcuM2865: When leaving RUN II state, the callout EcuM_OnExitRun shall be invoked and POST_RUN mode shall be indicated.

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

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

7.4.3.4 RUN III

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

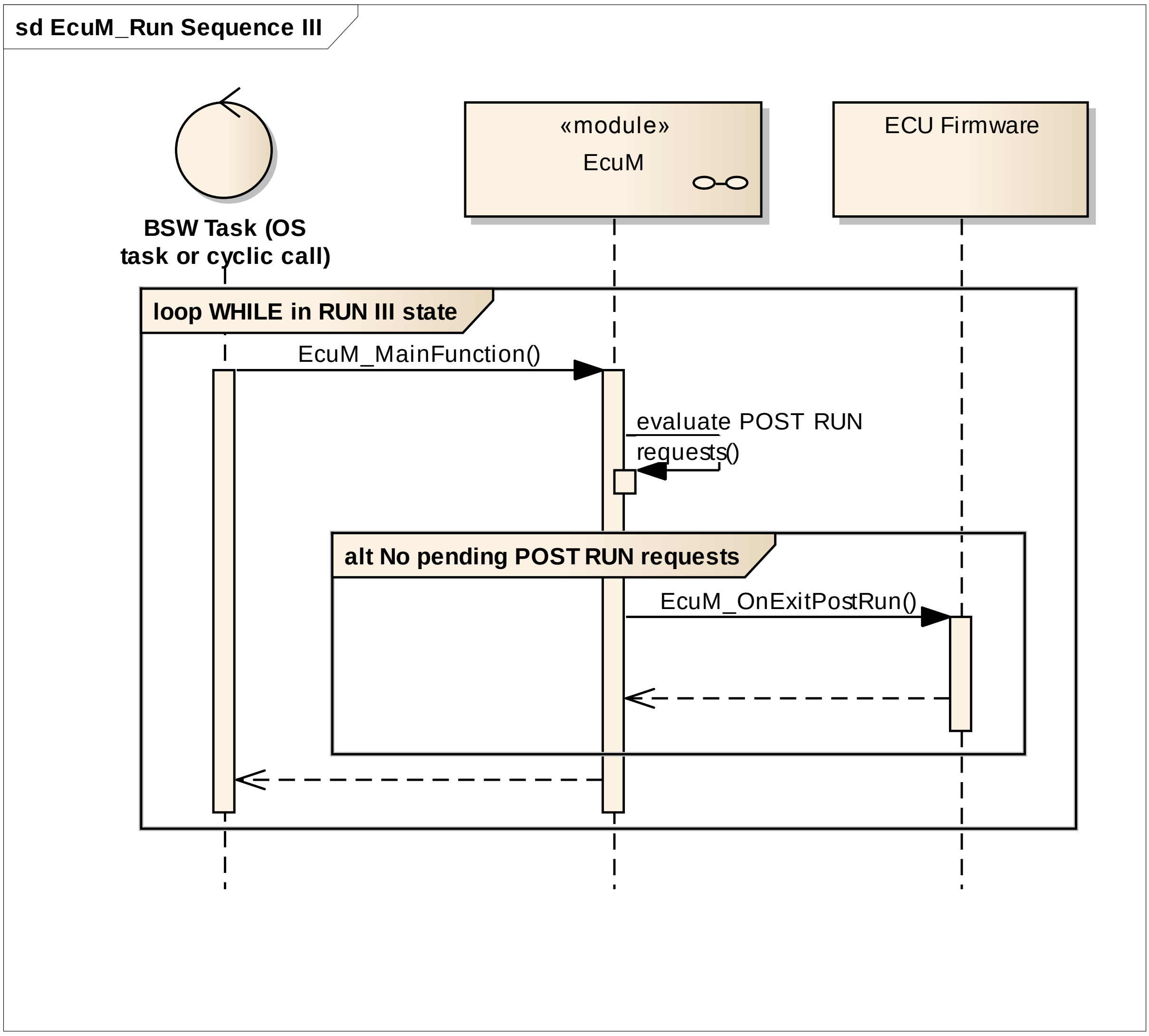

Figure 9 – RUN III State Sequence

7.4.3.5 Leaving RUN III State

EcuM2761: When the last POST_RUN request has been released and no RUN request has been issued, the ECU State Manager module shall advance to the SHUTDOWN state and shall invoke the callout EcuM_OnExitPostRun. The evaluation is done with the next cyclic invocation of EcuM_MainFunction.

EcuM2866: While in RUN III state, if a RUN request is received, the ECU State Manager module shall immediately enter RUN II state again.

7.5 SHUTDOWN State

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

EcuM2188: When SHUTDOWN state is entered and shutdown target is SLEEP, no wake up event shall be missed. If a valid wake up event occurs while the ECU is in transition to SLEEP the ECU shall as quickly as possible proceed to the WAKEUP state and shall not enter the SLEEP state.

EcuM2756: When a wake up event occurs during the shutdown phase and the shutdown target is OFF or RESET, then the shutdown shall complete but the ECU shall restart immediately thereafter.

7.5.1 State Breakdown Structure

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

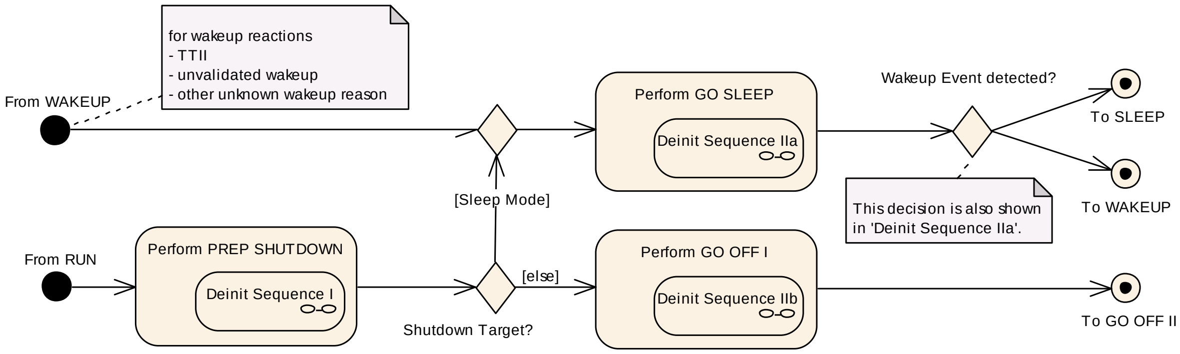

Figure 10 – Fine Structure of SHUTDOWN

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

7.5.2 High Level Sequence Diagram

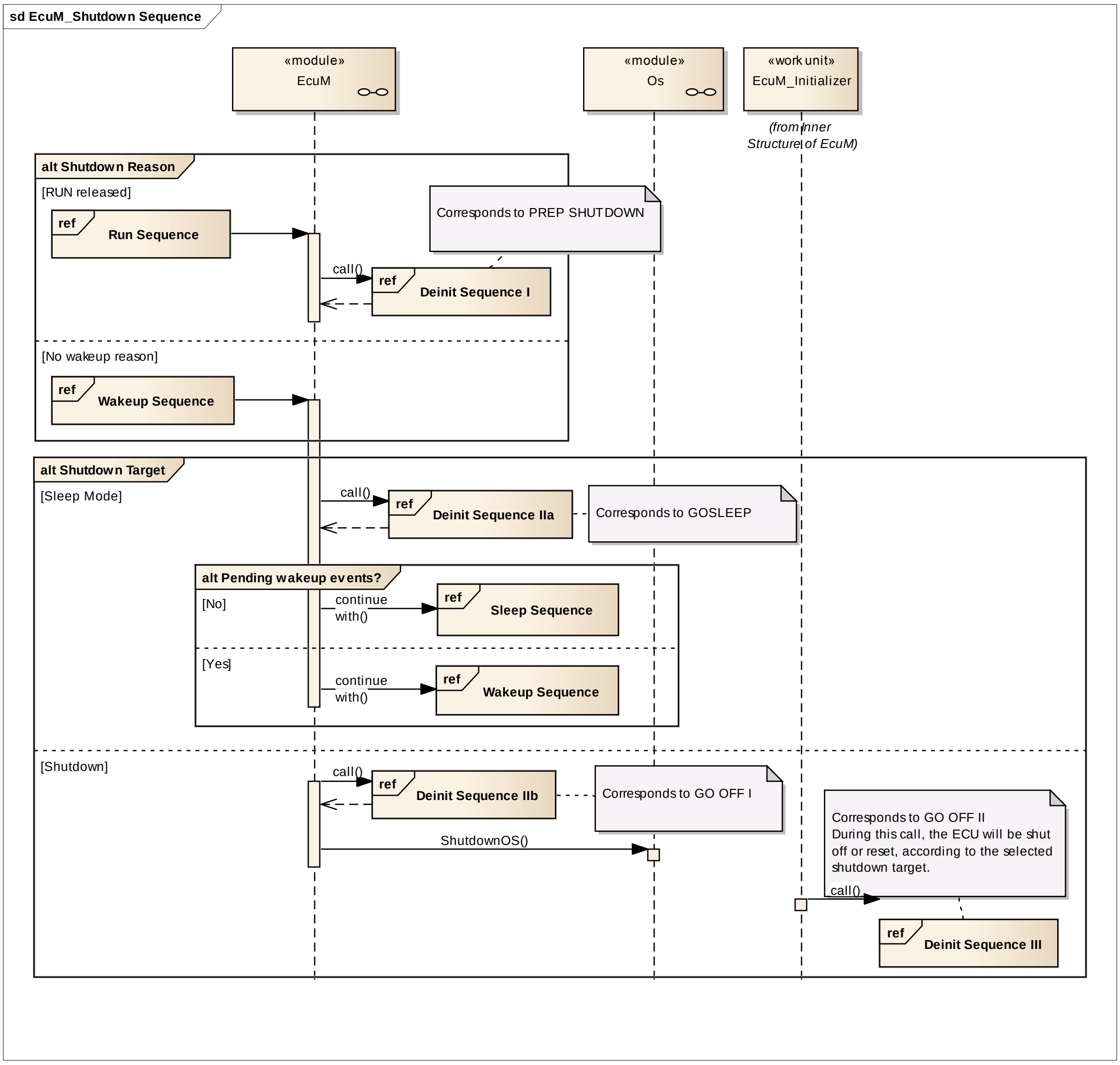

Figure 11 – Shutdown Sequence (high level diagram)

To see adjacent diagrams refer to

Figure 7 – RUN State Sequence (high level diagram)
Figure 19 – Wake up Sequence (high level diagram)
Figure 16 – Sleep Sequence (high level diagram)
Figure 1 – ECU Main States (top level diagram)

7.5.3 SHUTDOWN Activity Overview

Sub-state[18] Shutdown Activity	Comment	Optional[19]
PREP SHUTDOWN		
Callout EcuM_OnPrepShutdown		
Shutdown Diagnostic Event Manager		yes
Indicate mode change to RTE	Indicated mode is SLEEP if next state is GO SLEEP, indicated mode is SHUTDOWN if next state is GO OFF I.	
GO SLEEP		
Callout EcuM_OnGoSleep		
Enable all interrupts		
Save persistent data to NVRAM	An incoming wake up event will cancel an ongoing write job	yes
Check for pending wake up events	Purpose is to detect wake up events that occurred while interrupts were disabled	
Callout EcuM_EnableWakeupSources	See *8.6.4.5* EcuM_EnableWakeupSources	
Set Watchdog Manager mode for sleep		yes
Lock Scheduler	Prevent other tasks from running in SLEEP state.	
GO OFF I		
Callout EcuM_OnGoOffOne		
Stop RTE		
Deinit Communication Manager		yes
Save persistent data to NVRAM		yes
Set Watchdog Manager mode for shutdown		yes
Call BswM_DeInit		
Check for pending wake up events	Purpose is to detect wake up events that occurred during shutdown	
Set RESET as shutdown target	This action shall only be carried out when pending wake up events were detected	yes
ShutdownOS	Last operation in this OS task	
GO OFF II		
Callout EcuM_OnGoOffTwo		
Call Mcu_PerformReset or *Callout* EcuM_AL_SwitchOff	Depends on the selected shutdown target (RESET or OFF)	
The following modules need not to be shut down:		
NVRAM Manager		
All other modules are not shutdown automatically. The following basic software modules must not be shut down at all.		
None		

Table 3 - Shutdown Activities

[18] Rows marked with × are conditional.
[19] Optional activities can be switched on or off by configuration. It shall be the system designers choice if a module is compiled in or not for an ECU design. See chapter *10.2* Configurable Parameters for details.

7.5.4 Sub-State Descriptions

7.5.4.1 PREP SHUTDOWN

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

EcuM2288: If the shutdown target is not any of the sleep modes, then control has to be handed over to GO OFF I (ref. *7.5.4.3* GO OFF I) after activities of this state have finished.

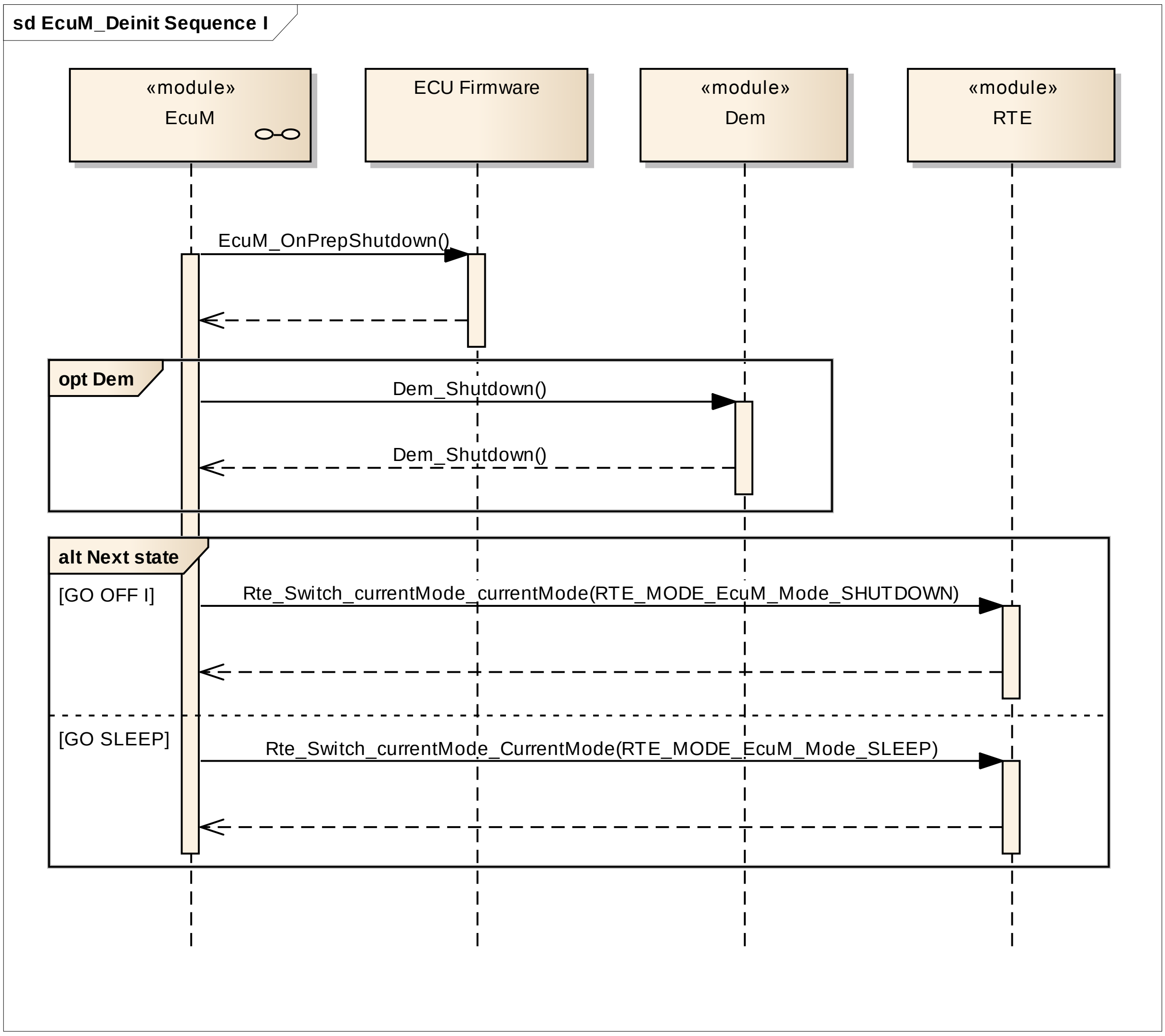

Figure 12 – Deinitialization Sequence I (PREP SHUTDOWN)

7.5.4.2 GO SLEEP

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

EcuM2389: To set up the wake up sources for the next sleep mode, the ECU State Manager module shall execute the callout EcuM_EnableWakeupSources for each wake up source that is configured in the target sleep mode.

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

Note:
In case of pending wake up events, after calling NvM_CancelWriteAll() the transition shall go to WAKEUP VALIDATION as for the "Power On Sequence" (see also Figure 28).

EcuM2863: The ECU Manager module shall invoke the callout `EcuM_GenerateRamHash` (see chapter 8.6.4.6) before halting the microcontroller, and the callout `EcuM_CheckRamHash` (see chapter 8.6.5.1) after the processor returns from halt.

Rationale for EcuM2863: 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.

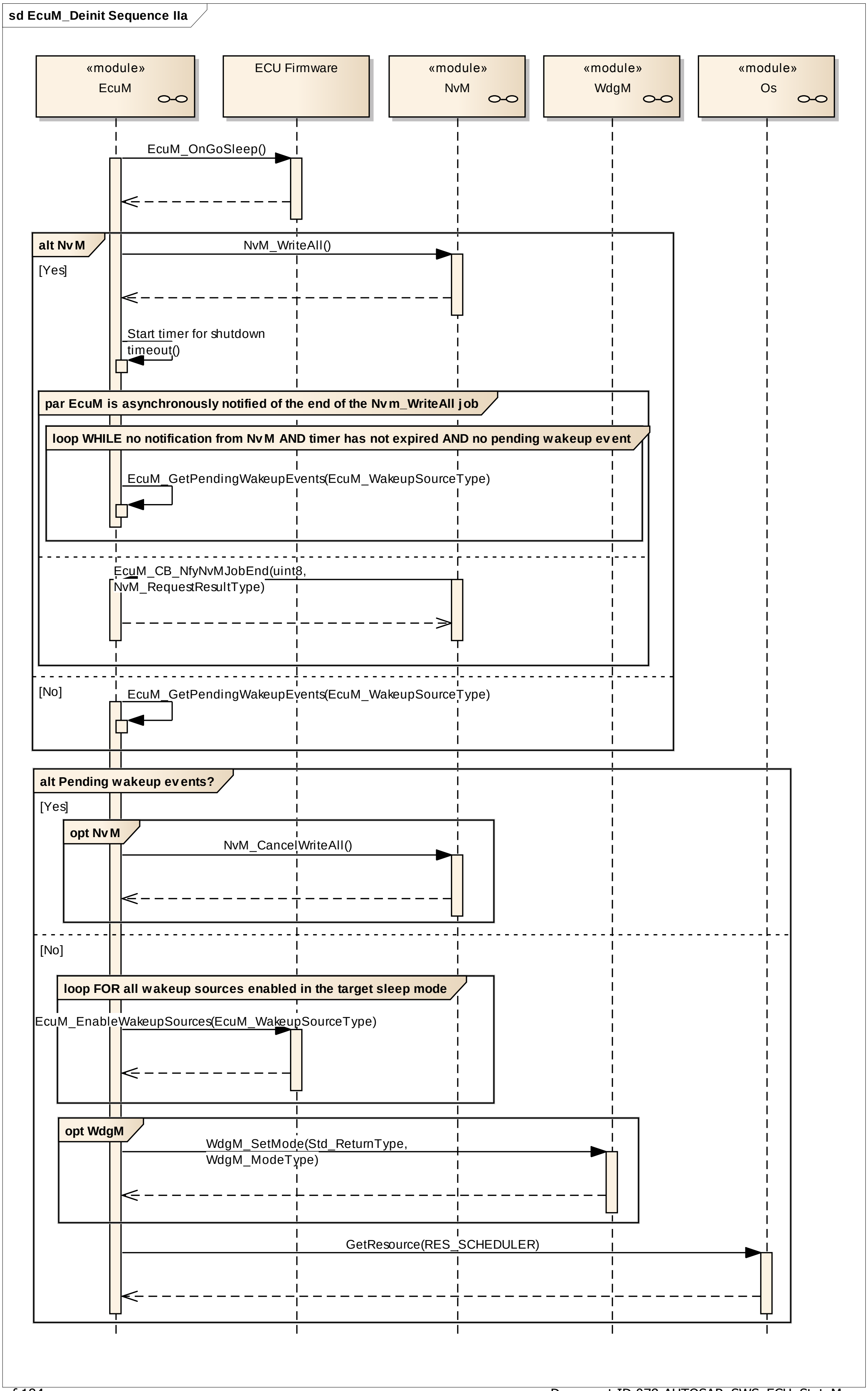
sd EcuM_Deinit Sequence IIa
«module»
EcuM
ECU Firmware
«module»
NvM
«module»
WdgM
«module»
Os
EcuM_OnGoSleep()
alt NvM
[Yes]
NvM_WriteAll()
Start timer for shutdown
timeout()
par EcuM is asynchronously notified of the end of the Nvm_WriteAll job
loop WHILE no notification from NvM AND timer has not expired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

Figure 13 – Deinitialization Sequence IIa (GOSLEEP)

7.5.4.3 GO OFF I

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

EcuM2328: As its last activity, the `ShutdownOS` service shall be called. This service will end up in the shutdown hook. The shutdown hook in turn shall call EcuM_Shutdown to terminate the shutdown process. EcuM_Shutdown will not return but switch off the ECU or issue a reset.

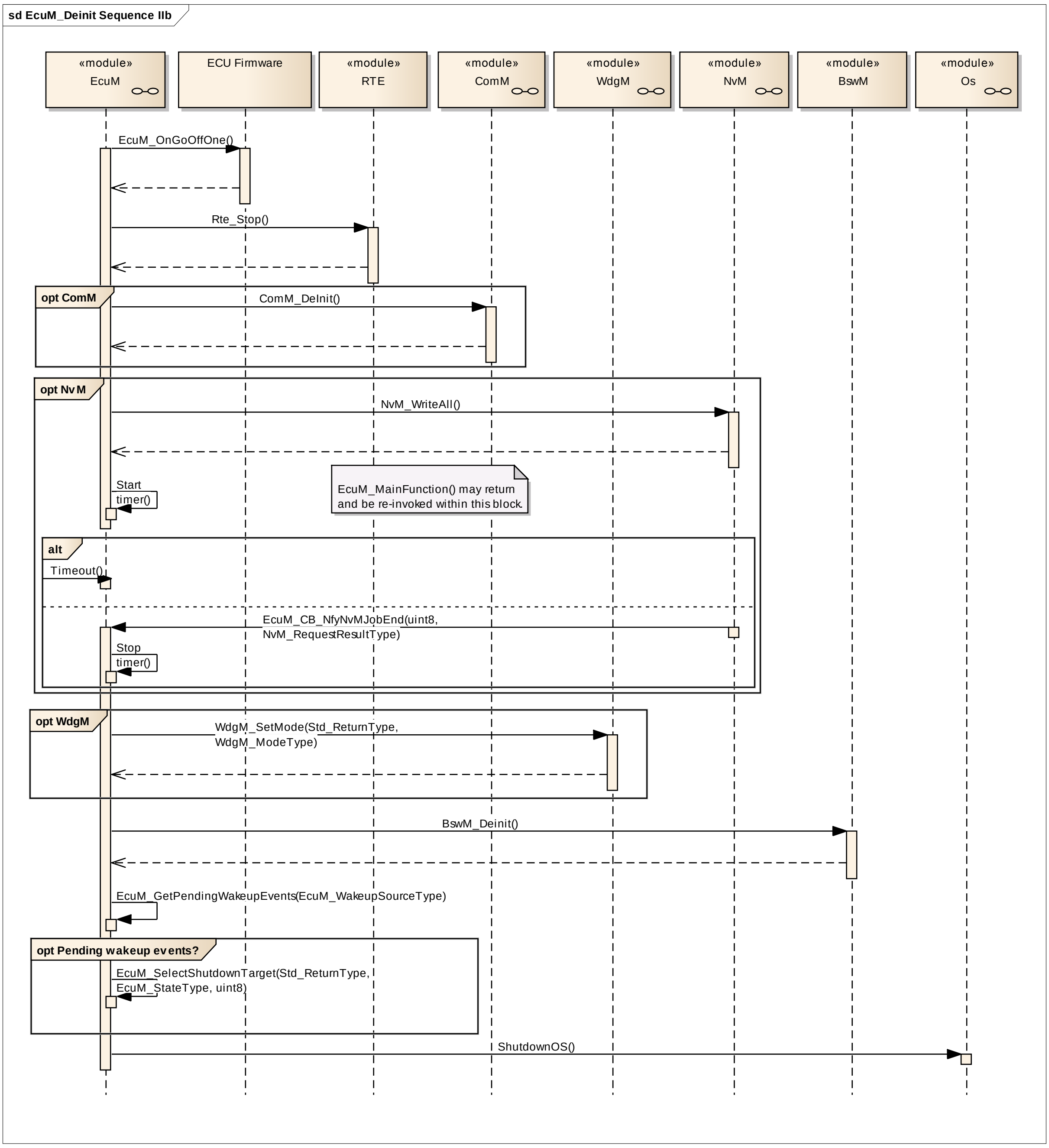

Figure 14 – Deinitialization Sequence IIb (GO OFF I)

7.5.4.4 GO OFF II

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

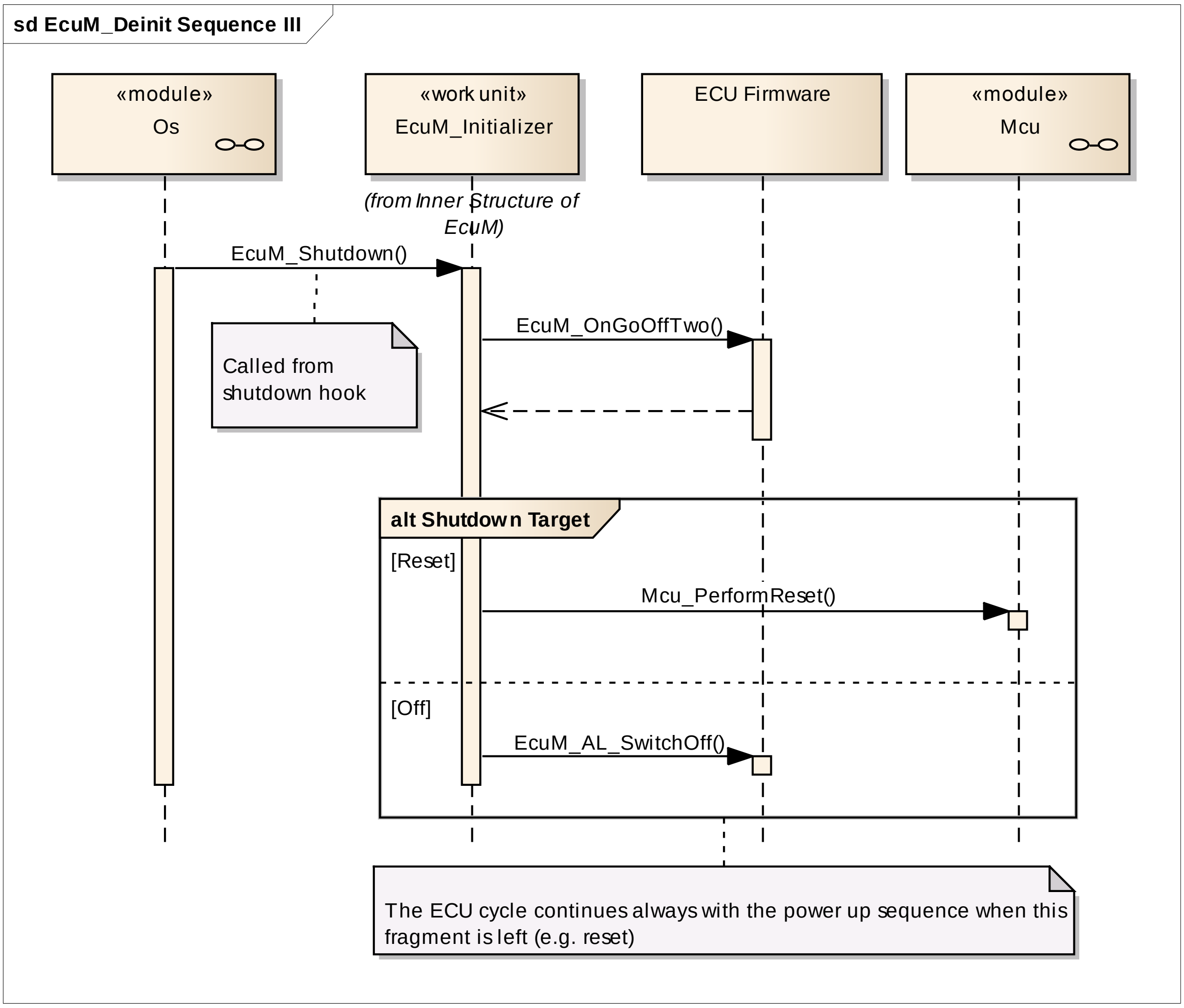

Figure 15 – Deinitialization Sequence III (GO OFF II)

The shutdown target RESET is reached by invoking the `Mcu_PerformReset` service of the MCU driver (see [12]).
The shutdown target OFF is implemented by the EcuM_AL_SwitchOff callout which must be filled at configuration time. See *8.6.4.7* EcuM_AL_SwitchOff for details.
EcuM_Initializer is only introduced to improve readability of the diagram. See also Figure 3 – Startup Sequence (high level diagram) and its comments.

7.6 SLEEP State

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

7.6.1 High Level Sequence Diagram

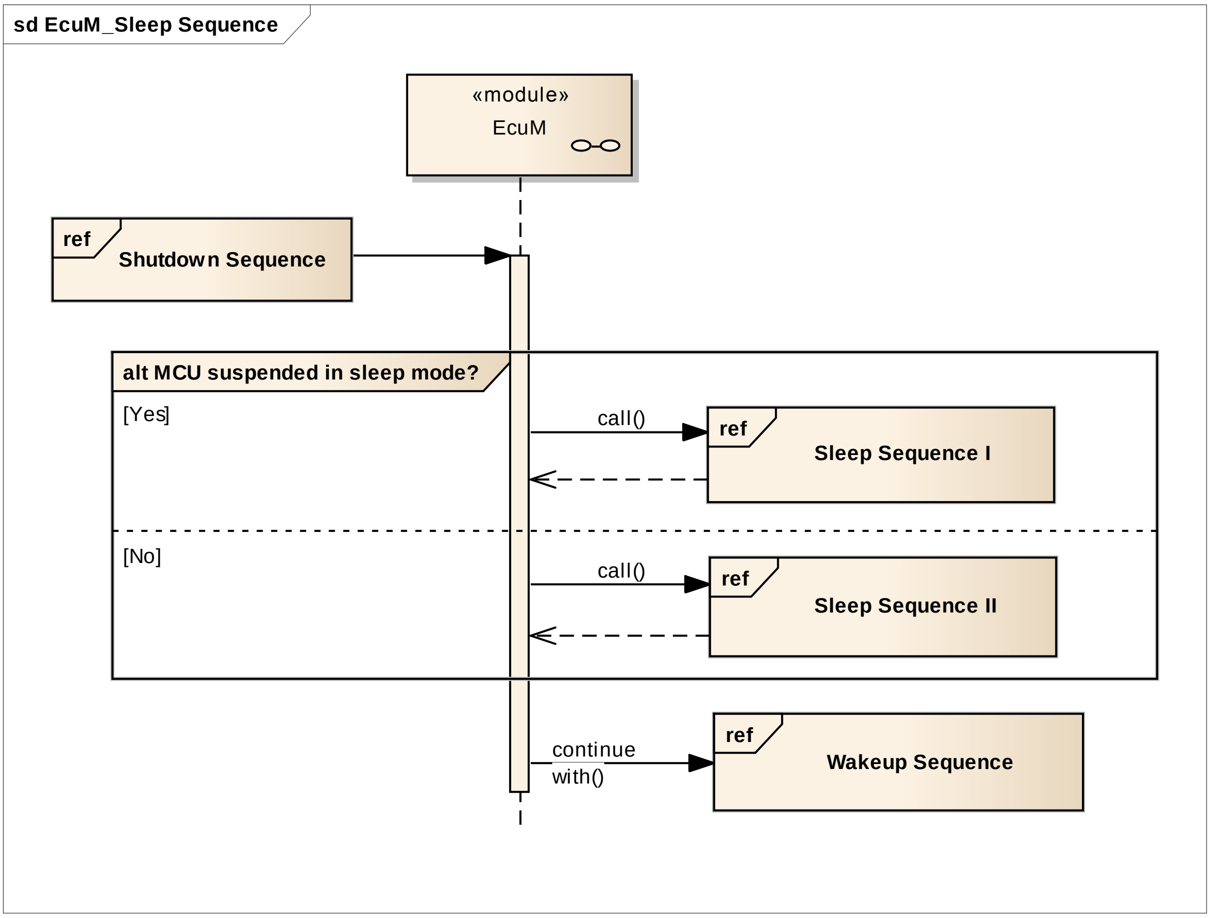

Figure 16 – Sleep Sequence (high level diagram)

To see adjacent diagrams refer to

Figure 11 – Shutdown Sequence (high level diagram)
Figure 19 – Wake up Sequence (high level diagram)
Figure 1 – ECU Main States (top level diagram)

7.6.2 Sub-State Descriptions

7.6.2.1 Shutdown Targets

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

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

EcuM2232: The default shutdown target shall be defined by configuration. This shutdown target shall be overridden by calling EcuM_SelectShutdownTarget.

The SLEEP state can define a configurable set of sleep modes, where each mode itself is a shutdown target (the bullet list above is a simplification). These sleep modes are hardware dependent and differ typically in clock settings or other low power features provided by the hardware. These different features are accessible through the MCU driver as so called MCU modes (see [12]). The ECU State Manager module allows to map these MCU modes to ECU sleep modes and hence they are addressable as shutdown targets. Further the configuration allows to define aliases for shutdown targets to simplify portability of code across different ECUs. See *10.2* Configurable Parameters container EcuMSleepMode for details.

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

7.6.2.2 Sleep Sequence I

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

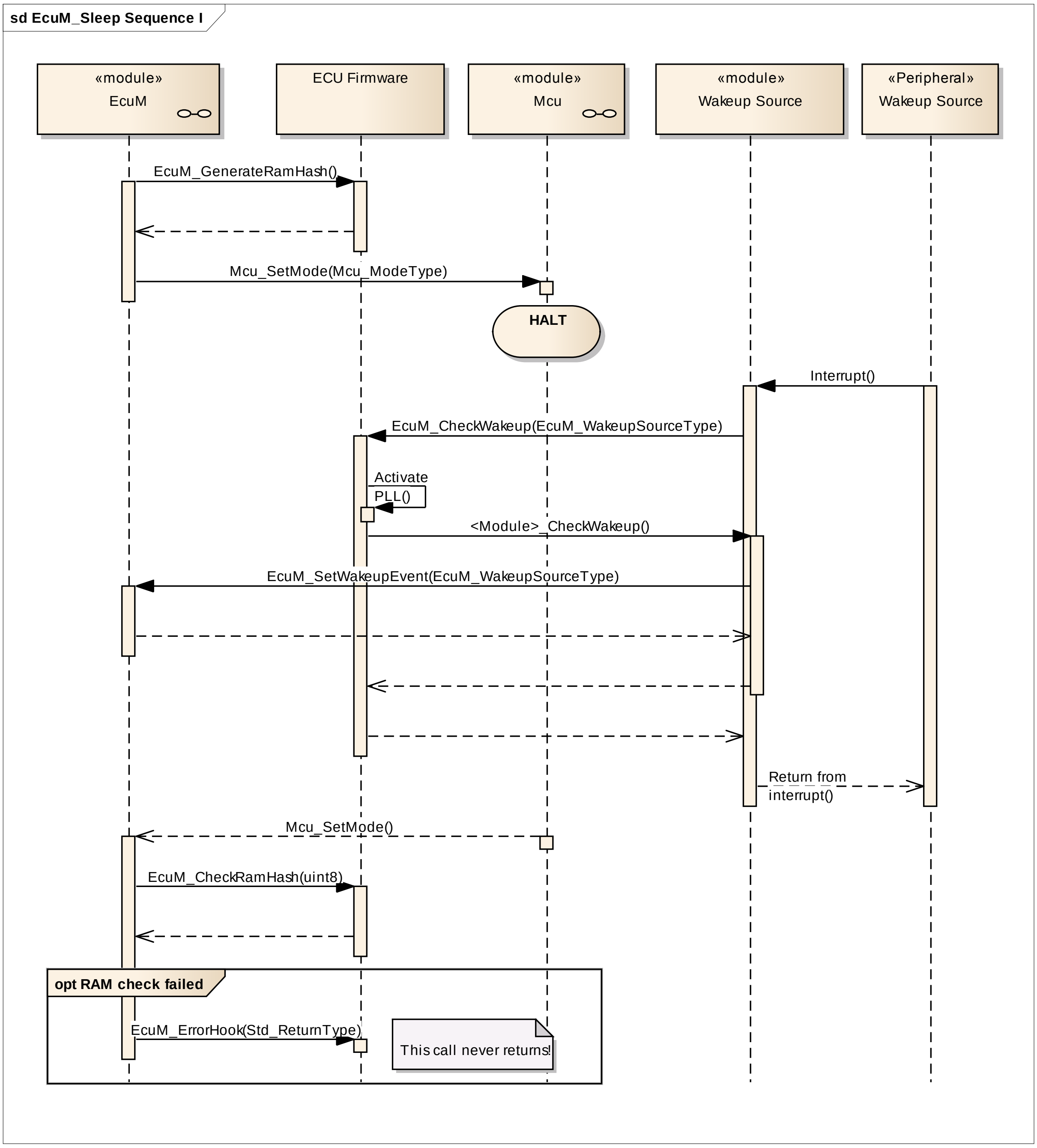

Figure 17 – Sleep Sequence I

A callout is invoked where the system designer can place a RAM integrity check. See also EcuM_GenerateRamHash `and EcuM2863`.

Either the module which provides the wakeup source is called directly from integration code "ECU Firmware" or via a <Bus> interface module.

7.6.2.3 Sleep Sequence II

Sleep Sequence II is executed in sleep modes that reduce the power consumption of the microcontroller but still execute code.

EcuM3020: In the Poll sequence the ECU State Manager module shall call the callouts `EcuM_SleepActivity()` and `EcuM_CheckWakeup()` in a blocking loop until a pending wake up event is reported.

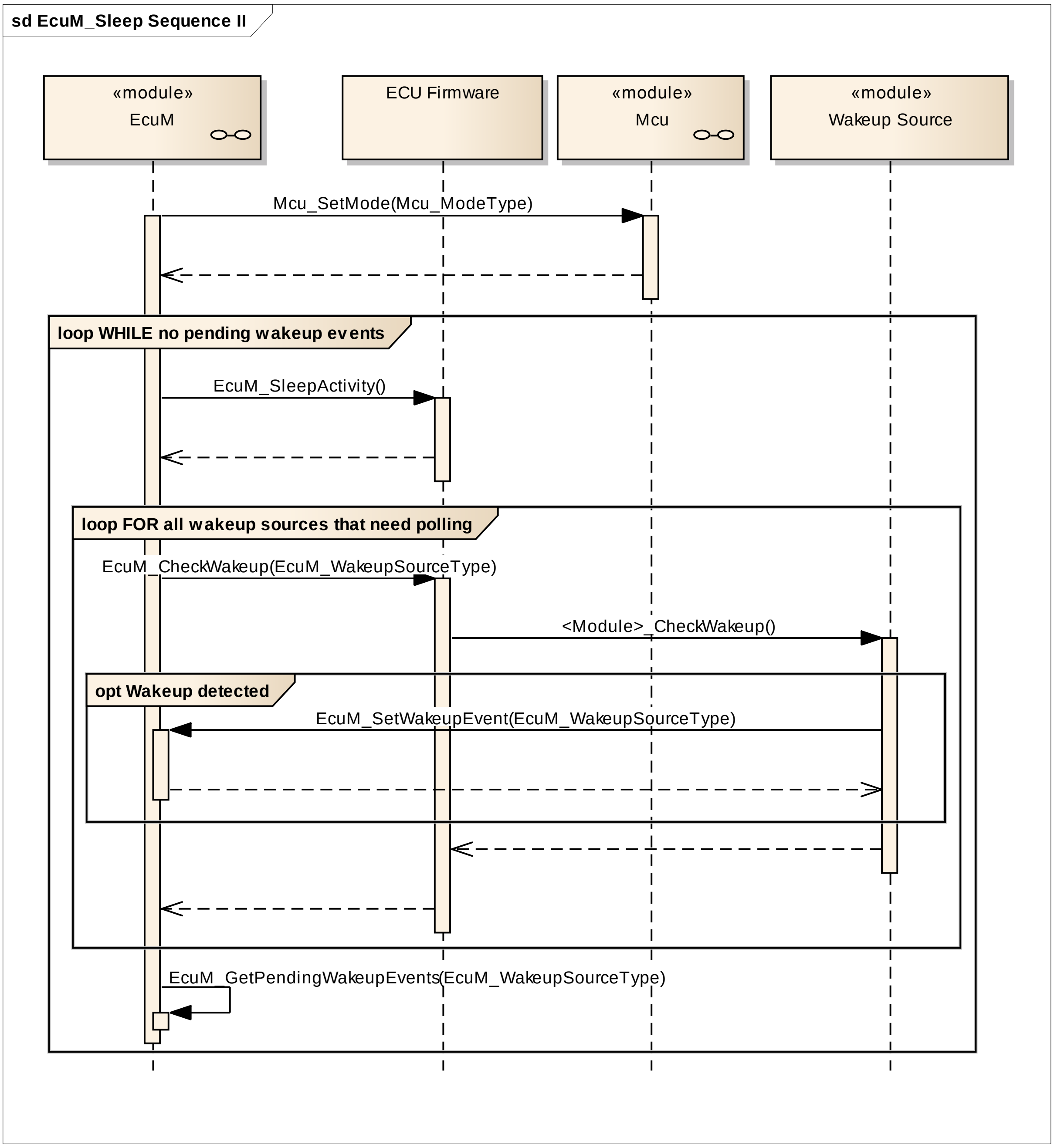

Figure 18 – Sleep Sequence II

Either the module which provides the wakeup source is called directly from integration code "ECU Firmware" or via a <Bus> interface module.

7.6.3 Leaving SLEEP State

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

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

7.7 WAKEUP State

7.7.1 High Level Sequence Diagram

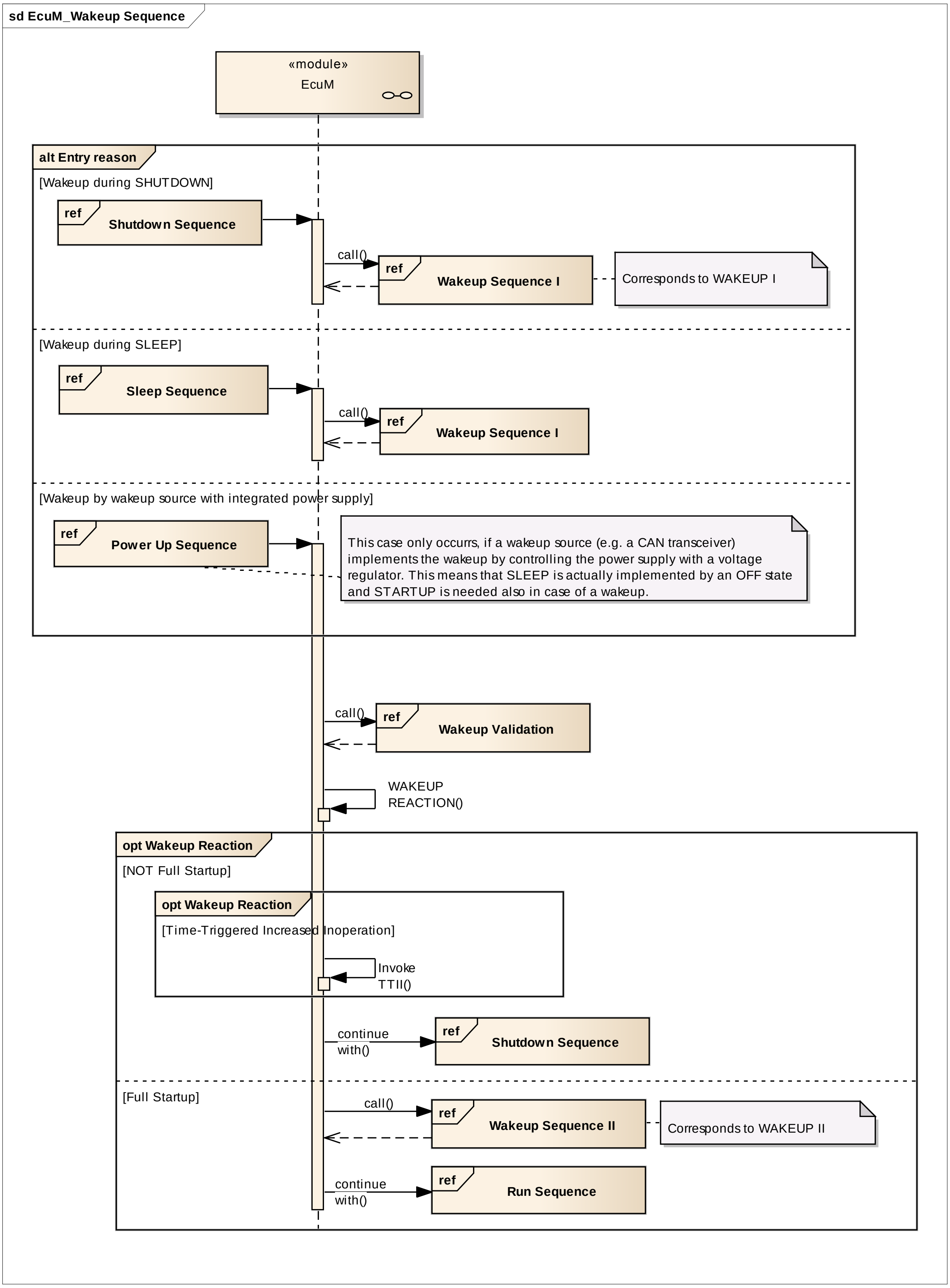

Figure 19 – Wake up Sequence (high level diagram)

To see adjacent diagrams, refer to

Figure 11 – Shutdown Sequence (high level diagram)

Figure 7 – RUN State Sequence (high level diagram)

7.7.2 State Breakdown Structure

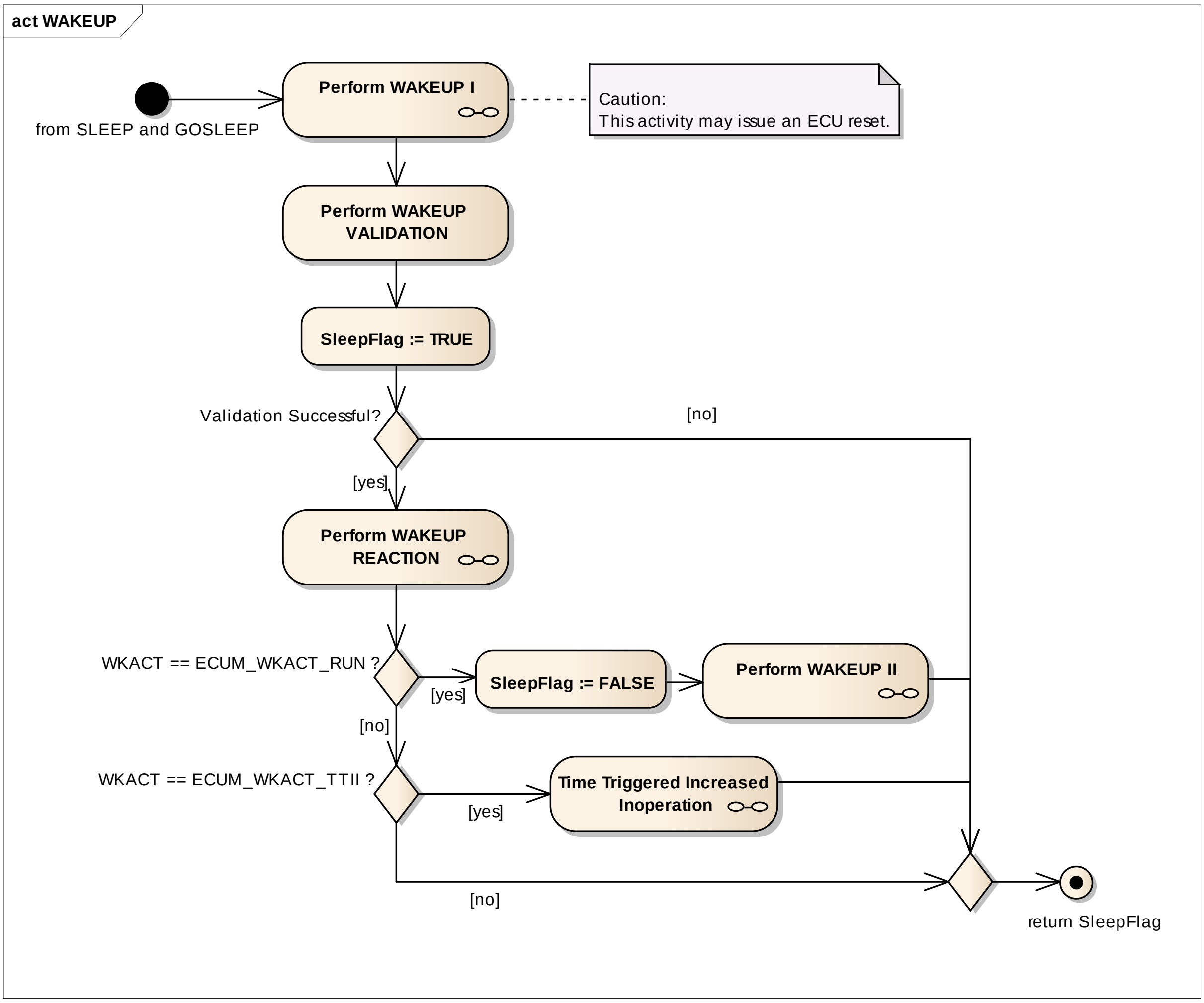

Figure 20 – WAKEUP State Breakdown

7.7.3 WAKEUP Activity Overview

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

Table 4 - Wake up Activities

[21] Rows marked with × are conditional.

7.7.4 Sub-State Descriptions

7.7.4.1 WAKEUP I

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

EcuM2539: During re-initialization, a driver must check if one of its assigned wake up sources was the reason for the previous wake up. If this test is true, it must invoke its 'wake up detected' callback (see [20] for an example), which in turn has to call the EcuM_SetWakeupEvent service. As a result, when WAKEUP I has finished, the ECU State Manager module has a list of wake up source candidates. These wake up source candidates still may need validation. See also *7.8* Wake up Validation Protocol for more information.

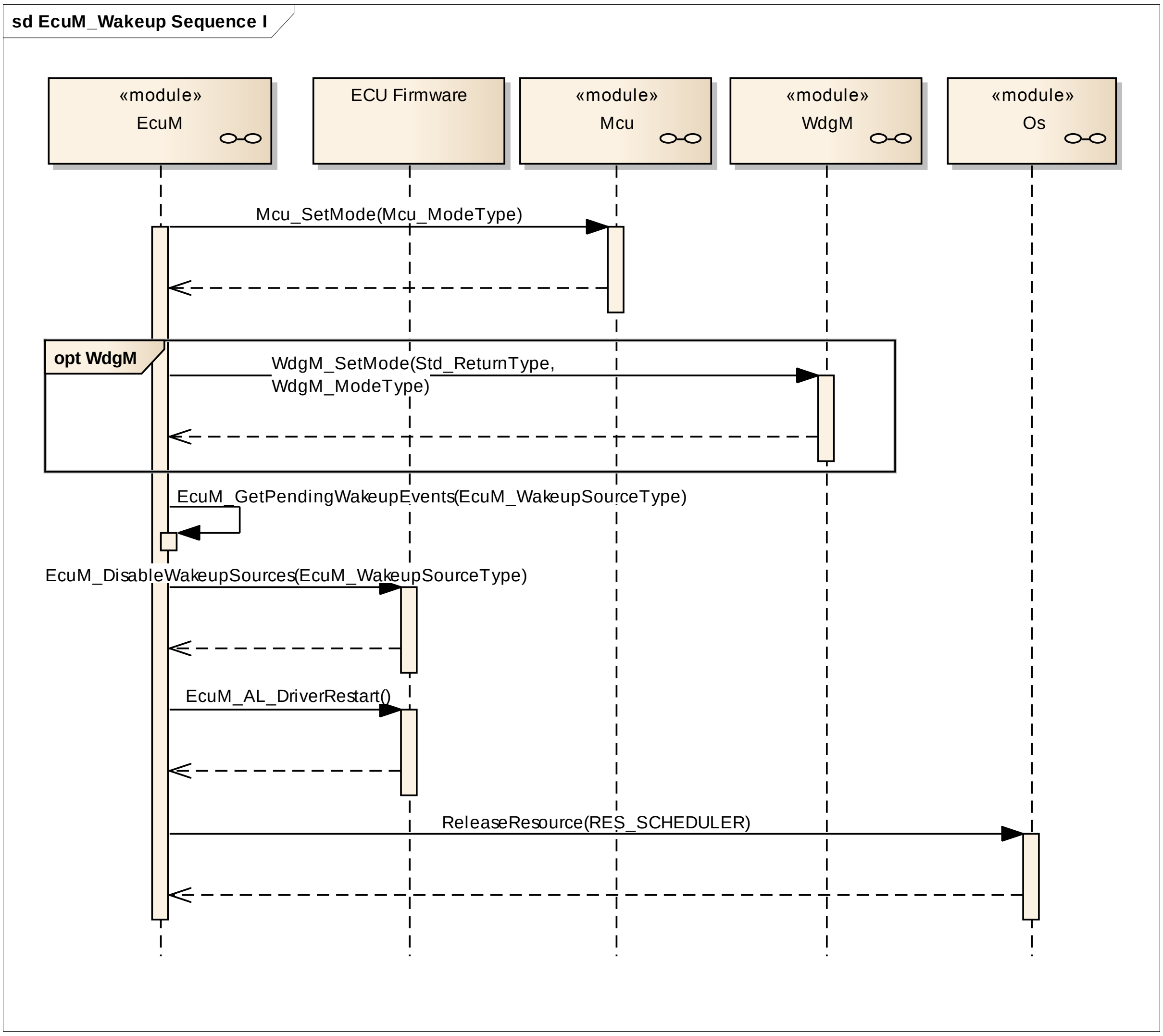

Figure 21 – Wake up Sequence I

EcuM2545: The driver should be implemented in a way that it only invokes the wake up callback once and then requires a dedicated service call to re-arm this mechanism. The driver then needs to be re-armed to fire the callback again.

7.7.4.2 WAKEUP VALIDATION

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

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

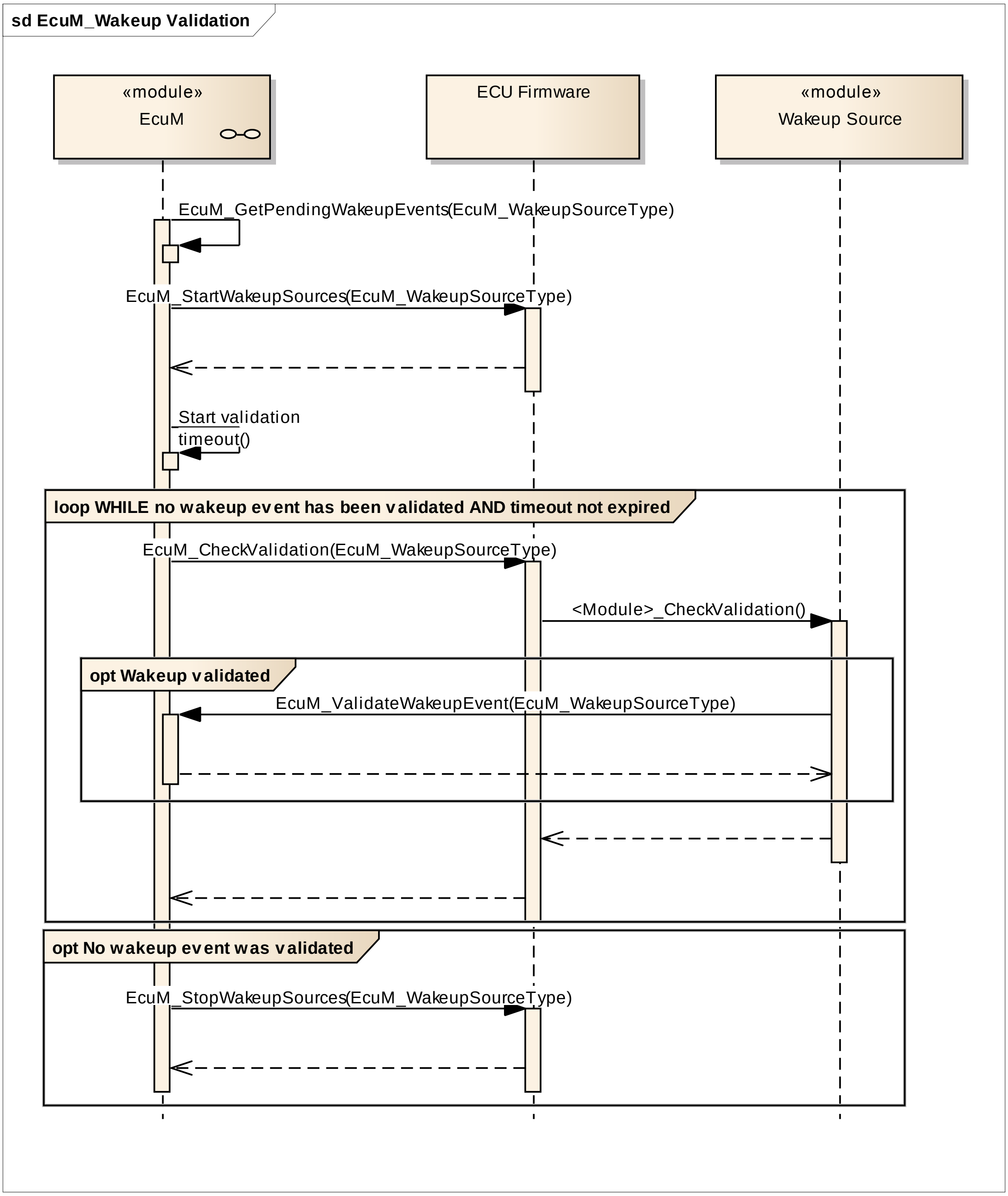

Figure 22 – Wake up Validation Sequence

EcuM2566: Wake up validation shall apply only to those wake up sources where it is required by configuration. If the validation protocol is not configured, then a call to EcuM_SetWakeupEvent shall also imply a call to EcuM_ValidateWakeupEvent.

EcuM2565: For each pending wake up event, for which validation is required, a validation timeout shall be started. The timeout is event specific and can be defined by configuration. Strictly spoken, it is sufficient for an implementation to provide only one timer, which is prolonged to the largest timeout when new wake up events are reported.

EcuM2567: If the last timeout expires without validation then the wake up validation is considered to have failed.

EcuM2568: If at least one of the pending events is validated then the entire validation has passed.
Pending events are validated with a call to EcuM_ValidateWakeupEvent. This call must be placed in the driver or the consuming stack on top of the driver (e.g. the handler). The best place to put this depends on hardware and software design. See also *7.8.5* Requirements for Drivers with Wake up Sources.

7.7.4.3 WAKEUP REACTION

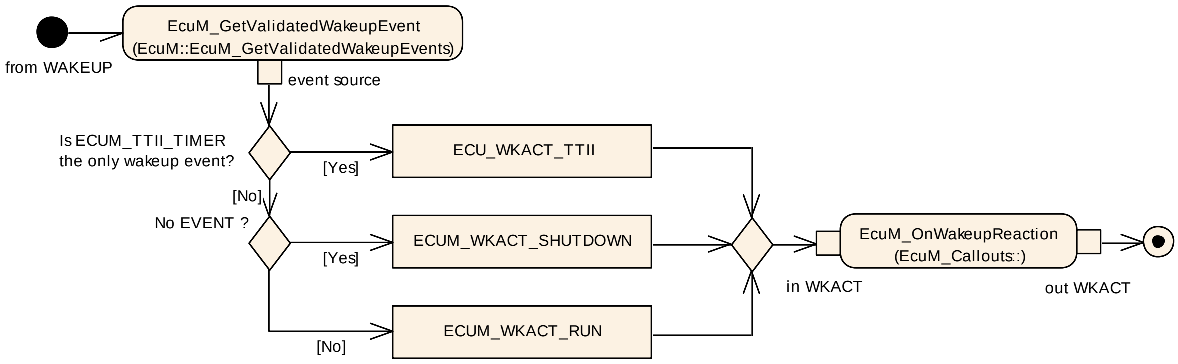

Figure 23 – Activity Diagram of WAKEUP REACTION

The WAKEUP REACTION state determines the appropriate wake up reaction (see *8.2.6* EcuM_WakeupReactionType) according to the wake up source (see *8.2.4* EcuM_WakeupSourceType).
As can be seen from, Figure 23 – Activity Diagram of WAKEUP REACTION there are the following wake up reactions:

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

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

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

7.7.4.4 WAKEUP II

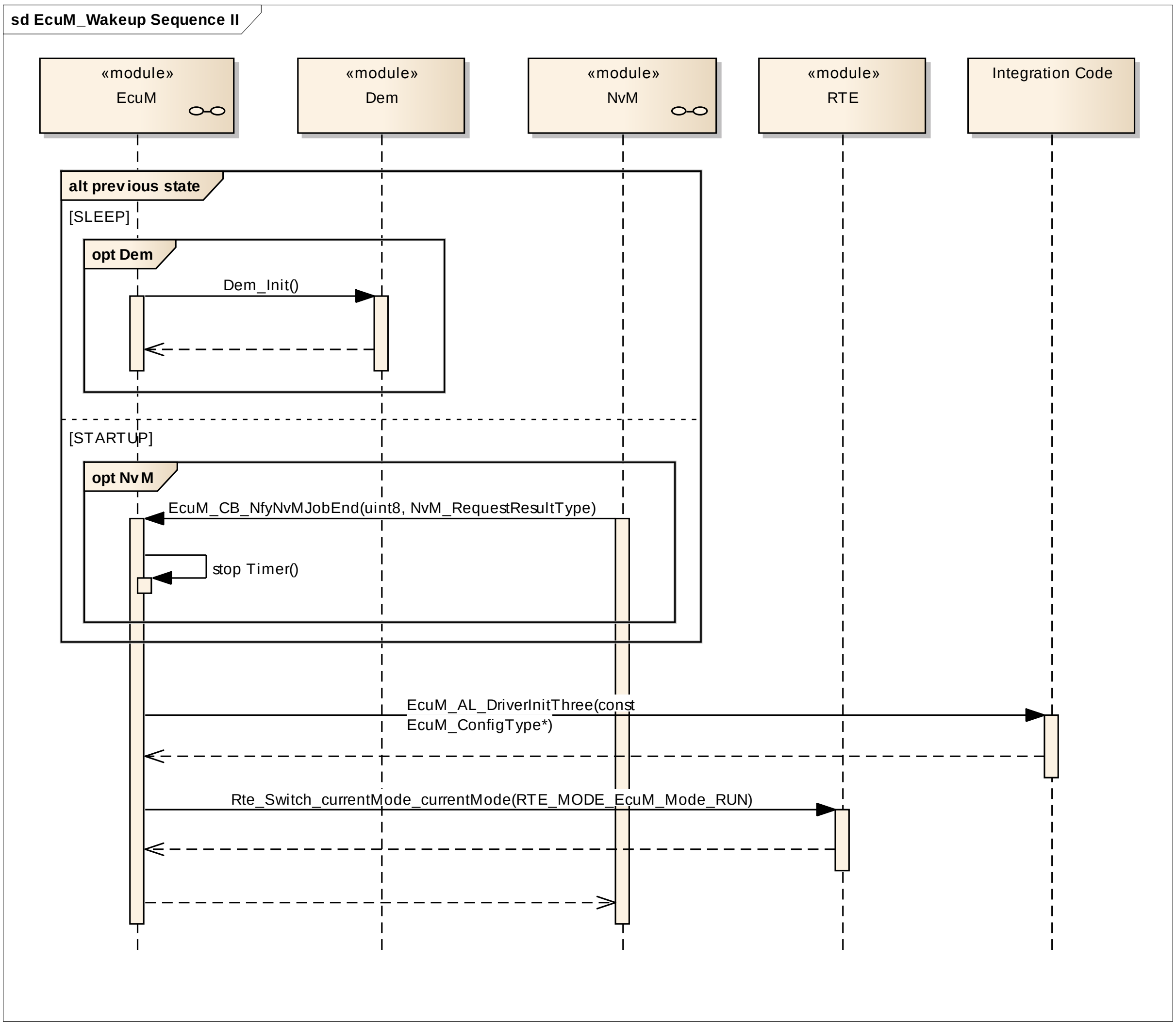

Figure 24 – Wake up Sequence II

7.8 Wake up Validation Protocol

7.8.1 Wake up of Communication Channels

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

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

The state machines of the communication channels are completely provided by the Communication Manager, see [5] for details.

According to the specification, the Communication Manager autonomously can fulfill the following tasks:

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

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

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

EcuM2479: The ECU State Manager module shall execute the Wake up Validation Protocol according to *7.8.3* Interaction of Wake up Sources and the ECU later in this chapter.

EcuM2480: If validation is successful, the ECU State Manager module shall inform the Communication Manager about the wake up event by invoking the Communication Manager's `ComM_EcuM_WakeUpIndication` service with the according channel as parameter.

In turn, the Communication Manager will use this event to bring the channel into full communication mode.

[EcuM3202] If at least one valid wake up is detected the ECU shall perform a startup as fast as possible.

[EcuM3203] If in addition to a validated wake up an "invalid" wake up occurs as well, it is tolerable to indicate it too. This does not contradict BSW09097 (Validation of physical channel wake-up), since the ECU has to start anyway.

7.8.2 Wake up of the Entire ECU

Before the ECU State Manager module can put the ECU into SLEEP state, the Communication Manager must have released all run requests[23]. This will only happen, if all communication state machines are in 'no communication' mode. But this, taking into account the previous paragraphs, implies that all wake up sources (e.g. bus transceivers) must have been put to standby state and the wake up source must have been armed. Thus, when a wake up occurs, all communication channels are in no communication state and there are no RUN requests.
The wake up procedure is identical to the previous chapter.

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

7.8.3 Interaction of Wake up Sources and the ECU State Manager module

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

EcuM2492: The ECU State Manager module has to collect all wake up indications of the responsible drivers of the wake up event.

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

EcuM2494: If wake up validation is required for this event, then the validation protocol applies. Otherwise the event is valid immediately.

EcuM2495: If the valid event is a wake up event from a communication interface then it is propagated to the Communication Manager.

EcuM2496: If the wake up event is validated (either immediately or by the wake up validation protocol), it is labelled as a wake up source and this information is made available to the application by the EcuM_GetValidatedWakeupEvents service.

7.8.4 Wake up Validation Timeout

It is the implementer's choice whether he wants to provide a single wake up validation timeout timer or one timer per wake up source. This timer must be configurable either per module or per wakeupSource.
The validation timeout period is defined by configuration (container EcuMWakeupSource, EcuMValidationTimeout).

The following requirements apply:

EcuM2709: The timer shall be started when the service EcuM_SetWakeupEvent is called.

EcuM2710: The timer shall be stopped and the validation is set to passed when the service EcuM_ValidateWakeupEvent is called.

EcuM2711: When the timer expires, validation is set to failed.

EcuM2712: Subsequent calls to EcuM_SetWakeupEvent for the same wake up source shall not prolong the timeout.
If only one timer is used, the following approach is proposed:

EcuM2714: If EcuM_SetWakeupEvent is called for a wake up source which did not fire yet during the same wake up cycle then the timeout should be prolonged for the validation timeout of that wake up source.
Wake up timeouts are defined by configuration in chapter *10.2* Configurable Parameters.

7.8.5 Requirements for Drivers with Wake up Sources

EcuM2571: The driver shall invoke the EcuM_SetWakeupEvent service with a configurable parameter identifying the source of the wake up once when the wake up event is detected.

EcuM2572: Wake-ups which occurred prior to driver initialization shall be detectable. This applies to initialization from SLEEP or from OFF state.

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

The callback invocation is enabled by calling the driver initialization service.

7.8.6 Requirements for Wake up Validation

EcuM2575: If the wake up source requires validation, this may be done by any but only by one appropriate module of the basic software. This may be a driver, an interface, a handler, or a manager.

Validation is done by calling the EcuM_ValidateWakeupEvent service.

7.8.7 Wake up Sources and Reset Reason

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

EcuM2625: The following wake up sources shall not require validation under no circumstances:

- `ECUM_WKSOURCE_POWER`
- `ECUM_WKSOURCE_RESET`
- `ECUM_WKSOURCE_INTERNAL_RESET`
- `ECUM_WKSOURCE_INTERNAL_WDG`
- `ECUM_WKSOURCE_EXTERNAL_WDG`.

7.8.8 Wake up Sources with Integrated Power Control

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

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

This behavior can be applied to all kinds of wake up sources.

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

7.8.9 Activity Diagram

.

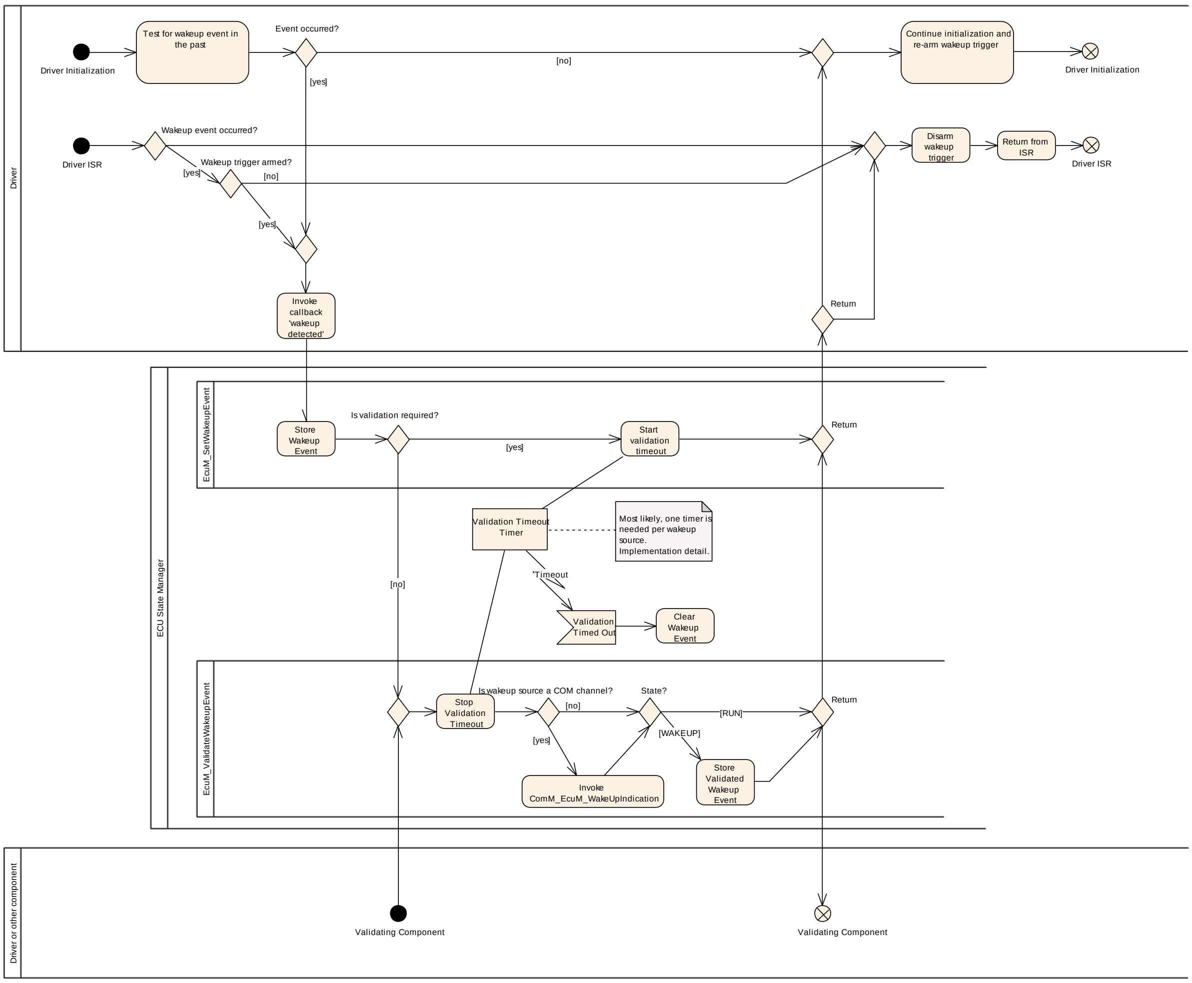

Figure 25 – Wake up Validation Protocol

7.9 Time Triggered Increased Inoperation

EcuM2653: TTII shall manage a list of sleep modes (shutdown targets). These sleep modes can be defined at configuration time. Typically the sleep modes are ordered to deepen the sleep phase of the ECU (decreased power consumption).

EcuM2654: An entry of the sleep mode list shall contain the following properties:

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

These properties shall be defined at configuration time (see *10.2* Configurable Parameters container EcuMTTII).

The TTII protocol is executed during the WAKEUP REACTION sub-state. Refer to chapter *7.7.4.3* WAKEUP REACTION and Figure 23 – Activity Diagram of WAKEUP REACTION.

EcuM2223: The entire TTII feature can be completely disabled by setting the *ECUM_TTII_ENABLED* configuration parameter to false. All further described activities are only applicable if TTII is enabled.

EcuM2222: A wake up source must be selected by configuration (*ECUM_TTII_WKSOURCE* configurable parameter) for use by the TTII protocol. Typically, the wake up source will be a timer, which serves as a timebase for TTII. Whenever the ECU is woken up by this configured wake up source, then the TTII protocol shall be executed.

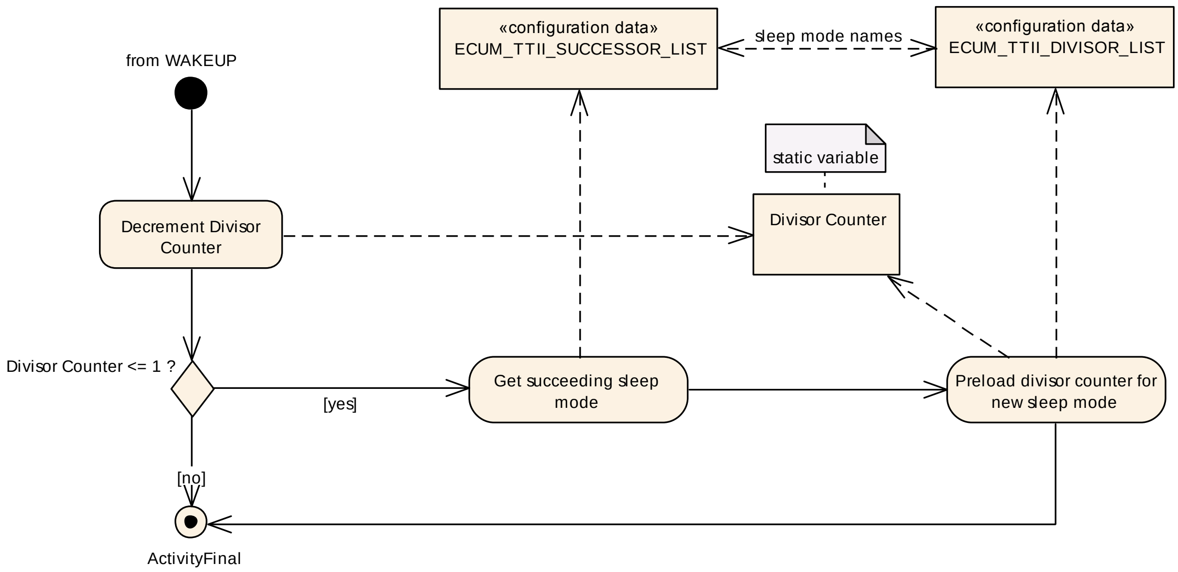

Figure 26 – Activity Diagram of TTII

7.10 AUTOSAR Ports

7.10.1 Scope of this Chapter

This chapter defines the AUTOSAR Interfaces of the ECU State Manager module.

7.10.2 Overview

The overall architecture of the ECU State Manager module service is depicted in the following picture.

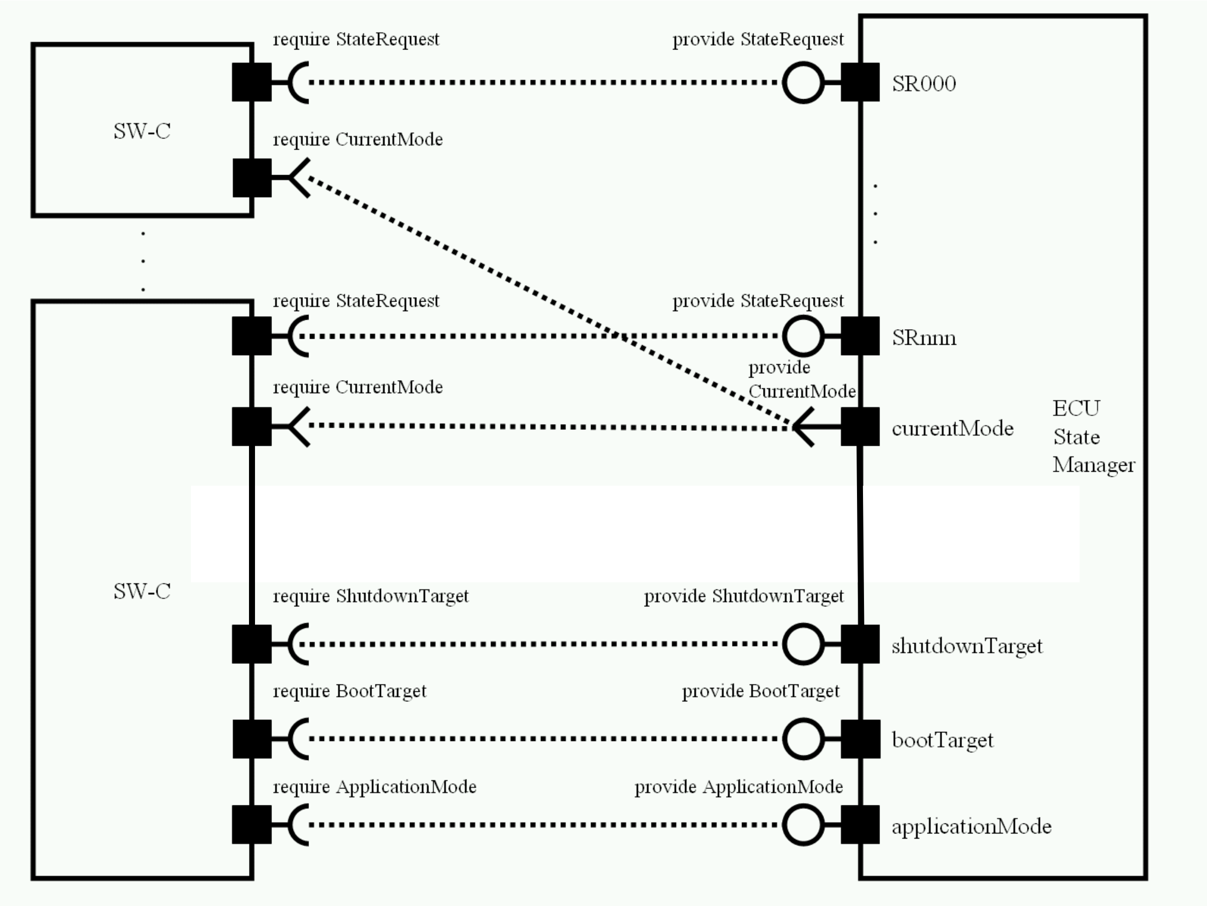

Figure 27 – ARPackage EcuM

7.10.3 Use Cases

EcuM2762: The ECU State Manager module shall provide AUTOSAR ports for the following functionalities:

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

EcuM2763: The ECU State Manager module shall provide also AUTOSAR ports for the following functionality:

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

The interfaces used in this chapter will be described in the following.

7.10.4 Specification of the Port Interfaces

This chapter specifies the Port Interfaces that are needed in order to operate the ECU State Manager module functionality over the VFB. The ports implementing the Port Interfaces described in this chapter will be defined in chapter 7.10.5.

7.10.4.1 Port Interface for Interface EcuM_StateRequest

7.10.4.1.1 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 states. See [rte_sws_1360] in [17] for a description of port-defined argument values.

7.10.4.1.2 Data Types

No data types are need for this interface

7.10.4.1.3 Port Interface

```
ClientServerInterface EcuM_StateRequest
{
      PossibleErrors {
      E_NOT_OK = 1 /* The request was not accepted by EcuM, a detailed
      error condition was sent to DET */
      };

      // The SW-C can request or release an ECU RUN or POST_RUN state when
      // requiring this interface
      RequestRUN(ERR{E_NOT_OK});
      ReleaseRUN(ERR{E_NOT_OK});
      RequestPOSTRUN(ERR{E_NOT_OK});
      ReleasePOSTRUN(ERR{E_NOT_OK});
};
```

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

7.10.4.2 Port Interface for Interface EcuM_CurrentMode

7.10.4.2.1 General Approach

EcuM2749: The mode port of the ECU State Manager module shall declare the following modes:

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

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

EcuM2750: State changes shall be notified to SW-Cs through the RTE mode ports when the state change occurs. The ECU State Manager module shall not wait until the RTE has performed the mode switch completely.

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

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

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

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

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

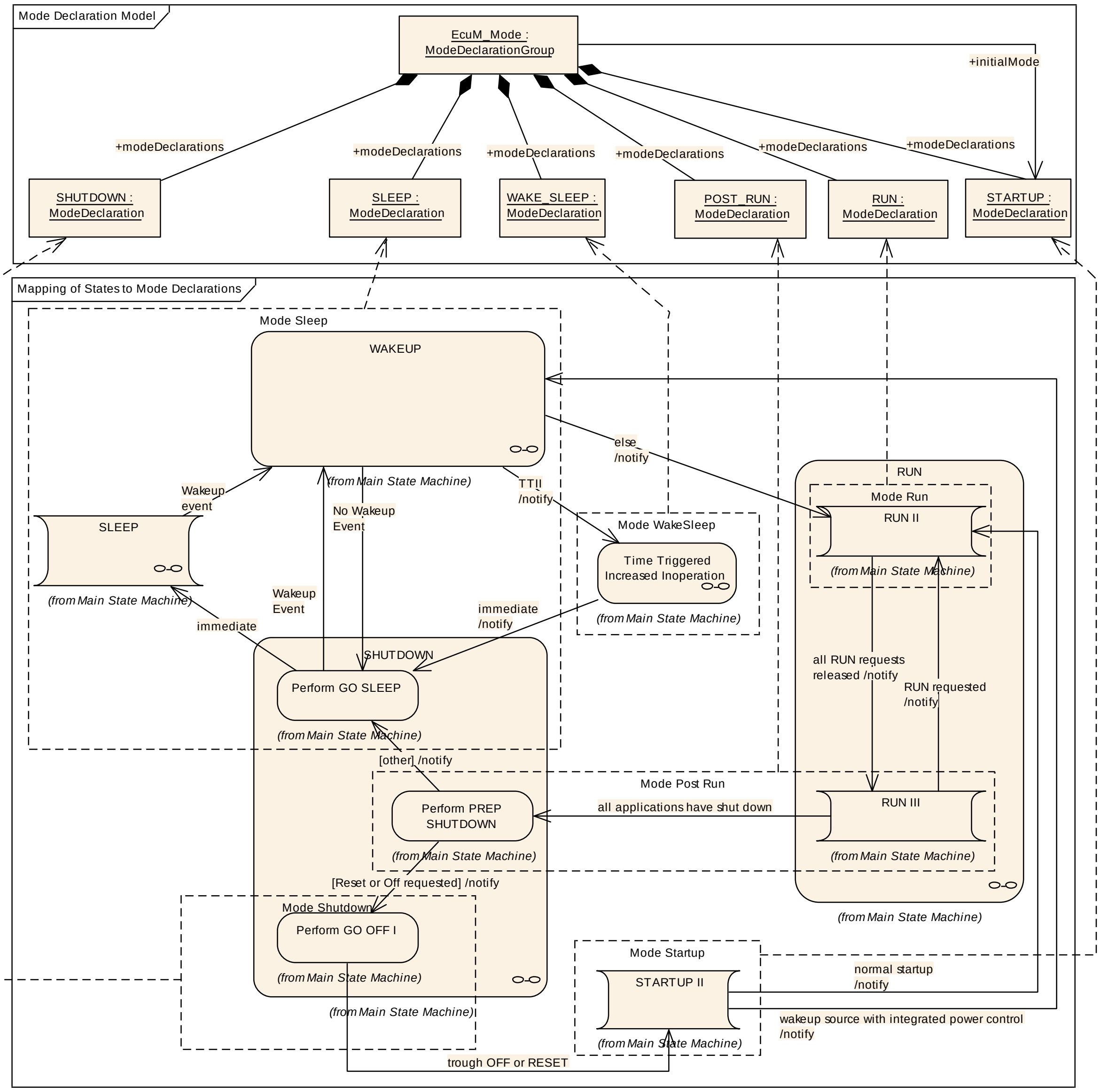

Figure 28 – Mapping of declared modes to ECU State Manager module states

EcuM2752: The ECU State Manager module shall notify WakeSleep mode and Sleep mode when transiting from WAKEUP to SHUTDOWN, but only if the selected shutdown target is SLEEP.
This allows the system designer to trigger runnables for wake-sleep operations or TTII.

7.10.4.2.2 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,
        WAKE_SLEEP,
        SHUTDOWN
      }
      initialMode = STARTUP
};
```

7.10.4.2.3 Port Interface

```
SenderReceiverInterface EcuM_CurrentMode {
      EcuM_Mode currentMode;
};
```

7.10.4.3 Ports and Port Interface for Interface EcuM_ShutdownTarget

7.10.4.3.1 General Approach

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

7.10.4.3.2 Data Types

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

```
PrimitiveTypeWithSemantics EcuM_StateType {
      IntegerType {
            LOWER-LIMIT=0x10, UPPER-LIMIT=0x90
      };
      0x10 -> ECUM_STATE_STARTUP
      0x11 -> ECUM_STATE_STARTUP_ONE
      0x12 -> ECUM_STATE_STARTUP_TWO
      0x20 -> ECUM_STATE_WAKEUP
      0x21 -> ECUM_STATE_WAKEUP_ONE
      0x22 -> ECUM_STATE_WAKEUP_VALIDATION
      0x23 -> ECUM_STATE_WAKEUP_REACTION
      0x24 -> ECUM_STATE_WAKEUP_TWO
      0x25 -> ECUM_STATE_WAKEUP_WAKESLEEP
      0x26 -> ECUM_STATE_WAKEUP_TTII
      0x30 -> ECUM_STATE_RUN
      0x32 -> ECUM_STATE_APP_RUN
      0x33 -> ECUM_STATE_APP_POST_RUN
      0x40 -> ECUM_STATE_SHUTDOWN
      0x44 -> ECUM_STATE_PREP_SHUTDOWN
      0x49 -> ECUM_STATE_GO_SLEEP
      0x4d -> ECUM_STATE_GO_OFF_ONE
      0x4e -> ECUM_STATE_GO_OFF_TWO
      0x50 -> ECUM_STATE_SLEEP
      0x80 -> ECUM_STATE_OFF
      0x90 -> ECUM_STATE_RESET
```

```
};
```

7.10.4.3.3 Port Interface

```
ClientServerInterface EcuM_ShutdownTarget
{
      // The SW-C can select a shutdown target when requiring
      // this interface
      PossibleErrors {
      E_NOT_OK = 1 /* The new shutdown target was not set */
      };

      // The SW-C selects a shutdown target
      SelectShutdownTarget(IN EcuM_StateType target, IN uint8 mode,
      ERR{E_NOT_OK});

      // The SW-C gets the currently selected shutdown target
      GetShutdownTarget(OUT EcuM_StateType target, OUT uint8 mode);

      // The SW-C gets the shutdown target of the previous shutdown process
      GetLastShutdownTarget(OUT EcuM_StateType target, OUT uint8 mode);
};
```

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

7.10.4.4 Port Interface for Interface EcuM_BootTarget

7.10.4.4.1 General Approach

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

7.10.4.4.2 Data Types

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

```
PrimitiveTypeWithSemantics EcuM_BootTargetType {
      IntegerType {LOWER-LIMIT=0, UPPER-LIMIT=1};
      0 -> ECUM_BOOT_TARGET_APP
      1 -> ECUM_BOOT_TARGET_BOOTLOADER

      // ECUM_BOOT_TARGET_APP: The ECU will boot into the application
      // ECUM_BOOT_TARGET_BOOTLOADER: The ECU will boot into the bootloader
};
```

7.10.4.4.3 Port Interface

```
ClientServerInterface EcuM_BootTarget
{
      PossibleErrors {
      E_NOT_OK = 1 /* The new boot target was not accepted by EcuM */
      };

      // The SW-C selects a boot target
      SelectBootTarget (IN EcuM_BootTargetType target, ERR{E_NOT_OK});

      // The SW-C gets informed of the current boot target
      GetBootTarget(OUT EcuM_BootTargetType target);
};
```

7.10.4.5 Port Interface for Interface EcuM_ApplicationMode

7.10.4.5.1 General Approach

A SW-C which wants to select an application mode shall require the client-server interface `EcuM_ApplicationMode`.

7.10.4.5.2 Data Types

The data type EcuM_`AppModeType` represents the application mode.

7.10.4.5.3 Port Interface

```
ClientServerInterface EcuM_ApplicationMode
{
      PossibleErrors {
      E_NOT_OK = 1 /* The new application mode was not accepted by EcuM */
      };

      // The SW-C selects an application mode
      SelectApplicationMode (IN EcuM_AppModeType appMode, ERR{E_NOT_OK});

      // The SW-C gets informed of the current application mode
      GetApplicationMode (OUT EcuM_AppModeType appMode);
};
```

7.10.5 Summary of ports

7.10.5.1 Definitions of interfaces

```
PrimitiveTypeWithSemantics EcuM_StateType {
      IntegerType {
            LOWER-LIMIT=0x10, UPPER-LIMIT=0x90
      };
      0x10 -> ECUM_STATE_STARTUP
      0x11 -> ECUM_STATE_STARTUP_ONE
      0x12 -> ECUM_STATE_STARTUP_TWO
      0x20 -> ECUM_STATE_WAKEUP
      0x21 -> ECUM_STATE_WAKEUP_ONE
```

```
        0x22 -> ECUM_STATE_WAKEUP_VALIDATION
        0x23 -> ECUM_STATE_WAKEUP_REACTION
        0x24 -> ECUM_STATE_WAKEUP_TWO
        0x25 -> ECUM_STATE_WAKEUP_WAKESLEEP
        0x26 -> ECUM_STATE_WAKEUP_TTII
        0x30 -> ECUM_STATE_RUN
        0x32 -> ECUM_STATE_APP_RUN
        0x33 -> ECUM_STATE_APP_POST_RUN
        0x40 -> ECUM_STATE_SHUTDOWN
        0x44 -> ECUM_STATE_PREP_SHUTDOWN
        0x49 -> ECUM_STATE_GO_SLEEP
        0x4d -> ECUM_STATE_GO_OFF_ONE
        0x4e -> ECUM_STATE_GO_OFF_TWO
        0x50 -> ECUM_STATE_SLEEP
        0x80 -> ECUM_STATE_OFF
        0x90 -> ECUM_STATE_RESET
};


ClientServerInterface EcuM_StateRequest
{
        PossibleErrors {
        E_NOT_OK = 1 /* The request was not accepted by EcuM, a detailed
        error condition was sent to DET */
        };

        // The SW-C can request or release an ECU RUN or POST_RUN state when
        // requiring this interface
        RequestRUN(ERR{E_NOT_OK});
        ReleaseRUN(ERR{E_NOT_OK});
        RequestPOSTRUN(ERR{E_NOT_OK});
        ReleasePOSTRUN(ERR{E_NOT_OK});
};

ModeDeclarationGroup EcuM_Mode {
        { STARTUP,
        RUN,
        POST_RUN,
        SLEEP,
        WAKE_SLEEP,
        SHUTDOWN
        }
        initialMode = STARTUP
};
SenderReceiverInterface EcuM_CurrentMode {
        EcuM_Mode currentMode;
};

ClientServerInterface EcuM_ShutdownTarget
{
        // The SW-C can select a shutdown target when requiring
        // this interface
        PossibleErrors {
        E_NOT_OK = 1 /* The new shutdown target was not set */
        };

        // The SW-C selects a shutdown target
        SelectShutdownTarget(IN EcuM_StateType target, IN uint8 mode,
        ERR{E_NOT_OK});

        // The SW-C gets the currently selected shutdown target
```

```
      GetShutdownTarget(OUT EcuM_StateType target, OUT uint8 mode);

      // The SW-C gets the shutdown target of the previous shutdown process
      GetLastShutdownTarget(OUT EcuM_StateType target, OUT uint8 mode);
};

PrimitiveTypeWithSemantics EcuM_BootTargetType {
      IntegerType {LOWER-LIMIT=0, UPPER-LIMIT=1};
      0 -> ECUM_BOOT_TARGET_APP
      1 -> ECUM_BOOT_TARGET_BOOTLOADER

      // Bootloader and application are two separated programs which in
      // many cases even can be flashed separately. The only way to get
      // from one image to another is through reset. The boot menu will
      // branch into the one or other image depending on the selected boot
      // target
};

ClientServerInterface EcuM_BootTarget
{
      PossibleErrors {
      E_NOT_OK = 1 /* The new boot target was not accepted by EcuM */
      };

      // The SW-C selects a boot target
      SelectBootTarget (IN EcuM_BootTargetType target, ERR{E_NOT_OK});

      // The SW-C gets informed of the current boot target
      GetBootTarget(OUT EcuM_BootTargetType target);
};

ClientServerInterface EcuM_ApplicationMode
{
      PossibleErrors {
      E_NOT_OK = 1 /* The new boot target was not accepted by EcuM */
      };

      // The SW-C selects an application mode
      SelectApplicationMode (IN EcuM_AppModeType appMode, ERR{E_NOT_OK});

      // The SW-C gets informed of the current application mode
      GetApplicationMode (OUT EcuM_AppModeType appMode);
};
```

7.10.5.2 Definition of the Service ECU State Manager module

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

```
/* This is the definition of the ECU State Manager as a service. This is
the "outside-view" of the ECU State Manager module, which must be visible
to the SW-C's / ECU-integrator */
Service EcuStateManager {
      // For each user the ECU State Manager module provides a port
      // to request/release RUN and POST_RUN states.
      // there are NU users;
      ProvidePort EcuM_StateRequest SR000;
      …
      ProvidePort EcuM_StateRequest SR<NU-1>;

      ProvidePort EcuM_CurrentMode currentMode;

      ProvidePort EcuM_ShutdownTarget shutdownTarget;

      ProvidePort EcuM_BootTarget bootTarget;

      ProvidePort EcuM_ApplicationMode applicationMode;
};
```

7.10.6 Runnables and Entry points

7.10.6.1 Internal behavior

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

```
InternalBehavior EcuStateManager {

      // Runnable entities of the EcuStateManager
      RunnableEntity RequestRUN
            symbol "EcuM_RequestRUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity ReleaseRUN
            symbol "EcuM_ReleaseRUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity RequestPOSTRUN
            symbol "EcuM_RequestPOST_RUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity ReleasePOSTRUN
            symbol "EcuM_ReleasePOST_RUN"
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectShutdownTarget
            symbol "EcuM_SelectShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetShutdownTarget
            symbol "EcuM_GetShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetLastShutdownTarget
            symbol "EcuM_GetLastShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectApplicationMode
            symbol "EcuM_SelectApplicationMode"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetApplicationMode
            symbol "EcuM_GetApplicationMode"
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectBootTarget
```

```
        symbol “EcuM_SelectBootTarget”
        canbeInvokedConcurrently = TRUE
    RunnableEntity GetBootTarget
        symbol “EcuM_GetBootTarget”
        canbeInvokedConcurrently = TRUE

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

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

7.11 Advanced Topics

7.11.1 Application Modes

Application Modes is a feature of the OS which allows to define different configurations, e.g. sets of tasks which will be started initially. The application mode is an in parameter of the `StartOS` service (ref. [5]). Since the ECU State Manager module is responsible for starting the OS, it has also responsibility for managing the application mode.

EcuM2700: An application mode change shall be accomplished by selecting the new application mode with the EcuM_SelectApplicationMode service (typically from RUN state) and a subsequent shutdown to the RESET shutdown target.

EcuM2243: The default application mode is set in the STARTUP I state in case of unintended restarts[24], see chapter *7.3.4.1* STARTUP I. After this point, the application mode can be modified by the application itself.

[24] e.g. like watchdog reset

7.11.2 Relation to Bootloader

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

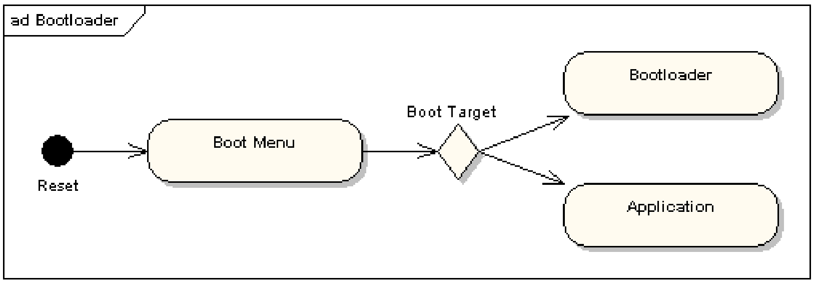

Figure 29 – Selection of Boot Targets

Bootloader and application are two separated programs which in many cases even can be flashed separately. The only way to get from one image to another is through reset. The boot menu will branch into the one or other image depending on the selected boot target.

7.11.3 Relation to Complex Drivers

EcuM2321: If the complex driver handles a wake up source, it must obey all rules of this specification which are related to handling wake up events.

EcuM2322: A complex driver may issue RUN requests.

7.11.4 Handling Errors during Startup and Shutdown

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

7.11.5 Configuration Alternative for Providing Wake-Sleep Operation

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

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

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

7.11.6 Selecting Scheduling Schemes for Startup and Shutdown

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

7.12 Error Classification

Type or error	*Relevance*	*Related error code*	*Value*
A service was called prior to initialization	Development	ECUM_E_NOT_INITED	0x10
A service was called which was disabled by configuration	Development	ECUM_E_SERVICE_DISABLED	0x11
A null pointer was passed as an argument	Development	ECUM_E_NULL_POINTER	0x12
A parameter was invalid (unspecific)	Development	ECUM_E_INVALID_PAR	0x13
RUN was requested multiple times by the same user ID	Development	ECUM_E_MULTIPLE_RUN_REQUESTS	0x14
RUN was released though it was not requested	Development	ECUM_E_MISMATCHED_RUN_RELEASE	0x15
A state, passed as an argument to a service, was out of range (specific parameter test)	Development	ECUM_E_STATE_PAR_OUT_OF_RANGE	0x16
An unknown wake up source was passed as a parameter to an API	Development	ECUM_E_UNKNOWN_WAKEUP_SOURCE	0x17
The RAM check during wake up failed	Production	ECUM_E_RAM_CHECK_FAILED	
The service EcuM_KillAllRUNRequests was issued	Production	ECUM_E_ALL_RUN_REQUESTS_KILLED	
Configuration data is inconsistent	Production	ECUM_E_CONFIGURATION_DATA_INCONSISTENT	

Table 5 - Error Classification

EcuM2759: All errors shall be reported as events.

EcuM2757: All errors shall be treated as errors immediately.

EcuM2758: All errors shall not be healable.

8 API specification

8.1 Imported Types

This chapter lists all data types included from the other modules:

EcuM2810:

Module	*Imported Type*
Adc	Adc_ConfigType
BswM	BswM_ConfigType
Can	Can_ConfigType
CanIf	CanIf_ConfigType
CanNm	CanNm_ConfigType
CanSM	CanSM_ConfigType
CanTrcv	CanTrcv_ConfigType
Com	Com_ConfigType
ComM	ComM_ConfigType
ComStack_Types	NetworkHandleType
Dcm	Dcm_ConfigType
Dem	Dem_EventIdType
Fr	Fr_ConfigType
FrIf	FrIf_ConfigType
FrNm	FrNm_ConfigType
FrSm	FrSm_ConfigType
FrTp	FrTp_ConfigType
Gpt	Gpt_ConfigType
Icu	Icu_ConfigType
IpduM	IpduM_ConfigType
Lin	Lin_ConfigType
LinIf	LinIf_ConfigType
	LinTp_ConfigType
Mcu	Mcu_ModeType
	Mcu_ResetType
	Mcu_ConfigType
Nm	Nm_ConfigType
NvM	NvM_RequestResultType
Os	AppModeType
PduR	PduR_PBConfigType
Port	Port_ConfigType
Pwm	Pwm_ConfigType
RTE	Rte_ModeType_EcuM_Mode
Spi	Spi_ConfigType
Std_Types	Std_ReturnType
	Std_VersionInfoType
Wdg	Wdg_ConfigType
WdgM	WdgM_ModeType
	WdgM_SupervisedEntityIdType
	WdgM_ConfigType

8.2 Type definitions

8.2.1 EcuM_ConfigType

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

EcuM2801: This structure shall hold the post-build configuration parameters for the ECU State Manager module as well as pointers to all ConfigType structures of modules that are initialized by the ECU State Manager module.

EcuM2793: The ECU State Manager module Configuration Tool shall specifically generate this structure for a given set of basic software modules that comprise the ECU configuration. The set of basic software modules is derived from the corresponding *EcuMModuleConfiguration* parameters.

EcuM2794: This structure shall contain an additional post-build configuration variant identifier (uint8/uint16/uint32 depending on algorithm to compute the identifier). See also chapter *10.4* Checking Configuration Consistency*.*

EcuM2795: This structure shall contain an additional hash code with is tested against the configuration parameter *EcuMConfigConsistencyHash* for checking consistency of the configuration data. See also chapter *10.4* Checking Configuration Consistency*.*

EcuM2800: The ECU State Manager module Configuration Tool shall also generate for each given ECU configuration an instance of this structure that is filled with the post-build configuration parameters of the ECU State Manager module as well as pointers to instances of configuration structures for the modules mentioned in **EcuM2793**. The pointers are derived from the corresponding *EcuMModuleConfiguration* parameters.

8.2.2 EcuM_StateType

Name:	EcuM_StateType		
Type:	uint8		
Range:	ECUM_SUBSTATE_MASK	0x0f	--
	ECUM_STATE_STARTUP	0x10	--
	ECUM_STATE_STARTUP_ONE	0x11	--
	ECUM_STATE_STARTUP_TWO	0x12	--
	ECUM_STATE_WAKEUP	0x20	--
	ECUM_STATE_WAKEUP_ONE	0x21	--
	ECUM_STATE_WAKEUP_VALIDATION	0x22	--
	ECUM_STATE_WAKEUP_REACTION	0x23	--
	ECUM_STATE_WAKEUP_TWO	0x24	--
	ECUM_STATE_WAKEUP_WAKESLEEP	0x25	--
	ECUM_STATE_WAKEUP_TTII	0x26	--
	ECUM_STATE_RUN	0x30	--

	ECUM_STATE_APP_RUN	0x32	--
	ECUM_STATE_APP_POST_RUN	0x33	--
	ECUM_STATE_SHUTDOWN	0x40	--
	ECUM_STATE_PREP_SHUTDOWN	0x44	--
	ECUM_STATE_GO_SLEEP	0x49	--
	ECUM_STATE_GO_OFF_ONE	0x4d	--
	ECUM_STATE_GO_OFF_TWO	0x4e	--
	ECUM_STATE_SLEEP	0x50	--
	ECUM_STATE_OFF	0x80	--
	ECUM_STATE_RESET	0x90	--
Description:	ECU State Manager states.		

EcuM507: Encodes states and sub-states of the ECU State Manager module. States are encoded in the hi-nibble, sub-state in the lo-nibble. The sub-state can be determined by ANDing the state value with `ECUM_SUBSTATE_MASK`.

EcuM2664: The ECU State Manager module shall define all states as listed in the `EcuM_StateType`.

8.2.3 EcuM_UserType

Name:	EcuM_UserType
Type:	uint8
Description:	Unique value for each user.

EcuM487: For each user, a unique value must be defined at system generation time. Ref. to 10.2 Configurable Parameters.

8.2.4 EcuM_WakeupSourceType

Name:	EcuM_WakeupSourceType		
Type:	uint32		
Range:	ECUM_WKSOURCE_INTERNAL_RESET	--	Internal reset of µC (bit 2) The internal reset typically only resets the µC core but not peripherals or memory controllers. The exact behavior is hardware specific. This source may also indicate an unhandled exception.
	ECUM_WKSOURCE_EXTERNAL_WDG	--	Reset by external watchdog (bit 4), if detection supported by hardware
	ECUM_WKSOURCE_INTERNAL_WDG	--	Reset by internal watchdog (bit 3)
	ECUM_WKSOURCE_POWER	--	Power cycle (bit 0)
	ECUM_WKSOURCE_ALL_SOURCES	--	~0 to the power of 29
	ECUM_WKSOURCE_RESET (default)	--	Hardware reset (bit 1). If hardware cannot distinguish between a power cycle and a reset reason, then this shall be the default wakeup source.
Description:	The bitfield provides one bit for each wakeup source. In WAKEUP state, all bits cleared indicates that no wakeup source is known. In STARTUP state, all bits cleared indicates that no reason for restart or reset is known. In this case, ECUM_WKSOURCE_RESET shall be assumed.		

EcuM2165: The list can be extended by configuration

EcuM2166: Extension values (see chapter *10.2* Configurable Parameters*)* must define single additional bits. The bit assignment shall be done by the configuration tool.

EcuM2601: If hardware cannot detect a specific wake up source, then the ECU State Manager module shall report a `ECUM_WKSOURCE_RESET` instead.

8.2.5 EcuM_WakeupStatusType

Name:	`EcuM_WakeupStatusType`		
Type:	`uint8`		
Range:	`ECUM_WKSTATUS_NONE`	0	No pending wakeup event was detected
	`ECUM_WKSTATUS_PENDING`	1	The wakeup event was detected but not yet validated
	`ECUM_WKSTATUS_VALIDATED`	2	The wakeup event is valid
	`ECUM_WKSTATUS_EXPIRED`	3	The wakeup event has not been validated and has expired therefore
Description:	The type describes the possible outcomes of the WAKEUP VALIDATION state. The type may be applied to one wakeup source or a collection of wakeup sources.		

See also *8.3.4.5* EcuM_GetStatusOfWakeupSource.

8.2.6 EcuM_WakeupReactionType

Name:	`EcuM_WakeupReactionType`		
Type:	`uint8`		
Range:	`ECUM_WKACT_RUN`	0	Initialization into RUN state
	`ECUM_WKACT_TTII`	2	Execute time triggered increased inoperation protocol and shutdown
	`ECUM_WKACT_SHUTDOWN`	3	Immediate shutdown
Description:	The type describes the possible outcomes of the WAKEUP REACTION state.		

8.2.7 EcuM_BootTargetType

Name:	`EcuM_BootTargetType`		
Type:	`uint8`		
Range:	`ECUM_BOOT_TARGET_APP`	0	The ECU will boot into the application
	`ECUM_BOOT_TARGET_BOOTLOADER`	1	The ECU will boot into the bootloader
Description:	--		

8.2.8 EcuM_AppModeType

Name:	`EcuM_AppModeType`		
Type:	`uint8`		
Range:	`ECUM_OSDEFAULTAPPMODE`	0	Default application mode
Description:	This data type represents the application mode, standardized for use by the EcuM RTE Services. Since the exact layout of AppModeType is not standardized by OSEK, the EcuM needs to ensure a correct mapping between EcuM_AppModeType and the OSEK-implementation specific AppModeType.		

8.3 Function Definitions

8.3.1 General

8.3.1.1 EcuM_GetVersionInfo

EcuM2813:

Service name:	EcuM_GetVersionInfo	
Syntax:	`void EcuM_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non 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.	

EcuM2728: This service returns the version information of this module. The version information includes:
- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

EcuM2729: This function shall be pre compile time configurable On/Off by the configuration parameter: ECUM_VERSION_INFO_API

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

8.3.2 Initialization and Shutdown

8.3.2.1 EcuM_Init

EcuM2811:

Service name:	EcuM_Init
Syntax:	`void EcuM_Init( )`
Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non 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)

8.3.2.2 EcuM_StartupTwo

EcuM2838:

Service name:	EcuM_StartupTwo

Syntax:	`void EcuM_StartupTwo(`
	
	`)`

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.

EcuM2806: This function must be called from a task which is started directly as a consequence of StartOS. I.e. either it must be called from an autostart task or it must be called from a task which is explicitely started.

8.3.2.3 EcuM_Shutdown

EcuM2812:

Service name:	EcuM_Shutdown

Syntax:	`void EcuM_Shutdown(`
	
	`)`

Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Non 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.

8.3.3 State Management

8.3.3.1 EcuM_GetState

EcuM2823:

Service name:	EcuM_GetState	

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

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

EcuM2423: The service must be accessible from an OS and an OS-free context as well as from an interrupt context.

8.3.3.2 EcuM_RequestRUN

EcuM2814:

Service name:	EcuM_RequestRUN	

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

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

EcuM2143: Requests 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 ignored.

EcuM2144: An implementation must track requests for each user known on the ECU. Run requests are specific to the user.

EcuM2668: RUN requests shall be ignored after EcuM_KillAllRUNRequests has been executed until the shutdown has completed.

Configuration of EcuM_RequestRUN: Ref. to *8.2.3* EcuM_UserType for more information about user IDs and their generation.

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

8.3.3.3 EcuM_ReleaseRUN

EcuM2815:

Service name:	EcuM_ReleaseRUN	

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

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	ID of the entity releasing the RUN state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The release request was accepted by EcuM E_NOT_OK: The release request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes below).
Description:	Releases a RUN request previously done with a call to EcuM_RequestRUN. The service is intended for implementing AUTOSAR ports.	

Configuration of EcuM_ReleaseRUN: Ref. to *8.2.3* EcuM_UserType for more information about user IDs and their generation.

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

8.3.3.4 EcuM_ComM_RequestRUN

EcuM2816:

Service name:	EcuM_ComM_RequestRUN	

Syntax:	`Std_ReturnType EcuM_ComM_RequestRUN(`
	`    NetworkHandleType channel`
	`)`	

Service ID[hex]:	0x0e	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	ID of the communication channel requesting the RUN state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request was accepted by EcuM E_NOT_OK: The request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes below).
Description:	The behavior is identical to EcuM_RequestRUN except that the parameter is not a	

	user but a communication channel.

EcuM2789: The ECU State Manager module shall track requests by communication channels in exactly the same way as it tracks other users.

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

8.3.3.5 EcuM_ComM_ReleaseRUN

EcuM2817:

Service name:	EcuM_ComM_ReleaseRUN	

Syntax:	`Std_ReturnType EcuM_ComM_ReleaseRUN(`
	`    NetworkHandleType channel`
	`)`	

Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	ID of the communication channel releasing the RUN state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The release request was accepted by EcuM E_NOT_OK: The release request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes below).
Description:	Releases a RUN request previously done with a call to EcuM_ComM_RequestRUN.	

EcuM2792: The service `EcuM_ComM_ReleaseRUN` shall clear all wake up events of the wakeup source corresponding to this channel.

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

8.3.3.6 EcuM_ComM_HasRequestedRUN

EcuM2818:

Service name:	EcuM_ComM_HasRequestedRUN	

Syntax:	`boolean EcuM_ComM_HasRequestedRUN(`
	`    NetworkHandleType channel`
	`)`	

Service ID[hex]:	0x1b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	ID of the communication channel being tested
Parameters (inout):	None	
Parameters (out):	None	
Return value:	boolean	true: The channel has reqeusted RUN state false: The channel has not requested RUN state

Description:	Returns if a channel has requested RUN state.

8.3.3.7 EcuM_RequestPOST_RUN

EcuM2819:

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

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

Configuration of EcuM_RequestPOST_RUN: Ref. to *8.2.3* EcuM_UserType for more information about user IDs and their generation.

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

8.3.3.8 EcuM_ReleasePOST_RUN

EcuM2820:

Service name:	EcuM_ReleasePOST_RUN	
Syntax:	`Std_ReturnType EcuM_ReleasePOST_RUN( EcuM_UserType user )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	ID of the entity releasing the POST RUN state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The release request was accepted by EcuM E_NOT_OK: The release request was not accepted by EcuM, a detailed error condition was sent to DET (see Error Codes

		below).
Description:	Releases a POST RUN request previously done with a call to EcuM_RequestPOST_RUN. The service is intended for implementing AUTOSAR ports.	

Configuration of `EcuM_ReleasePOST_RUN`: Ref. to *8.2.3* EcuM_UserType for more information about user IDs and their generation.

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

8.3.3.9 EcuM_KillAllRUNRequests

EcuM2821:

Service name:	EcuM_KillAllRUNRequests

Syntax:	`void EcuM_KillAllRUNRequests(`
	
	`)`

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

EcuM1872: The function `EcuM_KillAllRUNRequests` unconditionally releases all requests to RUN.
Note: As an effect the ECU State Manager module switches to RUN III state (see also **EcuM2311**), which allows for a controlled shutdown.

EcuM2600: As a consequence EcuM_RequestRUN must not accept any new requests unless the resulting shutdown has been completed.

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

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

8.3.3.10 EcuM_KillAllPostRUNRequests

[EcuM3204] :

Service name:	EcuM_KillAllPostRUNRequests

Syntax:	`void EcuM_KillAllPostRUNRequests(`
	
	`)`

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

[EcuM3205] The function `EcuM_KillAllPostRUNRequests` unconditionally releases all pending requests to PostRUN.

[EcuM3206⌉ As a consequence `EcuM_RequestPostRUN` must not accept any new requests unless the resulting shutdown has been completed.⌋()

8.3.3.11 EcuM_SelectShutdownTarget

EcuM2822:

Service name:	EcuM_SelectShutdownTarget	

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

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	target	The selected shutdown target.
	mode	An index like value which can be dereferenced to a sleep mode (EcuM_SleepModeConfigType).
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:	Selects the shutdown target.	

EcuM624: Parameter mode of the function EcuM_SelectShutdownTarget: The selected shutdown target. Only the following subset of the EcuM_StateType value range is accepted:

- `ECUM_STATE_SLEEP`
- `ECUM_STATE_RESET`
- `ECUM_STATE_OFF`

All other values will be rejected and result in a development error message `ECUM_E_STATE_PAR_OUT_OF_RANGE` must be thrown.

EcuM2185: The parameter mode of the function EcuM_SelectShutdownTarget shall be the identifier of a sleep mode. The mode parameter shall only be used if the target parameter equals `ECUM_STATE_SLEEP`. In all other cases, it shall be ignored. Only sleep modes that are defined at configuration time and are stored in the EcuMSleepMode container are allowed as parameters.

EcuM2585: An implementation of this service should not initiate any setup activities but only store the value for later use in the `SHUTDOWN` state.

EcuM2228: An implementation must preload the TTII divisor counter variable with the preload value defined in the *`ECUM_TTII_DIVISOR_LIST`*.
The service is intended for implementing AUTOSAR ports.

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

8.3.3.12 EcuM_GetShutdownTarget

EcuM2824:

Service name:	EcuM_GetShutdownTarget	

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

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	shutdownTarget	One of these values is returned: • ECUM_STATE_SLEEP • ECUM_STATE_RESET • ECUM_STATE_OFF
	sleepMode	If the return parameter is ECUM_STATE_SLEEP, this out parameter tells which of the configured sleep 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:	This function returns the selected shutdown target as set by EcuM_SelectShutdownTarget	

EcuM2788: Parameter sleepMode of the function `EcuM_GetShutdownTarget`: An implementation shall cope with NULL pointers by simply ignoring the out parameter in all cases. An implementation may assert the `ECUM_E_NULL_POINTER` development error.

8.3.3.13 EcuM_GetLastShutdownTarget

EcuM2825:

Service name:	EcuM_GetLastShutdownTarget	

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

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	shutdownTarget	One of these values is returned: • ECUM_STATE_SLEEP • ECUM_STATE_RESET • ECUM_STATE_OFF
	sleepMode	This parameter tells which of the configured sleep 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:	This service returns the shutdown target of the previous shutdown process.	

EcuM2336: Parameter sleepMode of the function EcuM_GetLastShutdownTarget: If the return parameter is `ECUM_STATE_SLEEP`, this out parameter tells which of the configured sleep modes was actually chosen.

EcuM2337: Parameter sleepMode of the function `EcuM_GetLastShutdownTarget`: An implementation shall cope with NULL pointers by simply ignoring the out parameter in all cases. An implementation may assert the `ECUM_E_NULL_POINTER` development error.

EcuM2156: The return value describes the ECU state from which the last wake up or power up occurred. This function shall return always the same value until the next shutdown.

EcuM2157: This function is intended for primary use in STARTUP or RUN state. Reasonable use cases exist there. To simplify implementation, it is acceptable if the value is set in late shutdown phase for use during the next startup. If so, implementation specific limitations must be clearly documented.

8.3.4 Wake up Handling

8.3.4.1 EcuM_GetPendingWakeupEvents

EcuM2827:

Service name:	EcuM_GetPendingWakeupEvents	

Syntax:	`EcuM_WakeupSourceType EcuM_GetPendingWakeupEvents(`
	
	`)`	

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

EcuM1156: Return code of the function `EcuM_GetPendingWakeupEvents`: Returns wake up events which have been set but not yet validated.

EcuM2172: The service must be callable from interrupt context, from OS context and an OS-free context.

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

8.3.4.2 EcuM_ClearWakeupEvent

EcuM2828:

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.	

EcuM2683: Clears all pending events passed in the in parameters from the internal variable (NAND-operation).

EcuM2807: The function must be callable from interrupt context, from OS context and an OS-free context.

8.3.4.3 EcuM_GetValidatedWakeupEvents

EcuM2830:

Service name:	EcuM_GetValidatedWakeupEvents	

Syntax:	`EcuM_WakeupSourceType EcuM_GetValidatedWakeupEvents(`
	
	`)`	

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

EcuM2533: Return code of `EcuM_GetValidatedWakeupEvents`: Returns the value from the internal variable.

EcuM2532: The service must be callable from interrupt context, from OS context and an OS-free context.

8.3.4.4 EcuM_GetExpiredWakeupEvents

EcuM2831:

Service name:	EcuM_GetExpiredWakeupEvents	

Syntax:	`EcuM_WakeupSourceType EcuM_GetExpiredWakeupEvents(`
	
	`)`	

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

EcuM2589: The service must be callable from interrupt context, from OS context and an OS-free context.

8.3.4.5 EcuM_GetStatusOfWakeupSource

EcuM2832:

Service name:	EcuM_GetStatusOfWakeupSource	

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

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

EcuM2754: When the EcuM_GetStatusOfWakeupSource service is called and parameter “sources” equals 0, then this service shall return `ECUM_WKSTATUS_NONE`. If parameter “sources” equals `ECUM_WKSOURCE_ALL_SOURCES`, then this service shall return the sum status of all configured wake up sources.

EcuM2864: If parameter “sources” contains an unknown (unconfigured) wake up source and is not `ECUM_WKSOURCE_ALL_SOURCES`, then the sum status of all known sources listed in parameter “sources” shall be returned. If Development Error Reporting is turned on, the service shall send the `ECUM_E_UNKNOWN_WAKEUP_SOURCE` error message to DET.

8.3.5 Miscellaneous

8.3.5.1 EcuM_SelectApplicationMode

EcuM2833:

Service name:	EcuM_SelectApplicationMode	
Syntax:	`Std_ReturnType EcuM_SelectApplicationMode( EcuM_AppModeType appMode )`	
Service ID[hex]:	0x0f	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	appMode	Application mode taken for next OS start
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The new application mode was accepted by EcuM E_NOT_OK: The new application mode was not accepted by EcuM
Description:	The implementation should store the application mode preferably in a non-initialized area of RAM. The service is intended for implementing AUTOSAR ports.	

EcuM2081: Parameter appMode of the function `EcuM_SelectApplicationMode`: The application mode taken for next OS start. The type is defined by the operating system.

8.3.5.2 EcuM_GetApplicationMode

EcuM2834:

Service name:	EcuM_GetApplicationMode	
Syntax:	`Std_ReturnType EcuM_GetApplicationMode( EcuM_AppModeType* appMode )`	
Service ID[hex]:	0x11	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	appMode	The currently selected application mode, see also EcuM_SelectApplicationMode
Return value:	Std_ReturnType	E_OK: The service always succeeds
Description:	The service is intended for implementing AUTOSAR ports.	

8.3.5.3 EcuM_SelectBootTarget

EcuM2835:

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:	Selects a boot target	

EcuM2247: The service must store the selected target in a way which is compatible with the boot loader. This may mean format AND location. The service is intended for implementing AUTOSAR ports.

Caveat of the function `EcuM_SelectBootTarget`: This service may be dependent on the boot loader used. This service is only intended for use by SW-C's related to diagnostics (boot management).

8.3.5.4 EcuM_GetBootTarget

EcuM2836:

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:	see EcuM_SelectBootTarget. The service is intended for implementing AUTOSAR ports.	

8.4 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.

8.4.1 EcuM_MainFunction

EcuM2837:

Service name:	EcuM_MainFunction

Syntax:	`void EcuM_MainFunction(`
	
	`)`

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

EcuM2594: This service must be called on a periodic basis from an adequate BSW task (i.e. a task under control of the BSW scheduler).
To determine the period, the system designer should consider the following timings:

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

EcuM2656: The service shall not be called from tasks which may invoke runnable entities.

8.5 Callback Definitions

8.5.1 Callbacks from NVRAM Manager

8.5.1.1 EcuM_CB_NfyNvMJobEnd

EcuM2839:

Service name:	EcuM_CB_NfyNvMJobEnd	

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

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

Configuration of `EcuM_CB_NfyNvMJobEnd`: NVRAM manager must be configured to call this callback as a multiple block job end notification. See [11] for details.

8.5.2 Callbacks from Wake up Sources

8.5.2.1 EcuM_CheckWakeup

See *8.6.6.2* EcuM_CheckWakeup for a description of the service.

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

8.5.2.2 EcuM_SetWakeupEvent

EcuM2826:

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.

EcuM1117: Takes the value and stores it in an internal variable (OR-operation).

EcuM2707: The service must start the wake up validation timeout timer according to chapter *7.8.4* Wake up Validation Timeout*.*

EcuM2867: If Development Error Reporting is turned on and parameter "sources" contains an unknown (unconfigured) wake up source, the service shall ignore the call and send the `ECUM_E_UNKNOWN_WAKEUP_SOURCE` error message to DET.

EcuM2171: The function must be callable from interrupt context, from OS context and an OS-free context.

8.5.2.3 EcuM_ValidateWakeupEvent

EcuM2829:

Service name:	EcuM_ValidateWakeupEvent	

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

Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	sources	Events to be 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 to wait for validation of the wakeup event. The validation is carried out with a call to this API service.	

EcuM2344: The validation shall be valid when ANDing the parameter `events` with the internal variable of pending wake up events results in a value other than null.

EcuM2645: The service shall invoke `ComM_EcuM_WakeUpIndication` of the Communication Manager for each wake up event if the EcuMComMChannelRef parameter in the EcuMWakeupSource configuration container for the corresponding wake up source is configured.

EcuM2868: If Development Error Reporting is turned on and parameter "sources" contains an unknown (unconfigured) wake up source, the service shall ignore the call and send the `ECUM_E_UNKNOWN_WAKEUP_SOURCE` error message to DET.

EcuM2345: The function must be callable from interrupt context, from OS context, and an OS-free context.

EcuM2790: The service shall return without effect for all sources except communication channels when called while ECU State Manager module is NOT in one of the states: `SHUTDOWN`, `SLEEP`, `WAKEUP I`, `WAKEUP VALIDATION`, and `STARTUP`.

EcuM2791: The service shall have full effect in any state for those sources which correspond to a communication channel (see **EcuM2645**), if RUN has not yet been requested for this channel.

8.5.2.4 EcuM_StartCheckWakeup

EcuM3200:

Service name:	EcuM_StartCheckWakeup	

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

Service ID[hex]:	0x1c	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	WakeupSource	--
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 is started. If EcuMCheckWakeupTimeout <= 0 the API call is ignored by the EcuM.	

8.5.2.5 EcuM_EndCheckWakeup

EcuM3201:

Service name:	EcuM_EndCheckWakeup	

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

Service ID[hex]:	0x1d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	WakeupSource	--
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. If CheckWakeupTimer is canceled the check of the this wakeup source is finished.	

8.6 Callout Definitions

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

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

8.6.1 Generic Callouts

8.6.1.1 EcuM_ErrorHook

EcuM2904

Service name:	EcuM_ErrorHook	

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

Service ID[hex]:	0x00	
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" occure. In this situation it is not possible to continue processing and the ECU must be stopped. The integrator may choose the modality how the ECU is stopped, i.e. reset, halt, restart, safe state etc.	

Invocation of EcuM_ErrorHook: in all states

Class of EcuM_ErrorHook: Mandatory

EcuM_ErrorHook is integration code and the integrator is free to define additional individual error codes to be passed as the `reason` parameter. These error codes shall not conflict with the development and production error codes as defined in Table 1 and Table 5 nor with the standard error codes used in this chapter, i.e. E_OK, E_NOT_OK, etc.

8.6.2 Callouts from STARTUP

8.6.2.1 EcuM_AL_DriverInitZero

Service name:	EcuM_AL_DriverInitZero
Syntax:	`void EcuM_AL_DriverInitZero(`

	)
Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callout shall provide driver initialization and other hardware-related startup activities for loading the post-build configuration data. Beware: Here only pre-compile and link-time configurable modules may be used.

Invocation of EcuM_AL_DriverInitZero: Early in STARTUP I

The ECU State Manager module configuration tool shall generate a default implementation of the EcuM_AL_DriverInitZero callout from the sequence of modules defined in the EcuMDriverInitListZero configuration container. See EcuM2559 and EcuM2730.

8.6.2.2 EcuM_DeterminePbConfiguration

Service name:	EcuM_DeterminePbConfiguration	

Syntax:	`EcuM_ConfigType* EcuM_DeterminePbConfiguration(`
	
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_ConfigType*	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
Description:	This callout should evaluate some condition, like port pin or NVRAM value, to determine which post-build configuration shall be used in the remainder of the startup process. It shall load this configuration data into a piece of memory that is accessible by all BSW modules and shall return a pointer to the EcuM post-build configuration as a base for all BSW module post-build configrations.	

Invocation of EcuM_DeterminePbConfiguration: Early in STARTUP I

Content is manually written.

8.6.2.3 EcuM_AL_DriverInitOne

Service name:	EcuM_AL_DriverInitOne

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

Service ID[hex]:	0x00

Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout shall provide driver initialization and other hardware-related startup activities in case of a power on reset.	

Invocation of EcuM_AL_DriverInitOne: In STARTUP I

The ECU State Manager module configuration tool shall generate a default implementation of the EcuM_AL_DriverInitOne callout from the sequence of modules defined in the EcuMDriverInitListOne configuration container. See EcuM2559 and EcuM2730.

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

8.6.2.4 EcuM_AL_DriverInitTwo

Service name:	EcuM_AL_DriverInitTwo	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout shall provide driver initialization of drivers which need OS and do not need to wait for the NvM_ReadAll job to finish.	

Invocation of EcuM_AL_DriverInitTwo: In STARTUP II

The ECU State Manager module configuration tool shall generate a default implementation of the EcuM_AL_DriverInitTwo callout from the sequence of modules defined in the EcuMDriverInitListTwo configuration container. See EcuM2559 and EcuM2730.

8.6.2.5 EcuM_AL_DriverInitThree

Service name:	EcuM_AL_DriverInitThree
Syntax:	`void EcuM_AL_DriverInitThree(`

	`const EcuM_ConfigType* ConfigPtr` `)`	
Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout shall provide driver initialization of drivers which need OS and need to wait for the NvM_ReadAll job to finish.	

Invocation of EcuM_AL_DriverInitThree: In STARTUP II

The ECU State Manager module configuration tool shall generate a default implementation of the EcuM_AL_DriverInitThree callout from the sequence of modules defined in the EcuMDriverInitListThree configuration container. See EcuM2559 and EcuM2730.

8.6.2.6 EcuM_OnRTEStartup

Service name:	EcuM_OnRTEStartup

Syntax:	`void EcuM_OnRTEStartup(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	--

Invocation of EcuM_OnRTEStartup: Just before calling RTE_Start

8.6.3 Callouts from RUN State

8.6.3.1 EcuM_OnEnterRun

Service name:	EcuM_OnEnterRun

Syntax:	`void EcuM_OnEnterRun(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None

Parameters (out):	None
Return value:	None
Description:	On entry of RUN state is very similar to "just after startup". This call allows the system designer to notify that RUN state has been reached.

Invocation of EcuM_OnEnterRun: On entry of RUN state.

8.6.3.2 EcuM_OnExitRun

Service name:	EcuM_OnExitRun

Syntax:	`void EcuM_OnExitRun(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the APP RUN state is about to be left.

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

8.6.3.3 EcuM_OnExitPostRun

Service name:	EcuM_OnExitPostRun

Syntax:	`void EcuM_OnExitPostRun(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the APP POST RUN state is about to be left.

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

8.6.4 Callouts from SHUTDOWN

8.6.4.1 EcuM_OnPrepShutdown

Service name:	EcuM_OnPrepShutdown

Syntax:	`void EcuM_OnPrepShutdown(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the PREP SHUTDOWN state is about to be entered.

Invocation of EcuM_OnPrepShutdown: On entry of PREP SHUTDOWN

8.6.4.2 EcuM_OnGoSleep

Service name:	EcuM_OnGoSleep

Syntax:	`void EcuM_OnGoSleep(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the GO SLEEP state is about to be entered.

Invocation of EcuM_OnGoSleep: On entry of GO SLEEP

8.6.4.3 EcuM_OnGoOffOne

Service name:	EcuM_OnGoOffOne

Syntax:	`void EcuM_OnGoOffOne(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the GO OFF I state is about to be entered.

Invocation of EcuM_OnGoOffOne: On entry of GO OFF I

8.6.4.4 EcuM_OnGoOffTwo

Service name:	EcuM_OnGoOffTwo

Syntax:	`void EcuM_OnGoOffTwo(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This call allows the system designer to notify that the GO OFF II state is about to be entered.

Invocation of EcuM_OnGoOffTwo: On entry of GO OFF II

8.6.4.5 EcuM_EnableWakeupSources

Service name:	EcuM_EnableWakeupSources	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Created to fix wakeup sequences	

EcuM2546: The ECU State Manager module needs to derive the wake up sources to be enabled for the from configuration information.

Invocation of EcuM_EnableWakeupSources: From GOSLEEP II

8.6.4.6 EcuM_GenerateRamHash

Service name:	EcuM_GenerateRamHash

Syntax:	`void EcuM_GenerateRamHash(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	see EcuM_CheckRamHash

Invocation of EcuM_GenerateRamHash: Just before putting the ECU physically to sleep

8.6.4.7 EcuM_AL_SwitchOff

Service name:	EcuM_AL_SwitchOff

Syntax:	`void EcuM_AL_SwitchOff(`
	
	`)`

Service ID[hex]:	0x00
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.

Invocation of EcuM_AL_SwitchOff: Last activity in SHUTDOWN II

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

8.6.5 Callouts from WAKEUP

8.6.5.1 EcuM_CheckRamHash

Service name:	EcuM_CheckRamHash	

Syntax:	`uint8 EcuM_CheckRamHash(`
	
	`)`	

Service ID[hex]:	0x00	
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.

Invocation of EcuM_CheckRamHash: Early in WAKEUP I

8.6.5.2 EcuM_DisableWakeupSources

Service name:	EcuM_DisableWakeupSources	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The callout shall set up the given wakeup source(s) so that they are not able to wakeup the ECU.	

Invocation of EcuM_DisableWakeupSources: In WAKEUP I

8.6.5.3 EcuM_AL_DriverRestart

Service name:	EcuM_AL_DriverRestart

Syntax:	`void EcuM_AL_DriverRestart(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callout shall provide driver initialization and other hardware-related startup activities in the wakeup case.

Invocation of EcuM_EcuM_AL_DriverRestart: In WAKEUP I

The ECU State 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 EcuM2561, EcuM2559 and EcuM2730.

8.6.5.4 EcuM_StartWakeupSources

Service name:	EcuM_StartWakeupSources	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The callout shall start the given wakeup source(s) so that they are ready to perform wakeup validation.	

Invocation of EcuM_StartWakeupSources: In WAKEUP VALIDATION

Hint: In `EcuM_StartWakeupSource()` the ICU notification can be disabled/enabled in oder to reduce redundant wake up interrupts during wake up validation.

8.6.5.5 EcuM_CheckValidation

Service name:	EcuM_CheckValidation	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout is called by the EcuM to validate a wakeup source. If a valid wakeup has been detected, it shall be reported to EcuM via EcuM_ValidateWakeupEvent().	

Invocation of EcuM_CheckValidation: In WAKEUP VALIDATION

8.6.5.6 EcuM_StopWakeupSources

Service name:	EcuM_StopWakeupSources	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters	None	

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

Invocation of EcuM_StopWakeupSources: In WAKEUP VALIDATION

Hint: In `EcuM_StopWakeupSource()` the ICU notification can be disabled/enabled in oder to reduce redundant wake up interrupts during wake up validation.

8.6.5.7 EcuM_OnWakeupReaction

Service name:	EcuM_OnWakeupReaction	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wact	The wakeup reaction computed by ECU State Manager
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupReactionType	All values: The desired wakeup reaction.
Description:	This callout gives the system designer the chance to intercept the automatic boot behavior and to override the wakeup reaction computed from wakeup source.	

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

8.6.6 Callouts from SLEEP State

8.6.6.1 EcuM_SleepActivity

Service name:	EcuM_SleepActivity

Syntax:	`void EcuM_SleepActivity(`
	
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This callout is invoked periodically in all reduced clock sleep modes. It is explicitely allowed to poll wakeup sources from this callout and to call wakeup notification functions to indicate the end of the sleep state to the ECU State Manager.

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

8.6.6.2 EcuM_CheckWakeup

Service name:	EcuM_CheckWakeup	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	wakeupSource	--
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout is called by the EcuM to poll a wakeup source. It shall also be called by the ISR of a wakeup source to set up the PLL and check other wakeup sources that may be connected to the same interrupt.	

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

8.7 Expected Interfaces

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

8.7.1 Mandatory Interfaces

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

EcuM2858:

API function	***Description***
BswM_Deinit	Deinitializes the BSW Mode Manager.
BswM_Init	Initializes the BSW Mode Manager.
CanSM_StartWakeupSource	This function shall be called by EcuM when a wakeup source shall be started.
CanSM_StopWakeupSource	This function shall be called by EcuM when a wakeup source shall be stopped.
ComM_DeInit	De-initializes (terminates) the AUTOSAR Communication Manager.
ComM_EcuM_RunModeIndication	Indication that ECU State Manager has entered “Run Mode” for that channel.
ComM_EcuM_WakeUpIndication	Notification of a wake up on the corresponding channel.
ComM_Init	Initializes the AUTOSAR Communication Manager and restarts the internal state machines.
GetResource	--
Mcu_GetResetReason	The service reads the reset type from the hardware, if supported.
Mcu_Init	This service initializes the MCU driver.
Mcu_PerformReset	The service performs a microcontroller reset.
Mcu_SetMode	This service activates the MCU power modes.
ReleaseResource	--
Rte_Start	--
Rte_Stop	--
Rte_Switch_currentMode_currentMode	--
SchM_Init	Function for initialization of the SchM module.
ShutdownOS	--
StartOS	--

Table 6 - Mandatory interfaces

8.7.2 Optional Interfaces

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

EcuM2859:

API function	***Description***
Adc_Init	Initializes the ADC hardware units and driver.
CanIf_Init	CANIF001: This service initializes internal and external interfaces of the

	CAN Interface for the further processing. All underlying CAN controllers and CAN transceivers still remain not operational. This service is called only ECU State Manager (EcuM). If a NULL pointer is passed for *ConfigPtr to this function the default configuration shall be used. In case only one configuration setup is used, a NULL pointer is sufficient to choose the one static existing configuration setup. Development errors: Invalid values of *ConfigPtr will be reported to the development error tracer (CANIF_E_PARAM_POINTER) only for post built use cases.
CanNm_Init	Initialize the complete CanNm module, i.e. all channels which are activated (see also configuration parameter CANNM_CHANNEL_ACTIVE) at configuration time are initialized. If a NULL pointer is passed as an argument to this function the default configuration shall be used. Caveats: This function has to be called after initialization of the CanIf. Configuration: Mandatory
CanSM_Init	This service initializes the CanSM module
CanTp_Init	This function initializes the CanTp module.
CanTrcv_Init	Initializes the CanTrcv module.
Can_Init	This function initializes the module.
Com_Init	This service initializes internal and external interfaces and variables of the AUTOSAR COM layer for the further processing. After calling this function the inter-ECU communication is still disabled.
Dcm_Init	Service for basic initialization of DCM module. If a NULL pointer is passed for ConfigPtr to this function the default configuration shall be used. In case only one configuration setup is used, a NULL pointer is sufficient to choose the one static existing configuration setup.
Dem_Init	Initializes this module.
Dem_PreInit	Initializes the internal states necessary to process events reported by BSW-modules
Dem_ReportErrorStatus	Reports errors to the DEM.
Dem_Shutdown	Shutdowns this module.
Det_Init	Service to initialize the Development Error Tracer.
Det_ReportError	Service to report development errors.
Ea_Init	Initializes the EEPROM abstraction module.
Fee_Init	Service to initialize the FEE module.
Fim_Init	This service initializes the FIM.
FrIf_Init	Initializes the FlexRay Interface.
FrNm_Init	Initializes the FlexRay NM and its internal state machine.
FrSm_Init	Initializes the FlexRay State Manager.
FrTp_Init	This service initializes all global variables of the FlexRay Transport Layer and sets all states to idle.
Fr_Init	Initalizes the Fr.
Gpt_Init	Initializes the hardware timer module.
Icu_Init	This function initializes the driver.
IoHwAb_Init<Init_Id>	Initializes either all the IO Hardware Abstraction software or is a part of the IO Hardware Abstraction.
IpduM_Init	Initializes the I-PDU Multiplexer.
LinIf_Init	Initializes the LIN Interface.

LinSM_init	This function initializes the LinSM. Note that in some implementations other values of the pointer than NULL may be considered faulty. Configuration dependent on Variant (see parameter)
LinTp_Init	Initializes the LIN Transport Layer.
Lin_Init	Initializes the LIN module.
Nm_Init	Initializes the NM Interface.
NvM_CancelWriteAll	Service to cancel a running NvM_WriteAll request.
NvM_Init	Service for resetting all internal variables.
NvM_ReadAll	Initiates a multi block read request.
NvM_WriteAll	Initiates a multi block write request.
PduR_Init	Initializes the PDU Router
Port_Init	Initializes the Port Driver module.
Pwm_Init	Service for PWM initialization.
SchM_Enter_EcuM	--
SchM_Exit_EcuM	--
Spi_Init	Service for SPI initialization.
WdgM_Init	Initializes the Watchdog Manager.
WdgM_SetMode	Sets the current mode of Watchdog Manager.
WdgM_UpdateAliveCounter	Gives alive indications to the Watchdog Manager.
Wdg_Init	Initializes the module.

Table 7 - Optional Interfaces

8.7.3 Configurable interfaces

There are no configurable interfaces.

8.8 API Parameter Checking

If development error detection is enabled for this module, then all services shall test input parameters and running conditions and use the following error codes in an adequate way:

- `ECUM_E_NOT_INITED`
- `ECUM_E_SERVICE_DISABLED`
- `ECUM_E_NULL_POINTER`
- `ECUM_E_INVALID_PAR`

Specific development errors are listed in the functions, where they do apply.

9 Sequence Charts

9.1 State Sequences

Sequence charts showing the behavior of the ECU State 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 3 – Startup Sequence (high level diagram)
- Figure 4 – Init Sequence I (STARTUP I)
- Figure 5 – Init Sequence II (STARTUP II)
- Figure 7 – RUN State Sequence (high level diagram)
- Figure 8 – RUN II State Sequence
- Figure 9 – RUN III State Sequence
- Figure 11 – Shutdown Sequence (high level diagram)
- Figure 12 – Deinitialization Sequence I (PREP SHUTDOWN)
- Figure 13 – Deinitialization Sequence IIa (GOSLEEP
- Figure 14 – Deinitialization Sequence IIb (GO OFF I)
- Figure 15 – Deinitialization Sequence III (GO OFF II)
- Figure 16 – Sleep Sequence (high level diagram)
- Figure 17 – Sleep Sequence I
- Figure 18 – Sleep Sequence II
- Figure 19 – Wake up Sequence (high level diagram)
- Figure 21 – Wake up Sequence I
- Figure 22 – Wake up Validation Sequence
- Figure 24 – Wake up Sequence II

9.2 Wake up Sequences

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

9.2.1 GPT Wake up Sequences

The General Purpose Timer (GPT) is one of the possible wake up sources. Usually the GPT is started before the ECU is put to sleep and the hardware timer causes an interrupt when it expires. The interrupt wakes the microcontroller, and executes the interrupt handler in the GPT module. It informs the ECU State Manager 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 souce identifier to each GPT channel. Figure 30 shows the corresponding sequence of calls.

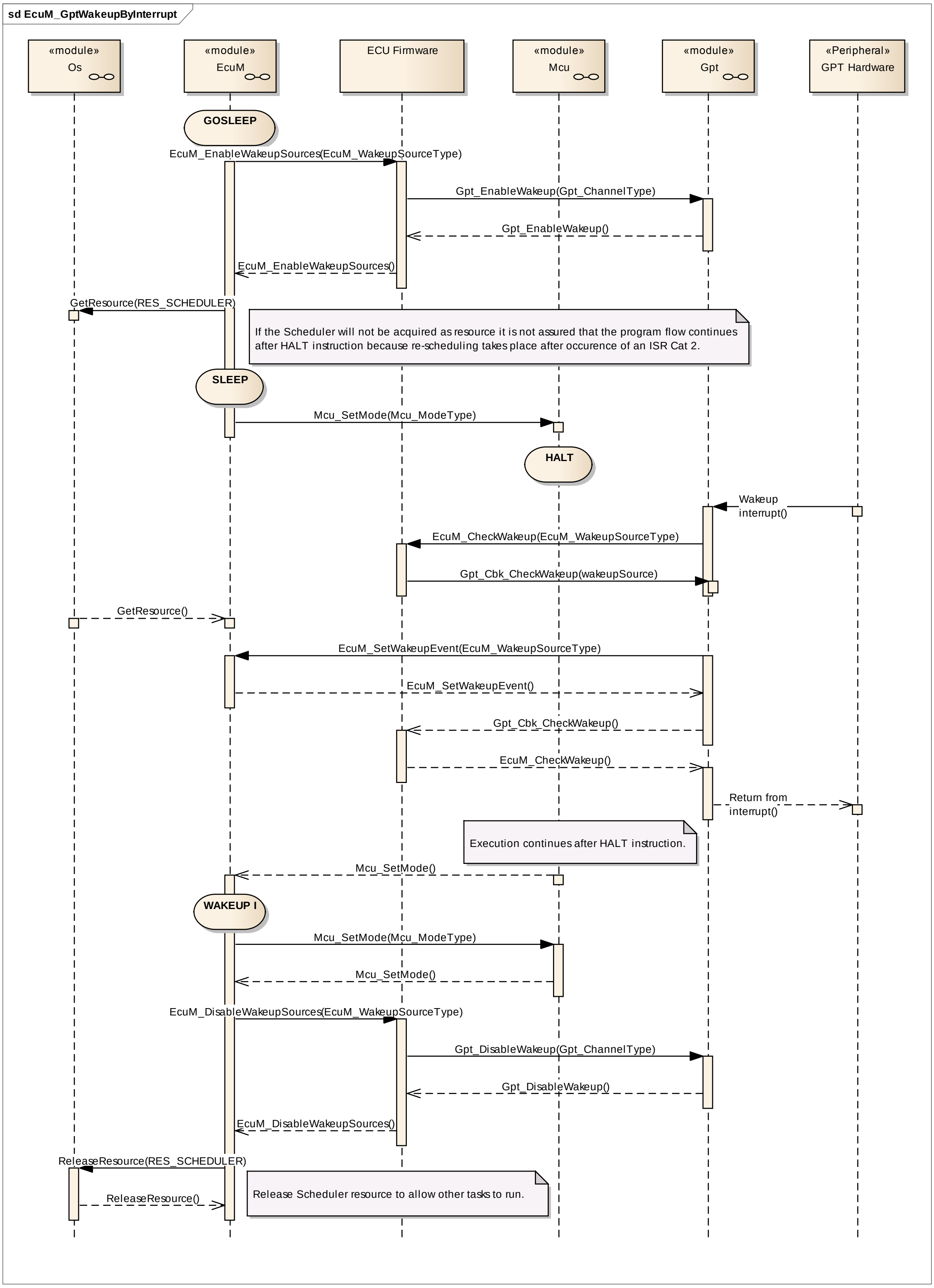

Figure 30 – GPT wake up by interrupt

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 31.

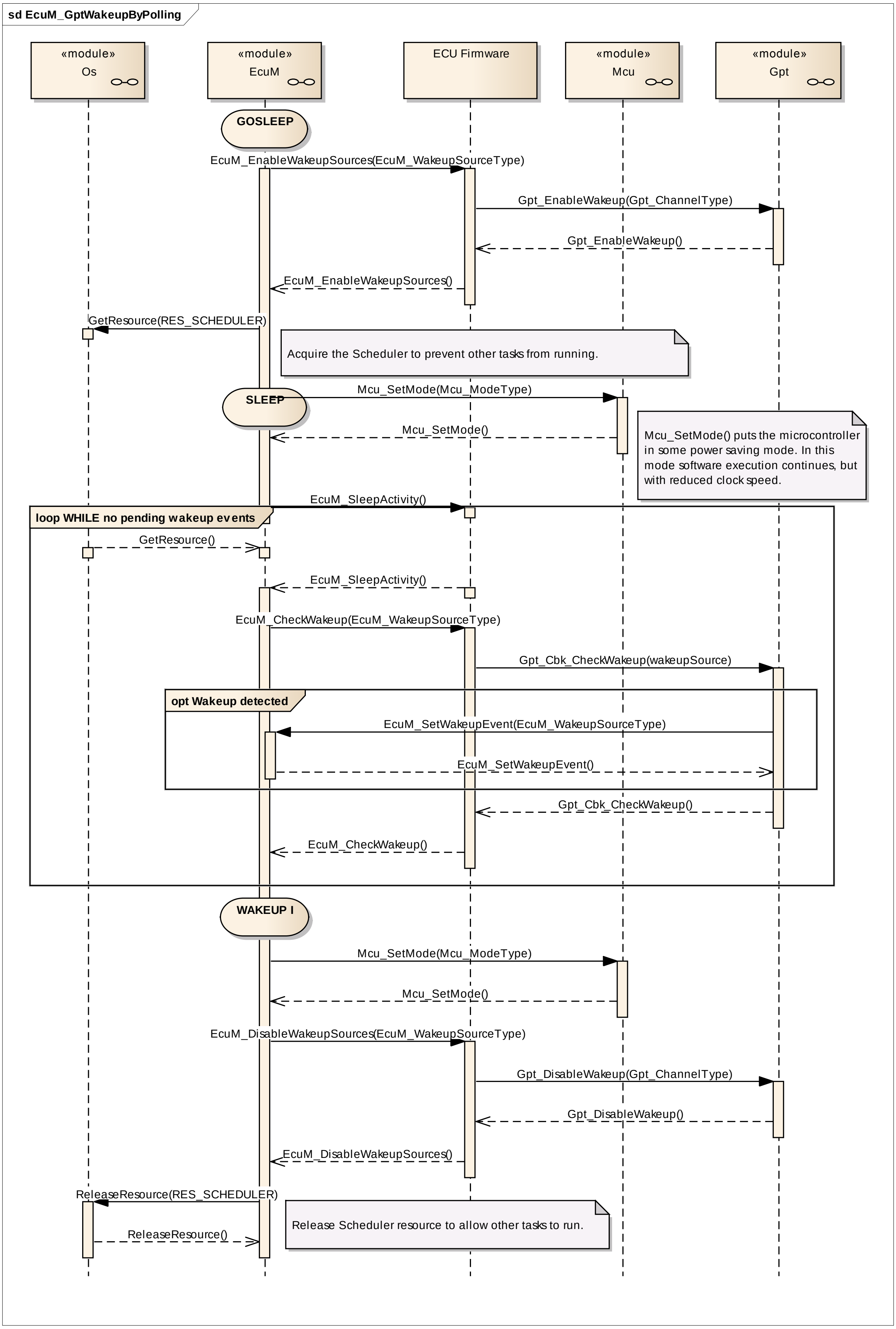

Figure 31 – GPT wake up by polling

9.2.2 ICU Wake up Sequences

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

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

Figure 32 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 18 – Sleep Sequence II.

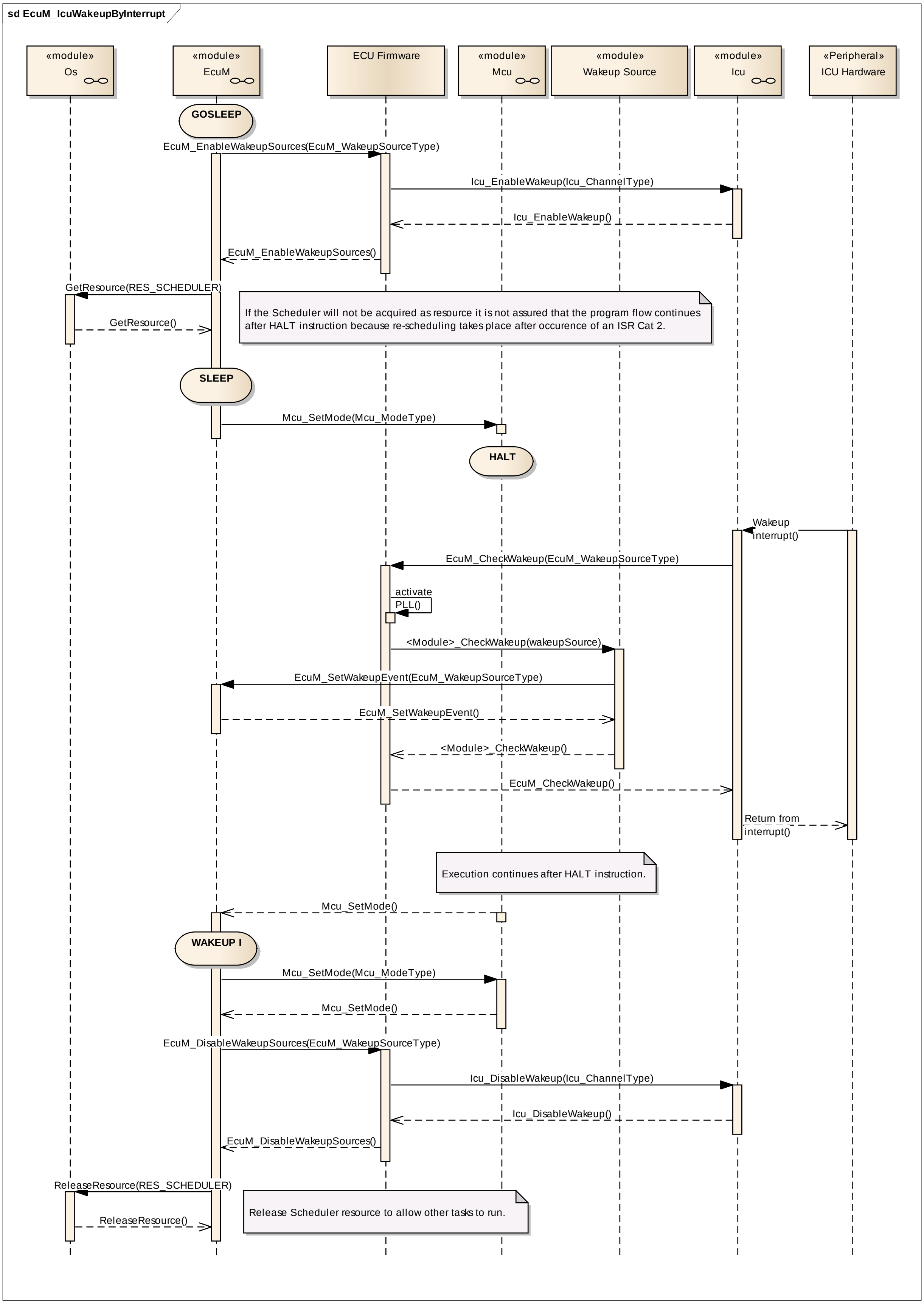

Figure 32 – ICU wake up by interrupt

9.2.3 CAN Wake up Sequences

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

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(), which could stand for a shared interrupt or just for a interrupt channel, to specific wake up sources which are passed to CanIf_CheckWakeup(). So here the parameter 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 “ECU Firmware”. Thus it is implementation specific if e.g. during EcuM_CheckWakeup() the CanIf is called to check the wake up source.

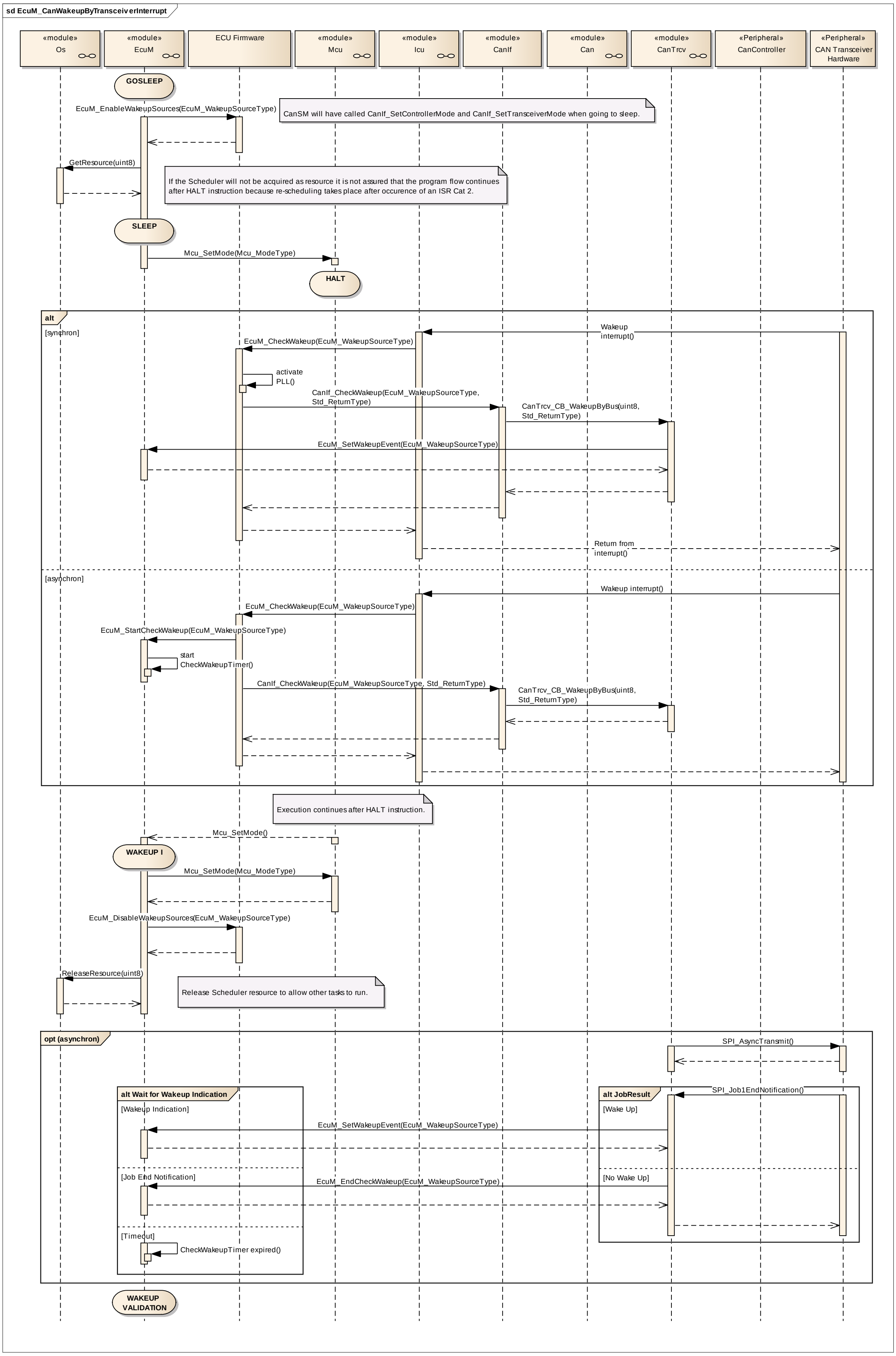
sd EcuM_CanWakeupByTransceiverInterrupt
«module» Os
«module» EcuM
ECU Firmware
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
CanSM will have called CanIf_SetControllerMode 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 occurence of an ISR Cat 2.
SLEEP
Mcu_SetMode(Mcu_ModeType)
HALT
alt
[synchron]
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
activate PLL()
CanIf_CheckWakeup(EcuM_WakeupSourceType, Std_ReturnType)
CanTrcv_CB_WakeupByBus(uint8, Std_ReturnType)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
[asynchron]
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
EcuM_StartCheckWakeup(EcuM_WakeupSourceType)
start CheckWakeupTimer()
CanIf_CheckWakeup(EcuM_WakeupSourceType, Std_ReturnType)
CanTrcv_CB_WakeupByBus(uint8, Std_ReturnType)
Execution continues after HALT instruction.
Mcu_SetMode()
WAKEUP I
Mcu_SetMode(Mcu_ModeType)
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
ReleaseResource(uint8)
Release Scheduler resource to allow other tasks to run.
opt (asynchron)
SPI_AsyncTransmit()
alt Wait for Wakeup Indication
[Wakeup Indication]
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
[Job End Notification]
EcuM_EndCheckWakeup(EcuM_WakeupSourceType)
[Timeout]
CheckWakeupTimer expired()
alt JobResult
SPI_Job1EndNotification()
[Wake Up]
[No Wake Up]
WAKEUP VALIDATION

Figure 33 – CAN transceiver wake up by interrupt

Figure 33 shows the CAN transceiver wake up via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.

Note that, for CAN the CAN Interface instead of the CAN Transceiver Driver or CAN Driver is responsible to report the wake up event to the ECU State Manager module via EcuM_SetWakeupEvent.

A CAN controller wake up by interrupt works similar to the GPT wake up. Here the interrupt handler and the CheckWakeup functionality are both encapsulated in the CAN Driver module, as shown in Figure 34.

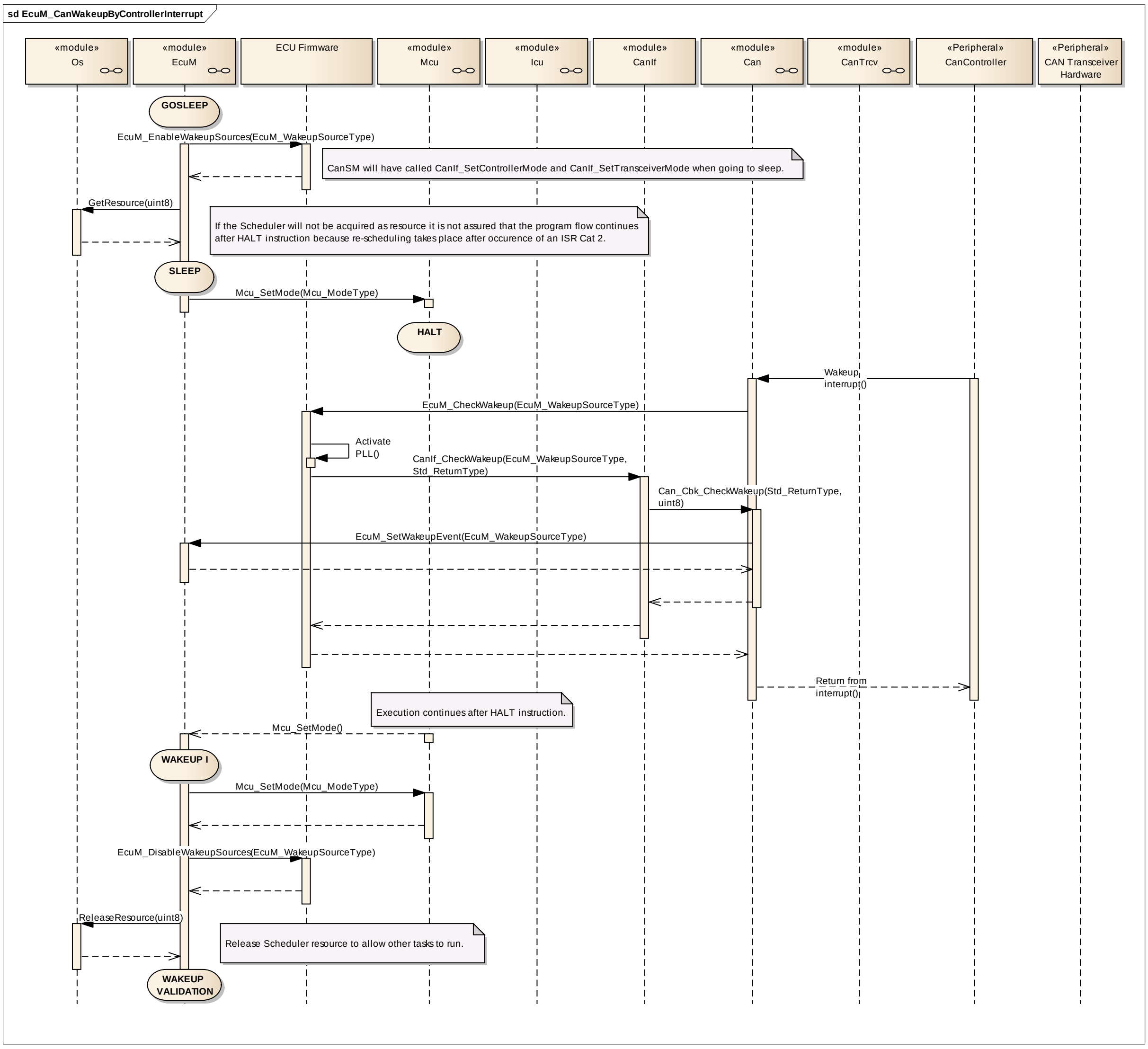

Figure 34 – CAN controller wake up by interrupt

Wake up by polling is possible both for CAN transceiver and CAN 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 35.

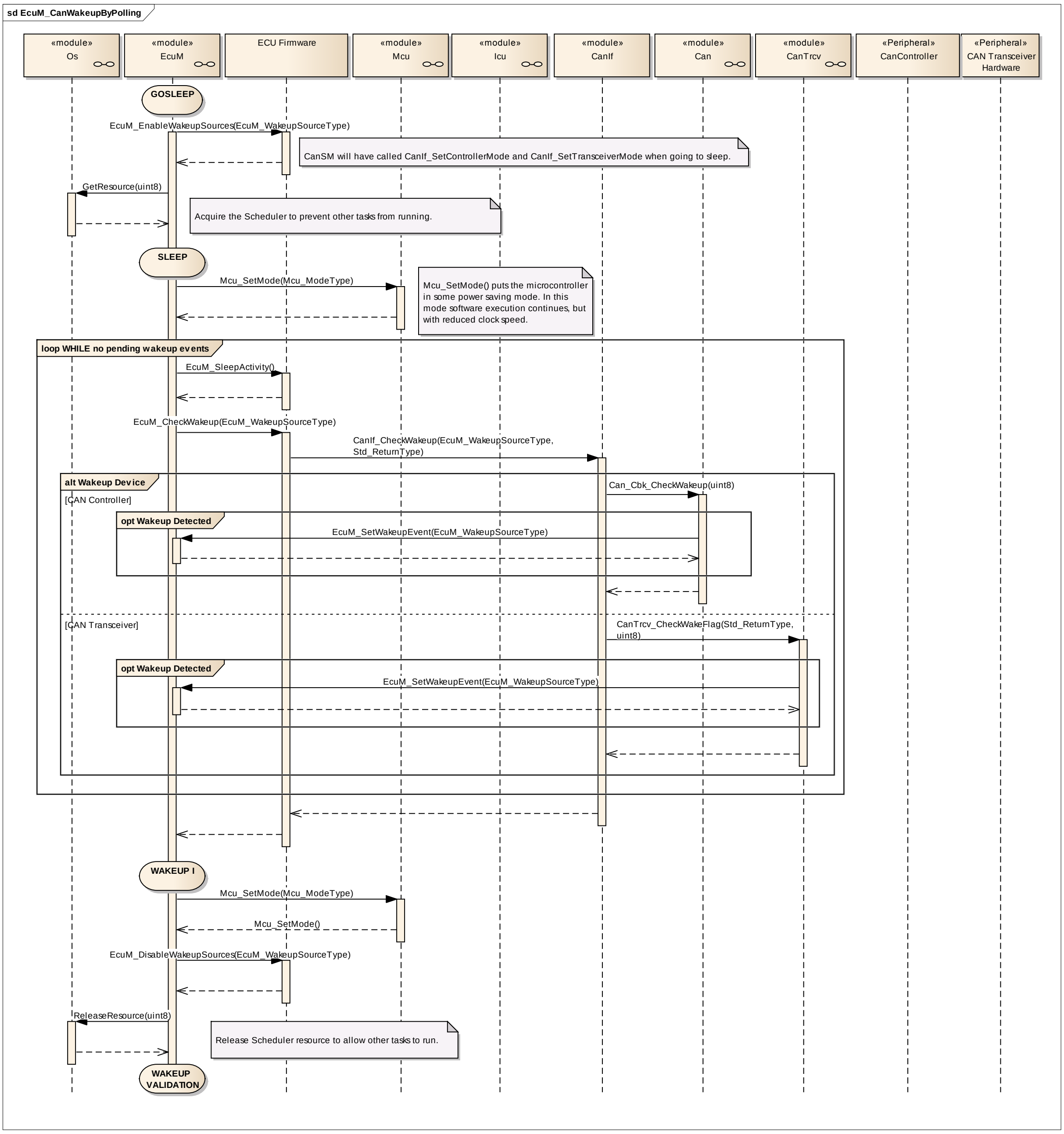

Figure 35 – CAN controller or transceiver wake up by polling

After the detection of a wake up event from the CAN transceiver or controller by either interrupt or polling, the wake up event still needs to be validated. This is done by switching on the corresponding CAN transceiver and controller in EcuM_StartWakeupSources. It depends on the used CAN transceivers and controllers, which function calls in Integrator Code EcuM_StartWakeupSource are necessary. In Figure 43 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 CAN Interface module are set to "Online", the CanIf will forward CAN messages.

The CAN interface module only 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 CAN Interface in Integrator Code EcuM_CheckValidation.
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_StopWakeupSources and go back to sleep.

The resulting sequence is shown in Figure 36.

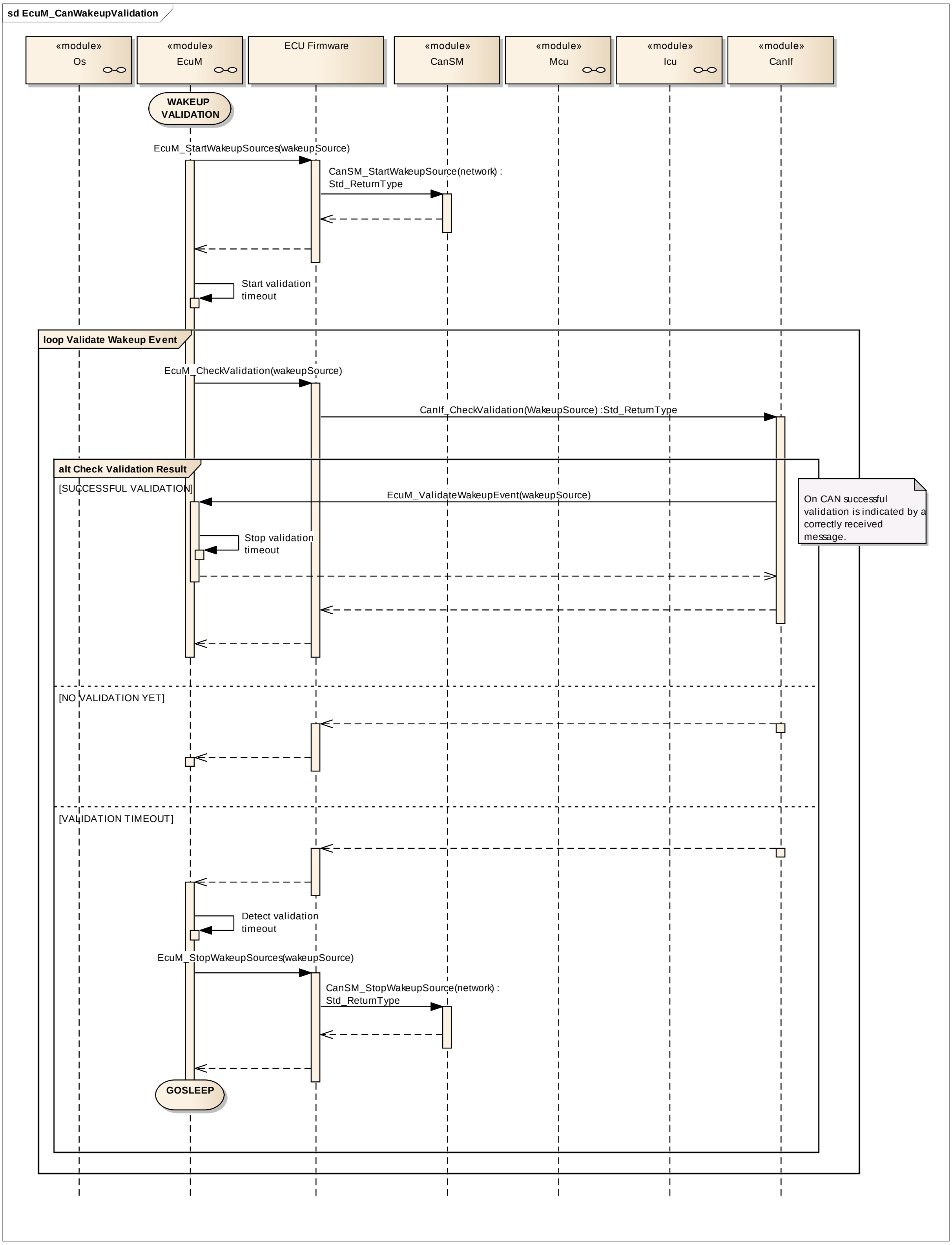

Figure 36 – CAN wake up validation

9.2.4 LIN wake up sequences

Figure 37 shows the LIN transceiver wake up via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.

Note that, for LIN the LIN Driver is always responsible to report the wake up event to the ECU State Manager module via EcuM_SetWakeupEvent.

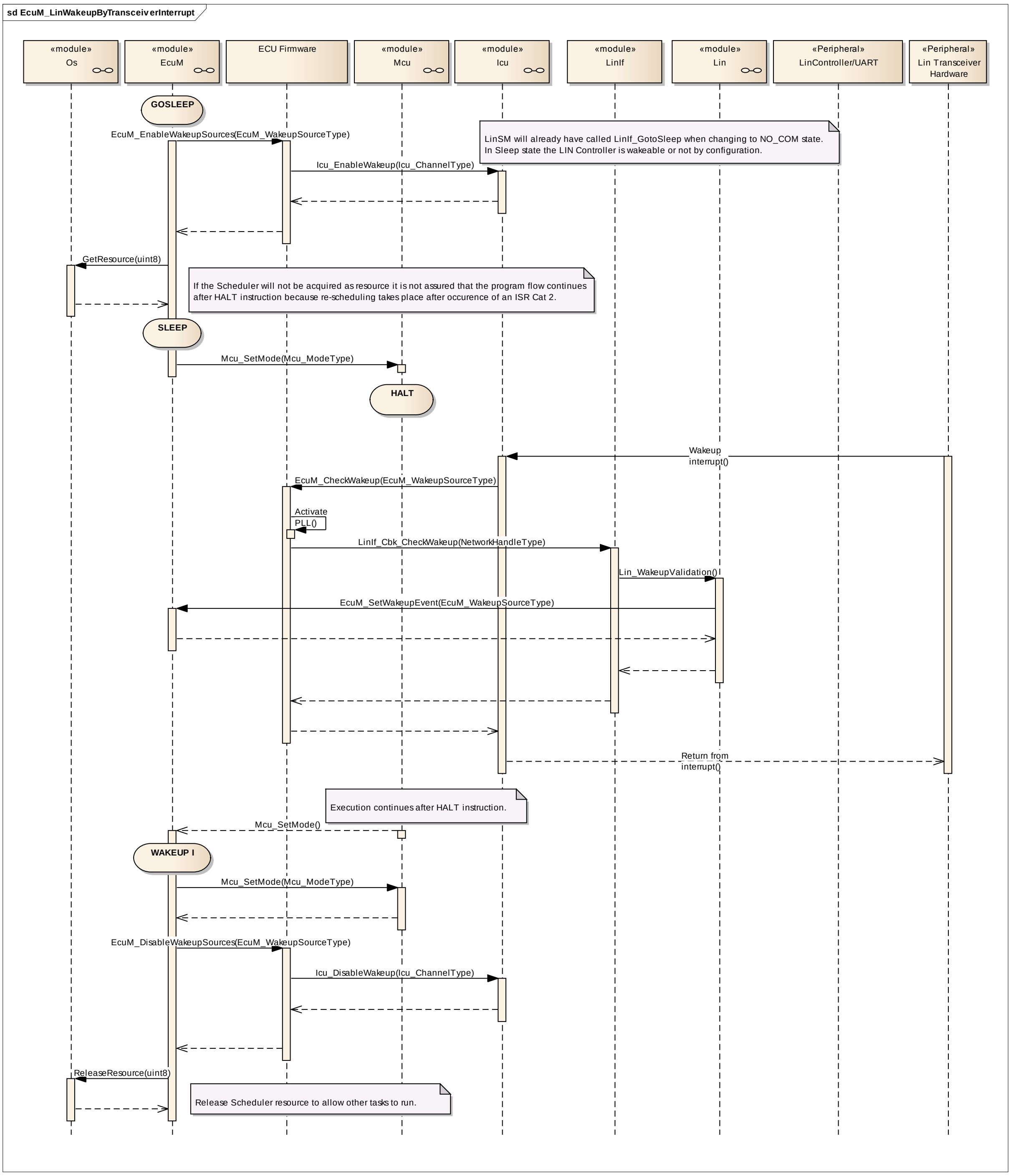

Figure 37 – LIN transceiver wake up by interrupt

As shown in Figure 38, 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.

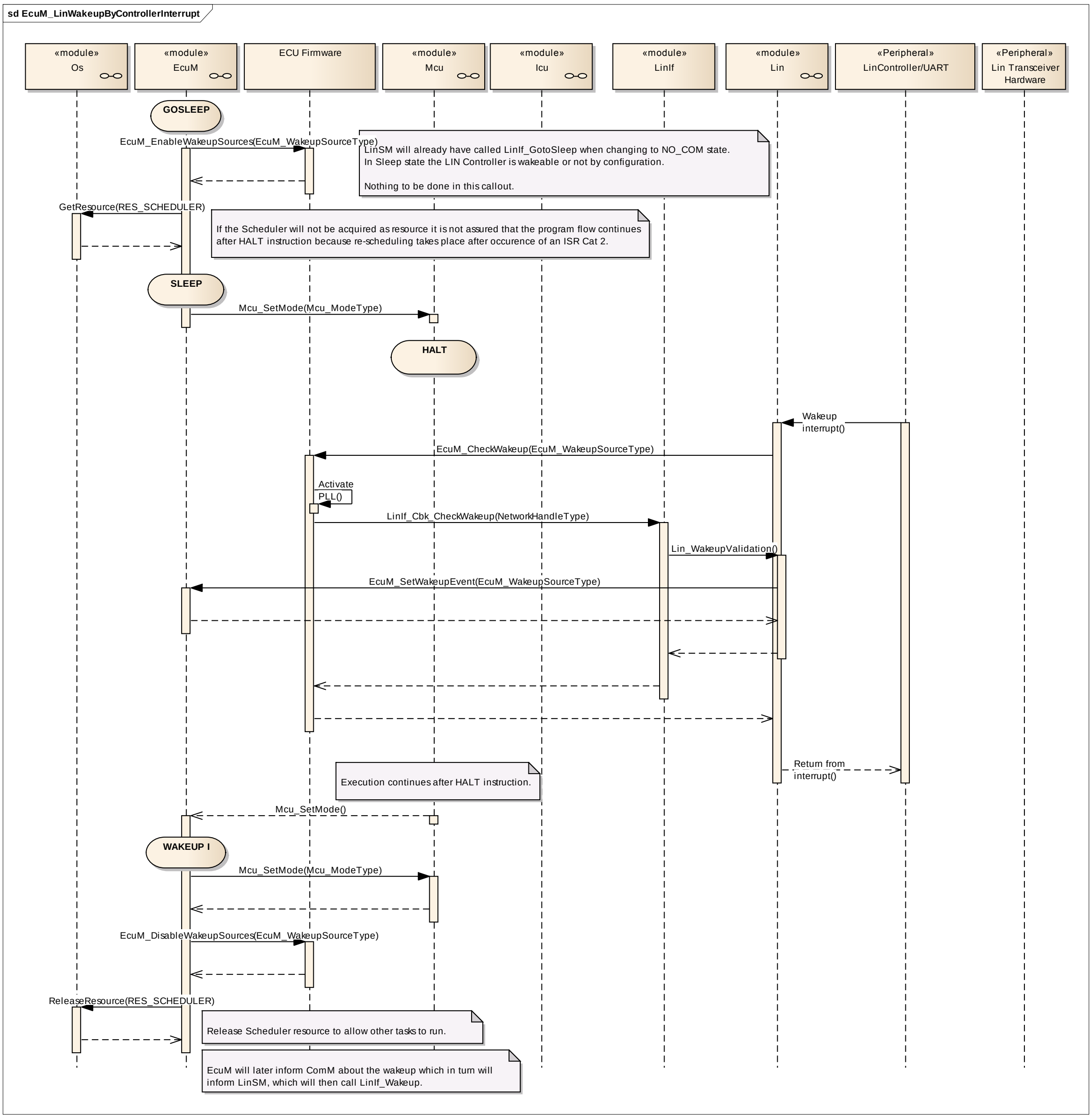

Figure 38 – LIN controller wake up by interrupt

Since there is no specific driver for the LIN transceiver, the LIN Driver is always checked for wake up events in the polling case. The sequence is shown in Figure 39.

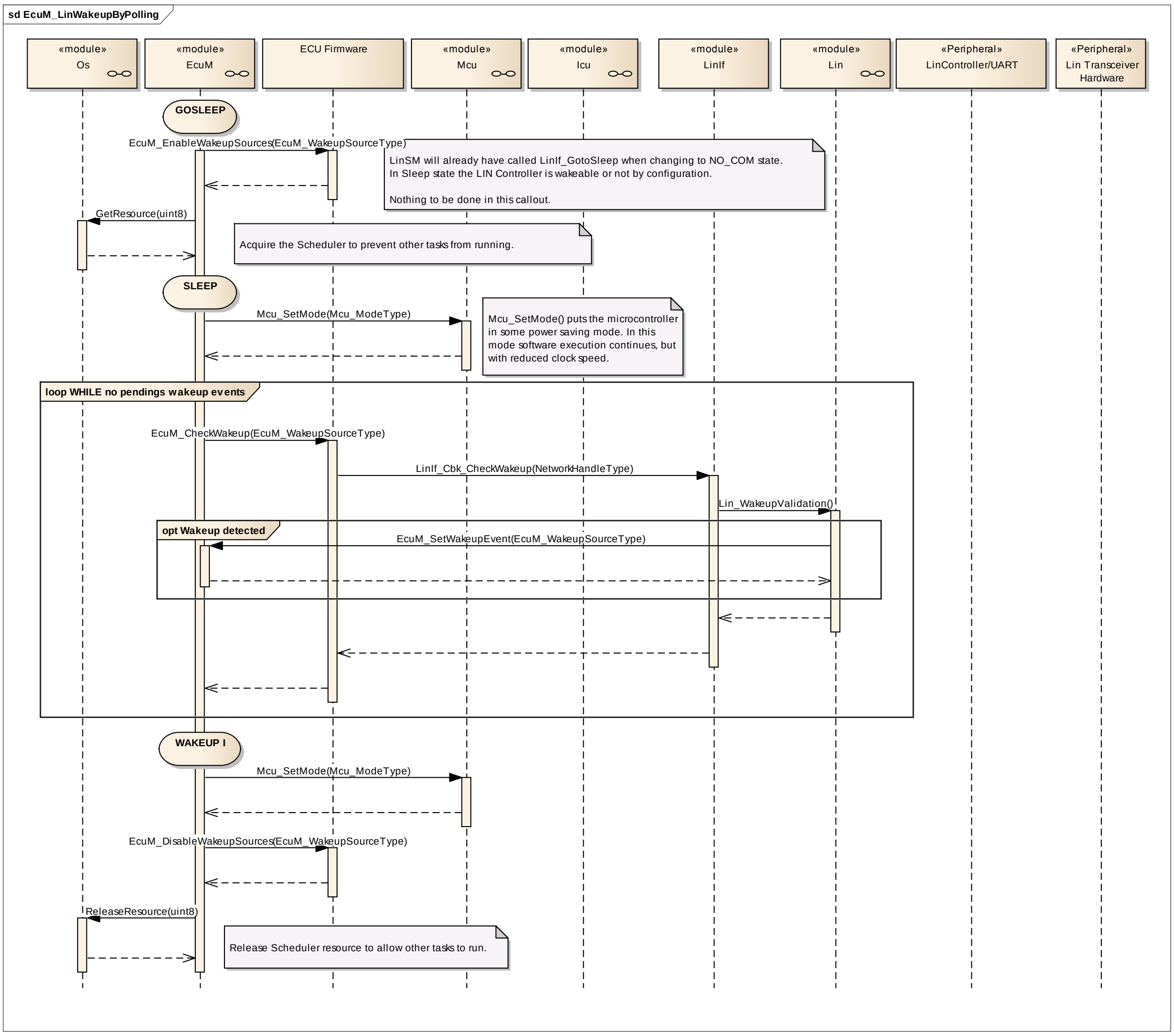

Figure 39 – LIN controller or transceiver wake up by polling

Note that, LIN does not require wake up validation.

9.2.5 FlexRay wake up sequences

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

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

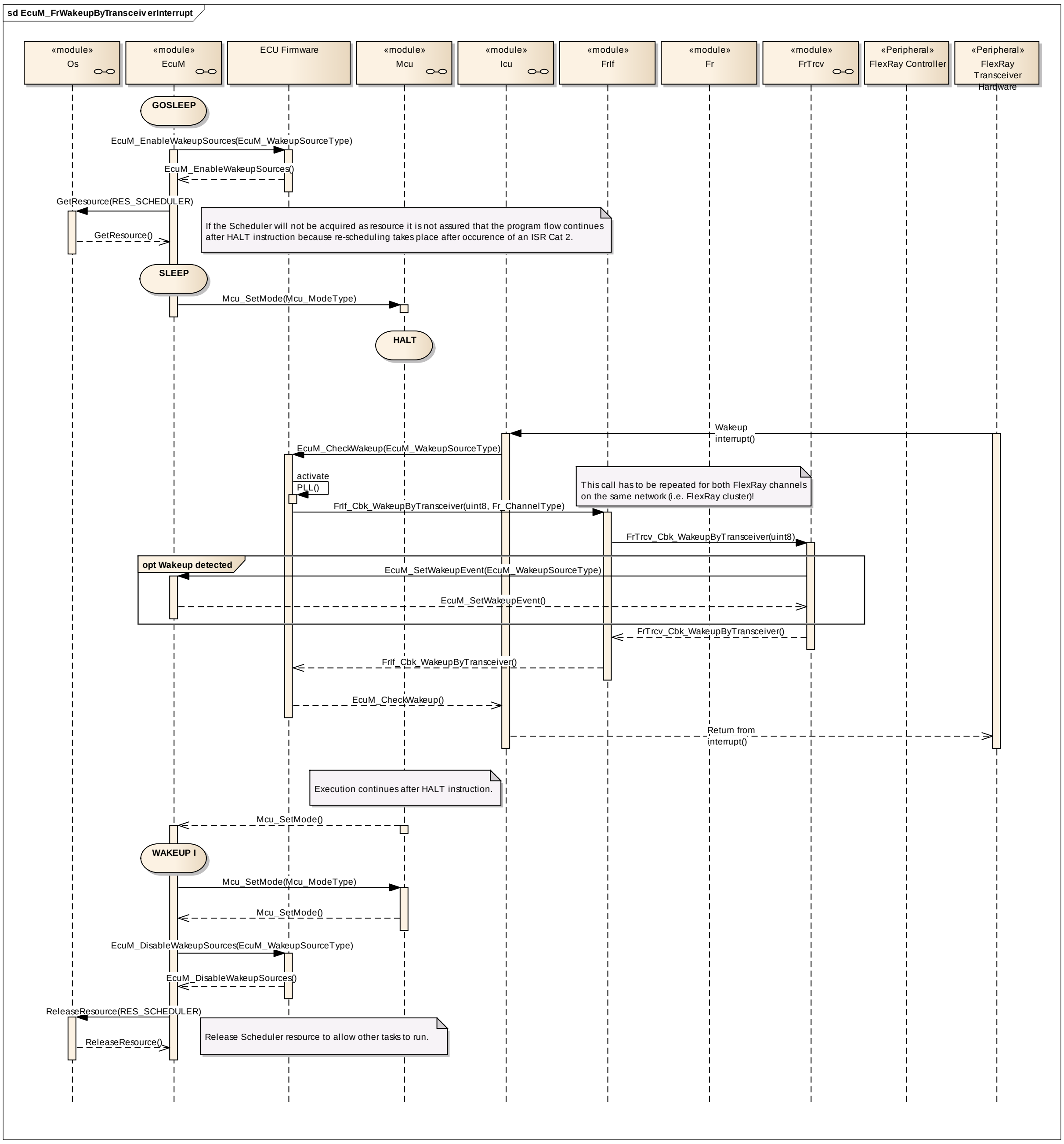

Figure 40 – FlexRay transceiver wake up by interrupt

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

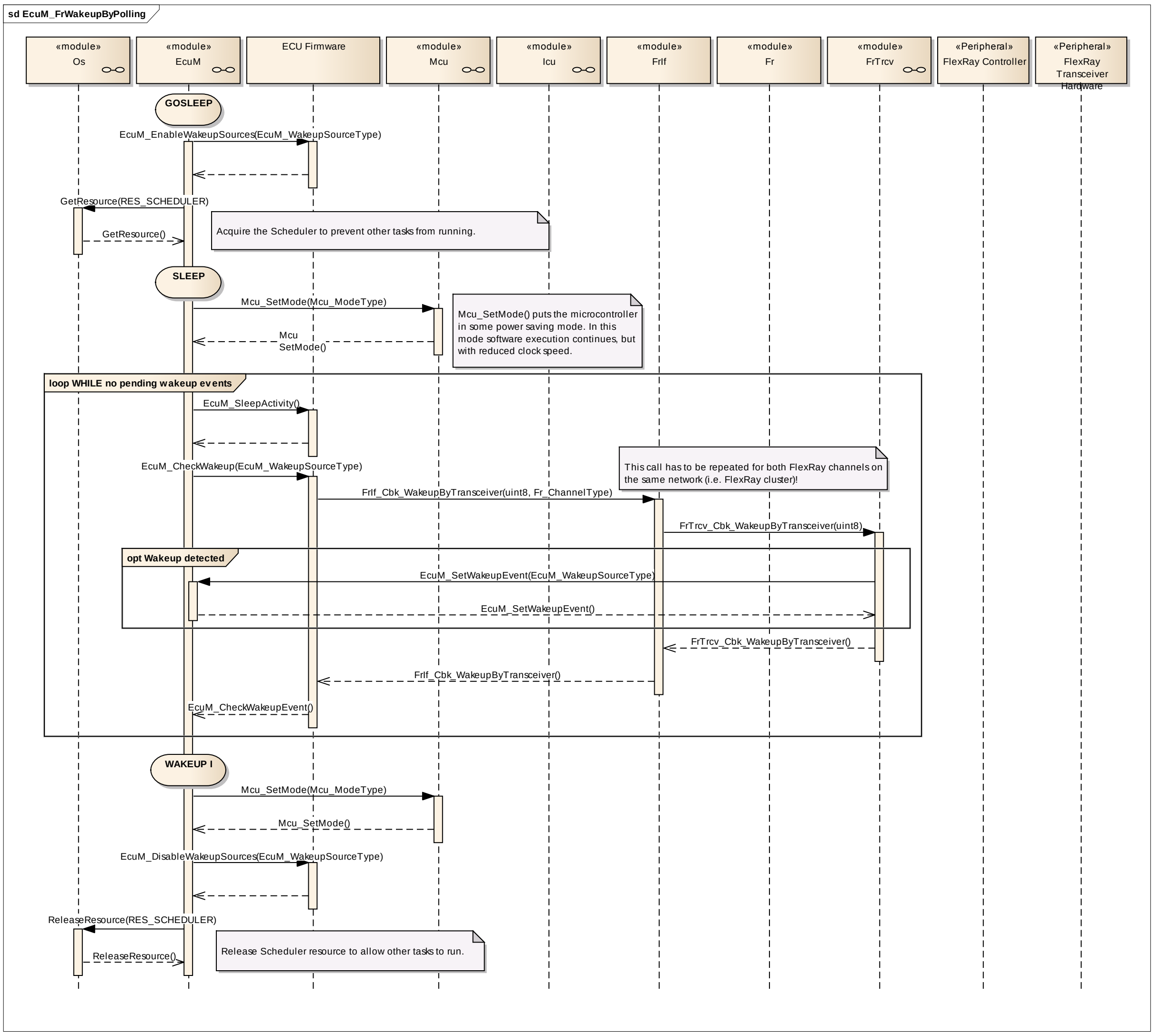

Figure 41 – FlexRay transceiver wake up by polling

10 Configuration specification

10.1 Configuration Variants

The ECU State Manager module has only one configuration variant.

10.2 Configurable Parameters

EcuM2809: 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.

10.2.1 EcuM

Module Name	*EcuM*
Module Description	Configuration of the EcuM (ECU State Manager) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMConfiguration	1	This container contains the configuration (parameters) of the ECU State Manager
EcuMGeneral	1	This container holds the general, pre-compile configuration parameters

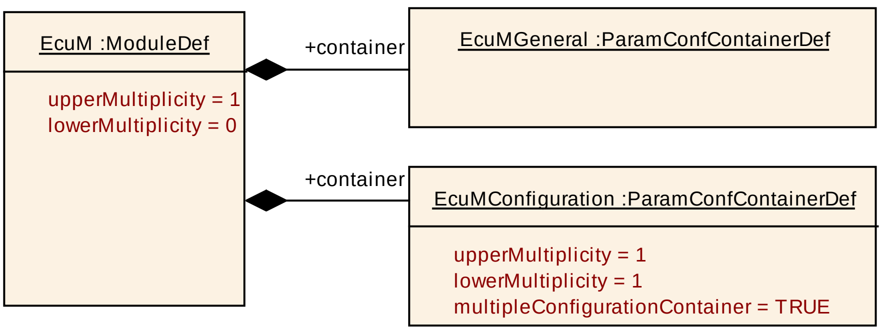

Figure 42 – Container EcuM

10.2.2 EcuMGeneral

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

SWS Item	:		
Name	EcuMDevErrorDetect {ECUM_DEV_ERROR_DETECT}		
Description	If false, no debug artifacts (e.g. calls to DET) shall remain in the executable object. Initialization of DET, however is controlled by configuration of optional BSW modules.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMIncludeDem {ECUM_INCLUDE_DEM}		
Description	If enabled and NVRAM manager is disabled, then an error shall be flagged by the configuration tool		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMIncludeDet {ECUM_INCLUDE_DET}		
Description	If defined, the according BSW module will be initialized by the ECU State Manager		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:
Name	EcuMIncludeNvramMgr {ECUM_INCLUDE_NVRAM_MGR}
Description	If NVRAM manager is enabled but both flash and EEPROM driver are missing, then an error shall be flagged by the configuration tool
Multiplicity	1
Type	BooleanParamDef
Default value	--

ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMIncludeWdgM {ECUM_INCLUDE_WDGM}		
Description	This configuration parameter defines whether the watchdog manager is supported by EcuM. This feature is presented for development purpose to compile out the watchdog manager in the early debugging phase		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMMainFunctionPeriod {ECUM_MAIN_FUNCTION_PERIOD}		
Description	This parameter defines the schedule period of EcuM_MainFunction. Unit: [s]		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	dependency: EcuM2594		

SWS Item	:		
Name	EcuMTTIIEnabled {ECUM_TTII_ENABLED}		
Description	Boolean switch to enable / disable TTII		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMVersionInfoApi		
Description	Switches the version info API on or off		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	

	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMTTIISleepModeRef {ECUM_TTII_WKSOURCE}		
Description	This configuration parameter references the initial sleep mode to be used by TTII when TTII is activated after a RUN mode. EcuM2785: Whenever RUN mode is reached, the TTII protocol shall be reset to use the wakeup source referenced by this parameter. This configuration parameter is a human readable name for a TTII wakeup source which is only needed by the configuration tool. For imlementation on the ECU, this parameter may be dropped and replaced by a generated list index of EcuM_TTII.		
Multiplicity	1		
Type	Reference to [EcuMSleepMode]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

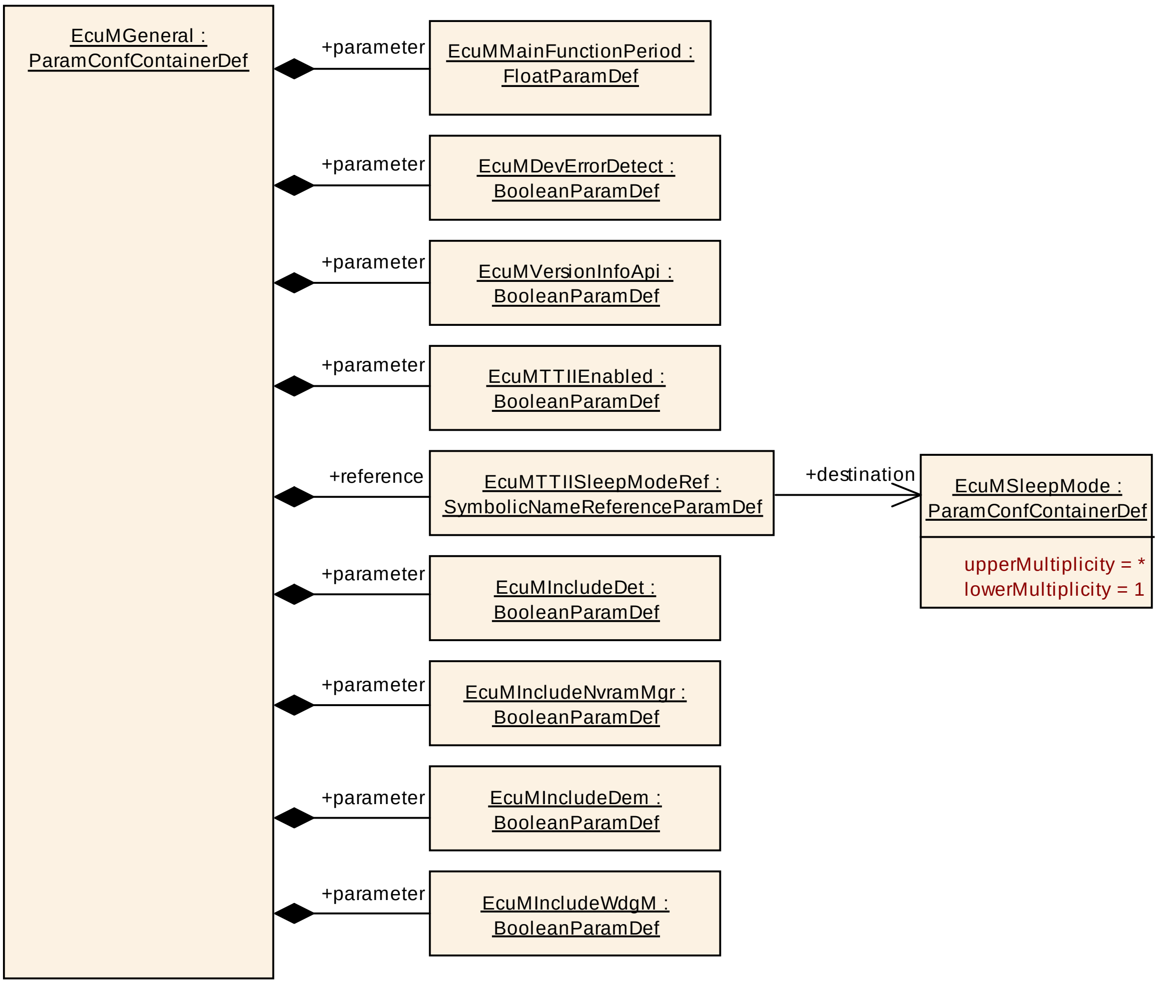

Figure 43 – Container EcuMGeneral

10.2.3 EcuMConfiguration

SWS Item	:
Container Name	EcuMConfiguration{EcuM_Configuration} [Multi Config Container]
Description	This container contains the configuration (parameters) of the ECU State Manager
Configuration Parameters	

SWS Item	:		
Name	EcuMConfigConsistencyHash {ECUM_CONFIGCONSISTENCY_HASH}		
Description	A hash value generated across all pre-compile and link-time parameters of all BSW modules. This hash value is compared against a field in the EcuM_ConfigType and hence allows checking the consistency of the entire configuration.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	X	VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMNvramReadallTimeout		
Description	Period given in seconds for which the ECU State Manager will wait until it considers a ReadAll job of the NVRAM Manager as failed.		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMNvramWriteallTimeout {ECUM_NVRAM_WRITEALL_TIMEOUT}		
Description	Period given in seconds for which the ECU State Manager will wait until it considers a WriteAll job of the NVRAM Manager as failed.		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:	
Name	EcuMRunMinimumDuration {ECUM_RUN_SELF_REQUEST_PERIOD}	
Description	Duration given in seconds for which the ECU State Manager will stay in RUN state even when no one requests RUN. This duration should be long at least as long as a SW-Cs needs to request RUN.	
Multiplicity	1	
Type	FloatParamDef	
Range	-INF .. INF	

Default value	--		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMSleepActivityPeriod {ECUM_SLEEP_ACTIVITY_PERIOD}		
Description	Period of the EcuM_SleepActivity callout. The period is given in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMDefaultAppMode {ECUM_DEFAULT_APP_MODE}		
Description	The default application mode loaded when the ECU comes out of reset.		
Multiplicity	1		
Type	Reference to [OsAppMode]		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMNormalMcuModeRef		
Description	This parameter is a reference to the normal MCU mode to be restored after a sleep.		
Multiplicity	1		
Type	Reference to [McuModeSettingConf]		
ConfigurationClass	**Pre-compile time**	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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 module IDs that will be initialised. Each module in the list will be called for initialisation in the list order. All modules in this list are initilialised before the OS is started and so these modules require no OS support.
EcuMDriverInitListThree	0..1	Container for Init Block III. This container holds a list of module IDs that will be initialised. Each module in the list will be called for initialisation in the list order. All modules in this list are

		initilialised after the OS is started and so these modules may use OS support. These modules may also rely on the Nvram ReadAll job to have provided all data.
EcuMDriverInitListTwo	0..1	Container for Init Block II. This container holds a list of module IDs that will be initialised. Each module in the list will be called for initialisation in the list order. All modules in this list are initilialised after the OS is started and so these modules may use OS support. These modules may not rely on the Nvram ReadAll job to have provided all data.
EcuMDriverInitListZero	0..1	Container for Init Block 0. This container holds a list of module IDs that will be initialised. Each module in the list will be called for initialisation in the list order. All modules in this list are initilialised 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 module IDs. EcuM2719: A configuration tool shall fill the callout EcuM_AL_DriverRestart with initialization calls to the listed drivers in the order in which they occur in the list. EcuM2720: Entries in this list must appear in the same order as in the combined list of EcuM_DriverInitListOne and EcuM_DriverInitListTwo. This list may be a real subset though. In all other cases, the generation tool shall report an error. The included container has the same structure as EcuM_DriverInitItem
EcuMModuleConfiguration	0..*	Collection of references to multiple configuration containers of BSW Modules.
EcuMSleepMode	1..*	This container describes one configured sleep mode.
EcuMTTII	0..*	This container describes a structure and the following configuration items describe its elements. This structures are concatenated to build a list as indicated by Figure 27 - Configuration Container Diagram. The list must contain at least on element when ECUM_TTII_ENABLED is set to true.
EcuMUserConfig	1..*	A list of identifiers that are needed to refer to a software component or another appropriate entity in the system which is designated to request the RUN state. Application requestors refer to entities above RTE, system requestors to entities below RTE (e.g. Communication Manager).
EcuMWakeupSource	1..*	This container describes one configured wakeup source.
EcuMWdgM	0..1	This container holds the configuration parameters for the interaction between the Watchdog Manager (WdgM) and EcuM. The WdgM mode to be selected in a specific Sleep Mode of EcuM is configured in the EcuMSleepMode container.

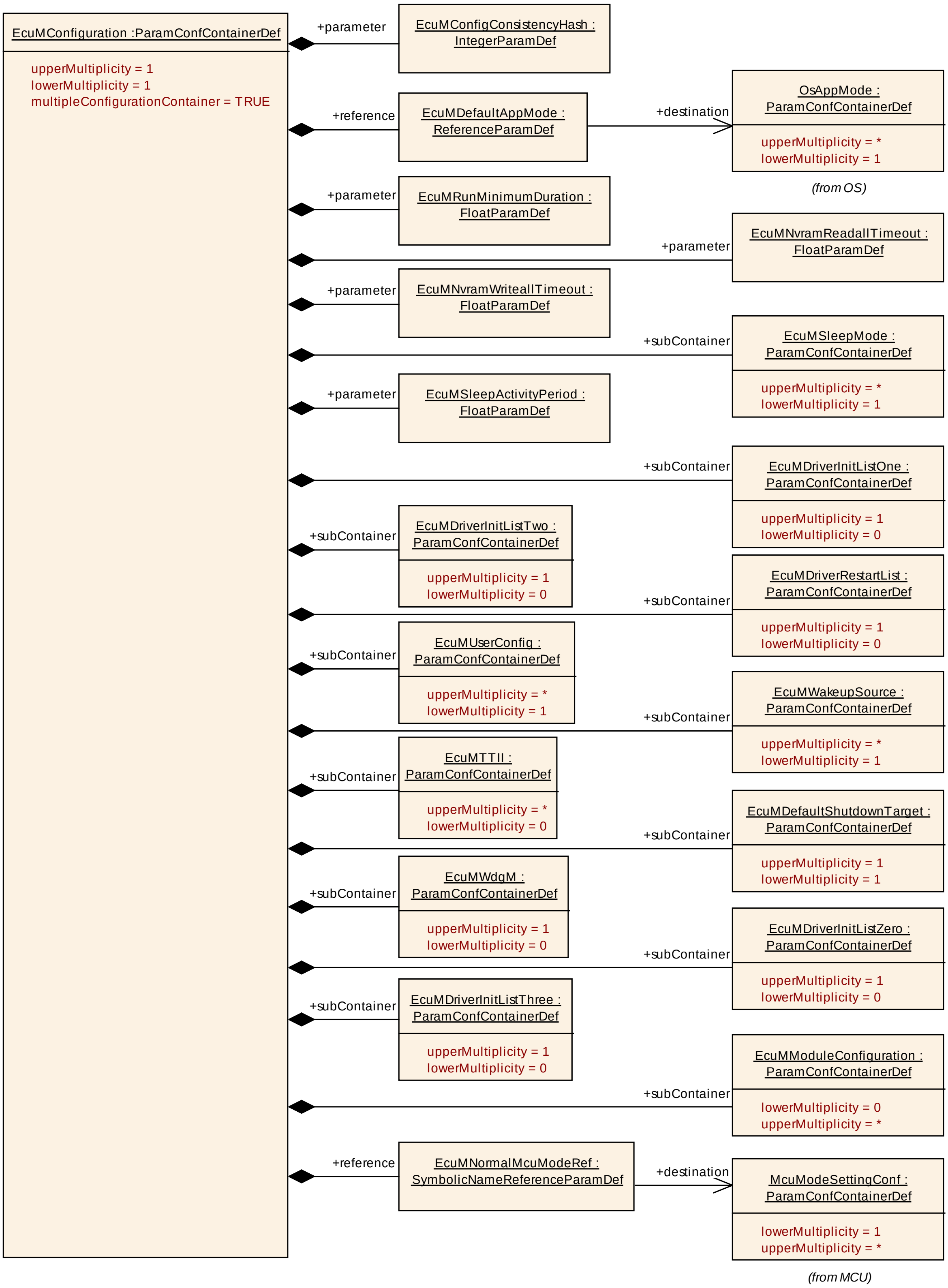

Figure 44 – Container EcuMConfiguration

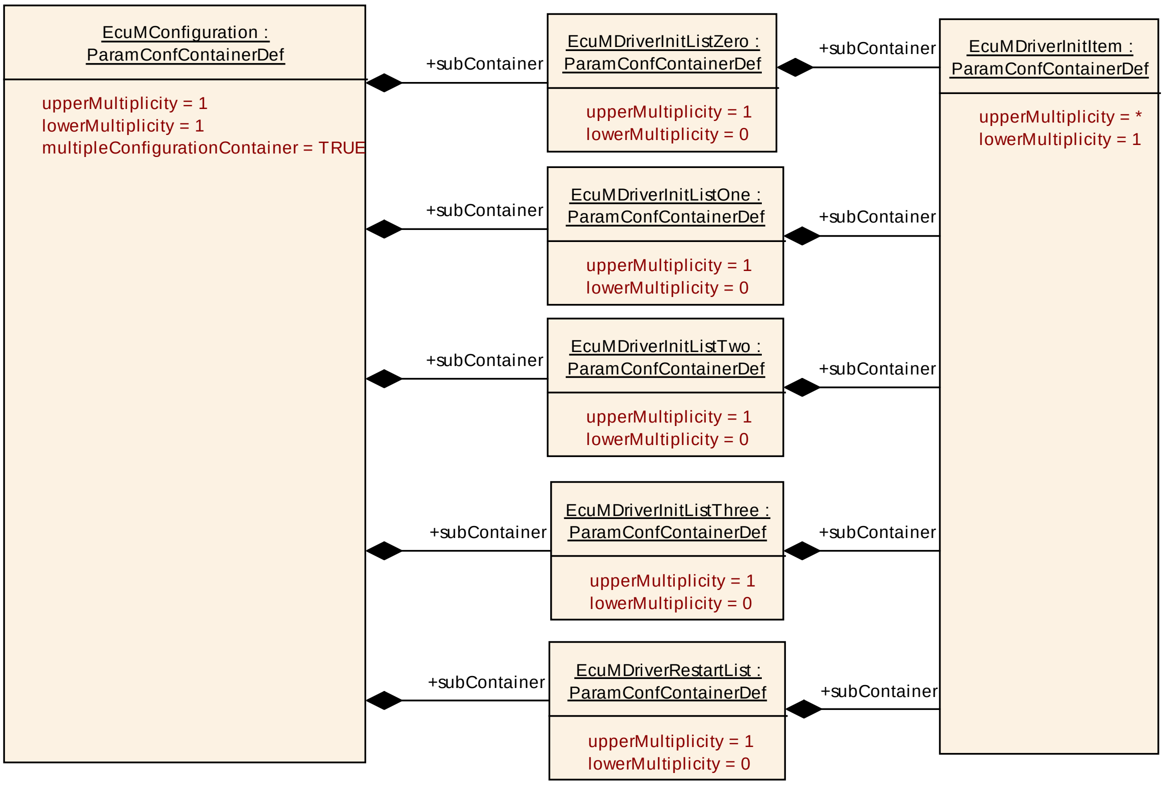

Figure 45 – Container EcuMConfiguration – Init Lists

10.2.4 EcuMDriverInitListZero

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

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	This container describes one entry in a driver init list.

10.2.5 EcuMDriverInitListOne

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

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***

EcuMDriverInitItem	1..*	This container describes one entry in a driver init list.

10.2.6 EcuMDriverInitListTwo

SWS Item	:
Container Name	EcuMDriverInitListTwo
Description	Container for Init Block II. This container holds a list of module IDs that will be initialised. Each module in the list will be called for initialisation in the list order. All modules in this list are initilialised after the OS is started and so these modules may use OS support. These modules may not rely on the Nvram ReadAll job to have provided all data.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	This container describes one entry in a driver init list.

10.2.7 EcuMDriverInitListThree

SWS Item	:
Container Name	EcuMDriverInitListThree
Description	Container for Init Block III. This container holds a list of module IDs that will be initialised. Each module in the list will be called for initialisation in the list order. All modules in this list are initilialised after the OS is started and so these modules may use OS support. These modules may also rely on the Nvram ReadAll job to have provided all data.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	This container describes one entry in a driver init list.

10.2.8 EcuMDriverRestartList

SWS Item	:
Container Name	EcuMDriverRestartList
Description	List of module IDs. EcuM2719: A configuration tool shall fill the callout EcuM_AL_DriverRestart with initialization calls to the listed drivers in the order in which they occur in the list. EcuM2720: Entries in this list must appear in the same order as in the combined list of EcuM_DriverInitListOne and EcuM_DriverInitListTwo. This list may be a real subset though. In all other cases, the generation tool shall report an error. The included container has the same structure as EcuM_DriverInitItem
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
EcuMDriverInitItem	1..*	This container describes one entry in a driver init list.

10.2.9 EcuMDriverInitItem

SWS Item	:
Container Name	EcuMDriverInitItem

Description	This container describes one entry in a driver init list.
Configuration Parameters	

SWS Item	:		
Name	EcuMModuleID {ModuleID}		
Description	Short name of the module to be initialized, e.g. Mcu, Gpt etc. In case EcuMModuleConfigRef is used the EcuMModuleID is optional (in case it is given it shall have the same value as the referenced EcuMModuleConfiguration).		
Multiplicity	0..1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMModuleService		
Description	The service to be called to initialize that module, e.g. Init, PreInit, Start etc. If the service is Init and the parameter EcuMModuleConfigurationRef has been set for that module, the corresponding pointer to the init structure (<Module>_ConfigType) shall be passed as an argument.		
Multiplicity	1		
Type	StringParamDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMModuleConfigRef		
Description	In case a BSW Module is configured with several instances in this ECU Configuration there shall be a reference which determines the to be used multipleConfigurationContainer for this BSW Module instance.		
Multiplicity	0..1		
Type	Reference to [EcuMModuleConfiguration]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

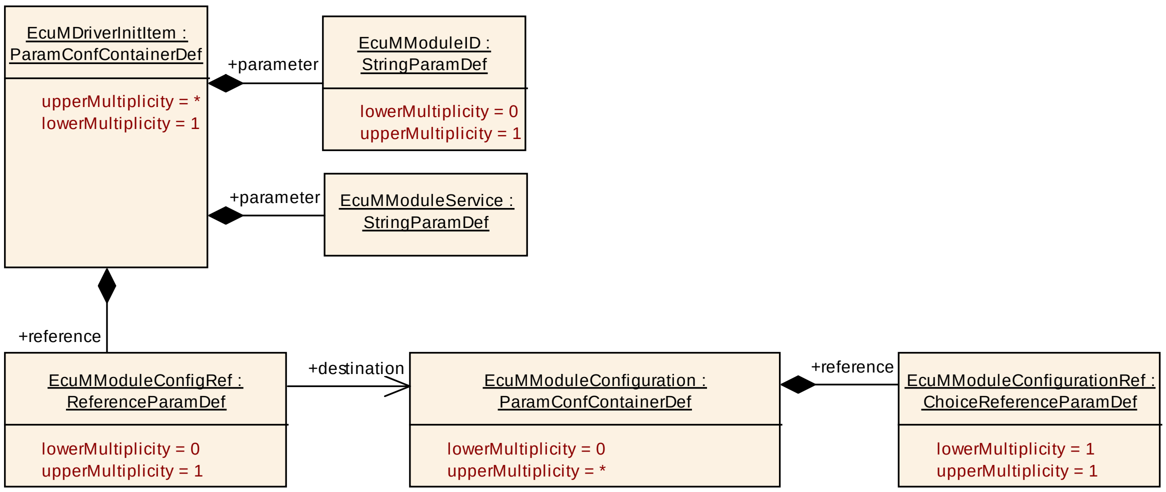

Figure 46 – Container EcuMDriverInititem

10.2.10 EcuMModuleConfiguration

SWS Item	:
Container Name	EcuMModuleConfiguration
Description	Collection of references to multiple configuration containers of BSW Modules.
Configuration Parameters	

SWS Item	:		
Name	EcuMModuleConfigurationRef {InitConfiguration}		
Description	This parameter contains a reference to the init structure of the corresponding BSW module.		
Multiplicity	1		
Type	Choice reference to [AdcConfigSet , CanConfigSet , CanIfInitConfiguration , CanNmGlobalConfig , CanStateManagerConfiguration , CanTrcvConfigSet , CddConfigSet , ComConfig , ComMConfigSet , DcmConfigSet , DemConfigSet , FlsConfigSet , FrIfConfig , FrIsoTpMultipleConfig , FrMultipleConfiguration , FrNmChannelConfig , FrSmCluster , FrTpMultipleConfig , GptChannelConfigSet , IcuConfigSet , IpduMConfig , LinGlobalConfig , LinIfGlobalConfig , LinSMChannel , LinTpGlobalConfig , McuModuleConfiguration , PduRGlobalConfig , PortConfigSet , PwmChannelConfigSet , SpiDriver , WdgMConfigSet , WdgModeConfig]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

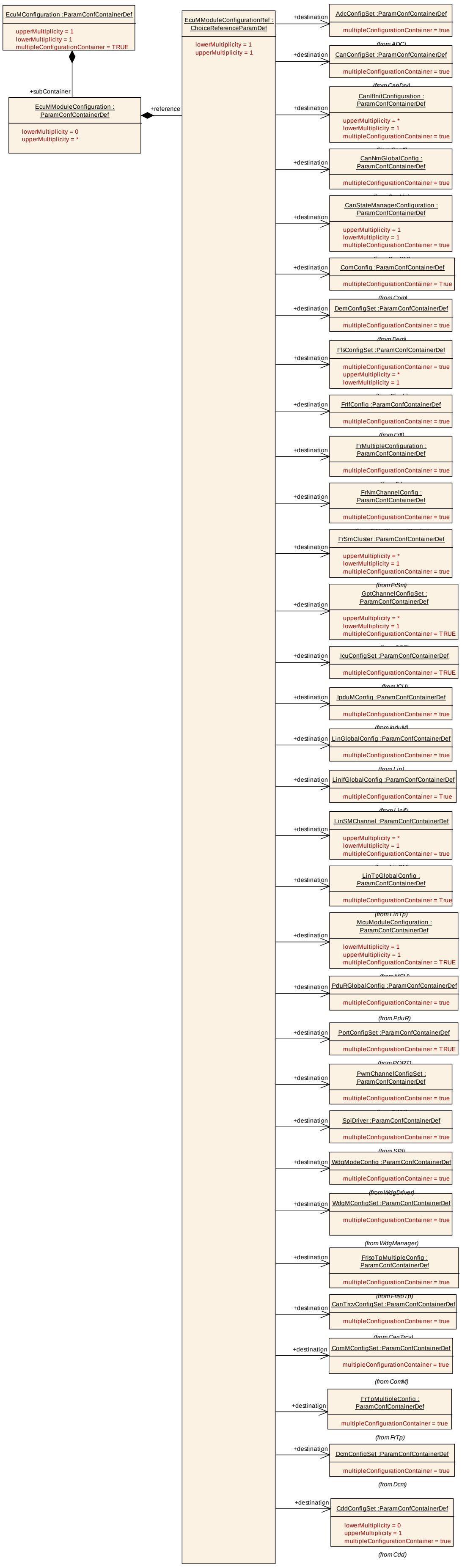
EcuMConfiguration :ParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 1
multipleConfigurationContainer = TRUE
+subContainer
EcuMModuleConfiguration : ParamConfContainerDef
lowerMultiplicity = 0
upperMultiplicity = *
+reference
EcuMModuleConfigurationRef : ChoiceReferenceParamDef
lowerMultiplicity = 1
upperMultiplicity = 1
+destination
AdcConfigSet :ParamConfContainerDef
CanConfigSet :ParamConfContainerDef
CanIfInitConfiguration : ParamConfContainerDef
CanNmGlobalConfig : ParamConfContainerDef
CanStateManagerConfiguration : ParamConfContainerDef
ComConfig :ParamConfContainerDef
DemConfigSet :ParamConfContainerDef
FlsConfigSet :ParamConfContainerDef
FrIfConfig :ParamConfContainerDef
FrMultipleConfiguration : ParamConfContainerDef
FrNmChannelConfig : ParamConfContainerDef
FrSmCluster :ParamConfContainerDef
GptChannelConfigSet : ParamConfContainerDef
IcuConfigSet :ParamConfContainerDef
IpduMConfig :ParamConfContainerDef
LinGlobalConfig :ParamConfContainerDef
LinIfGlobalConfig :ParamConfContainerDef
LinSMChannel :ParamConfContainerDef
LinTpGlobalConfig : ParamConfContainerDef
McuModuleConfiguration : ParamConfContainerDef
PduRGlobalConfig :ParamConfContainerDef
PortConfigSet :ParamConfContainerDef
PwmChannelConfigSet : ParamConfContainerDef
SpiDriver :ParamConfContainerDef
WdgModeConfig :ParamConfContainerDef
WdgMConfigSet :ParamConfContainerDef
FrIsoTpMultipleConfig : ParamConfContainerDef
CanTrcvConfigSet :ParamConfContainerDef
ComMConfigSet :ParamConfContainerDef
FrTpMultipleConfig : ParamConfContainerDef
DcmConfigSet :ParamConfContainerDef
CddConfigSet :ParamConfContainerDef
(from PduR)
(from WdgDriver)
(from WdgManager)
(from FrIsoTp)
(from ComM)
(from FrTp)
(from Dcm)
(from Cdd)

Figure 47 – Container EcuMModuleConfiguration

10.2.11 EcuMDefaultShutdownTarget

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

SWS Item	:		
Name	EcuMDefaultState {ECUM_DEFAULT_SHUTDOWN_TARGET}		
Description	This parameter describes the state part of the default shutdown target selected when the ECU comes out of reset. If EcuMStateSleep is selected, the parameter EcuMDefaultSleepModeRef selects the specific sleep mode.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	EcuMStateOff	Corresponds to ECUM_STATE_OFF in EcuM_StateType.	
	EcuMStateReset	Corresponds to ECUM_STATE_RESET in EcuM_StateType.	
	EcuMStateSleep	Corresponds to ECUM_STATE_SLEEP in EcuM_StateType.	
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMDefaultSleepModeRef		
Description	If EcuMDefaultShutdownTarget is EcuMStateSleep, this parameter selects the default sleep mode. Otherwise this parameter may be ignored.		
Multiplicity	1		
Type	Reference to [EcuMSleepMode]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

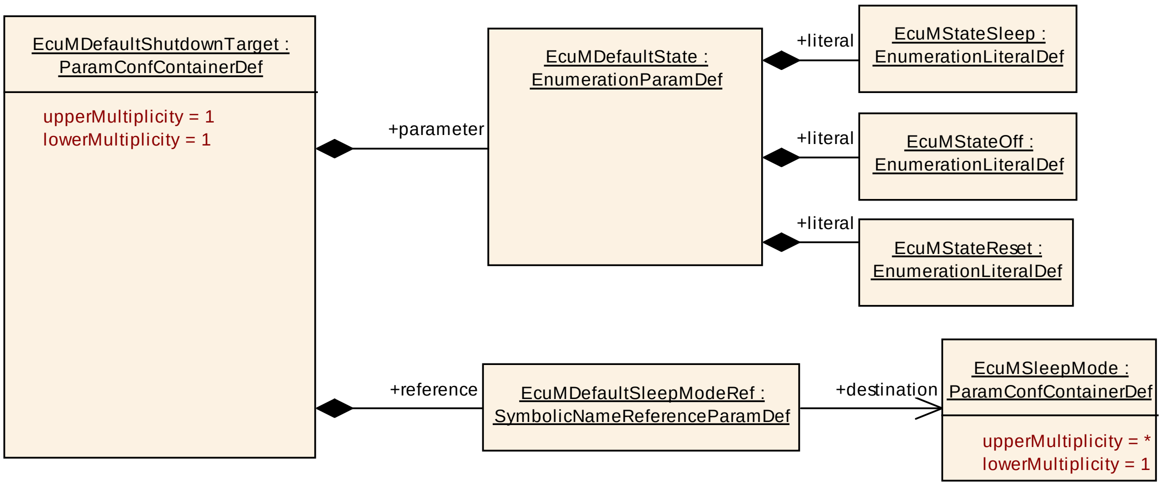

Figure 48 – Container EcuMDefaultShutdownTarget

10.2.12 EcuMWakeupSource

SWS Item	:
Container Name	EcuMWakeupSource{EcuM_WakupSource}
Description	This container describes one configured wakeup source.
Configuration Parameters	

SWS Item	:		
Name	EcuMCheckWakeupTimeout		
Description	Add a Timer to 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	FloatParamDef		
Range	0 .. 10		
Default value	0		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	:		
Name	EcuMResetReason {ResetReason}		
Description	This parameter describes the mapping of reset reasons detected by the MCU driver into wakeup sources.		
Multiplicity	0..*		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMValidationTimeout {ValidationTimeout}		
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.		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMWakeupSourceId {WakeupSourceName}		
Description	This parameter defines the identifier of this wakeup source.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD

	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMWakeupSourcePolling		
Description	This parameter describes if the wakeup source needs polling.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMComMChannelRef {ComChannel}		
Description	This parameter is a reference to a Network (channel) defined in the Communication Manager. No reference indicates that the wakeup source is not a communication channel.		
Multiplicity	0..1		
Type	Reference to [ComMChannel]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

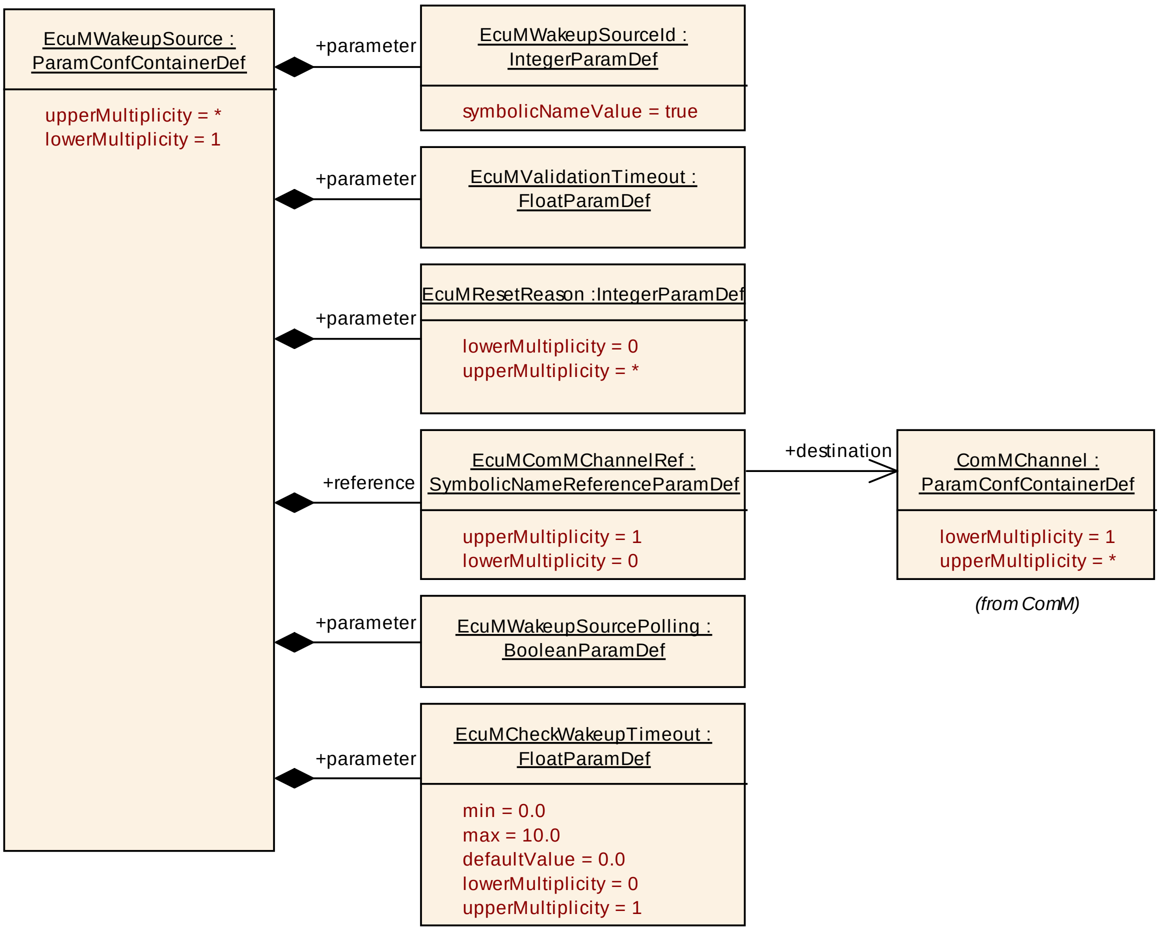

Figure 49 – Container EcuMWakeupSource

10.2.13 EcuMSleepMode

SWS Item	:
Container Name	EcuMSleepMode
Description	This container describes one configured sleep mode.
Configuration Parameters	

SWS Item	:		
Name	EcuMSleepModeId		
Description	This is the ID to identify this sleep mode in services like EcuM_SelectShutdownTarget.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMSleepModeName {SleepModeName}		
Description	This item allows to give symbolic names to the different sleep modes.		
Multiplicity	1		
Type	StringParamDef (Symbolic Name generated for this parameter)		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMSleepModeSuspend		
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	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMSleepModeMcuModeRef {SleepModeConfiguration}		
Description	This parameter is a reference to the corresponding MCU mode for this sleep mode.		
Multiplicity	1		
Type	Reference to [McuModeSettingConf]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD

	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMSleepModeWdgMModeRef		
Description	This parameter defines the Watchdog Manager mode that shall be active in this sleep mode.		
Multiplicity	0..1		
Type	Reference to [WdgMMode]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMWakeupSourceMask		
Description	This parameter is a reference to the wakeup source that shall be enabled for this sleep mode.		
Multiplicity	1..*		
Type	Reference to [EcuMWakeupSource]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

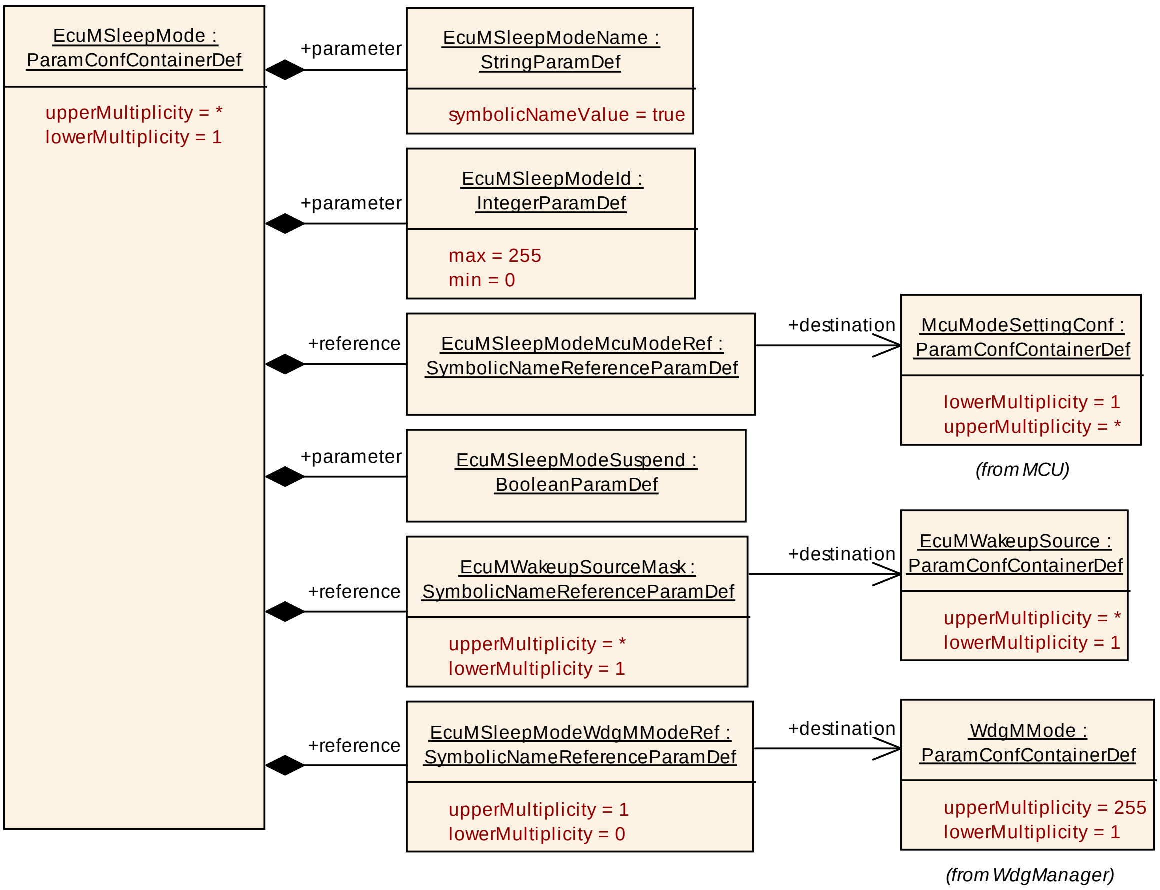

Figure 50 – Container EcuMSleepMode

10.2.14 EcuMTTII

SWS Item	:
Container Name	EcuMTTII
Description	This container describes a structure and the following configuration items describe its elements. This structures are concatenated to build a list as indicated by Figure 27 - Configuration Container Diagram. The list must contain at least on element when ECUM_TTII_ENABLED is set to true.
Configuration Parameters	

SWS Item	:		
Name	EcuMDivisor {Divisor}		
Description	This parameter defines the divisor preload value.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMSleepModeRef		
Description	This configuration parameter is a reference to a configured sleep mode that is used for TTII.		
Multiplicity	1		
Type	Reference to [EcuMSleepMode]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	:		
Name	EcuMSuccessorRef {Successor}		
Description	This parameter is a reference to the next sleep mode in the TTII protocol.		
Multiplicity	1		
Type	Reference to [EcuMSleepMode]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

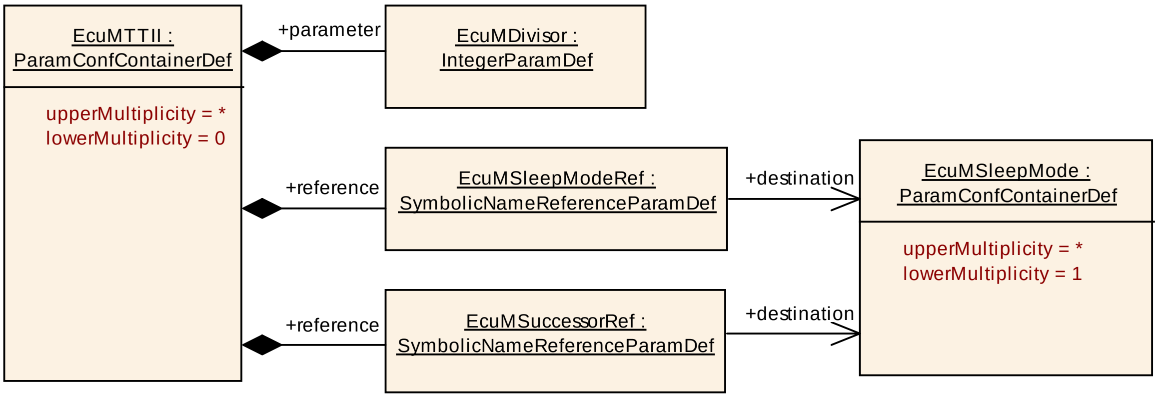

Figure 51 – Container EcuMTTII

10.2.15 EcuMUserConfig

SWS Item	:
Container Name	EcuMUserConfig{EcuM_User}
Description	A list of identifiers that are needed to refer to a software component or another appropriate entity in the system which is designated to request the RUN state. Application requestors refer to entities above RTE, system requestors to entities below RTE (e.g. Communication Manager).
Configuration Parameters	

SWS Item	:		
Name	EcuMUser {User}		
Description	--		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

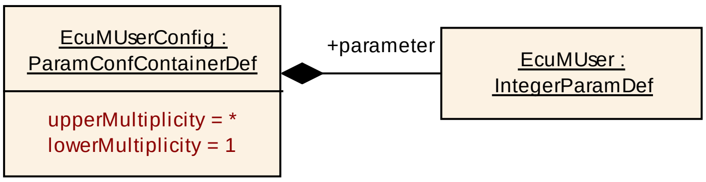

Figure 52 – Container EcuMUserConfig

10.2.16 EcuMWdgM

SWS Item	:
Container Name	EcuMWdgM
Description	This container holds the configuration parameters for the interaction between the Watchdog Manager (WdgM) and EcuM. The WdgM mode to be selected in a specific Sleep Mode of EcuM is configured in the EcuMSleepMode container.
Configuration Parameters	

SWS Item	:		
Name	EcuMSupervisedEntityRef		
Description	This parameter references the Supervised Entity ID that way configured for EcuM in the Watchdog Manager.		
Multiplicity	1		
Type	Reference to [WdgMSupervisedEntity]		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMWdgMPostRunModeRef		
Description	This parameter references the WdgM mode to be set when entering the POST RUN state of EcuM.		
Multiplicity	1		
Type	Reference to [WdgMMode]		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMWdgMRunModeRef		
Description	This parameter references the WdgM mode to be set when entering the RUN state of EcuM.		
Multiplicity	1		
Type	Reference to [WdgMMode]		
ConfigurationClass	**Pre-compile time**	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMWdgMShutdownModeRef		
Description	This parameter references the WdgM mode to be set when leaving the GO OFF I state of EcuM.		
Multiplicity	1		
Type	Reference to [WdgMMode]		
ConfigurationClass	**Pre-compile time**	--	

	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMWdgMStartupModeRef		
Description	This parameter references the WdgM mode to be set when entering the STARTUP II state of EcuM.		
Multiplicity	1		
Type	Reference to [WdgMMode]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	:		
Name	EcuMWdgMWakeupModeRef		
Description	This parameter references the WdgM mode to be set when entering the WAKEUP I state of EcuM.		
Multiplicity	1		
Type	Reference to [WdgMMode]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

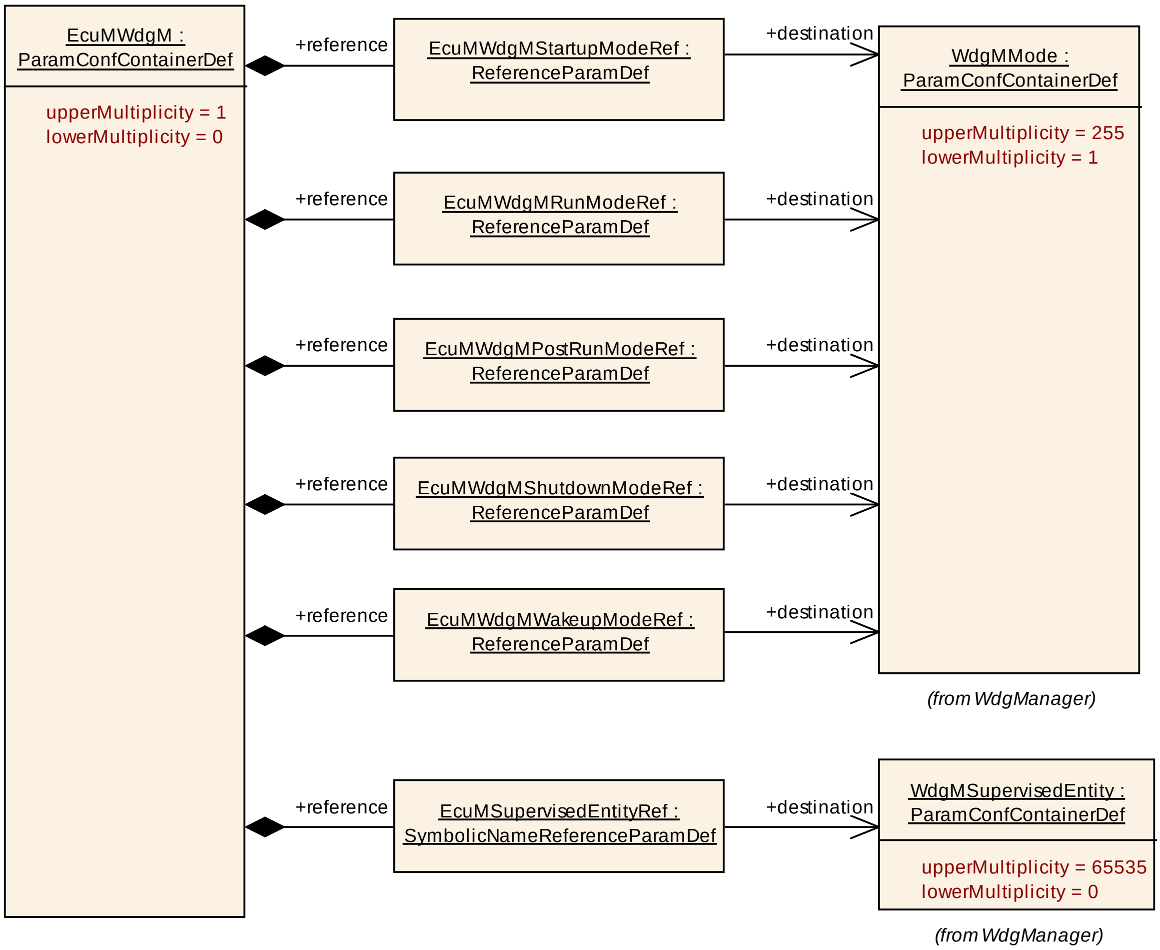

Figure 53 – Container EcuMWdgM

10.3 Published Parameters

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [22] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

10.4 Checking Configuration Consistency

10.4.1 The Necessity for Checking Configuration Consistency in the ECU State Manager

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

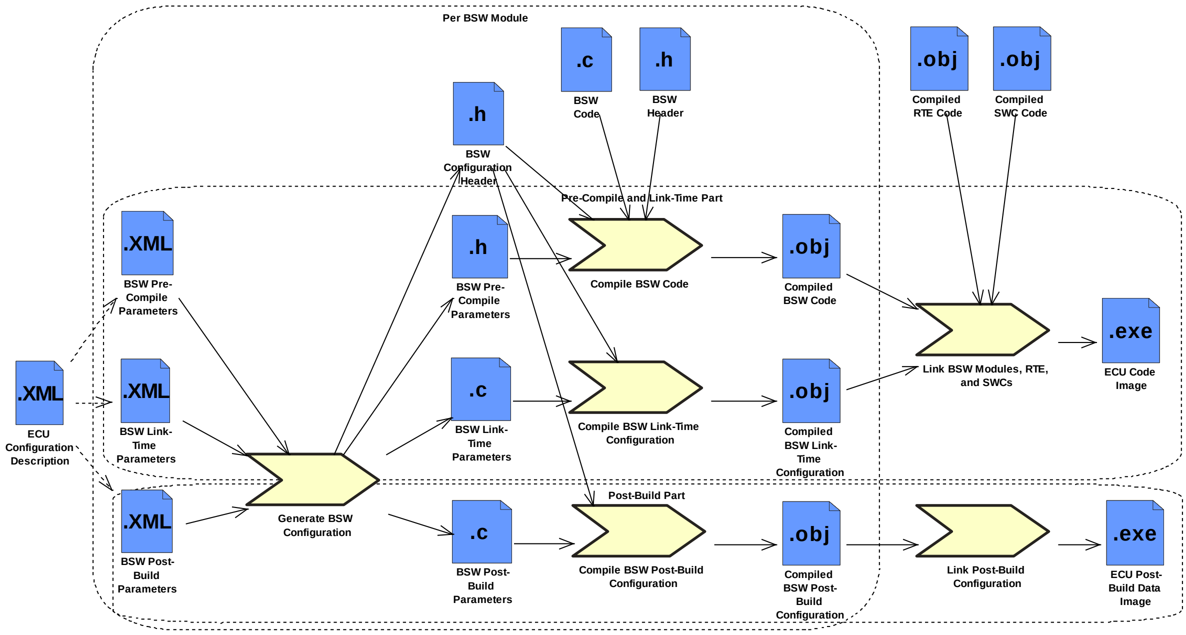

Figure 54 – BSW Configuration Steps

Example: The number of watchdogs to be triggered by the Watchdog Manager is set in the pre-compile parameter *WDGM_NUMBER_OF_WATCHDOG_INSTANCES*. For each of these watchdog instances the container *WdgMWatchdogInstance* contains three post-build parameters:

- *WDGM_WATCHDOG_INSTANCE_ID*,
- *WDGM_TRIGGER_SLOW_REFERENCE_CYCLE*, and
- *WDGM_TRIGGER_FAST_REFERENCE_CYCLE*.

The number of *WdgMWatchdogInstance* containers in the post-build data must exactly match the value of *WDGM_NUMBER_OF_WATCHDOG_INSTANCES*. Otherwise, wrong data will be read by the `WdgM_Init` function.

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

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

are exactly the same as

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

This can only be done at run-time.

EcuM2796: To avoid multiple checks scattered over the different BSW modules, the ECU State Manager module shall check the consistency once before initializing the first BSW module. This also implies that the ECU State Manager module must not only check the consistency of its own parameters but of all post-build configurable BSW modules.

EcuM2797: The ECU configuration tool shall compute a hash value over all pre-compile and link-time configuration parameters of all BSW modules and put that into the link-time configuration parameter *ECUM_CONFIGCONSISTENCY_HASH.* The hash value is necessary for two reasons. First, the pre-compile and link-time parameters are not accessible anymore at run-time. Second, the check must be very efficient at run-time. Comparing hundreds of parameters would cause an unacceptable delay in the ECU startup process.

EcuM2798: The EcuM configuration tool shall put the current value of the configuration parameter *ECUM_CONFIGCONSISTENCY_HASH* into a field in the EcuM_ConfigType structure, which contains the root of all post-build configuration parameters. EcuM shall check in EcuM_Init 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 becomes very efficient and independent of a certain hash computation algorithm. This allows for the use of complex hash computation algorithms, e.g. cryptographically strong hash functions.

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

EcuM2799: The used hash computation algorithm shall always produce the same hash value for the same set of configuration data, regardless of the order of configuration parameters in the XML files.

10.4.2 Example Hash Computation Algorithm

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

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

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

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

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

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

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