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

Document Version	3.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Ver.	Changed by	Change Description
24.11.2011	3.0.0	AUTOSAR Administration	• Fixed interoperability problems between EcuM and BswM • Terminology of ECU State Manager Flexible more consistently described • Modification of sleep sequences to minimize misses of wakeup interrupts
16.11.2010	2.1.0	AUTOSAR Administration	• Updated pseudo code for AUTOSAR Services • Update startup procedure for multi core systems
04.12.2009	2.0.0	AUTOSAR Administration	• Removed state machine to accommodate mode-dependent scheduling • Added Multi-Core support • Added Alarm Clock feature • Revised disclaimer
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised
11.12.2007	1.2.0	AUTOSAR Administration	• Fixed Wakeup mechanisms • Included optional triggering of Watchdog Manager during Startup, Shutdown, and Sleep • Extended startup sequence to have more flexibility and to directly initialize all other BSW modules • Generated APIs from BSW UML model • Generated configuration from Meta Model • Document meta information extended Small layout adaptations made

Document Change History			
Date	**Ver.**	**Changed by**	**Change Description**
31.01.2007	1.1.0	AUTOSAR Administration	• Corrected startup flow and wakeup concept. • Added specification for AUTOSAR ports. • Modified configuration for compliance with variant management. • Added new API services. • Legal disclaimer revised • Release Notes added • “Advice for users” revised “Revision Information” added
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

Known Limitations

- Run / PostRun Request Handling should be addressed in the context of flexible ECU management (see section 1.1 Backwards Compatibility to Previous ECU Manager Module Versions)
- The ECU Manager module interfaces must be specified as reentrant in the Multi-Core context.

1 Introduction and Functional Overview

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

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

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

There are actually two variants of AUTOSAR ECU management: flexible and fixed.

Flexible ECU management is much more powerful than in previous versions of the ECU Manager. Most notably, the fixed schema of ECU states and transitions between them has been eliminated to allow the following additional scenarios:

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

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

- RTE and BSW Scheduler module [14] are now amalgamated into one module: This module supports freely configurable BSW and application modes and their mode-switching facilities.
- BSW Mode Manager module [21]: This module implements configurable rules and action lists to evaluate the conditions for switching ECU modes and to implement the necessary actions to do so.

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

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

Fixed ECU Management continues ECU management in the form of previous AUTOSAR releases. It has a fixed set of ECU states and transistions between them and is sufficient for conventional ECUs that do not have special requirements such

as partial or fast startup, interleaved startup, and multiple operational states (multiple RUN states). Fixed ECU managament does not support Multicore ECUs, among other things.

This document specifies the ECU Manager module for flexible ECU management. [22] specifies the ECU Manager module for fixed ECU management.

1.1 Backwards Compatibility to Previous ECU Manager Module Versions

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

For more information about a configuration in respect to compatibility see the “Guide to Mode Management” [23].

2 Definitions and Acronyms

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

Term	***Description***
Callback	Refer to the Glossary [6]
Callout	'Callouts' are function stubs that the system designer can replace with code, usually at configuration time, to add functionality to the ECU Manager module. Callouts are separated into two classes. One class provides mandatory ECU Manager module functionality and serves as a hardware abstraction layer. The other class provides optional functionality.
Integration Code	Refer to the Glossary [6]
Mode	A Mode is a certain set of states of the various state machines (not only of the ECU Manager) that are running in the vehicle and are relevant to a particular entity, an application or the whole vehicle
Passive Wakeup	A wakeup caused from an attached bus rather than an internal event like a timer or sensor activity.
Phase	A logical or temporal assembly of ECU Manager's actions and events, e.g. STARTUP, UP, SHUTDOWN, SLEEP, … Phases can consist of Sub-Phases which are often called Sequences if they above all exist to group sequences of executed actions into logical units. Phases in this context are not the phases of the AUTOSAR Methodology.
Shutdown Target	The ECU must be shut down before it is put to sleep, before it is powered off or before it is reset. SLEEP, OFF, and RESET are therefore valid shutdown targets. By selecting a shutdown target, an application can communicate its wishes for the ECU behavior after the next shutdown to the ECU Manager module.
State	States are internal to their respective BSW component and thus not visible to the application. So they are only used by the BSW's internal state machine. The States inside the ECU Manager build the phases and therefore handle the modes.
Wakeup Event	A physical event which causes a wakeup. A CAN message or a toggling IO line can be wakeup events. Similarly, the internal SW representation, e.g. an interrupt, may also be called a wakeup event.
Wakeup Reason	The wakeup reason is the wakeup event that is the actual cause of the last wakeup.
Wakeup Source	The peripheral or ECU component which deals with wakeup events is called a wakeup source.

Acronym	***Description***
BswM	Basic Software Mode Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EcuM	ECU Manager
GPT	General Purpose Timer
ICU	Input Capture Unit
ISR	Interrupt Service Routine
MCU	Microcontroller Unit

NVRAM	Non-volatible random access memory
OS	Operating System
RTE	Runtime Environment
VFB	Virtual Function Bus

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

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

[4] Requirements on Mode Management
AUTOSAR_SRS_ModeManagement.pdf

[5] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

3.2 Related standards and norms

None

3.3 Related AUTOSAR Software Specifications

[6] Glossary
AUTOSAR_TR_Glossary.pdf

[7] Specification of Communication Manager
AUTOSAR_SWS_COMManager.pdf

[8] Specification of Watchdog Manager
AUTOSAR_SWS_WatchdogManager.pdf

[9] Specification of MCU Driver
AUTOSAR_SWS_MCUDriver.pdf

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

[11] Specification of EEPROM Interface
AUTOSAR_SWS_EEPROMDriver.pdf

[12] Specification of Flash Interface
AUTOSAR_SWS_FlashDriver.pdf

[13] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[14] Specification of RTE
AUTOSAR_SWS_RTE.pdf

[15] Specification of the Virtual Function Bus
AUTOSAR_EXP_VFB.pdf

[16] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[17] Specification of Development Error Tracer
AUTOSAR_SWS_ DevelopmentErrorTracer.pdf

[18] Specification of CAN Transceiver Driver
AUTOSAR_SWS_CANTransceiverDriver.pdf

[19] Specification of C Implementation Rules
AUTOSAR_TR_CImplementationRules.pdf

[20] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[21] Specification of BSW Mode Manager
AUTOSAR_SWS_BSWModeManager.pdf

[22] Specification of ECU State Manager Fixed
AUTOSAR_SWS_ECUStateManagerFixed.pdf

[23] Guide to Mode Management
AUTOSAR_Guide_ModeManagement.pdf

4 Constraints and Assumptions

4.1 Limitations

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

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

4.2 Hardware Requirements

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

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

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

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

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

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

4.3 Applicability to car domains

The ECU Manager module is applicable to all car domains.

5 Dependencies to other modules

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

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

5.1 SPAL Modules

5.1.1 MCU Driver

The MCU Driver is the first basic software module initialized by the ECU Manager module. When MCU_Init returns (see EcuM2858), the MCU module and the MCU Driver module are not necessarily fully initialized, however. Additional MCU module specific steps may be needed to complete the initialization. The ECU Manager module provides two callout where this additional code can be placed. Refer to section 7.3.2 Activities in StartPreOS Sequence for details.

5.1.2 Driver Dependencies and Initialization Order

BSW drivers may depend on each other. A typical example is the watchdog driver, which needs the SPI driver to access an external watchdog. This means on the one hand, that drivers may be stacked (not relevant to the ECU Manager module) and on the other hand that the called module must be initialized before the calling module is initialized.

The system designer is responsible for defining the initialization order at configuration time in `EcuMDriverInitListZero` (see `EcuM114_Conf`), `EcuMDriverInitListOne` (see `EcuM111_Conf`) and in `EcuMDriverRestartList` (see `EcuM115_Conf`).

5.2 Peripherals with Wakeup Capability

Wakeup sources must be handled and encapsulated by drivers.

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

The driver must invoke EcuM_SetWakeupEvent (see EcuM2826) to notify the ECU Manager module that a pending wakeup event has been detected. The driver must not only invoke EcuM_SetWakeupEvent while the ECU is waiting for a wakeup event during a sleep phase but also during the driver initialization phase and during normal operation when EcuM_MainFunction is running.

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

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

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

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

5.3 Operating System

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

5.4 BSW Scheduler

The ECU Manager module initializes the BSW Scheduler and the ECU Manager module also contains EcuM_MainFunction (see EcuM2837) which is scheduled to periodically evaluate wakeup requests and update the Alarm Clock.

5.5 BSW Mode Manager

ECU states are generally implemented as AUTOSAR modes and the BSW Mode Manager is responsible for monitoring changes in the ECU and affecting the corresponding changes to the ECU state machine as appropriate. Refer to the Specification of the Virtual Function Bus [15] for a discussion of AUTOSAR mode management and to the Guide to Mode Management [23] for ECU state machine implementation details and for guidelines about how to configure the BSW Mode Manager to implement the ECU state machine

The BSW Mode Manager can only manage the ECU state machine after mode management is operational – that is, after the SchM has been initialized and until the SchM is de-initialised or halted. The ECU Manager module takes control of the ECU when the BSW Mode manager is not operational.

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

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

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

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

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

5.6 Software Components

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

- Shutdown targets.

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

The following measures might help to resolve this issue.

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

5.7 File Structure

[EcuM3023][

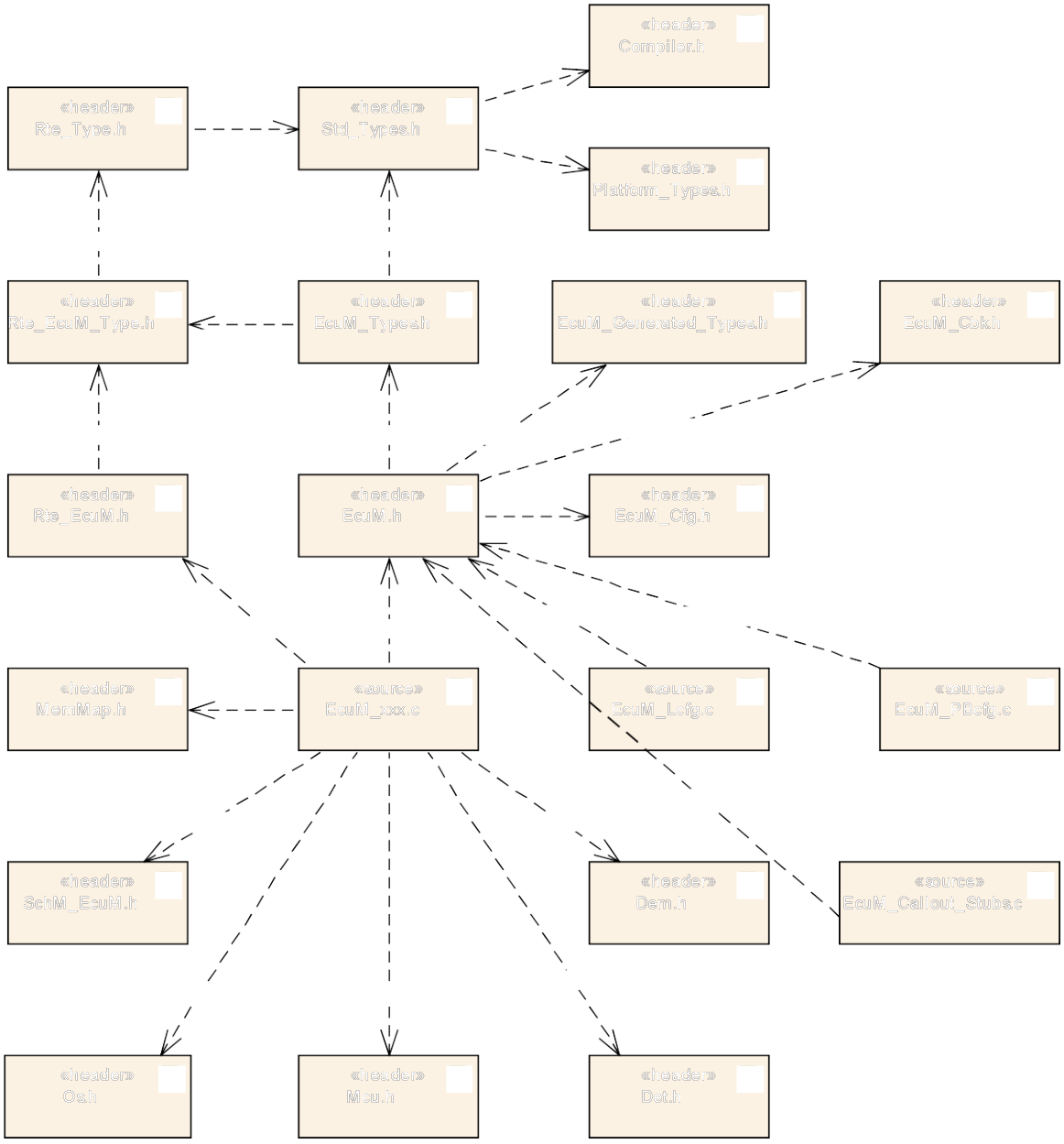

Figure 1 - ECU Manager Module Code File Structure

](BSW00300,BSW00346)

5.7.1 Code file structure

This specification does not define the code file structure completely.

[EcuM2988] [The code-file structure shall at least include files named:

- `EcuM_Lcfg.c` – for link time configurable parameters and
- `EcuM_PBcfg.c` – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.](BSW00380)

[EcuM2989] ⌈The module code for the entire ECU Manager module implementation, or all of its parts, shall be contained in one or more C files with names conforming to the pattern `EcuM_xxx.c`.⌋()

[EcuM2990] ⌈The ECU Manager module implementation shall provide a single `EcuM_Callout_Stubs.c` file which contains the stubs of the callouts realized in this implementation (see section 8.6 Callout Definitions for a list of the callouts that could possibly be implemented)⌋()

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

5.7.2 Header file structure

[EcuM2991] ⌈The ECU Manager module implementation shall provide a file named `EcuM.h` which contains fix type declarations, forward declarations to generated types, and function prototypes.⌋(BSW00447)

[EcuM2992] ⌈The ECU Manager module implementation shall provide a `EcuM_Generated_Types.h` file which contains generated type declarations that fulfill the forward declarations in `EcuM.h`.⌋(BSW00447)

[EcuM2993] ⌈The ECU Manager module implementation shall provide a `EcuM_Cfg.h` file which contains the configuration parameters.⌋(BSW00412, BSW00447)

[EcuM2994] ⌈The ECU Manager module implementation shall provide a `EcuM_Cbk.h` file which contains the callback/callout function prototypes.⌋(BSW00447)

[EcuM2676] ⌈`EcuM.h` shall contain API interface declarations for all ECU Manager module services.⌋(BSW00447)

[EcuM2677] ⌈`IEcuM_Cbk.h` shall contain all declarations necessary to interact with the callbacks and callouts of the ECU Manager module.⌋(BSW00447)

[EcuM2862] ⌈The ECU Manager module implementation shall include `SchM_EcuM.h` and `MemMap.h`.⌋(BSW00435,BSW00436, BSW00447)

Rationale for EcuM2862: `MemMap.h` makes it possible to map the code and the data of the ECU Manager module into specific memory sections.

[EcuM2875] ⌈The ECU Manager module shall include the `Dem.h` file.which contains the API and Event Id symbol definitions required to report errors. This document defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the

Event Id symbols and publishes the symbols in `Dem_IntErrId.h`.](BSW00385,BSW00409, BSW00447)

[EcuM3025] [The file EcuM_Types.h shall include Rte_EcuM_Type.h to include the types which are common used by BSW Modules and Software Components. EcuM_Types.h and EcuM.h shall only contain types, that are not already defined in Rte_EcuM_Type.h.](BSW00447)

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	EcuM3007
[BSW00404]	Reference to post build time configuration	
[BSW00405]	Reference to multiple configuration sets	
[BSW00345]	Pre-compile-time configuration	EcuM3007
[BSW159]	Tool-based configuration	not applicable (EcuM does not specify the configuration tool)
[BSW167]	Static configuration checking	not applicable (EcuM does not specify rules for configuration checking)
[BSW171]	Configurability of optional functionality	*10* Introduction and Functional Overview
[BSW00380]	Separate C-files for configuration parameters	*5.7* 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	EcuM2875
[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] and [BSW00438]		*10* Configuration specification
[BSW00402]	Published information	*EcuM001_PI*
[BSW00375]	Notification of wakeup reason	*8.3.4* Wakeup *EcumWakeupSource* (*EcuM150_Conf*)
[BSW101]	Initialization interface	*8.3.2.4* 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.)
[BSW00437]	NoInit--Area in RAM	not applicable
[BSW00435]	Header File Structure for the Basic Software Scheduler	EcuM2862
[BSW00	Module Header File Structure for the Basic	EcuM2862

436]	Software Memory Mapping	
[BSW00 447]	Standardizing Include file structure of BSW Modules Implementing Autosar Service	5.7.2 Header file structure
[BSW16 8]	Diagnostic Interface of SW components	not applicable (EcuM has no testing requirements)
[BSW00 407]	Function to read out published parameters	*8.3.1.1* EcuM_GetVersionInfo
[BSW00 423]	Usage of SW-C template to describe BSW modules with AUTOSAR interfaces	*7.10* AUTOSAR Ports
[BSW00 424]	BSW main processing function task allocation	Implementation of EcuM_MainFunction (see EcuM2837) according to this specification does not require extended task mechanisms.
[BSW00 425]	Trigger conditions for schedulable objects	*8.4.1* EcuM_MainFunction
[BSW00 426]	Exclusive areas in BSW modules	not applicable (EcuM does not specify directly accessible global data.)
[BSW00 427]	ISR description for BSW modules	not applicable (EcuM does not specify ISRs.)
[BSW00 428]	Execution order dependencies of main processing functions	There are no requirements of this sort.
[BSW00 429]	Restricted BSW OS functionality access	EcuM does not use any other than the allowed OS services.
[BSW00 431]	The BSW Scheduler module implements task bodies	EcuM does not define any task body.
[BSW00 432]	Modules should have separated main processing functions for a read/receive and write/transmit data path	not applicable (EcuM does not specify RxTx functionality.)
[BSW00 433]	Calling of main processing functions	EcuM does not call any main processing function.
[BSW00 434]	The Schedule Module shall provide an API for exclusive areas	not applicable (This is not an EcuM requirement)
[BSW00 336]	Shutdown interface	*8.3.2.6* EcuM_Shutdown
Fault Operation and Error Detection		
[BSW00 337]	Classification of errors	Table 7 - Error Classification
[BSW00 338]	Detection and reporting of development errors	Table 7 - Error Classification EcuM2983, EcuM2984
[BSW00 369]	Do not return development error codes via API	EcuM2986
[BSW00 339]	Reporting of production relevant error statuses	EcuM1987
[BSW00 417]	Reporting of Error Events by Non-Basic Software	not applicable
[BSW00 422]	Pre--de--bouncing of production relevant error status	not applicable
[BSW00 323]	API parameter checking	API Parameter Checking *EcuM3009*
[BSW00 4]	Version check	[
[BSW00 409]	Header files for production code error IDs	*5.7* File Structure
[BSW00 385]	List possible error notifications	Table 7 - Error Classification
[BSW00	Configuration for detecting errors	*7.12* Error Classification

386]		
[BSW161]	Microcontroller abstraction	not applicable **ECUM150_Conf :** (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.	*5.7* File Structure
[BSW00343]	Specification and configuration of time	*EcuMAlarmClock (EcuM184_Conf)*
[BSW160]	Human-readable configuration data	not applicable (This specification does not define the configuration file)
[BSW00453]	Harmonization of BSW Modules	not applicable
[BSW00456]	Header file for Harmonizing BSW Modules	*5.7* File Structure
[BSW00457]	Callback functions of software components	*8* API specification
[BSW007]	HIS MISRA C	The API definition complies with MISRA C. *8* API specification
[BSW00300]	Module naming conventions.	*5.7* 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
[BSW00450]	Main Function Processing for Un-Initialized Modules	not applicable
[BSW00327]	Error values naming convention	Table 7 - Error Classification
[BSW00335]	Status values naming convention	*8.2* Type definitions
[BSW00350]	Development error detection keyword	Table 7 - Error Classification
[BSW00442]	Debugging Support in Modules	**[**
[BSW00408]	Configuration parameter naming convention	10 Configuration specification
[BSW00410]	Compiler switches shall have defined values	not applicable (This specification does not define compiler switchers)

[BSW00411]	Get version info keyword	EcuM2813
[BSW00346]	Basic set of module files	*5.7* File Structure
[BSW158]	Separation of configuration from implementation	*5.7* File Structure *10* Configuration specification
[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	*8* API specification *7* Functional Specification
[BSW00312]	Shared code shall be re-entrant	*8* API specification *7* Functional Specification
[BSW006]	Platform independency	*8* API specification
[BSW00439]	Declaration of interrupt handlers and ISRs	Not applicable
[BSW00448]	Module SWS shall not contain requirements from Other Modules	*See chapter 1- 10*
[BSW00449]	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	Not applicable
[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
[BSW00	Return type of callback functions	*8.5* Callback Definitions

359]		
[BSW00 360]	Parameters of callback functions	*8.5* Callback Definitions
[BSW00 440]	Function prototype for callback functions of AUTOSAR Services	*8.5* Callback Definitions
[BSW00 329]	Avoidance of generic interfaces	*8* API specification
[BSW00 330]	Usage of macros/inline functions instead of functions	not applicable (Requirement to implementation)
[BSW00 331]	Separation of error and status values	*8.2* Type definitions
[BSW00 443]	Enabling / disabling defensive behavior of BSW	EcuM196_Conf
[BSW00 444]	Error reporting and logging for defensive behavior of BSW	**[**
[BSW00 445]	Protection against untimely call of BSW initialization	Table 7 - Error Classification
[BSW00 446]	Protection against untimely call of BSW de-initialization	not applicable
[BSW00 9]	Module user documentation	Fulfilled by usage of template/formal review
[BSW00 401]	Documentation of multiple instances of configuration parameters	*10* Configuration specification
[BSW17 2]	Compatibility and documentation of scheduling strategy	EcuM2836
[BSW01 0]	Memory resource documentation	not applicable (requirement to implementation)
[BSW00 333]	Documentation of callback function context	*8.5* Callback Definitions
[BSW00 374]	Module vendor identification	EcuM2728, EcuM2729
[BSW00 379]	Module identification	EcuM2728, EcuM2729
[BSW00 3]	Version identification	EcuM4034
[BSW00 318]	Format of module version numbers	EcuM4034
[BSW00 321]	Enumeration of module version numbers	EcuM4034
[BSW00 341]	Microcontroller compatibility documentation	not applicable (requirement to implementation)
[BSW00 334]	Provision of XML file	not applicable (provided by system team)

Document: Requirements on Mode Management [4]

Requirement		***Satisfied by***
[BSW09 122]	Configuration of users of the ECU Manager	EcuM487,
[BSW09 100]	Selection of wakeup sources shall be configurable	EcuM2389, `EcuMWakeupSource` (EcuM150_Conf)
[BSW09 116]	Requesting and releasing the RUN state	not applicable
[BSW09 114]	Starting/invoking the shutdown process	EcuM2818, EcuM2822, EcuM624, EcuM2185, EcuM2585
[BSW09	ECU Manager shall take over control	EcuM2952, EcuM2953

104]	after OS shutdown	
[BSW09 113]	Initialization of Basic Software modules	Table 2 – StartPostOS Sequence
[BSW09 127]	De-initialization of BSW	*7.4* SHUTDOWN Phase
[BSW09 128]	Support of several shutdown targets	EcuM2822, EcuM2824, EcuM2825 `EcuMDefaultShutdownTarget` (EcuM105_Conf)
[BSW09 119]	Support of several sleep modes	*`EcuMSleepMode`* (EcuM131_Conf), *7.5* SLEEP Phase
[BSW09 102]	API for selecting the sleep mode	not applicable
[BSW09 072]	Force ECU shutdown	*7.1.3* SHUTDOWN Phase, *8.3.3* Shutdown Management
[BSW09 017]	Provide ECU state information	not applicable
[BSW09 136]	Centralized Wakeup Management	*`EcuMWakeupSource`* (EcuM150_Conf), *8.3.4* Wakeup Handling
[BSW09 098]	Registration of wakeup reasons	*9.2* Wakeup Sequences, EcuM2826
[BSW09 097]	Validation of physical channel wakeup	*9.2* Wakeup Sequences
[BSW09 126]	Provide an API for querying of wakeup reason	*EcuM2827, EcuM2828, EcuM2830, EcuM2831*
[BSW09 101]	Provide an API to query the reset reason	not applicable
[BSW09 234]	Initialization of Basic Software modules	*`EcuMDriverInitListZero`* (EcuM114_Conf) *`EcuMDriverInitListOne`* (EcuM111_Conf) *`EcuMDriverRestartList`* (EcuM115_Conf) *`EcuMDriverInittItem`* (EcuM110_Conf) EcuM2559, EcuM2730, EcuM2947
[BSW09 235]	Support of several shutdown targets	8.3.3 Shutdown Management; *`EcuMDefaultShutdownTarget`* (EcuM105_Conf)
[BSW09 227]	Configuration of privileged users	*`EcuMUserConfig`* (EcuM147_Conf)
[BSW09 185]	Provide a persistent Alarm Clock to be used by local SW-Cs	7.6.1 Alarm Clock Handling, *`EcuMAlarmClock`* (EcuM184_Conf)
[BSW09 186]	Alarm Clock shall be active while the ECU is powered	**[, [, [, [, [, [, [**
[BSW09 187]	Cancellation of Alarms in Case of wakeup	**[**
[BSW09 188]	Cancellation of Alarms in Case of startup	**[**
[BSW09 189]	Consideration of the earliest expiring Wakeup only	7.8 Alarm Clock
[BSW09 190]	Provision of an Interface to set relative Alarms	**[**
[BSW09 199]	Provision of an Interface to set absolute Alarms	**[**
[BSW09 194]	Provision of an Interface to set the Clock	**[**
[BSW09 195]	Protection against untimely Call of EcuM_Init	not applicable
[BSW09 197]	Protection against erroneous Call of EcuM_KillAllRUNRequests	not applicable
[BSW09 198]	Protection against erroneous Call of EcuM_SelectShutdownTarget	not applicable

[BSW09 236]	Distinguish cores	7.9 MultiCore
[BSW09 237]	Starting of RTE	not applicable
[BSW09 238]	State changes are ECU global	7.9 MultiCore
[BSW09 239]	Synchronized Shutdown	7.9 MultiCore

7 Functional Specification

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

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

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

This document only specifies the functionality that remains in the ECU Manager module. For a complete picture of ECU State Management, refer to the specifications of the other relevant modules, i.e., RTE and BSW Scheduler module [14] and BSW Mode Manager module [21].

Refer to the Guide to Mode Management [23] for some example use cases for ECU states and the interaction between the involved BSW modules.

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

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

7.1 Phases of the ECU Manager Module

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

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

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

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

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

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

Figure 2 shows an overview over the the phases of the Flexible ECU Manager module.
The STARTUP phase lasts until the mode management facitliies are running. Basically the STARTUP phase consists of the minimal activities needed to start mode management: initializing low-level drivers, starting the OS and initializing the BSW Scheduler and the BSW Mode Manager modules. Similarly the SHUTDOWN phase is the reverse of the STARTUP phase is where mode management is de-initialized.

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

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

A diagram which maps the new *Phases* to the old *States* of the ECU Manager of AUTOSAR 3 and to the *States* of the ECU Manager Fixed of AUTOSAR 4 can be found in the “Guide to Mode Management” [23].

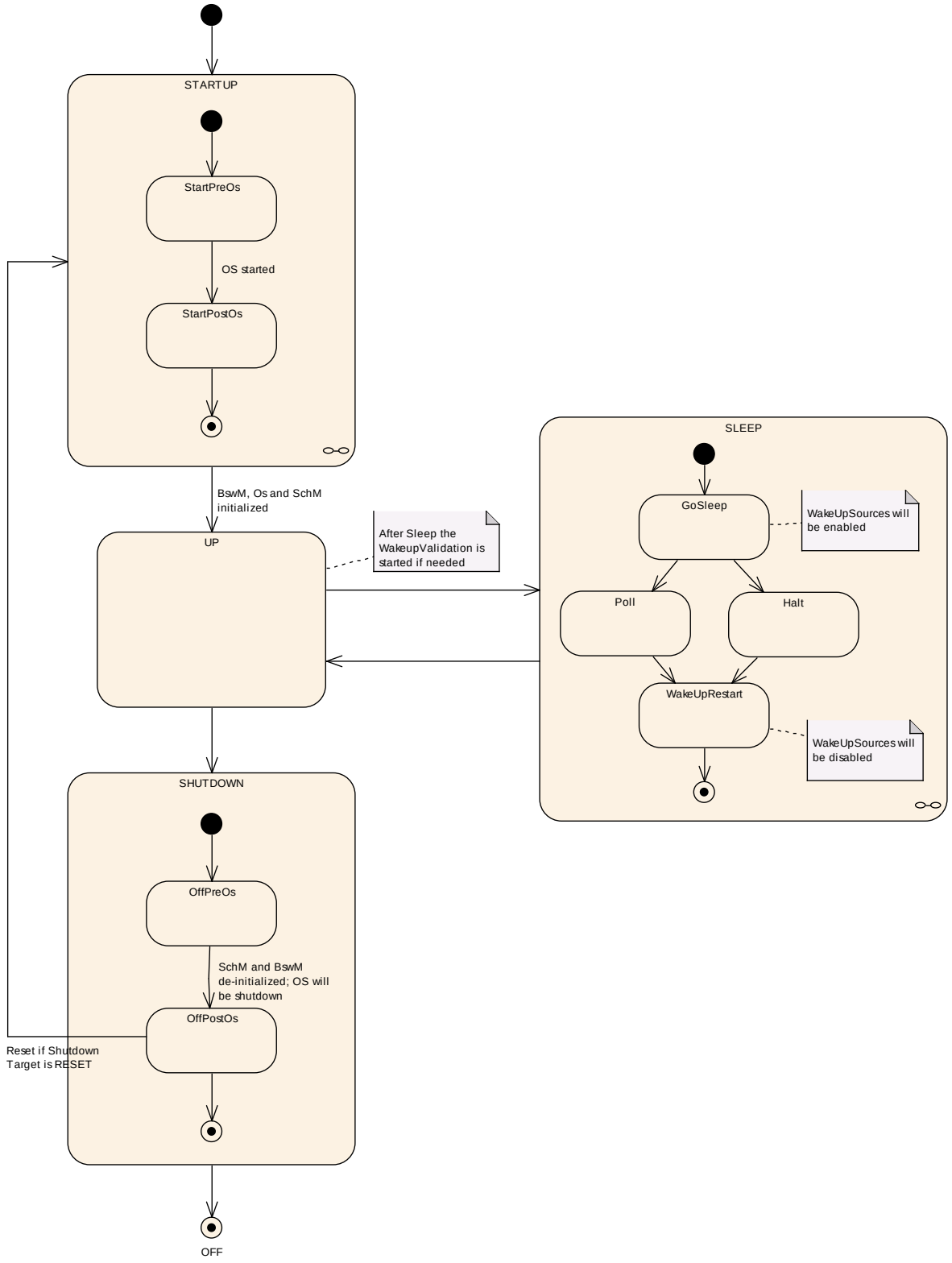

Figure 2 – Phases of the ECU Manager

7.1.1 STARTUP Phase

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

7.1.2 UP Phase

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

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

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

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

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

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

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

Refer to the Guide to Mode Management [23] for guidance on how to design mode-driven ECU management and for configuring the BSW Mode Manager accordingly.

7.1.3 SHUTDOWN Phase

[EcuM3022][The SHUTDOWN phase handles the controlled shutdown of basic software modules and finally results in the selected shutdown target OFF or RESET.](BSW09072)

7.1.4 SLEEP Phase

The ECU saves energy in the SLEEP phase. Typically, no code is executed but power is still supplied, and if configured accordingly, the ECU is wakeable in this state[2]. The ECU Manager module provides a configurable set of (hardware) sleep modes which typically are a trade off between power consumption and time to restart the ECU.

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

7.1.5 OFF Phase

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

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

7.2 Structural Description of the ECU Manager

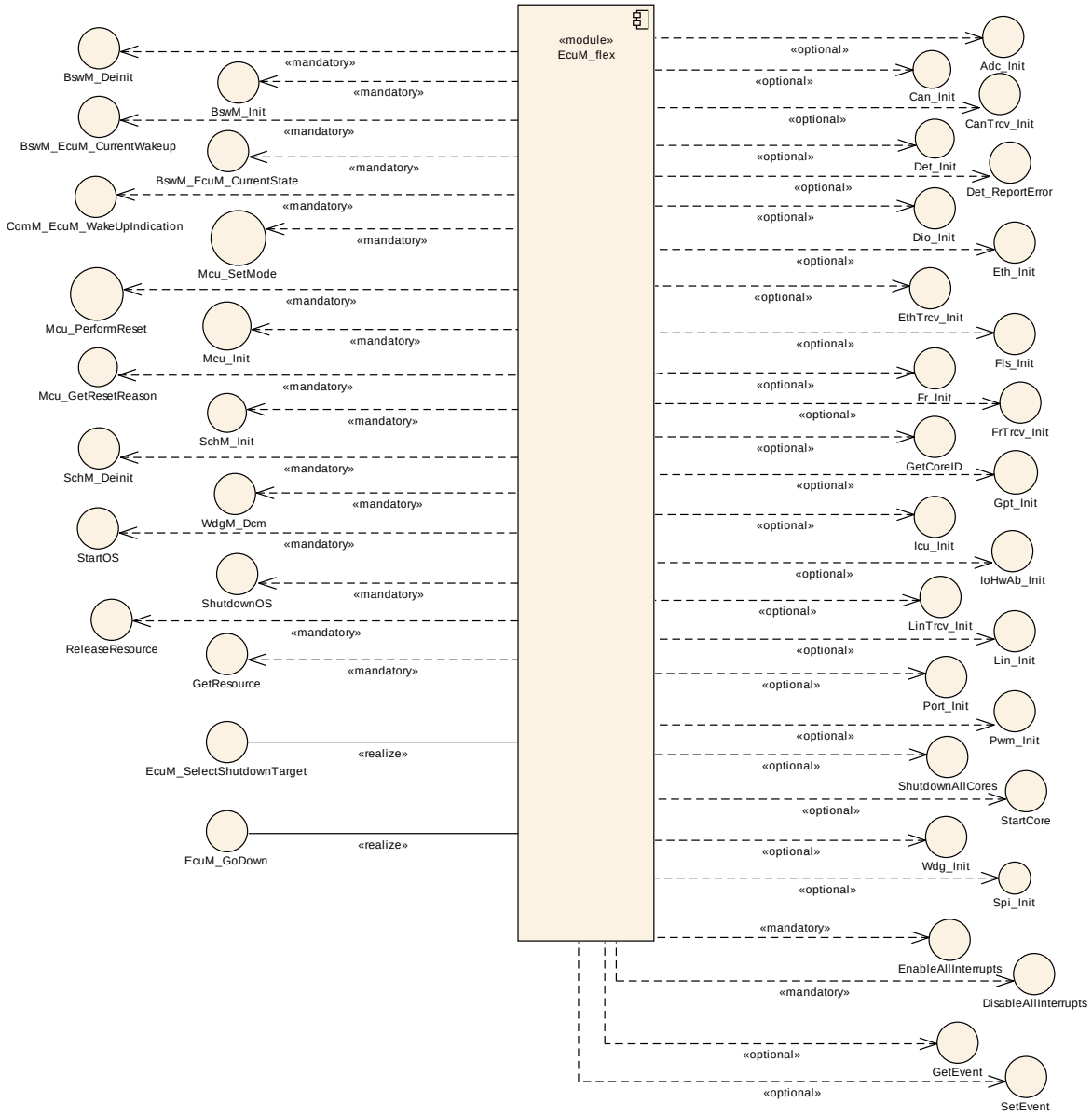

Figure 3 – ECU Manager Module Relationships

Figure 3 illustrates the ECU Manager module's relationship to the interfaces of other BSW modules. In most cases, the ECU Manager module is simply responsible for initialization[3]. There are however some modules that have a functional relationship with the ECU Manager module, which is explained in the following paragraphs.

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

7.2.1 Standardized AUTOSAR Software Modules

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

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

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

7.2.2 Software Components

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

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

7.3 STARTUP Phase

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

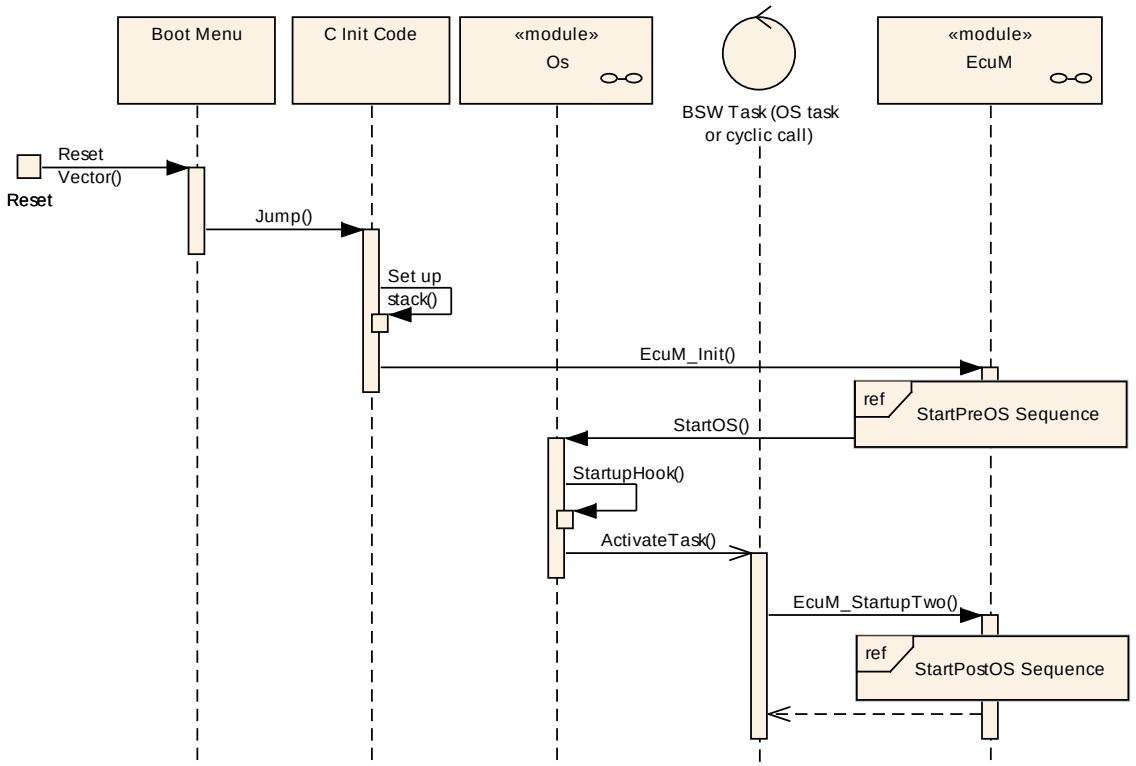

Figure 4: STARTUP Phase

Figure 4 shows the startup behavior of the ECU. When invoked through EcuM_Init, the ECU Manager module takes control of the ECU startup procedure. With the call to `StartOS`, the ECU Manager module temporarily relinquishes control. To regain control, the Integrator has to implement an OS task that is automatically started and calls EcuM_StartupTwo as its first action.

7.3.1 Activities before EcuM_Init

The ECU Manager module assumes that before EcuM_Init (see EcuM2811) is called a minimal initialization of the MCU has taken place, so that a stack is set up and code can be executed, also that C initialization of variables has been performed.

7.3.2 Activities in StartPreOS Sequence

[EcuM2411]⌈Table 1 shows the activities in StartPreOS Sequence and the order in which they shall be executed in EcuM_Init (see EcuM2811).

StartPreOS Sequence			
	Initialization Activity	Comment	Opt.[4]
	Callout EcuM_AL_SetProgrammableInterrupts	On ECUs with programmable interrupt priorities, these priorities must be set before the OS is started.	yes
	Callout EcuM_AL_DriverInitZero	Init block 0 This callout may only initialize BSW modules that do not use post-build configuration parameters. The callout may not only contain driver initialization but also any kind of pre-OS, low level initialization code. See 7.3.5 Driver Initialization	yes
	Callout EcuM_DeterminePbConfiguration	This callout is expected to return a pointer to a fully initialized EcuM_ConfigType structure containing the post-build configuration data for the ECU Manager module and all other BSW modules.	no
	Check consistency of configuration data	If check fails the EcuM_ErrorHook is called. See *7.3.4* Checking Configuration Consistency for details on the consistency check.	no
	Callout EcuM_AL_DriverInitOne	Init block I The callout may not only contain driver initialization but any kind of pre-OS, low level initialization code. See 7.3.5 Driver Initialization	yes
	Get reset reason	The reset reason is derived from a call to `Mcu_GetResetReason` and the mapping defined via the `EcuMWakeupSource` configuration containers. See 8.5.1.2 EcuM_SetWakeupEvent and 8.3.4.3 EcuM_GetValidatedWakeupEvents (see EcuM2830)	no
	Select default shutdown target	See EcuM2181	no
	Start OS	Start the AUTOSAR OS, see EcuM2603 and EcuM2141	no

Table 1 – StartPreOS Sequence

⌋()

[EcuM2623] ⌈The ECU Manager module shall remember the wakeup source resulting from the reset reason translation (see table 1).⌋()

Rationale for EcuM2623: The wakeup sources must be validated by the EcuM_MainFunction (see section 7.6.4 Activities in the WakeupValidation Sequence).

[EcuM2684] ⌈When activated through the EcuM_Init (see EcuM2811) function, the ECU Manager module shall perform the actions in the StartPreOS Sequence (see Table 1 – StartPreOS Sequence).⌋()

[4] Optional activities can be switched on or off by configuration. See section 10.2 Common Containers and configuration parameters for details.

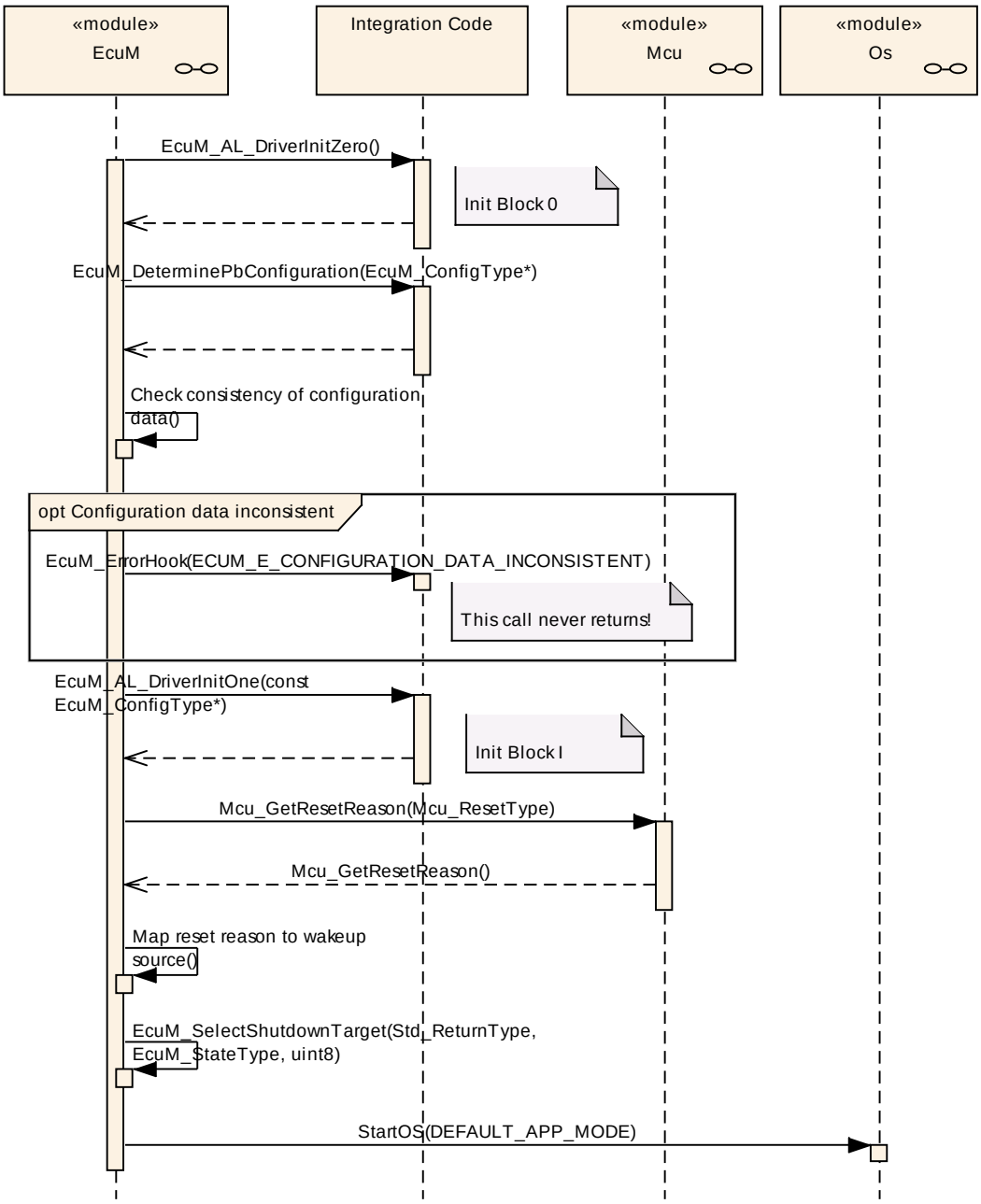

Figure 5 – StartPreOS Sequence

The StartPreOS Sequence is intended to prepare the ECU to initialize the OS and should be kept as short as possible. Drivers should be initialised in the UP phase when possible and the callouts should also be kept short. Interrupts should not be used during this sequence. If interrupts have to be used, only category I interrupts are allowed in the StartPreOS Sequence 1[5].

[5] 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.

Initialization of drivers and hardware abstraction modules is not strictly defined by the ECU Manager. Two callouts EcuM_AL_DriverInitZero (see EcuM2905) and EcuM_AL_DriverInitOne (see EcuM2907) are provided to define the init blocks 0 and I. These blocks contain the initialization activities associated with the StartPreOS sequence.

`MCU_Init` does not provide complete MCU initialization. Additionally, hardware dependent steps have to be executed and must be defined at system design time. These steps are supposed to be taken within the EcuM_AL_DriverInitZero (see *8.6.2.2* EcuM_AL_DriverInitZero, EcuM2905) or EcuM_AL_DriverInitOne callouts (see *8.6.2.4* EcuM_AL_DriverInitOne, EcuM2907). Details can be found in the Specification of MCU Driver [9].

[EcuM2181] ⌈The ECU Manager module shall call EcuM_SelectShutdownTarget (see EcuM2822) with the configured default shutdown target (see section *7.7* Shutdown Targets and `EcuMDefaultShutdownTarget` `EcuM105_Conf`).⌋()

[EcuM2603] ⌈The StartPreOS Sequence shall initialize all basic software modules that are needed to start the OS.⌋()

7.3.3 Activities in the StartPostOS Sequence

StartPostOS Sequence			
	Initialization Activity	Comment	Opt.[6]
	Init BSW Scheduler	Initialize the semaphores for critical sections used by BSW modules	no
	Init BSW Mode Manager		no

Table 2 – StartPostOS Sequence

[EcuM2932] ⌈When activated through the EcuM_StartupTwo (see EcuM2838) function, the ECU Manager module shall perform the actions in StartPostOS Sequence (see Table 2 – StartPostOS Sequence).⌋(BSW09113)

[6] Optional activities can be switched on or off by configuration. See section 10.2 Common Containers and configuration parameters for details.

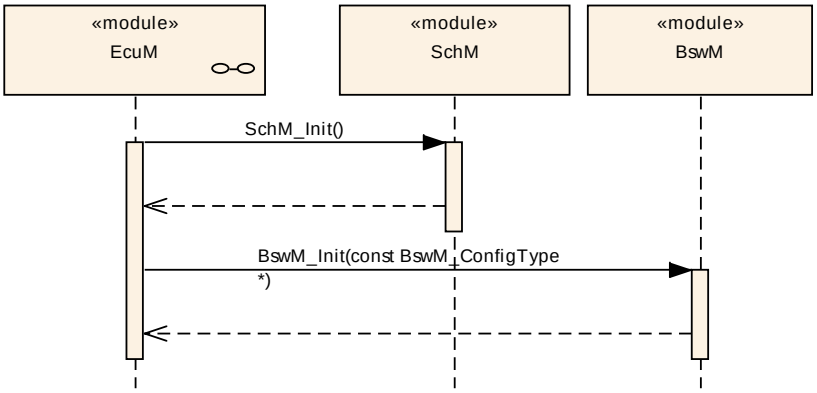

Figure 6 – StartPostOS Sequence

7.3.4 Checking Configuration Consistency

7.3.4.1 The Necessity for Checking Configuration Consistency in the ECU Manager

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

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

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

are exactly the same as

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

This can only be done at run-time.

Explanation for EcuM2796: The ECU Manager module checks the consistency once before initializing the first BSW module to avoid multiple checks scattered over the different BSW modules.

This also implies that:

[EcuM2796] ⌈The ECU Manager module shall not only check the consistency of its own parameters but of all post-build configurable BSW modules before initializing the first BSW module.⌋()

The ECU Manager Configuration Tool must compute a hash value over all pre-compile and link-time configuration parameters of all BSW modules and store the value in the link-time *ECUM_CONFIGCONSISTENCY_HASH* (see ECUM102_Conf) configuration parameter. The hash value is necessary for two reasons. First, the pre-compile and link-time parameters are not accessible at run-time. Second, the check must be very efficient at run-time. Comparing hundreds of parameters would cause an unacceptable delay in the ECU startup process.

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

[EcuM2798] ⌈The ECU Manager module shall check in EcuM_Init (see EcuM2811) that the field in the structure is equal to the value of *ECUM_CONFIGCONSISTENCY_HASH.*⌋()

By computing hash values at configuration time and comparing them at run-time the EcuM code can be very efficient and is furthermore independent of a particular hash computation algorithm. This allows the use of complex hash computation algorithms, e.g. cryptographically strong hash functions.

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

[EcuM2799] ⌈The hash computation algorithm used to compute a hash value over all pre-compile and link-time configuration parameters of all BSW modules shall always

produce the same hash value for the same set of configuration data regardless of the order of configuration parameters in the XML files.]()

7.3.4.2 Example Hash Computation Algorithm

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

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

Additionally, not only the values of the configuration parameters but also their names must be taken into account in the hash algorithm. One possibility is to build a text file that contains the names of the configuration parameters and containers, separate them from the values using a delimiter, e.g. a colon, and putting each parameter as a line into a text file.

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

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

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

7.3.5 Driver Initialization

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

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

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

[EcuM2559] [The configuration of the ECU Manager module shall specify the order of initialization calls inside init block 0 and init block 1. (see `EcuMDriverInitListZero` EcuM114_Conf and `EcuMDriverInitListOne` EcuM111_Conf).](BSW00416,BSW09234)
[EcuM2730] [The ECU Manager module shall call each driver’s init function with the parameters derived from the driver’s `EcuMModuleService` configuration container (see EcuM124_Conf).](BSW09234)

[EcuM2947] ⌈For re-initialization during WakeupRestart, the integrator shall integrate a restart block into the integration code for EcuM_AL_DriverRestart (see EcuM2923) using the `EcuMDriverRestartList` (see EcuM115_Conf)⌋(BSW09234)

[EcuM2562] ⌈`EcuMDriverRestartList` (see EcuM115_Conf) may contain drivers that serve as wakeup sources. EcuM_AL_DriverRestart (see EcuM2923) shall re-arm the trigger mechanism of these drivers' 'wakeup detected' callback (see Section 7.6.4 Activities in the WakeupRestart Sequence).⌋()

[EcuM2563] ⌈When hardware has been put into a sleep mode during SHUTDOWN then this hardware must be restarted by its driver. The ECU Manager module shall invoke in the WakeupRestart Sequence (see Section 7.6.4 Activities in the WakeupRestart Sequence).⌋()

[EcuM2561] ⌈The ECU Manager module shall initialize the drivers in `EcuMDriverRestartList` in the same order as in the combined list of init block 0 and init block 1.⌋()

Hint for EcuM2561: `EcuMDriverRestartList` will typically only contain a subset of the combined list of init block 0 and init block 1 drivers.

Table 3 shows one possible (and recommended) sequence of activities for the Init Blocks 0 and I. Depending on hardware and software configuration, BSW modules may be added or left out and other sequences may also be possible.

Recommended Init Block		
	Init Activity	Comment
Init Block 0[7]		
	Development Error Tracer	This should always be the first module to be initialized, so that other modules can report development errors.
	Diagnostic Event Manager	Pre-Initialization
	Any drivers needed to access post-build configuration data	These drivers shall not depend on the post-build configuration or on OS features.
Init Block I[8]		
	MCU Driver	
	General Purpose Timer	
	Watchdog Driver	Internal watchdogs only, external ones may need SPI
	Watchdog Manager	
	ADC Driver	
	ICU Driver	
	PWM Driver	

Table 3 - Driver Initialization Details, Sample Configuration

[7] Drivers in Init Block 0 are listed in the *`EcuMDriverInitListZero`* configuration container.
[8] Drivers in Init Block I are listed in the *`EcuMDriverInitListOne`* configuration container.

7.3.6 DET Initialization

The Development Error Tracer module is a BSW module which contains software used for debugging. The DET must be both initialized (by calling `Det_Init`) and started (by calling `Det_Start`) before becoming operational. Refer to [17] Specification of Development Error Tracer for details.

In production environments, the DET module must not be compiled in and in development environments, at least one module must use the DET before its initialization is relevant to the system.

[EcuM2783] ⌈If at least one module is configured to track development errors, the ECU Manager module shall initialize the DET before all other drivers during the StartPreOS sequence (see Section7.3.2 Activities in StartPreOS Sequence).⌋()

Rational for EcuM2783: Other modules cannot report development errors before the DET is initialized.

[EcuM2634] ⌈The ECU Manager module shall not start the DET by default.⌋()

Rationale for EcuM2634: The system designer has to configure the point where DET is started, preferably into the EcuM_AL_DriverInitOne callout (see EcuM2907). The best point for starting DET depends on its implementation and behavior.

7.4 SHUTDOWN Phase

Refer to Section *7.1.3* SHUTDOWN Phase for an overview of the SHUTDOWN phase. `EcuM_GoOff` initiates the SHUTDOWN Phase.

[EcuM2756]⌈When a wakeup event occurs during the shutdown phase, the ECU Manager module shall complete the shutdown and restart immediately thereafter.⌋

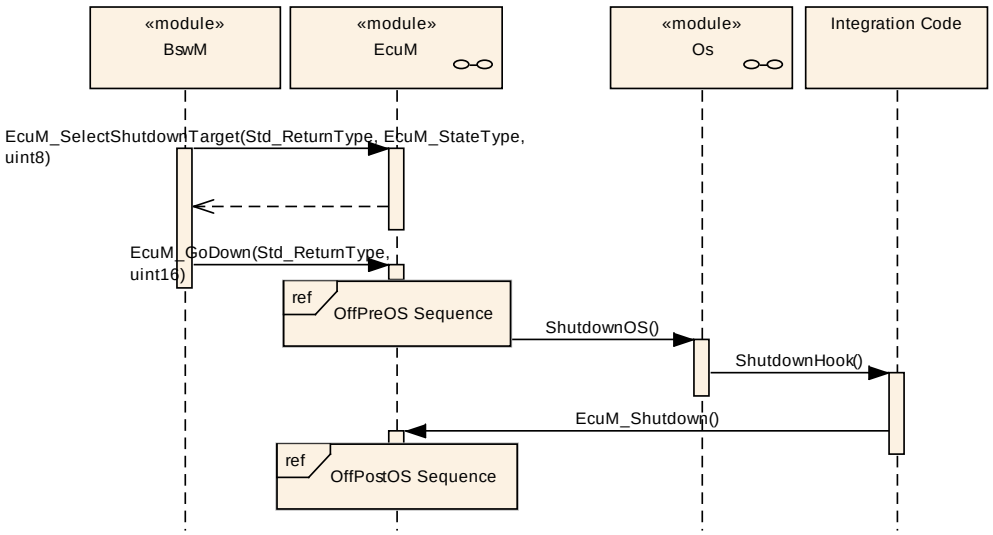

Figure 8 – SHUTDOWN Phase

7.4.1 Activities in the OffPreOS Sequence

[EcuM3021] ⌈

OffPreOS Sequence			
	Shutdown Activity	Comment	Opt.[9]
	De-init BSW Mode Manager		no
	De-init BSW Scheduler		no
	Check for pending wakeup events	Purpose is to detect wakeup events that occurred during shutdown	no
	Set RESET as shutdown target, if wakeup events are pending	This action shall only be carried out when pending wakeup events were detected to allow an immediate startup	no
	ShutdownOS	Last operation in this OS task	no

Table 4 – OffPreOS Sequence

⌋(BSW09127)

[EcuM2952] ⌈As its last activity, the ECU Manager module shall call the `ShutdownOS` function.⌋(BSW09104)

[9] 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 . See section 10.2 Common Containers and configuration parameters for details.

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

[EcuM2953] ⌈The shutdown hook shall call EcuM_Shutdown (see EcuM2812) to terminate the shutdown process. EcuM_Shutdown (see EcuM2812) shall not return but switch off the ECU or issue a reset.⌋(BSW09104)

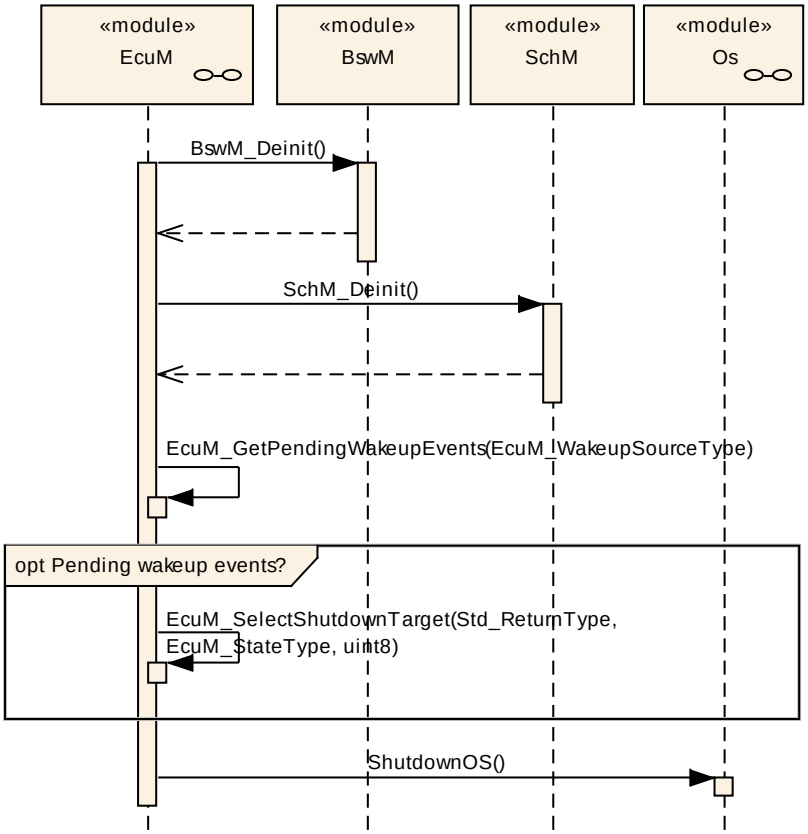

Figure 9 – OffPreOS Sequence

7.4.2 Activities in the OffPostOS Sequence

The OffPostOS sequence implements the final steps to reach the shutdown target after the OS has been shut down. EcuM_Shutdown (see EcuM2812) initiates the sequence.

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

OffPostOS Seqeunce			
	Shutdown Activity	Comment	Opt.[10]
	Callout `EcuM_OnGoOffTwo`		no
	Callout EcuM_AL_Reset or Callout EcuM_AL_SwitchOff	Depends on the selected shutdown target (RESET or OFF)	no

Table 5 – OffPostOS Sequence

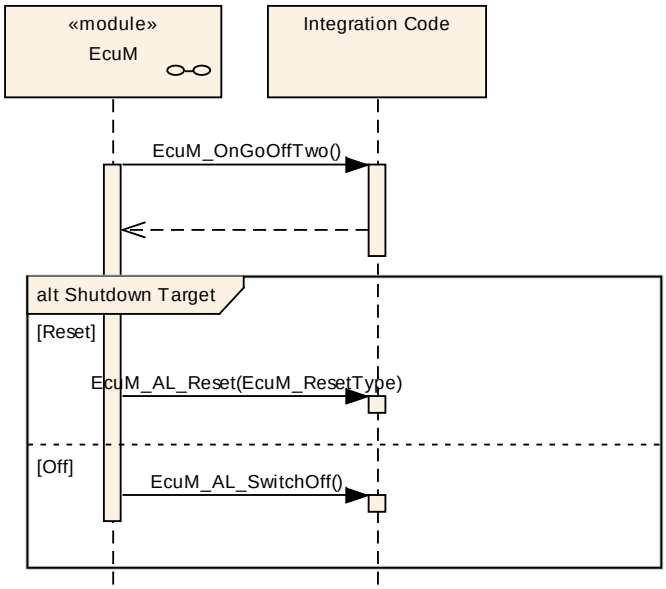

Figure 10 – OffPostOS Sequence

[EcuM4074] ⌈When the shutdown target is RESET, the ECU Manager module shall call the EcuM_AL_Reset callout. See section 8.6.3.4 EcuM_AL_Reset (EcuM4065) for details.⌋()

[EcuM4075] ⌈When the shutdown target is OFF, the ECU Manager module shall call the EcuM_AL_SwitchOff callout. See section *8.6.3.3* EcuM_AL_SwitchOff (EcuM2920) for details.⌋()

7.5 SLEEP Phase

Refer to Section *7.1.4* SLEEP Phase for an overview of the SLEEP phase. EcuM_GoHalt or EcuM_GoPoll initiate the SLEEP phase.

EcuM_GoHalt and EcuM_GoPoll initiate two control streams that differ structurally in the mechanisms used to realize sleep. They share the sequences for preparing for and recovering from sleep, however.

[10] 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. See section 10.2 Common Containers and configuration parameters for details.

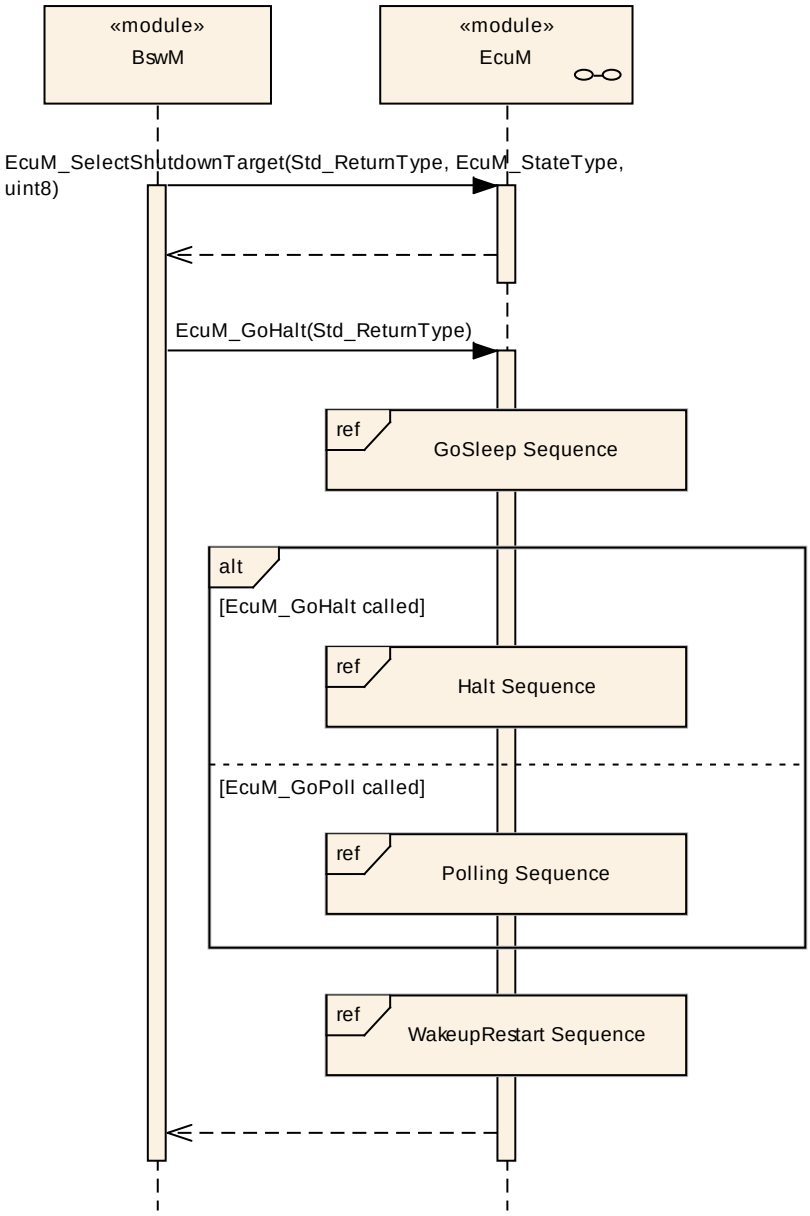

Figure 11 – SLEEP Phase

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

7.5.1 Activities in the GoSleep Sequence

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

[EcuM2389][To set the wakeup sources up for the next sleep mode, the ECU Manager module shall execute the EcuM_EnableWakeupSources callout (see

EcuM2546) for each wakeup source that is configured in `EcuMWakeupSourceMask` (see EcuM152_Conf) for the target sleep mode.](BSW09100)

[EcuM2951] [In contrast to the SHUTDOWN phase, the ECU Manager module shall not shut down the OS when entering the SLEEP phase. The sleep mode, i.e. combination of the EcuM SLEEP phase and the Mcu Mode, shall be transparent to the OS.]()

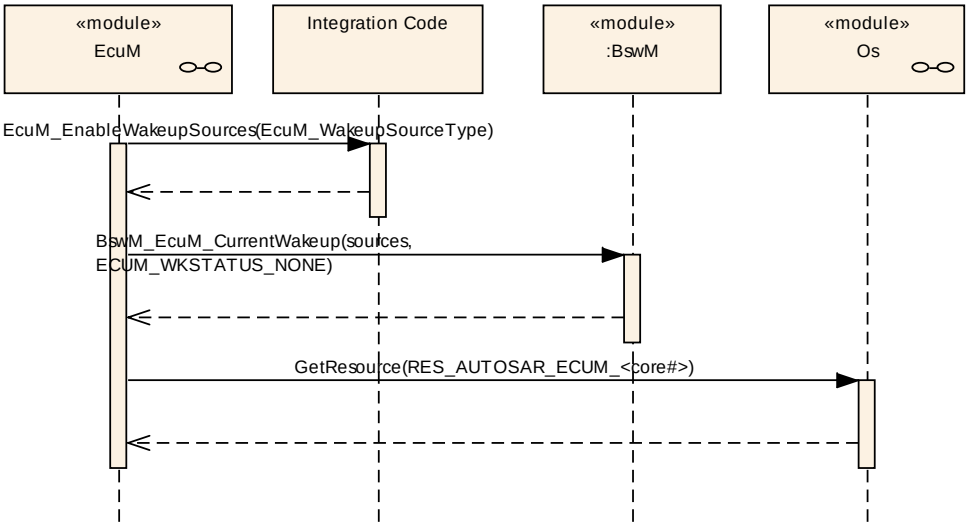

Figure 12 – GoSleep Sequence

[EcuM3010] [When operating on a multicore ECU ECUM shall reserve a dedicated resource (RES_AUTOSAR_ECUM) for each core, which is allocated during GoSleep.]()

7.5.2 Activities in the Halt Sequence

[EcuM2960] [The ECU Manager module shall execute the Halt Sequence in sleep modes that halt the microcontroller. In these sleep modes the ECU Manager module does not execute any code.]()

[EcuM2863] [The ECU Manager module shall invoke the EcuM_GenerateRamHash (see EcuM2919) callout before halting the microcontroller the `EcuM_CheckRamHash` (see EcuM2921) callout after the processor returns from halt.
In case of applied multi core and existence of "slave" EcuM(s) this check should be executed on the "master" EcuM only. The "master" EcuM generates the hash out of all data that lie within its reach. Private data of "slave" EcuMs are out of scope.]()

Rationale for 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.

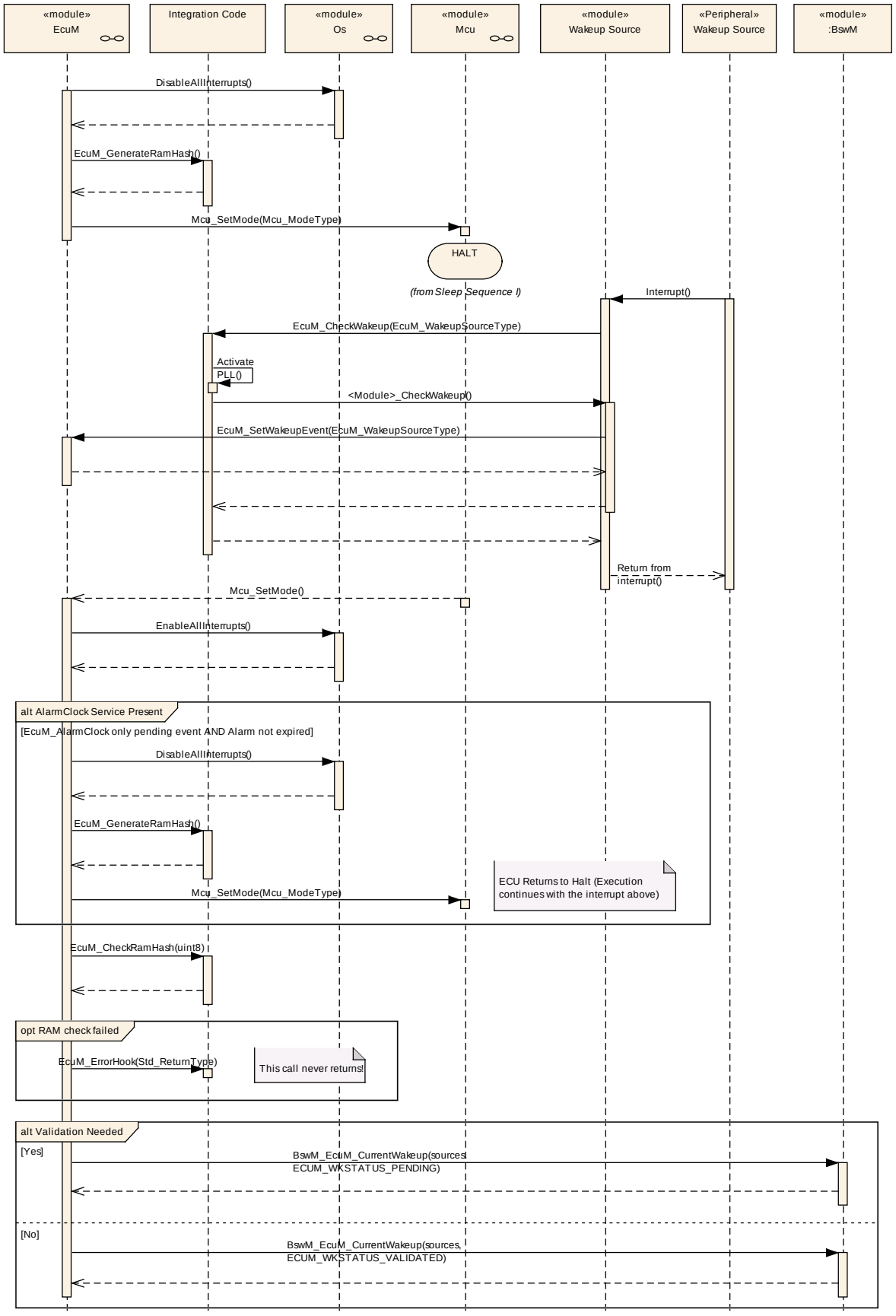

Figure 13 – Halt Sequence

[EcuM2961] ⌈The ECU Manager module shall invoke the EcuM_GenerateRamHash (see EcuM2919) where the system designer can place a RAM integrity check.⌋()

7.5.3 Activities in the Poll Sequence

[EcuM2962] ⌈The ECU Manager module shall execute the Poll Sequence in sleep modes that reduce the power consumption of the microcontroller but still execute code.⌋()
[EcuM3020] ⌈In the Poll sequence the EcuM shall call the callouts EcuM_SleepActivity() and EcuM_CheckWakeup() in a blocking loop until a pending wakeup event is reported.⌋()

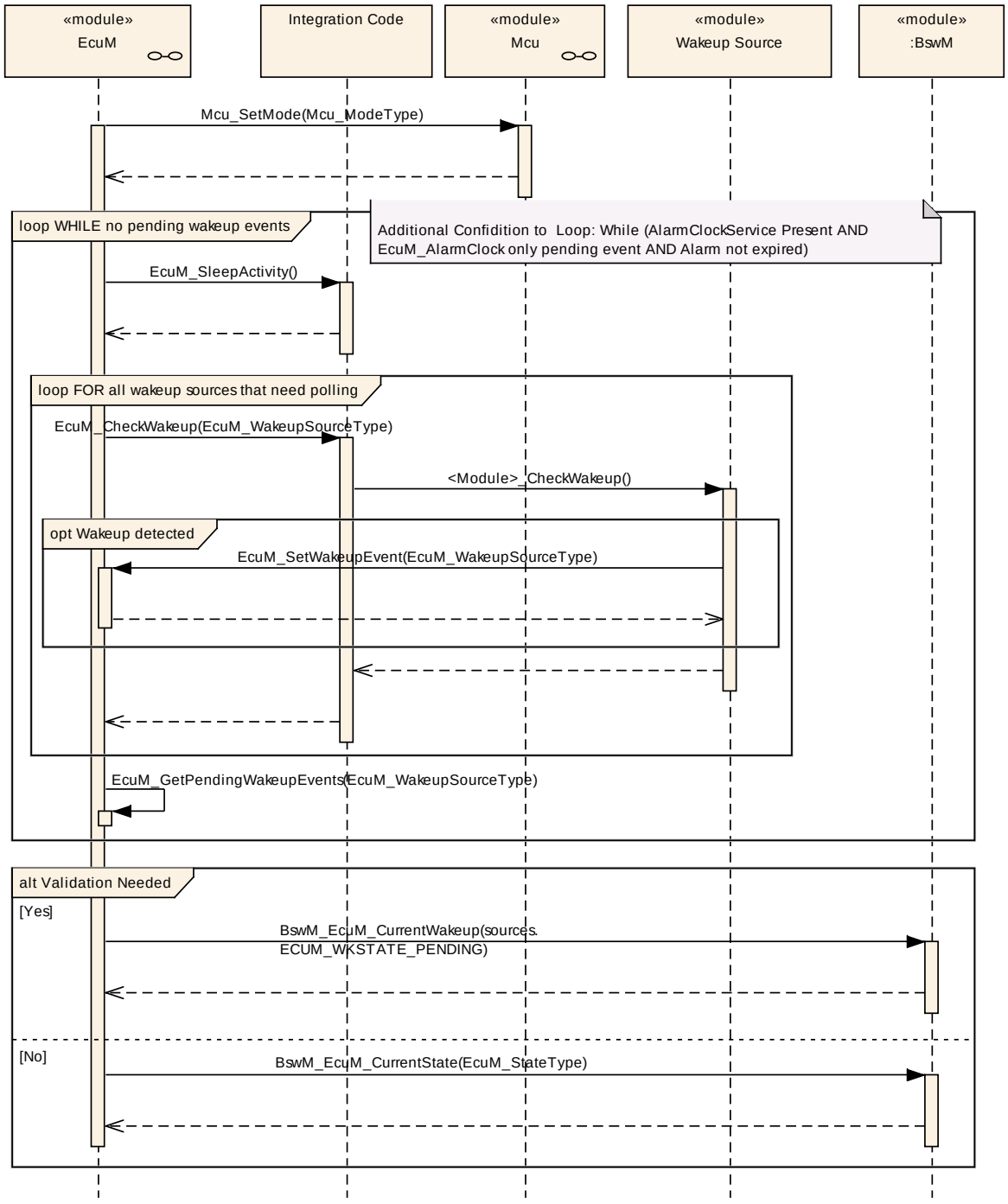

Figure 14 – Poll Sequence

7.5.4 Leaving Halt or Poll

[EcuM2963] [If a wakeup event (e.g. toggling a wakeup line, communication on a CAN bus etc.) occurs while the ECU is in Halt or Poll, then the ECU Manager module shall regain control and exit the SLEEP phase by executing the WakeupRestart sequence (see section 7.5.5 Activities in the WakeupRestart Sequence).

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

[EcuM4001] [If irregular events (a hardware reset or a power cycle) occur while the ECU is in Halt or Poll, the ECU Manager module shall restart the ECU in the STARTUP phase.]()

7.5.5 Activities in the WakeupRestart Sequence

WakeupRestart[11]			
	Wakeup Activity	Comment	Opt.
	Restore MCU normal mode	Selected MCU mode is configured in the configuration parameter `EcuMNormalMcuModeRef`	
	Get the pending wakeup sources		
	Callout EcuM_DisableWakeupSources	Disable currently pending wakeup source but leave the others armed so that later wakeups are possible.	
	Callout EcuM_AL_DriverRestart	Initialize drivers that need restarting	
	Unlock Scheduler	From this point on, all other tasks may run again.	

Table 6 - WakeupRestart Activities

The ECU Manager module invokes the EcuM_AL_DriverRestart (see EcuM2923) callout which is intended for re-initializing drivers. Among others, drivers with wakeup sources typically require re-initialization. For more details on driver initialization refer to section 7.3.5 Driver Initialization.

During re-initialization, a driver must check if one of its assigned wakeup sources was the reason for the previous wakeup. If this test is true, the driver must invoke its 'wakeup detected' callback (see the Specification of CAN Transceiver Driver [18] for an example), which in turn must call the EcuM_SetWakeupEvent (see EcuM2826) function.

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

[EcuM2539] [If the ECU Manager module has a list of wakeup source candidates when the WakeupRestart Sequence has finished, the ECU Manager module shall validate these wakeup source candidates in `EcuM_MainFunction`. See *section 7.6.4* Activities in the WakeupValidation Sequence.*]()*

[11] Rows marked with × are conditional.

[EcuM4066

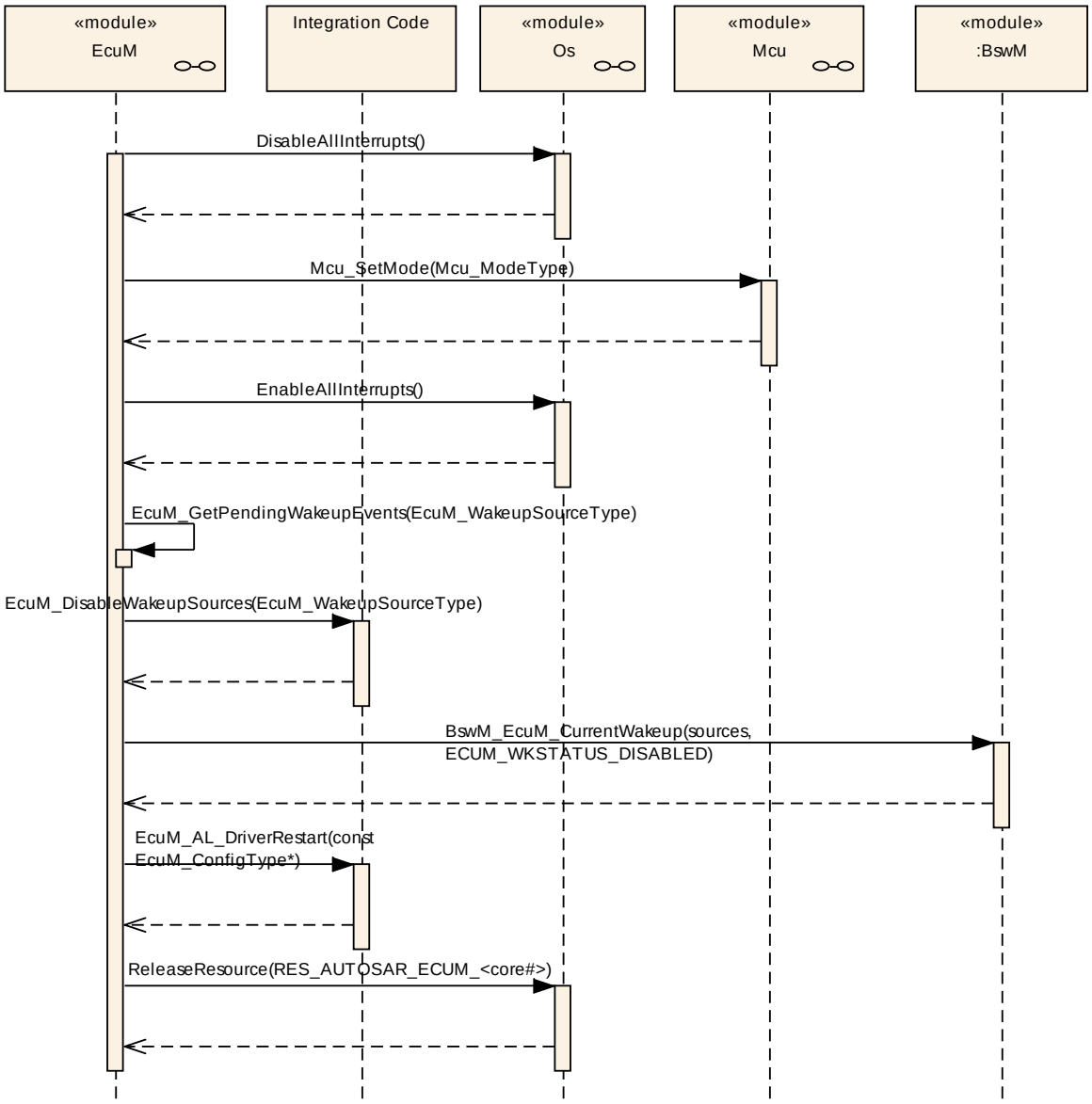

Figure 15 - WakeupRestart Sequence

7.6 UP Phase

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

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

7.6.1 Alarm Clock Handling

See section 7.8.2.1 EcuM Clock Time in the UP Phase for implementation details.

[EcuM4002] ⌈When the Alarm Clock service is present (see `EcuMAlarmClockPresent` EcuM199_Conf) the `EcuM_MainFunction` shall update the Alarm Clock Timer⌋()

7.6.2 Wakeup Source State Handling

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

Wakeup sources can be in the following states:

- NONE - no wakeup event detected or wakeup event cleared,
- PENDING - wakeup event detected but not yet validated,
- VALIDATED - wakeup event detected and validated
- EXPIRED - wakeup event detected and validation failed
- DISABLED – the wakeup source has been disabled and will currently not be processed.

Figure 16 illustrates the relationship between the wakeup source states and the conditions functions that evoke state changes.

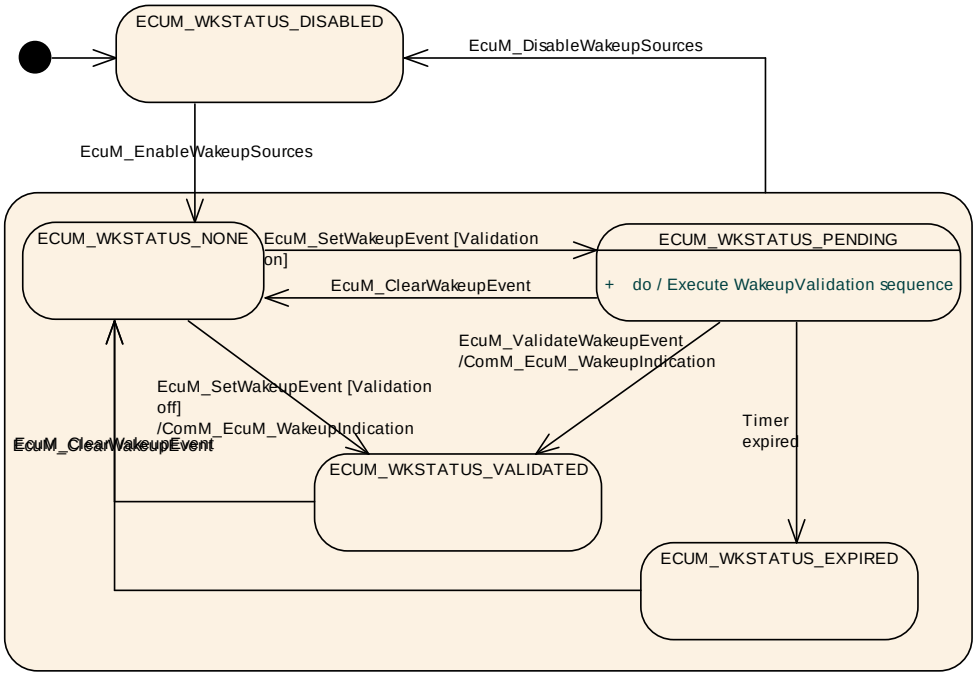

Figure 16 - Wakeup Source States

[EcuM4003] ⌈When an ECU Manager action causes the state of a wakeup source to change, the ECU Manager module shall issue a mode request to the BswM to change the wakeup source's mode to the new the wakeup source state.⌋()

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

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

If the wakeup source is valid, the BswM returns the ECU to its RUN state. If all wakeup events have gone back to NONE or EXPIRED, the BswM prepares the BSW for SLEEP or OFF again and invokes to `EcuM_GoPoll` or `EcuM_GoHalt` or `EcuM_GoDown` depending on the last shutdown target.

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

7.6.3 Internal Representation of Wakeup States

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

- EcuM_GetPendingWakeupEvents
- EcuM_GetValidatedWakeupEvents
- EcuM_GetExpiredWakeupEvents

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

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

The ECU Manager module can manage up to 32 wakeup sources. The state of the wakeup sources is typically represented at the EcuM interfaces named above by means of an EcuM_WakeupSourceType bitmask where the individual wakeup sources correspond to a fixed bit position. There are 5 predefined bit positions and the rest can be assigned by configuration. See section 8.2.4 EcuM_WakeupSourceType for details.

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

variables" in the respective interface definitions to define the semantics of the interface.

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

7.6.4 Activities in the WakeupValidation Sequence

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

The validation mechanism is the same for all wakeup sources. When a wakeup event occurs, the ECU is woken up from its SLEEP state and execution resumes within the `MCU_SetMode` service of the MCU driver[12]. When the WakeupRestart Sequence has finished, the ECU Manager module will have a list of pending wakeup events to be validated (see EcuM2539). The ECU Manager module then releases the BSW Scheduler and all BSW MainFunctions; most notably in this case, the EcuM MainFunction can resume processing.

Implementation hint: Since SchM will be running at the end of the StartPostOS and WakeupRestart sequences, there is the possibility that the EcuM_MainFunction will initiate validation for a source whose stack has not yet been initialized. The integrator should configure appropriate modes which indicate that the stack is not available and disable the `EcuM_MainFunction` accordingly (see [14]).

[12] 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.

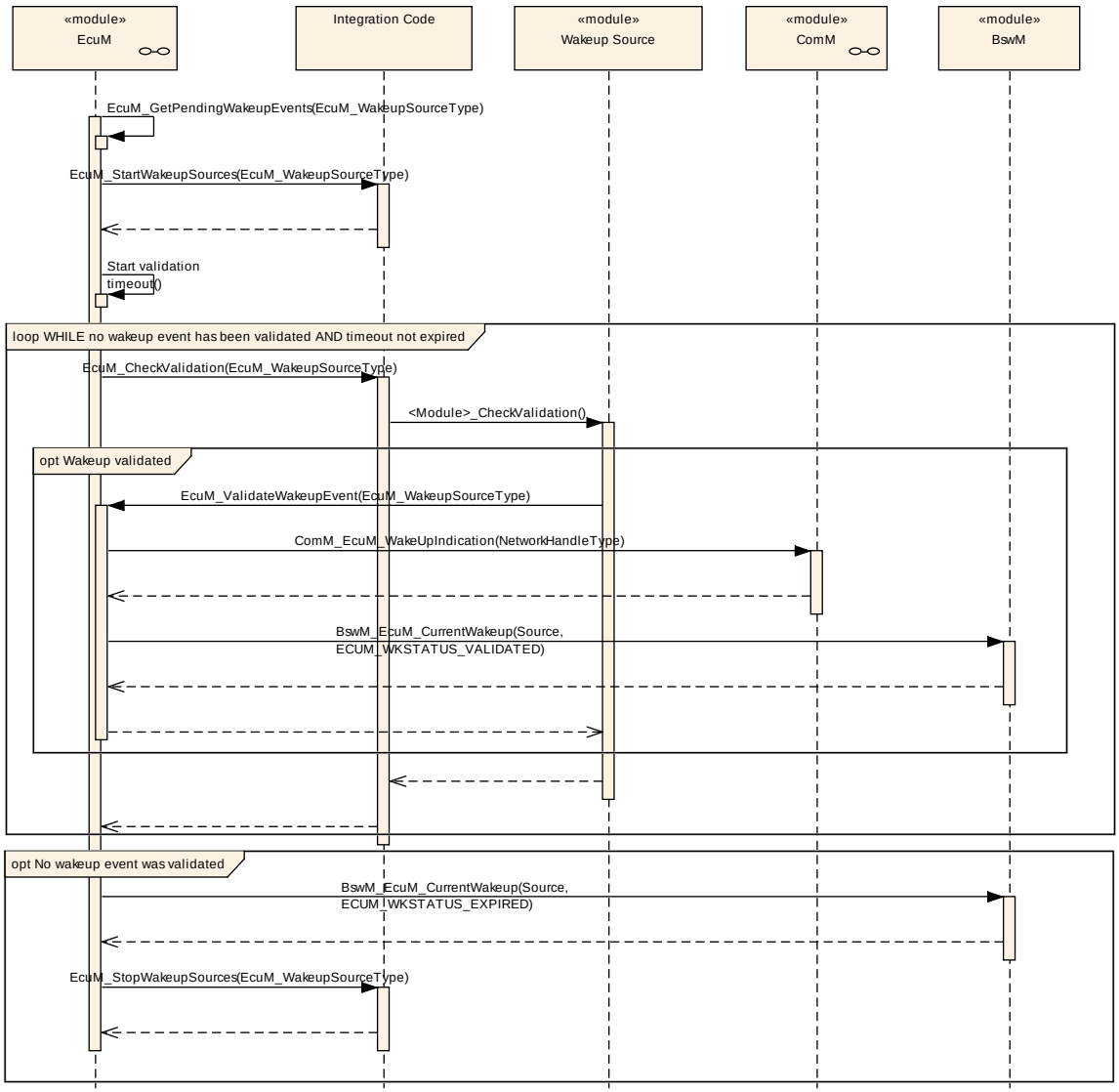

Figure 17 – The WakeupValidation Sequence

[EcuM2566] ⌈The ECU Manager module shall only invoke wakeup validation on those wakeup sources where it is required by configuration. If the validation protocol is not configured (see `EcuMValidationTimeout` EcuM150_Conf), then a call to EcuM_SetWakeupEvent (see EcuM2826) shall also imply a call to EcuM_ValidateWakeupEvent (see EcuM2829).⌋()

[EcuM2565] ⌈The ECU Manager module shall start a validation timeout for each pending wakeup event that should be validated. The timeout shall be event-specific (see `EcuMValidationTimeout` EcuM150_Conf).⌋()

Implementation hint for EcuM2565: It is sufficient for an implementation to provide only one timer, which is prolonged to the largest timeout when new wakeup events are reported.

[EcuM4081]⌈When the validation timeout expires for a pending wakeup event, the `EcuM_MainFunction` sets (OR-operation) set the bit in the internal expired wakeup events variable (see section 7.6.3 Internal Representation of Wakeup States).⌋()

[EcuM4082] ⌈When the validation timeout expires for a pending wakeup event, the `EcuM_MainFunction` shall invoke `BswM_EcuM_CurrentWakeup` with an EcuM_WakeupSourceType bitmask parameter with the bit corresponding to the wakeup event set and state value parameter set to ECUM_WKSTATUS_EXPIRED.⌋()

The BswM will be configured to monitor the wakeup validation through mode switch requests coming from the EcuM as the wakeup sources are validated or the timers expire. If the last validation timeout (see EcuM2565) expires without validation then the BswM shall consider wakeup validation to have failed. If at least one of the pending events is validated then the entire validation shall have passed.

Pending events are validated with a call of EcuM_ValidateWakeupEvent (see EcuM2829). This call must be placed in the driver or the consuming stack on top of the driver (e.g. the handler). The best place to put this depends on hardware and software design. See also section 7.6.4.4 Requirements for Drivers with Wakeup Sources .

7.6.4.1 Wakeup of Communication Channels

If a wakeup occurs on a communication channel, the corresponding bus transceiver driver must notify the ECU Manager module by invoking EcuM_SetWakeupEvent (see EcuM2826) function. Requirements for this notification are described in section *5.2* Peripherals with Wakeup Capability.

[EcuM2479] ⌈The ECU Manager module shall execute the Wakeup Validation Protocol upon the EcuM_SetWakeupEvent (see EcuM2826) function call according to *7.6.4.2* Interaction of Wakeup Sources and the ECU Manager later in this chapter.⌋()

7.6.4.2 Interaction of Wakeup Sources and the ECU Manager

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

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

- After exiting the Halt or Poll sequences. In this scenario, the ECU Manager module invokes EcuM_AL_DriverRestart (see EcuM2923) to re-initialize of the relevant drivers, which in turn get a chance to scan their hardware e.g. for pending wakeup interrupts.
- If the wakeup source is actually in sleep mode, the driver must scan autonomously for wakeup events; either by polling or by waiting for an interrupt.

[EcuM2975] ⌈If a wakeup event requires validation then the ECU Manager module shall invoke the validation protocol⌋()

[EcuM2976] ⌈If a wakeup event does not require validation, the ECU Manager module shall issue a mode switch request to set the event's mode to `ECUM_WKSTATUS_VALIDATED`.⌋()

[EcuM2496] ⌈If the wakeup event is validated (either immediately or by the wakeup validation protocol), the ECU Manager module shall make the information that it is a source of the current ECU wakeup through the EcuM_GetValidatedWakeupEvents (see EcuM2830) function.⌋()

7.6.4.3 Wakeup Validation Timeout

[EcuM4004] ⌈The ECU Manager Module shall either provide a single wakeup validation timeout timer or one timer per wakeup source.⌋()

The following requirements apply:

[EcuM2709] ⌈The ECU Manager module shall start the wakeup validation timeout timer when EcuM_SetWakeupEvent (see EcuM2826) is called.⌋()

[EcuM2710] ⌈EcuM_ValidateWakeupEvent shall stop the wakeup validation timeout timer (see EcuM2829).⌋()

[EcuM2712] ⌈If EcuM_SetWakeupEvent (see EcuM2826) is called subsequently for the same wakeup source, the ECU Manager module shall not restart the wakeup validation timeout.⌋()

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

If EcuM_SetWakeupEvent (see EcuM2826) is called for a wakeup source that did not yet fire during the same wakeup cycle then the ECU Manager module should prolong the validation timeout of that wakeup source.

Wakeup timeouts are defined by configuration (see EcuM148_Conf).

7.6.4.4 Requirements for Drivers with Wakeup Sources

The driver must invoke EcuM_SetWakeupEvent (see EcuM2826) once when the wakeup event is detected and supply a EcuM_WakeupSourceType parameter identifying the source of the wakeup (see EcuM2165, EcuM2166) as specified in the configuration (see `EcuMWakeupSourceId`, EcuM151_Conf).

[EcuM2572] ⌈The ECU Manager module shall detect wakeups that occurr prior to driver initialization, both from Halt/Poll or from OFF.⌋()
The driver must provide an API to configure the wakeup source for the SLEEP state, to enable or disable the wakeup source, and to put the related peripherals to sleep. This requirement only applies if hardware provides these capabilities.

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

7.6.5 Requirements for Wakeup Validation

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

Validation is done by calling the EcuM_ValidateWakeupEvent (see EcuM2829) function.

[EcuM2601] ⌈If hardware cannot detect a specific wakeup source, then the ECU Manager module shall report an ECUM_WKSOURCE_RESET instead.⌋()

7.6.6 Wakeup Sources and Reset Reason

The ECU Manager module API only provides one type (EcuM_WakeupSourceType, see *8.2.4* EcuM_WakeupSourceType), which can describe all reasons why the ECU starts or wakes up.

[EcuM2625] ⌈The ECU Manager module shall never invoke validation for the following wakeup sources:

- ECUM_WKSOURCE_POWER
- ECUM_WKSOURCE_RESET
- ECUM_WKSOURCE_INTERNAL_RESET
- ECUM_WKSOURCE_INTERNAL_WDG
- ECUM_WKSOURCE_EXTERNAL_WDG.

⌋()

7.6.7 Wakeup Sources with Integrated Power Control

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

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

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

- The CAN transceiver is initialized during one of the driver initialization blocks (under BswM control by default). This is configured or generated code, i.e. code which is under control of the system designer.

- The CAN transceiver driver API provides functions to find out if it was the CAN transceiver which started the ECU due to a passive wakeup. It is the system designer's responsibility to check the CAN transceiver for wakeup reasons and pass this information on to the ECU Manager module by using the EcuM_SetWakeupEvent (see EcuM2826) and EcuM_ClearWakeupEvent (see EcuM2828) functions.

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

7.7 Shutdown Targets

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

- Off[13]
- Sleep
- Reset

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

In previous versions of the ECU Manager module, sleep targets were treated specially, as the sleep modes realized in the ECU depended on the capabilities of the ECU. These sleep modes depend on hardware and differ typically in clock settings or other low power features provided by the hardware. These different features are accessible through the MCU driver as so-called MCU modes (see [9]).

There are also various modalities for performing a reset which are controlled, or triggered, by different modules:

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

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

Refer to section 8.3.3 Shutdown Management for the shutdown management facility's interface definitions.

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

7.7.1 Sleep

[EcuM2188] ⌈No wakeup event shall be missed after entering the SLEEP phase. If a valid wakeup event occurs while the ECU is in transition to SLEEP the ECU Manager module shall proceed as quickly as possible to the WakeupReaction Sequence and shall not enter the Halt or Poll Sequences.⌋()

[EcuM2957] ⌈The ECU Manager module may define a configurable set of sleep modes (see *EcuMSleepMode* EcuM131_Conf) where each mode itself is a shutdown target.⌋()

[EcuM2958] ⌈The ECU Manager module shall allow mapping the MCU sleep modes to ECU sleep modes and hence allow them to be addressed as shutdown targets.⌋()

[EcuM2959] ⌈The ECU Manager module shall allow aliases to be defined for shutdown targets (see EcuM180_Conf).⌋()

Rationale for EcuM2959: This is to simplify portability of code across different ECUs.

7.7.2 Reset

[EcuM4005] ⌈The ECU Manager module shall define a configurable set of reset modes (see *EcuMResetMode* EcuM172_Conf and section 8.2.7 EcuM_ResetType EcuM4044), where each mode itself is a shutdown target. The set will minimally contain targets for

- Mcu_PerformReset
- WdgM_PerformReset
- Toggle I/O Pin via DIO / SPI⌋()
-

[EcuM4006] ⌈The ECU Manager module shall allow defining aliases for reset targets (See EcuM180_Conf).⌋()

[EcuM4007] ⌈The ECU Manager module shall define a configurable set of reset causes (see *EcuMShutdownCause* Ecum175_Conf and section 8.2.8 EcuM_ShutdownCauseType EcuM4045). The set shall minimally contain targets for

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

and the time of the reset.⌋()

[EcuM4008] ⌈The ECU Manager Module shall offer facilities (see section 8.3.3 Shutdown Management) to BSW modules and SW-Cs to

- Record a shutdown cause
- Get a set of recent shutdown causes⌋()

7.8 Alarm Clock

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

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

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

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

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

[EcuM4069] ⌈When the Alarm Clock service is present (see `EcuMAlarmClockPresent` EcuM199_Conf) the EcuM Manager module shall maintain an EcuM clock whose time shall be the time in seconds since battery connect.⌋()

[EcuM4086] ⌈The EcuM clock shall track time in the UP and SLEEP phases.⌋()

[EcuM4087] ⌈Hardware permitting, the EcuM clock time shall not be reset by an ECU reset.⌋()

[EcuM4088] ⌈There shall be one and only one wakeup source assigned to the EcuM Clock (see `EcuMAlarmWakeupSource` EcuM200_Conf).⌋()

7.8.1 Alarm Clocks and Users

SW-Cs and BSW modules can each maintain an alarm clock (user alarm clock). Each user alarm clock (see `EcuMAlarmClock` EcuM184_Conf) is associated with an `EcuMUser` (see EcuM195_Conf) which identifies the respective SW-C or BSW module.

[EcuM4070] ⌈Each EcuM User shall have at most one user alarm clock.⌋()

[EcuM4071] ⌈An EcuM User shall not be able to set the value of another user's alarm clock.⌋()

[EcuM4072] ⌈The ECU Manager module shall set always the master alarm clock value to the value of the earliest user alarm clock value.⌋()

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

[EcuM4073] ⌈Only authorized EcuM Users can set the EcuM clock time (see EcuM197_Conf, a user list in EcuM168_Conf⌋()

Rationale for EcuM4073: Generally EcuM Users shall not be able to set the EcuM clock time. The EcuM clock time can be set to an arbitrary time to allow testing alarms that take days to expire.

7.8.2 EcuM Clock Time

[EcuM4089] ⌈If the underlying hardware mechanism is tick based, the ECUM shall "correct" the time accordingly⌋()

7.8.2.1 EcuM Clock Time in the UP Phase

The EcuM_MainFunction increments the EcuM clock during the UP Phase. It uses standard OS mechanisms (alarms / counters) to derive its time. Note the difference in granularity between the counters and EcuM time, which is measured in seconds (EcuM4069).

7.8.2.2 EcuM Clock Time in the Sleep Phase

There are two alternatives to increment the EcuM clock during sleep depending on whether `EcuM_GoHalt` or `EcuM_GoPoll` were called.

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

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

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

[EcuM4009] ⌈When leaving the Sleep state the ECU Manager Module will abort any active user alarm clock and the master alarm clock. This means that both clock induced and wakeups due to other events will result in clearing all alarms.⌋(BSW09187)

[EcuM4010] ⌈User alarms and the master alarm shall be cancelled during the StartPreOS Sequence, in the WakeupReaction Seqence and the OffPreOS Sequence.⌋(BSW09188)

7.9 MultiCore

In its current release, the MultiCore OS only supports communication with SW-Cs and CDDs on the slave cores.

The bulk of the BSW is placed on a single core: the master core. Only the set of BSW modules needed to support the OS and SW-Cs runs on slave cores. ECU management on the slave cores must therefore also only support that subset of the BSW.

It is assumed, at least for the current release that all modules are instantiated as a single image, that is there are not separate module variants for the master and slave cores. In the case of the ECU Manager module, this means that there is only one ECU Manager Module image on the ECU that is executed on every core.
This has the effect that code that is superfluous on slave cores is not compiled out and that at points where the ECU manager behavior differs on the different cores, that the ECU Manager must first test whether it is on a master or a slave core and act appropriately.

This also has the effect that the EcuM services that implement the EcuM AUTOSAR services (i.e. the ShutdownTarget, BootTarget and AlarmClock interfaces) are also available on all cores. See section 7.10.1 Overview of the EcuM AUTOSAR Service for a discussion about this is handled.

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

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

7.9.1 Master Core

The master core contains the set of BSW modules that would constitute the entire set of BSW modules on a single-core system. Most notably for ECU management, there is only one BswM on the ECU, and that on the master core.

There is a “master” ECU Manager module located on the master core, which performs the actions initiated by the BswM: GoOff, GoHalt and GoPoll as well as the startup and initialization routines. The remaining port and external interfaces as outlined in section 7.10 AUTOSAR Ports are also located on the master core.

7.9.2 Slave Core

Only the Startup I drivers for the hardware controlled by the core, a ”slave” ECU Manager module, the OS (+IOC), SchM, RTE, SW-Cs and CDDs run on the slave cores.

7.9.3 Master Core – Slave Core Signalling

This section discusses the general mechanisms with which BSW can communicate over cores. It presupposed general knowledge of the IOC, which is described and specified in the “Specification of Operating System”.

7.9.3.1 BSW Level

The Multi-Core Operating System provides a basic mechanism for synchronizing the starts of the operating systems on the master and slave cores. The BSW Mode Manager on the master core is responsible for starting and stopping the RTE, using a task that starts or stops the RTE on the slave cores.

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

7.9.3.2 EcuM Level

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

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

The Multi-Core Operating System extends the OSEK SetEvent function across cores. A task on one core can wait for an event set on another core. Figure 18 illustrates

how this applies to the problem of synchronizing the cores before calling ShutdownAllCores (whereby the de-intialization details have been omitted). The Set/WaitEvent functions accept a bitmask which can be used to indicate shutdown-readiness on the individual slave cores. Each SetEvent call from a “slave” ECU Manager module will stop the “master” ECU Manager module’s wait. The “master” ECU Manager module must therefore track the state of the individual slave cores and set the wait until all cores have registered their readiness.

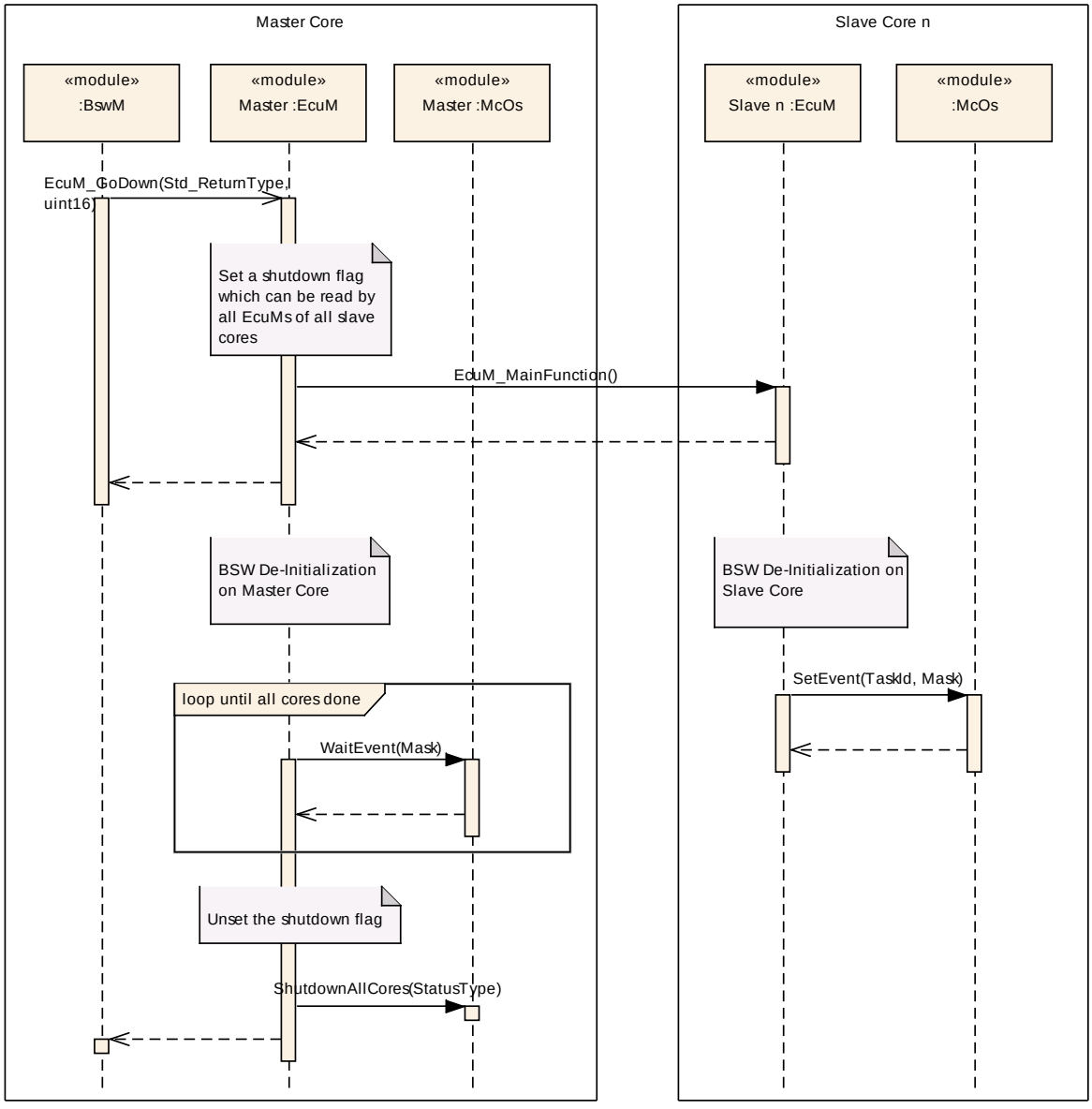

Figure 18: Master / Slave Core Shutdown Synchronization

7.9.4 UP Phase

From the hardware perspective, it is possible that wakeup interrupts could occur on all cores. Wakeup sources are nonetheless restricted to the master core in the current release.

[EcuM4011] ⌈The EcuM_MainFunction shall only run on the master core⌋()

[EcuM4012] ⌈The ECU Manager module shall only process wakeup events on the master core.⌋()

Rationale for EcuM4012: Otherwise the message must be relayed from the slave cores to the master core. Additionally, the communication stack software is on the master core.

As in the single-core case, the BswM (as configured by the integrator) has the responsibility for controlling ECU resources, establishing that the ECU must be powered down or halted as well as de-initializing the appropriate applications and BSW before handing control over to the EcuM on the master core. The BswM is located on the master core so that mode requests from slave cores must be routed over the IOC.

7.9.5 STARTUP Phase

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

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

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

StartPostOS is executed on each core and the SchM is initialized on each core. The BswM is initialised on the master core.

There is a (configurable) mode for the ECU (which is known globally). Control will pass to the BswM on the main core and startup will continue using modes as designed and configured by the integrator. The integrator is responsible for configuring a mode change after Com_Init and NvM_Init have been called on the master core. After that, the RTE can be started on the slave cores (and the master core for that matter).

Note that COM, DEM and FIM only run on the master core, so that handling their initialization after the NvM_ReadAll has finished is just a problem for the master core.

[EcuM4013] ⌈There shall be a single, common EcuMDriveInitListZero (see EcuM114_Conf) and EcuMDriverInitListOne (see EcuM111_Conf) for all slave cores. EcuMDriverInitListZero_slave and EcuMDriverInitListOne_slave shall be empty.⌋()

Rationale for EcuM4013: All BSW drivers shall run on the master core. SW-Cs or CDDs on slave cores can use IOC mechanisms to communicate with drivers or other BSW modules located on the master core.

Implementation hint. Custom initlialization required by a CDD running on a slave core can be executed in the “main” before the call to EcuM_Init or in the StartupHook.

[EcuM4014] ⌈The ECU Manager module shall not call BswM_Init on slave cores.⌋()

7.9.5.1 Master Core STARTUP
[EcuM4015]⌈

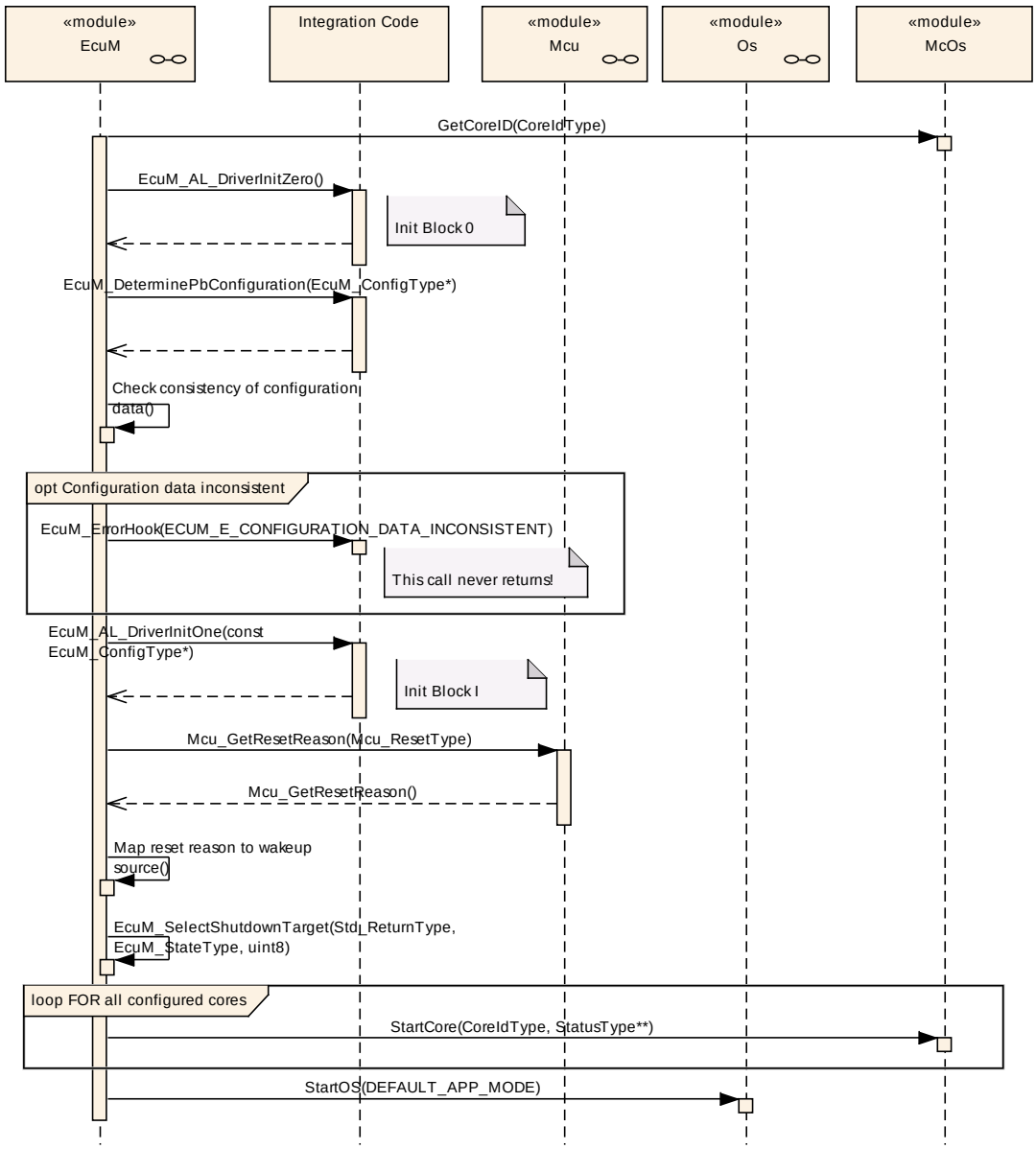

Figure 19 -Master Core StartPreOS Sequence

⌋()

[EcuM4016]⌈

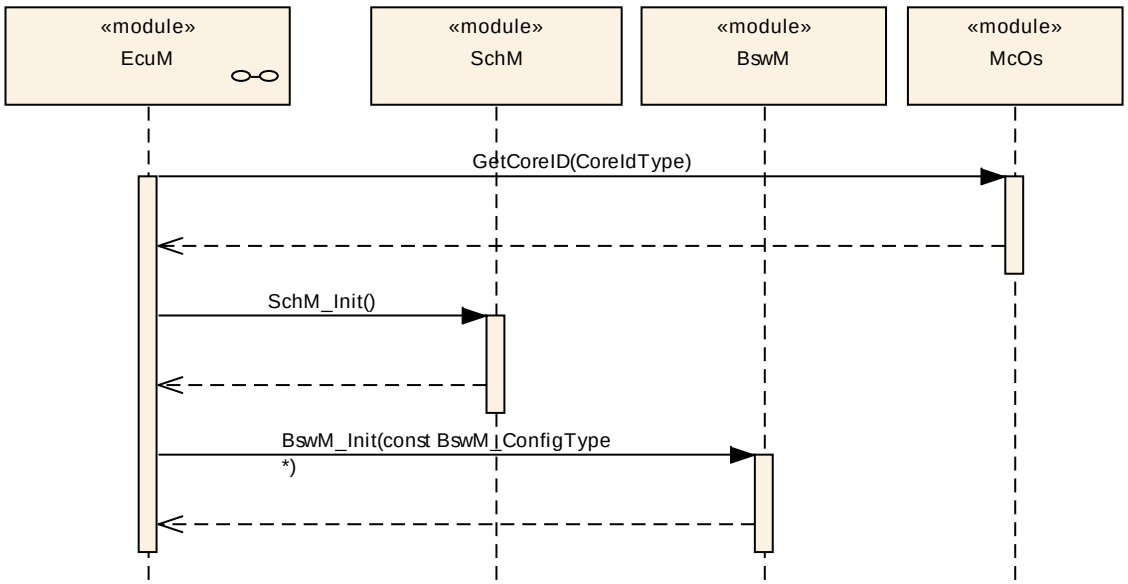

Figure 20 - Master Core StartPostOS Sequence

⌋()

7.9.5.2 Slave Core STARTUP

[EcuM4017]⌈

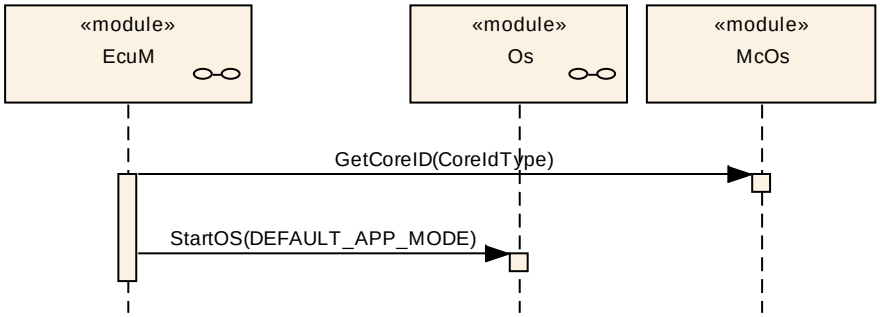

Figure 21 - Slave Core StartPreOS Sequence

⌋()

[EcuM4018][

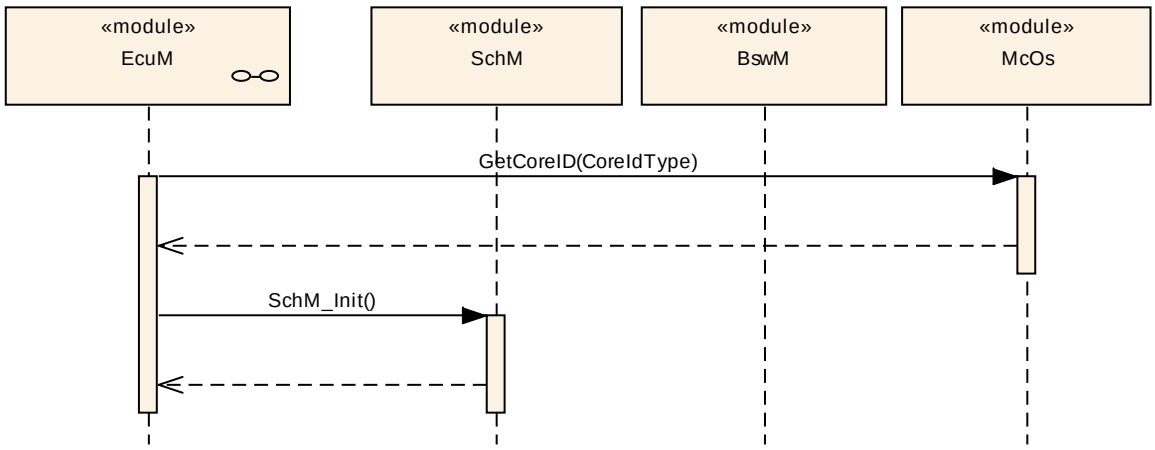

Figure 22 - Slave Core StartPostOS Sequence

]()

7.9.6 SHUTDOWN Phase

Individual core shutdown (i.e. while the rest of the ECU continues to run) is currently not supported. All cores are shut down simultaneously. The way the master ECU Manager requests the shutdown of all slave ECU Manager Modules is described in 7.9.3.2 EcuM Level.

The “master” ECU Manager module calls a single ShutdownAllCores rather than somehow calling ShutdownOs on the individual cores. The ShutdownAllCores stops the OS on all cores and stops all cores as well.

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

The BswM (which is (only) on the master core) ascertains that the ECU should be shut down and distributes an appropriate mode switch to each core (see section 7.9.3 Master Core – Slave Core Signalling for details). The SW-Cs and CDDs on the slave cores must catch this mode switch, de-initialize appropriately and send appropriate signals to the BswM to indicate their readiness.

The BswM continues with SW-C and BSW de-initialization on the master core as appropriate until it ultimately calls GoOff on the master core and on the slave cores. The “master” EcuM de-initializes the BswM, and the SchM. The EcuMs on the slave cores de-initialize their SchMs and then send a signal to indicate that the core is ready for ShutdownOS (again, see section 7.9.3 Master Core – Slave Core Signalling for details). Note that in a Multi-Core environment, neither “master” nor “slave” ECU Managers call ShutdownOS at the end of the OffPreOS Sequence.

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

7.9.6.1 Master Core SHUTDOWN

[EcuM4019][

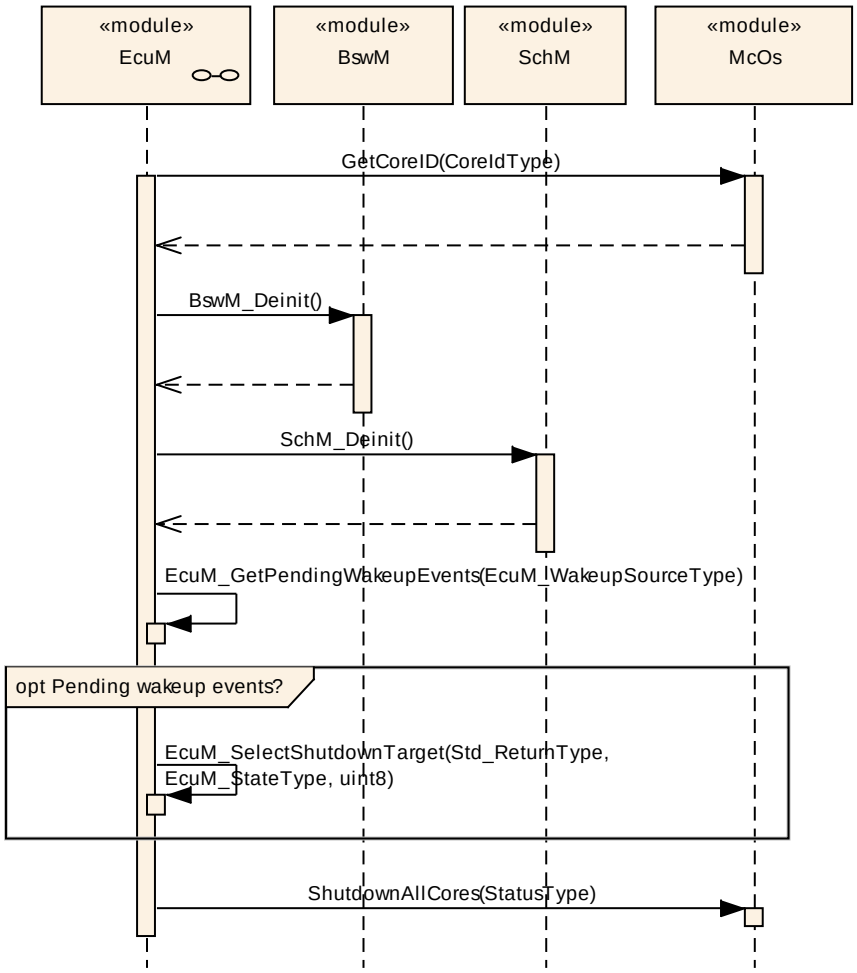

Figure 23 - Master Core OffPreOS Sequence

]()

[EcuM4020]⌈

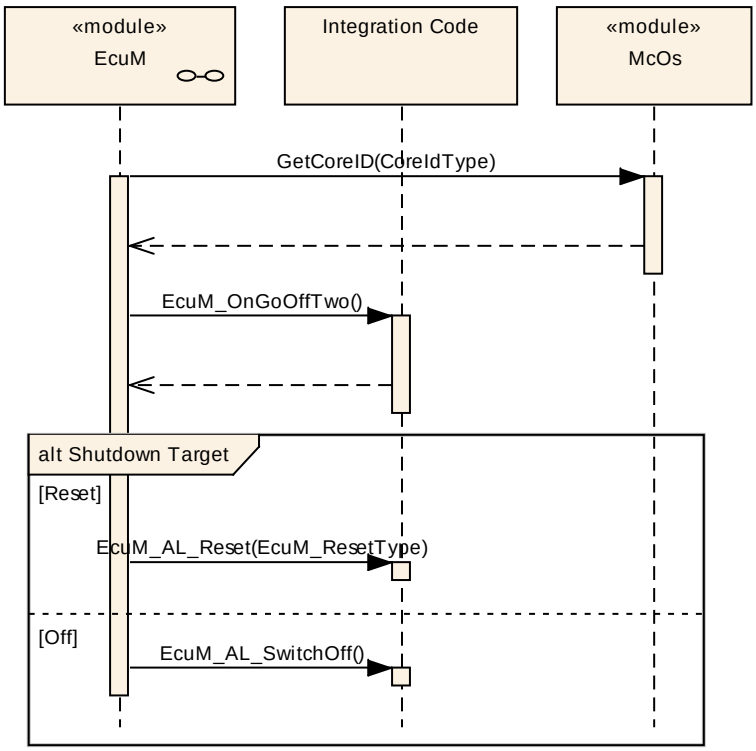

Figure 24 - Master Core OffPostOS Sequence

⌋()

7.9.6.2 Slave Core SHUTDOWN

[EcuM4021]⌈

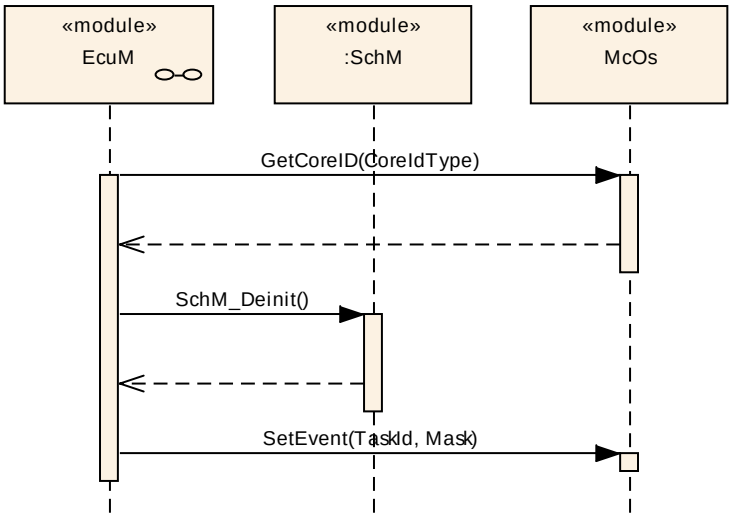

Figure 25 - Slave Core OffPreOS Sequence

]()

[EcuM4022][

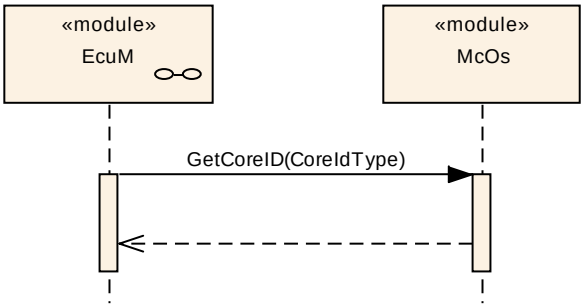

Figure 26 - Slave Core OffPostOS Sequence

]()

7.9.7 SLEEP Phase

Individual core sleep (while the rest of the ECU continues to run) is currently not supported.

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

The BswM on the master core ascertains that sleep should be initiated and distributes an appropriate ECU mode to each core. The SW-Cs and CDDs on the slave cores must catch this mode switch, de-initialize appropriately and send appropriate mode requests to the BswM to indicate their readiness.

The slave cores must be halted so that they cannot generate interrupts (which disturb the state of the RAM). The “halt”s must be synchronized so that all slave cores are halted before the master core computes the checksum. The ECU Manager module on the master core uses the same “signal” mechanism as for synchronizing cores on GoOff.

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

7.9.7.1 Master Core SLEEP

[EcuM4023][

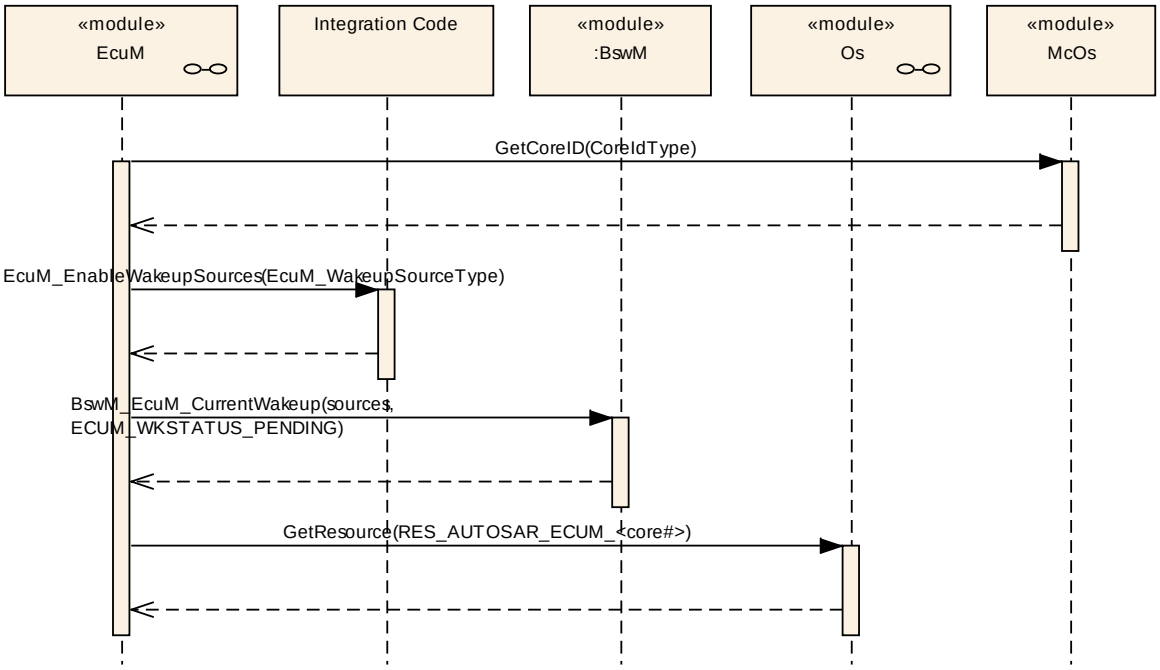

Figure 27 - Master Core GoSleep Sequence

]()

[EcuM4024]⌈

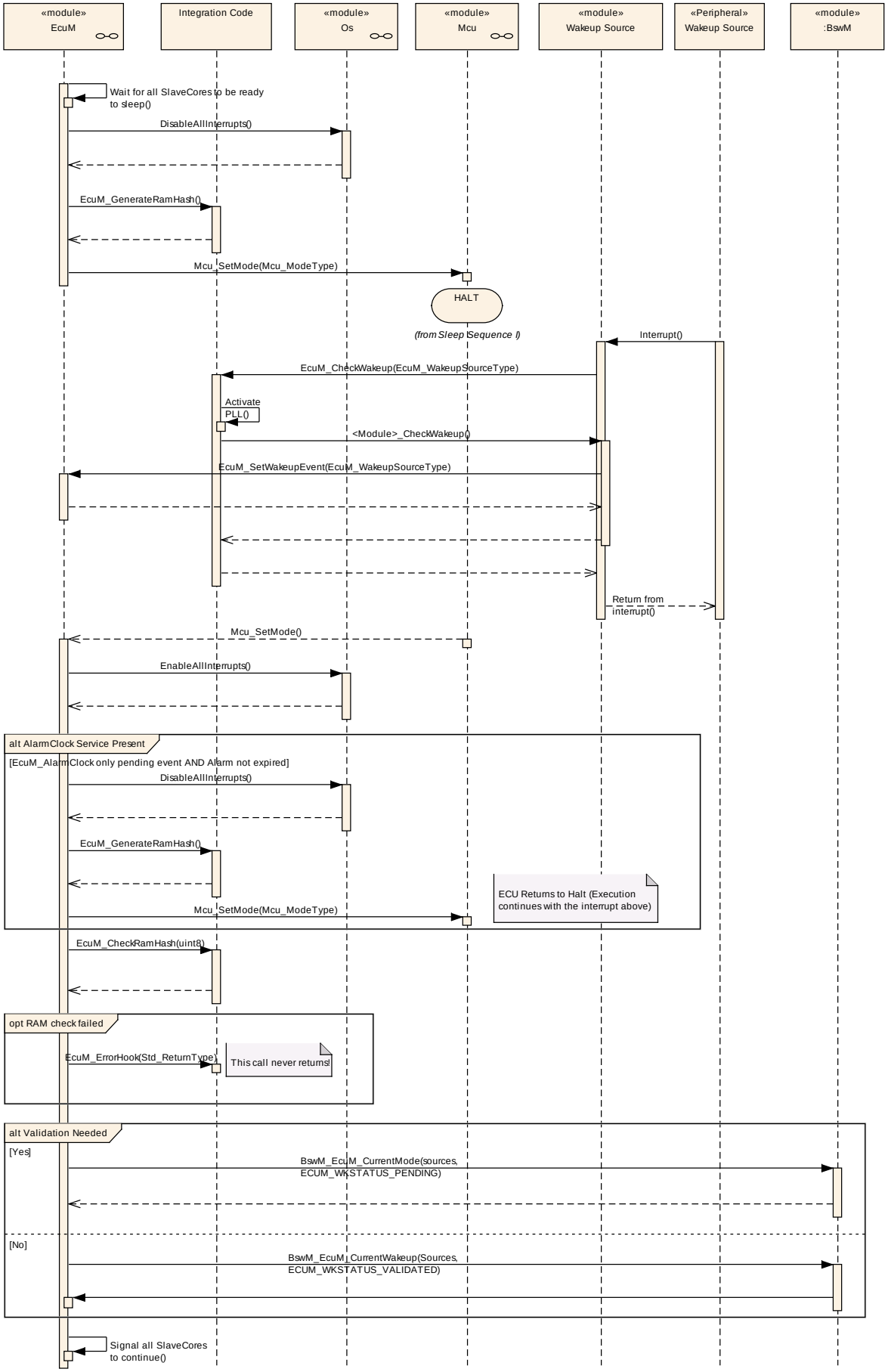

Figure 28: AUTOSAR confidential Sequence

](BSW09239)

[EcuM4025][

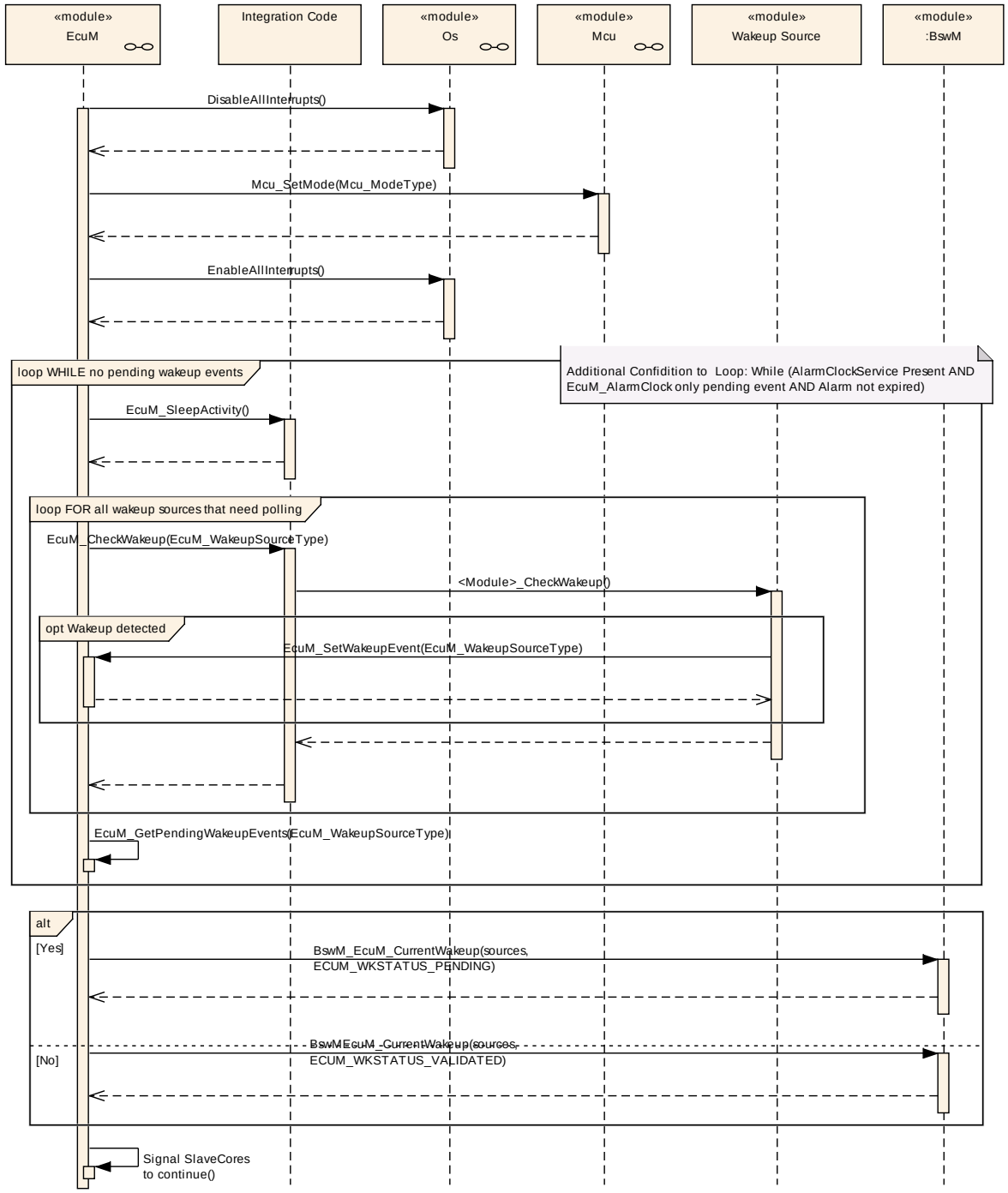

Figure 29 - Master Core Poll Sequence

](

[EcuM4026][

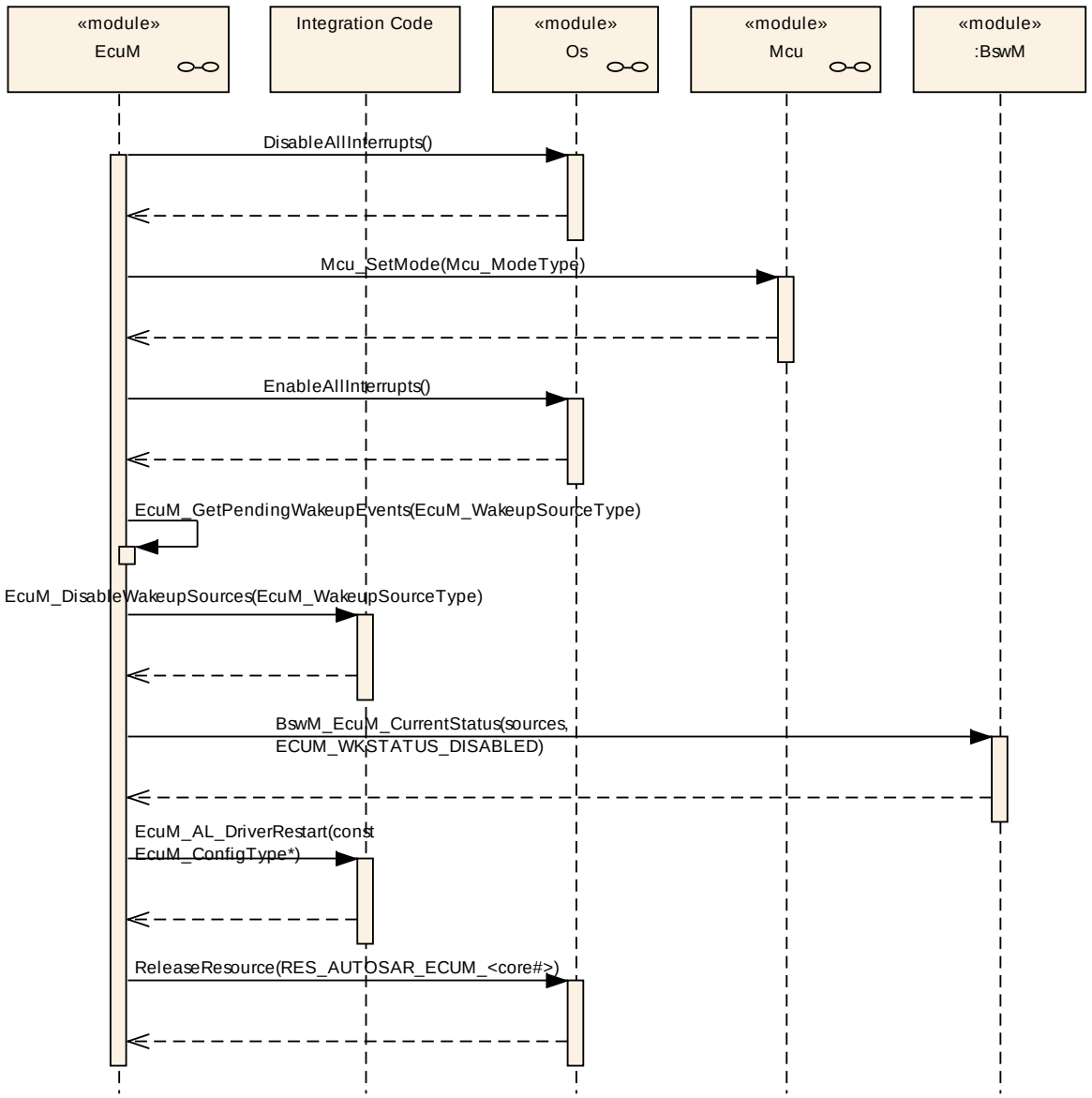

Figure 30 - Master Core WakeupRestart Sequence

]()

7.9.7.2 Slave Core SLEEP

[EcuM4027][

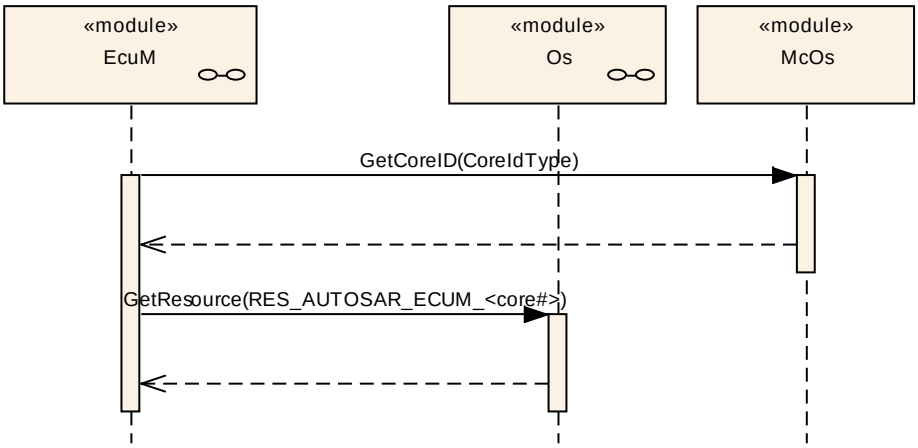

Figure 31 - Slave Core GoSleep Sequence

]()

[EcuM4028][

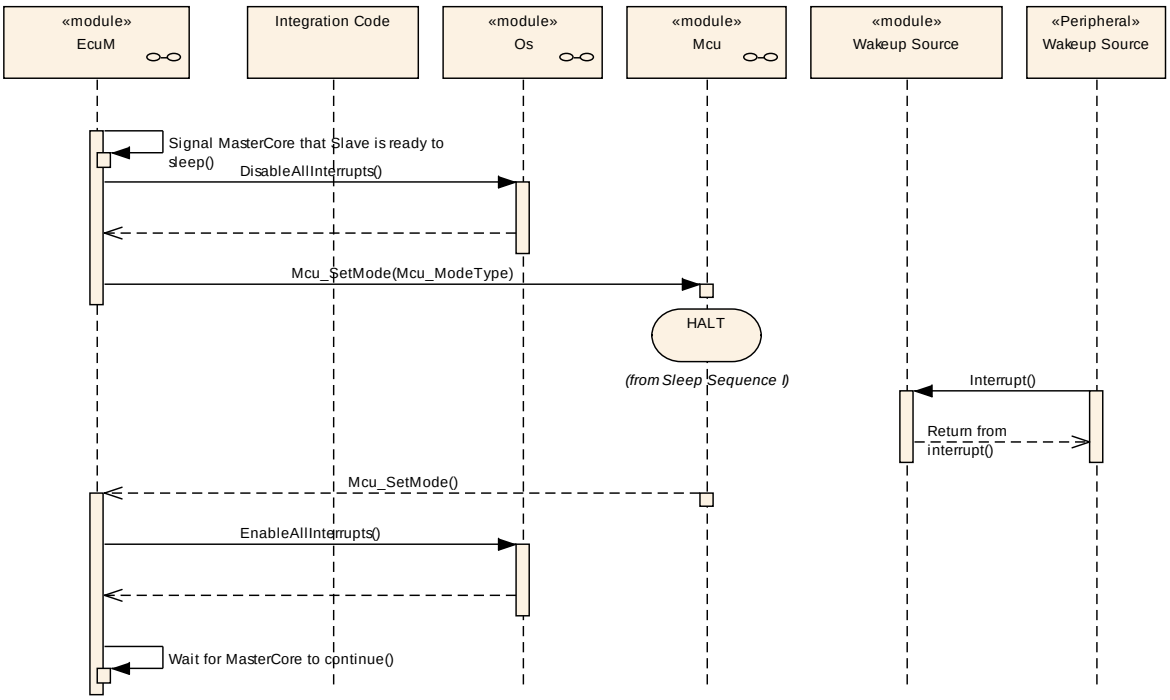

Figure 32 - Slave Core Halt Sequence

]()

[EcuM4029]⌈

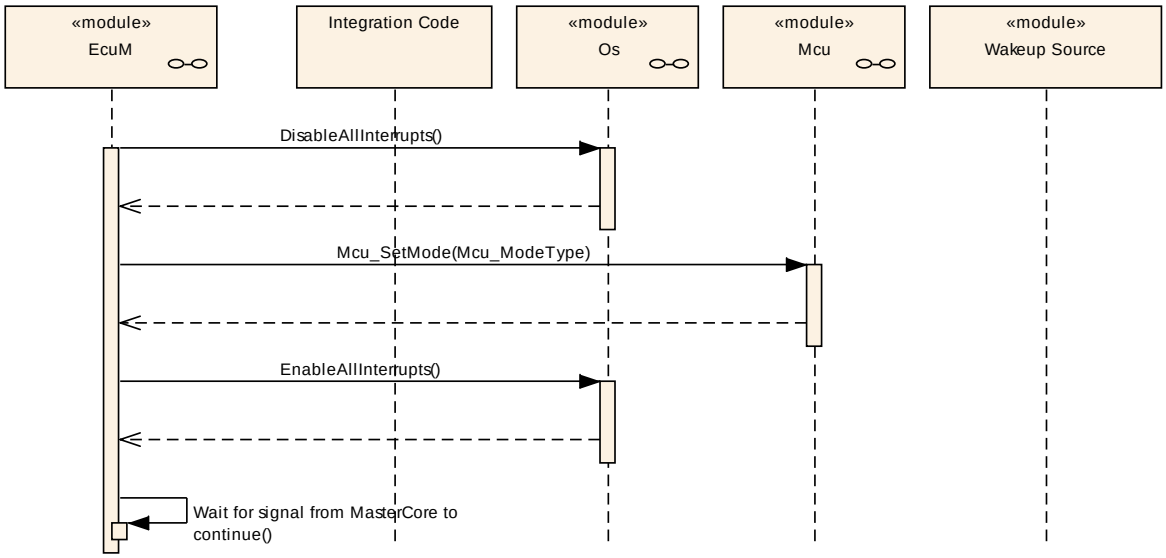

Figure 33 - Slave Core Poll Sequence

⌋()

[EcuM4030]⌈

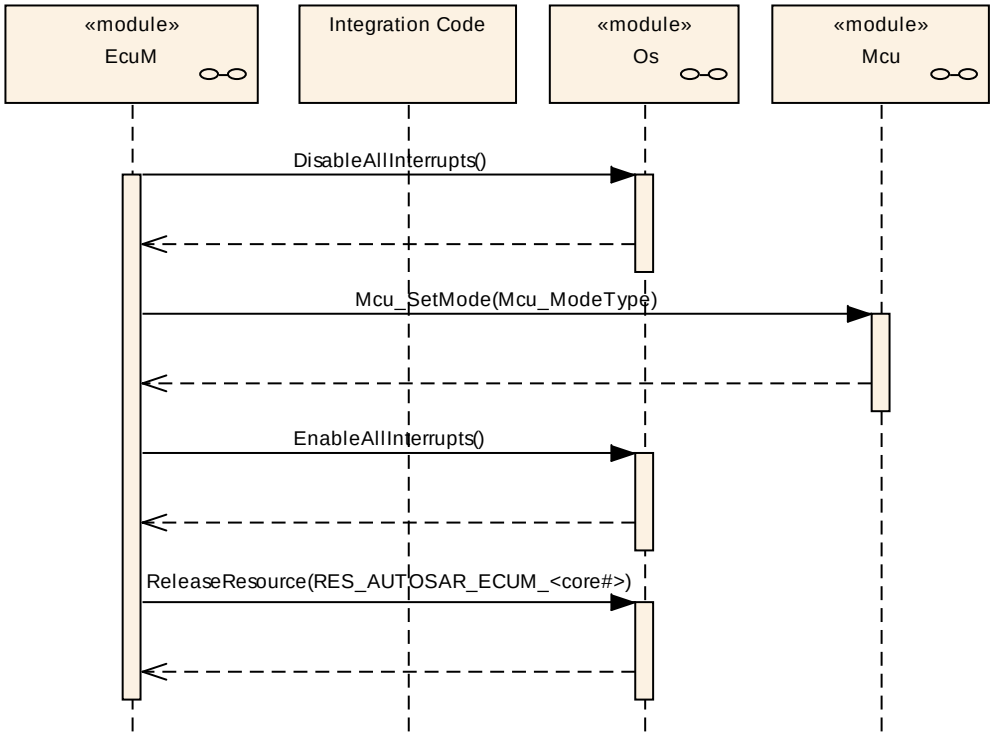

Figure 34 - Slave Core WakeupRestart Sequence

⌋()

7.10 AUTOSAR Ports

Here are the Autosar Ports described.

7.10.1 Overview of the EcuM AUTOSAR Service

[EcuM2977] ⌈The overall architecture of the ECU Manager module service shall be as depicted in Figure 35:

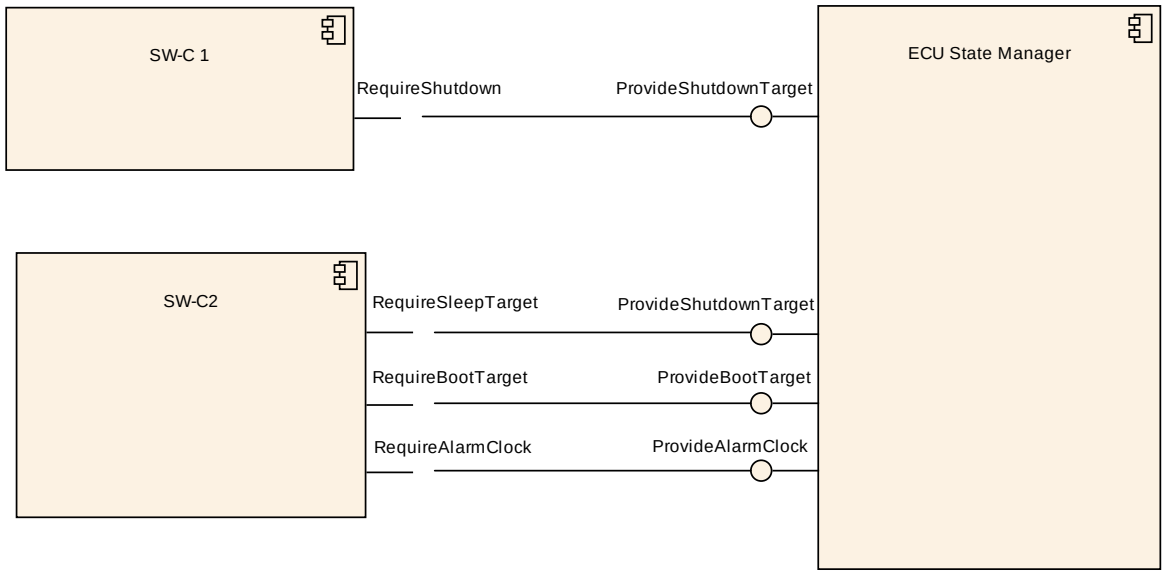

Figure 35 – ARPackage of the ECU Manager module

⌋()

In the case of a MultiCore ECU, the EcuM AUTOSAR service should only be offered on the master core – the core that contains the rest of the BSW modules.

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

[EcuM2763] ⌈The ECU Manager module shall provide AUTOSAR ports to:

- Manage sleep and shutdown modalities
  - select the shutdown target (mode) and get the current and last sleep targets
  - select and get the reset modality
  - get the time and cause of previous resets
- select and get the boot target
- set, reset and retrieve alarm clock values.⌋()

7.10.2 Specification of the Port Interfaces

This chapter specifies the port interfaces needed to access the ECU Manager module over the VFB. The ports implementing the Port Interfaces described in this chapter are defined in chapter *7.10.3*.

7.10.2.1 Ports and Port Interface for EcuM_ShutdownTarget Interface

7.10.2.1.1 General Approach

The `EcuM_ShutdownTarget` client-server interface allows an SW-C to select a shutdown target which will be respected during the next shutdown phase.

Note that the ECU Manager module does not offer a port interface to allow a SW-C to initiate shutdown, however.

7.10.2.1.2 Data Types

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

```
ImplementationDataType EcuM_StateType {
      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
};
```

This data type represents the modes of the ECU Manager module.

```
ImplementationDataType EcuM_ModeType {
      category TYPE_REFERENCE
      ImplementationDataType uint8
};
```

This data type represents the time of the ECU Manager module.

```
ImplementationDataType EcuM_TimeType {
      category TYPE_REFERENCE
      ImplementationDataType uint32
};
```

This data typre represents the cause for the shutdown of the ECU. Concrete values for implementation can be found in chapter 8.2.8.

```
ImplementationDataType EcuM_ShutdownCauseType;
```

Most states pertain to the ECU Manager Fixed and are kept for compatibility purposes. Only the shutdown targets are relvant the ECU Manager Flexible's port interface:

- ECUM_STATE_SLEEP
- ECUM_STATE_RESET
- ECUM_STATE_OFF

7.10.2.1.3 Port Interface

[EcuM3011] [The ClientServerInterface EcuM_ShutdownTarget shall look as follows

```
ClientServerInterface EcuM_ShutdownTarget
{
      // The SW-C can select a shutdown target when it requires
      // 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 EcuM_ModeType mode,
      ERR{E_NOT_OK});
      // The SW-C gets the currently selected shutdown target
      GetShutdownTarget(OUT EcuM_StateType target, OUT EcuM_ModeType mode);

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

      // The SW-C selects the cause corresponding to the next shutdown
      // target
      PossibleErrors {
      E_NOT_OK = 1 /* The new shutdown cause was not set */
      };
      SelectShutdownCause(IN EcuM_ShutdownCauseType target,
      ERR{E_NOT_OK});

      // The SW-C gets the cause corresponding to the next shutdown target
      PossibleErrors {
      E_NOT_OK = 1 /* The shutdown cause has not been set */
      };
```

```
      GetShutdownCause(OUT EcuM_ShutdownCauseType target, ERR{E_NOT_OK});

      // The SW-C gets the cause data from the previous shutdown process
      PossibleErrors {
      E_NOT_OK = 1 /* No shutdown causes */
      };
      GetMostRecentShutdown(OUT EcuM_StateType target, OUT EcuM_ModeType
mode, OUT
      EcuM_ShutdownCauseType cause, OUT EcuM_TimeType time, ERR{E_NOT_OK});

      // The SW-C gets the cause data from the previous shutdown process
      PossibleErrors {
      E_NOT_OK = 1 /* No more shutdown causes */
      };
      GetNextRecentShutdown(OUT EcuM_StateType target, OUT EcuM_ModeType
mode, OUT
      EcuM_ShutdownCauseType cause, OUT EcuM_TimeType time, ERR{E_NOT_OK});
};]()
```

[EcuM2979] [The `mode` parameter shall determine the specific sleep or reset mode (see EcuM132_Conf) relevant to `SelectShutdownTarget`, `GetShutdownTarget` and `GetLastShutdownTarget`. The ECU Manager module shall only use the `mode` parameter is if the `target` parameter is equal to ECUM_STATE_SLEEP or ECUM_STATE_RESET, otherwise it shall be ignored.]()

7.10.2.2 Port Interface for EcuM_BootTarget Interface

7.10.2.2.1 General Approach

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

7.10.2.2.2 Data Types

The following data type represents the boot targets the ECU Manager module can be configured with.

```
ImplementationDataType EcuM_BootTargetType {
      0 -> ECUM_BOOT_TARGET_APP
      1 -> ECUM_BOOT_TARGET_OEM_BOOTLOADER
      2 -> ECUM_BOOT_TARGET_SYS_BOOTLOADER

      // ECUM_BOOT_TARGET_APP: The ECU will boot into the application
      // ECUM_BOOT_TARGET_OEM_BOOTLOADER: The ECU will boot into the OEM
      bootloader
      // ECUM_BOOT_TARGET_SYS_BOOTLOADER: The ECU will boot
         into the system supplier bootloader
};
```

7.10.2.2.3 Port Interface

[EcuM3012] [The ClientServerInterface EcuM_BootTarget shall look as follows

```
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);
};
```
]()

7.10.2.3 Port Interface for EcuM_AlarmClock Interface

7.10.2.3.1 General Approach

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

The EcuM_AlarmClock interface uses port-defined argument values to identify the user that manages its alarm clock. See [rte_sws_1350] in the Specification of RTE [14] for a description of port-defined argument values.

7.10.2.3.2 Data Types

The EcuM_AlarmClock service does not have any specific data types.

7.10.2.3.3 Port Interface

[EcuM3013] [The ClientServerInterface EcuM_AlarmClock shall look as follows

```
ClientServerInterface EcuM_AlarmClock
{
      // The SW-C selects its alarm relative to the current time
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_EARLIER_ACTIVE = 3 /* An earlier alarm is already set */
      };
      SelectRelWakeupAlarm (IN EcuM_UserType user, IN EcuM_TimeType time,
            ERR{E_NOT_OK, E_NOT_SUPPORTED, E_EARLIER_ACTIVE});

      // The SW-C selects its alarm to an absolute point in time
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_EARLIER_ACTIVE = 3 /* An earlier alarm is already set */
      E_PAST = 4 /* The desired point in time has already passed */
      };
      SelectAbsWakeupAlarm (IN EcuM_UserType user, IN EcuM_TimeType time,
            ERR{E_NOT_OK, E_NOT_SUPPORTED, E_EARLIER_ACTIVE, E_PAST});

      // The SW-C cancels its alarm
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
```

```
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_NOT_ACTIVE = 5 /* No active alarm found */
      };
      AbortWakeupAlarm (IN EcuM_UserType user,
           ERR{E_NOT_OK, E_NOT_SUPPORTED, E_NOT_ACTIVE});

      // The SW-C gets the current (EcuM clock) time
      PossibleErrors {
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      };
      GetCurrentTime (OUT EcuM_TimeType time, ERR{E_NOT_SUPPORTED});

      // The SW-C gets the absolute time in seconds of the next wakeup
      // (master alarm)
      PossibleErrors {
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      };
      GetWakeupTime (OUT EcuM_TimeType time, ERR{E_NOT_SUPPORTED});

      // The SW-C sets Ecum Clock (the absolute time since battery connect)
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_NOT_ALLOWED = 6 /* Service is privileged (BSW only) */
      };
      SetClock (IN EcuM_TimeType time, ERR{E_NOT_OK, E_NOT_SUPPORTED,
                E_NOT_ALLOWED});
};
```

]()

7.10.3 Summary of ports

7.10.3.1 Definitions of interfaces

```
ImplementationDataType EcuM_StateType {
      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_ShutdownTarget
```

```
{
      // The SW-C can select a shutdown target when it requires
      // 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 EcuM_ModeType mode,
      ERR{E_NOT_OK});
      // The SW-C gets the currently selected shutdown target
      GetShutdownTarget(OUT EcuM_StateType target, OUT EcuM_ModeType mode);

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

      // The SW-C selects the cause corresponding to the next shutdown
      // target
      PossibleErrors {
      E_NOT_OK = 1 /* The new shutdown cause was not set */
      };
      SelectShutdownCause(IN EcuM_ShutdownCauseType target,
      ERR{E_NOT_OK});

      // The SW-C gets the cause corresponding to the next shutdown target
      PossibleErrors {
      E_NOT_OK = 1 /* The shutdown cause has not been set */
      };
      GetShutdownCause(OUT EcuM_ShutdownCauseType target, ERR{E_NOT_OK});

      // The SW-C gets the cause data from the previous shutdown process
      PossibleErrors {
      E_NOT_OK = 1 /* No shutdown causes */
      };
      GetMostRecentShutdown(OUT EcuM_StateType target, OUT EcuM_ModeType
mode, OUT
      EcuM_ShutdownCauseType cause, OUT EcuM_TimeType time, ERR{E_NOT_OK});

      // The SW-C gets the cause data from the previous shutdown process
      PossibleErrors {
      E_NOT_OK = 1 /* No more shutdown causes */
      };
      GetNextRecentShutdown(OUT EcuM_StateType target, OUT EcuM_ModeType
mode, OUT
      EcuM_ShutdownCauseType cause, OUT EcuM_TimeType time, ERR{E_NOT_OK});
};

ImplementationDataType EcuM_BootTargetType {
      0 -> ECUM_BOOT_TARGET_APP
      1 -> ECUM_BOOT_TARGET_OEM_BOOTLOADER
      2 -> ECUM_BOOT_TARGET_SYS_BOOTLOADER

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

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_AlarmClock
{
      // The SW-C selects its alarm relative to the current time
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_EARLIER_ACTIVE = 3 /* An earlier alarm is already set */
      };
      SelectRelWakeupAlarm (IN EcuM_UserType user, IN EcuM_TimeType time,
            ERR{E_NOT_OK, E_NOT_SUPPORTED, E_EARLIER_ACTIVE});

      // The SW-C selects its alarm to an absolute point in time
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_EARLIER_ACTIVE = 3 /* An earlier alarm is already set */
      E_PAST = 4 /* The desired point in time has already passed */
      };
      SelectAbsWakeupAlarm (IN EcuM_UserType user, IN EcuM_TimeType time,
            ERR{E_NOT_OK, E_NOT_SUPPORTED, E_EARLIER_ACTIVE, E_PAST});

      // The SW-C cancels its alarm
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_NOT_ACTIVE = 5 /* No active alarm found */
      };
      AbortWakeupAlarm (IN EcuM_UserType user,
           ERR{E_NOT_OK, E_NOT_SUPPORTED, E_NOT_ACTIVE});

      // The SW-C gets the current (EcuM clock) time
      PossibleErrors {
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      };
      GetCurrentTime (OUT EcuM_TimeType time, ERR{E_NOT_SUPPORTED});

      // The SW-C gets the absolute time in seconds of the next wakeup
      // (master alarm)
      PossibleErrors {
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      };
      GetWakeupTime (OUT EcuM_TimeType time, ERR{E_NOT_SUPPORTED});

      // The SW-C sets Ecum Clock (the absolute time since battery connect)
      PossibleErrors {
      E_NOT_OK = 1 /* Service failed, e.g. a NULL pointer was passed*/
      E_NOT_SUPPORTED = 2 /* Service not supported by this hardware */
      E_NOT_ALLOWED = 6 /* Service is privileged (BSW only) */
      };
      SetClock (IN EcuM_TimeType time, ERR{E_NOT_OK, E_NOT_SUPPORTED,
```

```
                E_NOT_ALLOWED});
};
```

7.10.3.2 Definition of the ECU Manager Service

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

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

[EcuM3017] ⌈The following pseudo code defines the interface of the ECU Manager Service

```
/* This is the definition of the ECU Manager module as a service. This is
the external view of the ECU Manager module, which is visible to the SW-Cs
/ ECU-integrator */
Service EcuStateManager {

      ProvidePort EcuM_ShutdownTarget shutdownTarget;

      ProvidePort EcuM_BootTarget bootTarget;

      ProvidePort EcuM_AlarmClock alarmClock
};
```

⌋()

7.10.4 Runnables and Entry points

7.10.4.1 Internal behavior

[EcuM3018] ⌈The definition of the internal behavior of the the ECU Manager module shall be as follows. This detailed description is only needed for the configuration of the local RTE.

```
InternalBehavior EcuStateManager {

      // Runnable entities of the EcuStateManager
      RunnableEntity SelectShutdownTarget
            symbol "EcuM_SelectShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetShutdownTarget
            symbol "EcuM_GetShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetLastShutdownTarget
            symbol "EcuM_GetLastShutdownTarget"
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetMostRecentShutdown
            symbol "EcuM_GetMostRecentShutdown"
            canbeInvokedConcurrently = TRUE
```

```
      RunnableEntity GetNextRecentShutdown
            symbol “EcuM_NextRecentShutdown”
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectShutdownCause
            symbol “EcuM_SelectShutdownCause”
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetShutdownCause
            symbol “EcuM_GetShutdownCause”
            canbeInvokedConcurrently = TRUE
      RunnableEntity SelectBootTarget
            symbol “EcuM_SelectBootTarget”
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetBootTarget
            symbol “EcuM_GetBootTarget”
            canbeInvokedConcurrently = TRUE
      RunnableEntity SetRelWakeupAlarm
            symbol “EcuM_SetRelWakeupAlarm”
            canbeInvokedConcurrently = TRUE
      RunnableEntity SetAbsWakeupAlarm
            symbol “EcuM_SetAbsWakeupAlarm”
            canbeInvokedConcurrently = TRUE
      RunnableEntity AbortWakeupAlarm
            symbol “EcuM_AbortWakeupAlarm”
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetCurrentTime
            symbol “EcuM_GetCurrentTime”
            canbeInvokedConcurrently = TRUE
      RunnableEntity GetWakeupTime
            symbol “EcuM_GetWakeupTime”
            canbeInvokedConcurrently = TRUE
      RunnableEntity SetClock
            symbol “EcuM_SetClock”
            canbeInvokedConcurrently = TRUE


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

7.11 Advanced Topics

7.11.1 Relation to Bootloader

The Bootloader is not part of AUTOSAR. Still, the application needs an interface to activate the bootloader. For this purpose, two functions are provided: EcuM_SelectBootTarget (see EcuM2835) and EcuM_GetBootTarget (see EcuM2836).

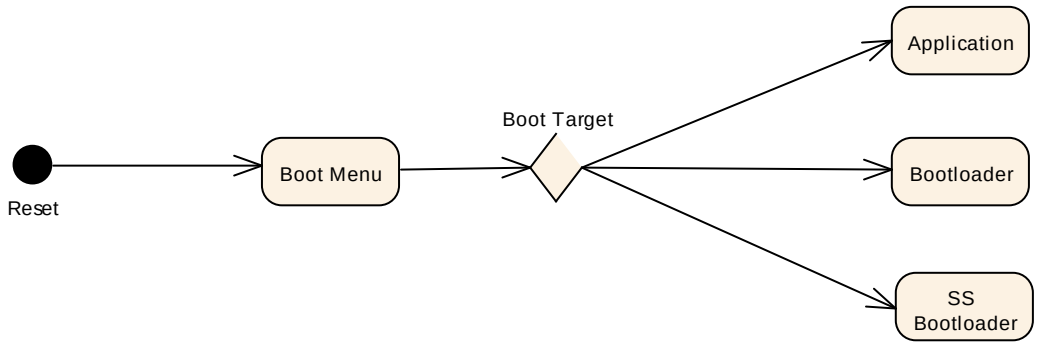

Figure 36 – Selection of Boot Targets

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

7.11.2 Relation to Complex Drivers

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

7.11.3 Handling Errors during Startup and Shutdown

[EcuM2980] ⌈The ECU Manager module shall ignore all types of errors that occur during initialization, e.g. values returned by init functions⌋()

Initialization is a configuration issue (see *`EcuMDriverInitListZero`* (EcuM114_Conf), `EcuMDriverListOne` (EcuM111_Conf) and *`EcuMDriverRestartList`* (EcuM115_Conf)) and therefore cannot be standardized.

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

also responsible for taking any special measures to react to errors occurring during their initialization.

7.12 Error Classification

[EcuM4031] ⌈Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file `Dem_IntErrId.h` and included via `Dem.h`. These values shall be different from the values of the development errors specified in Table 7.⌋(BSW00386)

[EcuM2982] ⌈Development error values are of type uint8.⌋()

[EcuM4032]⌈

Type or error	*Relevance*	*Related error code*	*Value*
A service was called prior to initialization	Development	`ECUM_E_UNINIT`	`0x10`
A function 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`
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 wakeup source was passed as a parameter to an API	Development	`ECUM_E_UNKNOWN_WAKEUP_SOURCE`	`0x17`
The RAM check during wakeup failed (see section 7.5.2 Activities in the Halt Sequence	Production	`ECUM_E_RAM_CHECK_FAILED`	`Assigned by the DEM`
Postbuild configuration data is inconsistent (see section 7.3.2 Activities in StartPreOS Sequence)	Production	`ECUM_E_CONFIGURATION_DATA_INCONSISTENT`	`Assigned by the DEM`
Defensive behavior checks have detected	Production	`ECUM_E_IMPROPER_CALLER`	`Assigned by the DEM`

improper use of the module (see section 8.3.2 Initialization and Shutdown Sequences)			
API service called with a NULL pointer. In case of this error, the API service shall return immediately without any further action, beside reporting this development error.	Development	ECUM_E_PARAM_POINTER	0x03

Table 7 - Error Classification

⌋(BSW00327, BSW00337,BSW00338,BSW00350,BSW00385,BSW00445)

7.13 Error detection

[EcuM2759] ⌈The ECU Manager Module shall report all errors as events.⌋()

[EcuM2757] ⌈The ECU Manager shall treat all errors immediately as errors.⌋()
[EcuM2758] ⌈The ECU Manager shall not recover from an error.⌋()

[EcuM4033] ⌈In the unrecoverable error situations defined in the first column of Table 7, the ECU Manager module shall call the EcuM_ErrorHook callout with the parameter value set to the corresponding related error code.⌋()

[EcuM2983] ⌈The detection of development errors is configurable (ON / OFF) at pre-compile time. The switch ECUM_DEV_ERROR_DETECT (see Chapter *10*) shall activate or deactivate the detection of all development errors.⌋(BSW00338)

[EcuM2984] ⌈If the ECUM_DEV_ERROR_DETECT switch is enabled, the ECU Manager module shall check API parameters. The detailed description of the detected errors can be found in Section 7.12 Error Classification and in Chapter *8*.⌋(BSW00338)

[EcuM2985] ⌈The ECU Manager module shall not switch the detection of production code errors off.⌋()

7.14 Error notification

[EcuM2986] ⌈If the pre-processor switch `ECUM_DEV_ERROR_DETECT` is set (see chapter *10*), the ECU Manager module shall report detected development errors to the `Det_ReportError` function of the Development Error Tracer (DET)⌋(BSW00369)

[EcuM2987] ⌈The ECU Manager module shall report production errors to the Diagnostic Event Manager (DEM). When the RAM check fails on wakeup (see section 7.5.2 Activities in the Halt Sequence) the ECU Manager module shall invoke EcuM_ErrorHook with the parameter `ECUM_E_RAM_CHECK_FAILED`. It is left integrator's discretion to allow EcuM_ErrorHook to relay the error to the DEM when he judges that the DEM will not write damaged NVRAM blocks.⌋(BSW00339)

7.15 Version Check

[EcuM4034] ⌈The ECUM module shall perform Inter Module Checks to avoid integration of incompatible files.
The imported included files shall be checked by preprocessing directives

The following version numbers shall be verified:
- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION

Where <MODULENAME> is the module short name of the other (external) modules which provide header files included by the ECUM module.

If the values are not identical to the expected values, an error shall be reported.⌋(BSW004,BSW003,BSW00318,BSW00321)

7.16 Debug Support

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

[EcuM4035] ⌈Each variable that shall be accessible by AUTOSAR Debugging shall be defined as global variable.⌋(BSW00442)

[EcuM4036] ⌈All type definitions of variables which shall be debugged shall be accessible by the header file EcuM.h.⌋()

[EcuM4037] ⌈The declaration of variables in the header file shall be such that it is possible to calculate the size of the variables by C-"sizeof".⌋()

8 API specification

8.1 Imported Types

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

[EcuM2810]⌈

Module	*Imported Type*
BswM	BswM_ConfigType

⌋(BSW00301)

8.2 Type definitions

8.2.1 EcuM_ConfigType

[EcuM4038]⌈

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

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

[EcuM2794] ⌈The structure defined in the `EcuM_ConfigType` type shall contain an additional post-build configuration variant identifier (uint8/uint16/uint32 depending on algorithm to compute the identifier). See also Chapter *7.3.4* Checking Configuration Consistency.⌋()

[EcuM2795] ⌈The structure defined by the `EcuM_ConfigType` type shall contain an additional hash code that is tested against the configuration parameter `EcuMConfigConsistencyHash` (see EcuM102_Conf) for checking consistency of the configuration data. See also section 7.3.4 Checking Configuration Consistency.⌋()

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

8.2.2 EcuM_StateType

[EcuM4039]⌈

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.		

⌋(BSW00331)

[EcuM507] ⌈The EcuM_StateType shall encode states and sub-states of the ECU Manager module. States shall be encoded in the high-nibble, sub-states in the low-nibble.

Hint for EcuM507: The sub-state encoded in EcuM_StateType can be determined by applying a bitwise AND to the state value and ECUM_SUBSTATE_MASK (first entry in the range section, above)**.** ⌋(BSW00335)

[EcuM2664] ⌈The ECU Manager module shall define all states as listed in the EcuM_StateType.⌋()

8.2.3 EcuM_UserType

[EcuM4067]⌈

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

⌋()

[EcuM487], ⌈The integrator shall define a unique value for each user at system generation time. See EcuM147_Conf⌋(BSW09122)

8.2.4 EcuM_WakeupSourceType

[EcuM4040]⌈

Name:	EcuM_WakeupSourceType		
Type:	uint32		
Range:	ECUM_WKSOURCE_POWER	- -	Power cycle (bit 0)
	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.
	ECUM_WKSOURCE_INTERNAL_RESET	- -	Internal reset of µC (bit 2) The internal reset typically only resets the µC core but not peripherals or memory controllers. The exact behavior is hardware specific. This source may also indicate an unhandled exception.
	ECUM_WKSOURCE_INTERNAL_WDG	- -	Reset by internal watchdog (bit 3)
	ECUM_WKSOURCE_EXTERNAL_WDG	- -	Reset by external watchdog (bit 4), if detection supported by hardware
Description:	EcuM_WakeupSourceType defines a bitfield with 5 pre-defined positions (see Range). The bitfield provides one bit for each wakeup source. In WAKEUP, all bits cleared indicates that no wakeup source is known. In STARTUP, all bits cleared indicates that no reason for restart or reset is known. In this case, ECUM_WKSOURCE_RESET shall be assumed.		

⌋()

[EcuM2165] ⌈Additional wakeup sources (to the pre-defined sources) shall be assigned individually to bitfield positions 5 to 31 by configuration. The bit assignment shall be done by the configuration tool.⌋()

[EcuM2166] ⌈The EcuMWakeupSourceId (see EcuM151_Conf) field in the EcuMWakeupSource container shall define the position corresponding to that wakeup source in all instances the EcuM_WakeupSourceType bitfield.⌋()

8.2.5 EcuM_WakeupStateType

[EcuM4041]⌈

Name:	EcuM_WakeupStatusType		
Type:	uint8		
Range:	ECUM_WKSTATUS_NONE	0	No pending wakeup event was detected
	ECUM_WKSTATUS_PENDING	1	The wakeup event was detected but not yet validated
	ECUM_WKSTATUS_VALIDATED	2	The wakeup event is valid
	ECUM_WKSTATUS_EXPIRED	3	The wakeup event has not been validated and has expired therefore
	ECUM_WKSTATUS_DISABLED	4	The wakeup source is disabled and does not detect wakeup events.
Description:	The type describes the possible states of a wakeup source.		

⌋()

NOTE: This declaration has to be changed to a mode. The name has to be changed.

8.2.6 EcuM_BootTargetType

[EcuM4042][

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

]()

8.2.7 EcuM_ResetType

[EcuM4044][

Name:	EcuM_ResetType		
Type:	uint8		
Range:	ECUM_RESET_MCU	0	Microcontroller reset via Mcu_PerformReset
	ECUM_RESET_WDG	1	Watchdog reset via WdgM_PerformReset
	ECUM_RESET_IO	2	Reset by toggeling an I/O line.
Description:	This type describes the reset mechanisms supported by the ECU State Manager. It can be extended by configuration.		

]()

8.2.8 EcuM_ShutdownCauseType

[EcuM4045][

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

]()

8.3 Function Definitions

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

8.3.1 General

8.3.1.1 EcuM_GetVersionInfo

[EcuM2813][

Service name:	EcuM_GetVersionInfo

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

](BSW00407,BSW00411)

[EcuM2728] [The function `EcuM_GetVersionInfo` shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor-specific version numbers (BSW00407)](BSW00407,BSW00374,BSW00379)

[EcuM2729] [The `EcuM_GetVersionInfo` function shall be configurable to ON/OFFat pre compile time through the `ECUM_VERSION_INFO_API` configuration parameter (see EcuM149_Conf).](BSW00374,BSW00379)

[EcuM2935] [If source code for caller and callee of `EcuM_GetVersionInfo` is available, the ECU Manager module should realize `EcuM_GetVersionInfo` as a macro, defined in the module's header file.]()

8.3.2 Initialization and Shutdown Sequences

8.3.2.1 EcuM_GoDown

[EcuM4046][

Service name:	EcuM_GoDown	
Syntax:	`Std_ReturnType EcuM_GoDown( uint16 caller )`	
Service ID[hex]:	0x1f	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	caller	Module ID of the calling module. Only special modules are allowed to call this function.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_NOT_OK: The shutdown request was not accepted. E_OK: This cannot occur because if the request was accepted, this call will not return.
Description:	Instructs the ECU State Manager module to perform a power off or a reset depending on the selected shutdown target.	

]()

[EcuM4047] ⌈If defensive behavior is enabled, the `EcuM_GoDown` shall check if the given CallerID is in the list of allowed CallerIDs. If the check fails, `EcuM_GoDown` shall report the error status `ECUM_E_IMPROPER_CALLER` to the DEM and return without any effect.⌋(BSW00444)

8.3.2.2 EcuM_GoHalt

[EcuM4048]⌈

Service name:	EcuM_GoHalt	
Syntax:	`Std_ReturnType EcuM_GoHalt( void )`	
Service ID[hex]:	0x20	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_NOT_OK: The request was not accepted, e.g. due to a wrong shutdown target. E_OK: If the call successfully returns, the ECU has left the sleep again.
Description:	Instructs the ECU State Manager module to go into a sleep mode where the microcontroller is halted, depending on the selected shutdown target.	

⌋()

8.3.2.3 EcuM_GoPoll

[EcuM4049]⌈

Service name:	EcuM_GoPoll	
Syntax:	`Std_ReturnType EcuM_GoPoll( void )`	
Service ID[hex]:	0x21	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_NOT_OK: The request was not accepted, e.g. due to a wrong shutdown target. E_OK: If the call successfully returns, the ECU has left the sleep again.
Description:	Instructs the ECU State Manager module to go into a polling sleep mode depending on the selected shutdown target.	

⌋()

8.3.2.4 EcuM_Init

[EcuM2811] ⌈

Service name:	EcuM_Init
Syntax:	`void EcuM_Init( void`

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

](BSW00358,BSW00414,BSW101)

8.3.2.5 EcuM_StartupTwo

[EcuM2838] [

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

]()

[EcuM2806] [Caveats of `EcuM_StartupTwo`: This function must be called from a task, which is started directly as a consequence of StartOS. I.e. either the `EcuM_StartupTwo` function must be called from an autostart task or the `EcuM_StartupTwo` function must be called from a task, which is explicitly started.]()

Clarification to EcuM2806 : The OS offers different mechanisms to activate a task on startup. Normally EcuM_StartupTwo would be configured as an autostart task in the default application mode.

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

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

8.3.2.6 EcuM_Shutdown

[EcuM2812][

Service name:	EcuM_Shutdown

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

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

](BSW0036,BSW09114)

8.3.3 Shutdown Management

8.3.3.1 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	The identfier of a sleep mode (if target is ECUM_STATE_SLEEP) or a reset mechanism (if target is ECUM_STATE_RESET) as defined by configuration.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The new shutdown target was set E_NOT_OK: The new shutdown target was not set
Description:	EcuM_SelectShutdownTarget selects the shutdown target. EcuM_SelectShutdownTarget is part of the ECU Manager Module port interface.	

](BSW09114,BSW09128,BSW09235)

[EcuM624] [The `EcuM_SelectShutdownTarget` function shall set the shutdown target to the value of the `mode` parameter. Only the following subset of the `EcuM_StateType` values are valid `mode` parameter values:

- ECUM_STATE_RESET
- ECUM_STATE_SLEEP
- ECUM_STATE_OFF

If the `mode` parameter is not a valid value, the `EcuM_SelectShutdownTarget` function shall not change the shutdown target and if Development Error Reporting is turned on, the `EcuM_SelectShutdownTarget` function shall additionally send an `ECUM_E_STATE_PAR_OUT_OF_RANGE` error message to the DET module.](BSW09114,BSW09235)

[EcuM2185] ⌈The parameter mode of the function `EcuM_SelectShutdownTarget` shall be the identifier of a sleep or reset mode. The mode parameter shall only be used if the target parameter equals `ECUM_STATE_SLEEP` or `ECUM_STATE_RESET`. In all other cases, it shall be ignored. Only sleep or reset modes that are defined at configuration time and are stored in the `EcuMCommonConfiguration` container (see EcuM181_Conf) are allowed as parameters.⌋(BSW09114)

[EcuM2585] ⌈`EcuM_SelectShutdownTarget` shall not initiate any setup activities but only store the value for later use in the SHUTDOWN or SLEEP phase.⌋(BSW09114)

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

8.3.3.2 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 out parameter "shutdownTarget" is ECUM_STATE_SLEEP, sleepMode tells which of the configured sleep modes was actually chosen. If "shutdownTarget" is ECUM_STATE_RESET, sleepMode tells which of the configured reset modes was actually chosen.
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service has failed, e.g. due to NULL pointer being passed
Description:	EcuM_GetShutdownTarget returns the currently selected shutdown target as set by EcuM_SelectShutdownTarget. EcuM_GetShutdownTarget is part of the ECU Manager Module port interface.	

⌋(BSW09128,BSW09235)

[EcuM2788] ⌈If the pointer to the `sleepMode` parameter is NULL, `EcuM_GetShutdownTarget` shall simply ignore the `sleepMode` parameter. If Development Error Detection is activated, `EcuM_GetShutdownTarget` shall send the ECUM_E_NULL_POINTER development error to the DET module.⌋()

8.3.3.3 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	If the out parameter "shutdownTarget" is ECUM_STATE_SLEEP, sleepMode tells which of the configured sleep modes was actually chosen. If "shutdownTarget" is ECUM_STATE_RESET, sleepMode tells which of the configured reset modes was actually chosen.
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service has failed, e.g. due to NULL pointer being passed
Description:	EcuM_GetLastShutdownTarget returns the shutdown target of the previous shutdown process. EcuM_GetLastShutdownTarget is part of the ECU Manager Module port interface.	

⌋**(**BSW09128,BSW09235**)**

[EcuM2156] ⌈`EcuM_GetLastShutdownTarget` shall return the ECU state from which the last wakeup or power up occurred in the `shutdownTarget` parameter. `EcuM_GetLastShutdownTarget` shall always return the same value until the next shutdown.⌋(BSW09235)

[EcuM2336] ⌈If the call of `GetLastShutdownTarget()` passes `ECU_STATE_SLEEP` in the parameter `shutdownTarget`, in the parameter `sleepMode` it returns which of the configured sleep modes was actually chosen.
If the call of `GetLastShutdownTarget()` passes `ECU_STATE_RESET` in the parameter `shutdownTarget`, in the parameter `sleepMode` it returns which of the configured reset modes was actually chosen.⌋()

[EcuM2337] ⌈If the pointer to the `sleepMode` parameter is NULL, `EcuM_GetLastShutdownTarget` shall simply ignore the `sleepMode` parameter and return the last shutdown target regardless of whether it was SLEEP or not. If Development Error Detection is activated, `EcuM_GetShutdownTarget` shall send the ECUM_E_NULL_POINTER development error to the DET module.⌋()

[EcuM2157] ⌈`EcuM_GetLastShutdownTarget` may return a shutdown target in a STARTUP phase that set late in a previous SHUTDOWN phase. If so, implementation specific limitations shall be clearly documented.⌋()

Rationale for EcuM2157
The `EcuM_GetLastShutdownTarget` function is intended primarily for use in the ECU STARTUP or RUN states. To simplify implementation, it is acceptable if the value is set in late shutdown phase for use during the next startup.

8.3.3.4 EcuM_SelectShutdownCause

[EcuM4050][

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

]()

8.3.3.5 EcuM_GetShutdownCause

[EcuM4051][

Service name:	EcuM_GetShutdownCause	
Syntax:	`Std_ReturnType EcuM_GetShutdownCause( EcuM_ShutdownCauseType* shutdownCause )`	
Service ID[hex]:	0x1c	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	shutdownCause	The selected cause of the next shutdown.
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service has failed, e.g. due to NULL pointer being passed
Description:	EcuM_GetShutdownCause returns the selected shutdown cause as set by EcuM_SelectShutdownCause. EcuM_GetShutdownCause is part of the ECU Manager Module port interface.	

]()

8.3.3.6 EcuM_GetMostRecentShutdown

[EcuM4052][

Service name:	EcuM_GetMostRecentShutdown
Syntax:	`Std_ReturnType EcuM_GetMostRecentShutdown( EcuM_StateType* target, uint8* mode, EcuM_ShutdownCauseType* cause,`

	uint32* time)	
Service ID[hex]:	0x1d	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	target	One of these values is returned: • ECUM_STATE_SLEEP • ECUM_STATE_RESET • ECUM_STATE_OFF
	mode	This parameter tells which of the configured sleep modes (target is ECUM_STATE_SLEEP) or which of the reset mechanisms (target is ECUM_STATE_RESET) was actually chosen.
	cause	The selected shutdown cause
	time	Absolute time of the shutdown if supported by hardware.
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, or no information is available
Description:	EcuM_GetMostRecentShutdown returns information about the most recent shutdown operation. EcuM_GetMostRecentShutdown is part of the ECU Manager Module port interface.	

]()

8.3.3.7 EcuM_GetNextRecentShutdown

[EcuM4053][

Service name:	EcuM_GetNextRecentShutdown	
Syntax:	Std_ReturnType EcuM_GetNextRecentShutdown(EcuM_StateType* target, uint8* mode, EcuM_ShutdownCauseType* cause, uint32* time)	
Service ID[hex]:	0x1e	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	target	One of these values is returned: • ECUM_STATE_SLEEP • ECUM_STATE_RESET • ECUM_STATE_OFF
	mode	This parameter tells which of the configured sleep modes (target is ECUM_STATE_SLEEP) or which of the reset mechanisms (target is ECUM_STATE_RESET) was actually chosen.
	cause	The selected shutdown cause
	time	Absolute time of the shutdown if supported by hardware.
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, or no information is available
Description:	EcuM_GetNextRecentShutdown returns information about the next recent shutdown operation. All stored shutdown information can be read by first calling EcuM_GetMostRecentShutdown and then looping over EcuM_GetNextRecentShutdown until an error is returned.	

	EcuM_GetNextRecentShutdown is part of the ECU Manager Module port interface.

]()

8.3.4 Wakeup Handling

8.3.4.1 EcuM_GetPendingWakeupEvents

[EcuM2827][

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

](BSW09126)

[EcuM1156] ⌈`EcuM_GetPendingWakeupEvents` shall return wakeup events which have been set to pending but not yet validated as bits set in a EcuM_WakeupSourceType bitmask.⌋()

[EcuM2172] ⌈`EcuM_GetPendingWakeupEvents` shall be callable from interrupt context, from OS context and an OS-free context.⌋()

[EcuM3003] ⌈Caveat of `EcuM_GetPendingWakeupEvents`: This function only returns the wakeup events with status `ECUM_WKSTATUS_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.	

](BSW09126)

[EcuM2683] ⌈`EcuM_ClearWakeupEvent` clears (NAND-operation) all pending events passed as a bit set in the `sources` in parameter (EcuM_WakeupSourceType bitmask) from the internal pending wakeup events variable, the internal validated events variable and the internal expired events variable (see section 7.6.3 Internal Representation of Wakeup States).⌋()

[EcuM2807] ⌈`EcuM_ClearWakeupEvent` shall 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( void )`	
Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant, Non-Interruptible	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupSourceType	All wakeup events
Description:	Gets validated wakeup events.	

⌋(BSW09126)

[EcuM2533] ⌈`EcuM_GetValidatedWakeupEvents` shall return wakeup events which have been set to validated in the internal validated events variable (see section 7.6.3 Internal Representation of Wakeup States) as bits set in a EcuM_WakeupSourceType bitmask.⌋()

[EcuM2532] ⌈`EcuM_GetValidatedWakeupEvents` shall 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( void )`	
Service ID[hex]:	0x19	
Sync/Async:	Synchronous	
Reentrancy:	Non-Reentrant, Non-Interruptible	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	EcuM_WakeupSourceType	All wakeup events: Returns all events that have been set and for which validation has failed. Events which do not need validation must never be reported by this

		function.
Description:	Gets expired wakeup events.	

](BSW09126)

[EcuM4076] [`EcuM_GetExpiredWakeupEvents` shall return wakeup events which have been set to validated in the internal expired events variable (see section 7.6.3 Internal Representation of Wakeup States) as bits set in a EcuM_WakeupSourceType bitmask.]()

[EcuM2589] [`EcuM_GetExpiredWakeupEvents` shall be callable from interrupt context, from OS context and an OS-free context.]()

8.3.5 Alarm Clock

8.3.5.1 EcuM_SetRelWakeupAlarm

[EcuM4054][

Service name:	EcuM_SetRelWakeupAlarm	

Syntax:	`Std_ReturnType EcuM_SetRelWakeupAlarm(`
	`  EcuM_UserType user,`
	`  uint32 time`
	`)`	

Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	The user that wants to set the wakeup alarm.
	time	Relative time from now in seconds.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed E_NOT_SUPPORTED: The service is not supported by this hardware E_EARLIER_ACTIVE: An earlier alarm is already set
Description:	EcuM_SetRelWakeupAlarm sets a user's wakeup alarm relative to the current point in time. EcuM_SetRelWakeupAlarm is part of the ECU Manager Module port interface.	

](BSW09186,BSW09190)

[EcuM4055] [If the relative time from now is earlier than the current wakeup time, `EcuM_SetRelWakeupAlarm` shall update the wakeup time.](BSW09186)

[EcuM4056] [If the relative time from now is later than the current wakeup time, `EcuM_SetRelWakeupAlarm` shall not update the wakeup time and shall return E_EARLIER_ACTIVE.](BSW09186)

8.3.5.2 EcuM_SetAbsWakeupAlarm

[EcuM4057][

Service name:	EcuM_SetAbsWakeupAlarm
Syntax:	`Std_ReturnType EcuM_SetAbsWakeupAlarm(`

	`EcuM_UserType user,`
	`uint32 time`
	)	

Service ID[hex]:	0x23	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	The user that wants to set the wakeup alarm.
	time	Absolute time in seconds. Note that, absolute alarms use knowledge of the current time.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed E_NOT_SUPPORTED: The service is not supported by this hardware E_EARLIER_ACTIVE: An earlier alarm is already set E_PAST: The given point in time has already passed
Description:	EcuM_SetAbsWakeupAlarm sets the user's wakeup alarm to an absolute point in time. EcuM_SetAbsWakeupAlarm is part of the ECU Manager Module port interface.	

](BSW09186,BSW09199)

[EcuM4058] [If the `time` parameter earlier than the current wakeup time, `EcuM_SetRelWakeupAlarm` shall update the wakeup time.](BSW09186)

[EcuM4059] [If the `time` parameter is later than the current wakeup time, `EcuM_SetRelWakeupAlarm` shall not update the wakeup time and shall return E_EARLIER_ACTIVE.](BSW09186)

[EcuM4060] [If the `time` parameter is earlier than now, `EcuM_SetRelWakeupAlarm` shall not update the wakeup time and shall return E_PAST.](BSW09186)

[EcuM3019] [E_EARLIER_ACTIVE and E_NOT_SUPPORTED shall be of type Std_ReturnType and represent the following values

- E_NOT_SUPPORTED = 2
- E_EARLIER_ACTIVE = 3

]()

8.3.5.3 EcuM_AbortWakeupAlarm

[EcuM4061][

Service name:	EcuM_AbortWakeupAlarm	

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

Service ID[hex]:	0x24	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	The user that wants to cancel the wakeup alarm.
Parameters (inout):	None	
Parameters (out):	None	

Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed E_NOT_SUPPORTED: The service is not supported by this hardware E_NOT_ACTIVE: No owned alarm found
Description:	Ecum_AbortWakeupAlarm aborts the wakeup alarm previously set by this user. EcuM_AbortWakeupAlarm is part of the ECU Manager Module port interface.	

]()

8.3.5.4 EcuM_GetCurrentTime

[EcuM4062][

Service name:	EcuM_GetCurrentTime	

Syntax:	`Std_ReturnType EcuM_GetCurrentTime(`
	`  uint32* time`
	`)`	

Service ID[hex]:	0x25	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	time	Absolute time in seconds since battery connect.
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_SUPPORTED: The service is not supported by this hardware
Description:	EcuM_GetCurrentTime returns the current value of the EcuM clock (i.e. the time since battery connect). EcuM_GetCurrentTime is part of the ECU Manager Module port interface.	

]()

8.3.5.5 EcuM_GetWakeupTime

[EcuM4063][

Service name:	EcuM_GetWakeupTime	

Syntax:	`Std_ReturnType EcuM_GetWakeupTime(`
	`  uint32* time`
	`)`	

Service ID[hex]:	0x26	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	time	Absolute time in seconds for next wakeup. 0xFFFFFFFF means no active alarm.
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_SUPPORTED: The service is not supported by this hardware
Description:	EcuM_GetWakeupTime returns the current value of the master alarm clock (the minimum absolute time of all user alarm clocks). EcuM_GetWakeupTime is part of the ECU Manager Module port interface.	

]()

8.3.5.6 EcuM_SetClock

[EcuM4064][

Service name:	EcuM_SetClock	
Syntax:	`Std_ReturnType EcuM_SetClock( EcuM_UserType user, uint32 time )`	
Service ID[hex]:	0x27	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	user	User that wants to set the clock
	time	Absolute time in seconds since battery connect.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed E_NOT_SUPPORTED: The service is not supported by this hardware E_NOT_ALLOWED: This service is is privileged (BSW only)
Description:	EcuM_SetClock sets the EcuM clock time to the provided value. This API is useful for testing the alarm services; Alarms that take days to expire can be tested. EcuM_SetClock is part of the ECU Manager Module port interface.	

](BSW09194)

8.3.6 Miscellaneous

8.3.6.1 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:	EcuM_SelectBootTarget selects a boot target. EcuM_SelectBootTarget is part of the ECU Manager Module port interface.	

]()

[EcuM2247] [The service `EcuM_SelectBootTarget` shall store the selected target in a way that is compatible with the boot loader.]()

Explanation for EcuM2247: This may mean format AND location. The implementer must ensure that the boot target information is placed at a safe location which then can be evaluated by the boot manager after a reset.

[EcuM3000] ⌈Caveat for the function `EcuM_SelectBootTarget:` This service may depend on the boot loader used. This service is only intended for use by SW-C's related to diagnostics (boot management).⌋()

8.3.6.2 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:	EcuM_GetBootTarget returns the current boot target - see EcuM_SelectBootTarget. EcuM_GetBootTarget is part of the ECU Manager Module port interface.	

⌋(BSW172)

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

⌋(BSW00425,BSW00373,BSW00376)
`EcuM_MainFunction` should be called on a periodic basis from an appropriate BSW task (i.e. a task under control of the BSW scheduler).

To determine the period, the system designer should consider:

- The function will perform wakeup validation (see 7.8 Wakeup Validation Protocol). The shortest validation timeout typically should limit the period.

- As a rule of thumb, the period of this function should be approximately half as long as the shortest validation timeout.

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

Terms and definitions:

Fixed cyclic:
Fixed cyclic means that one cycle time is defined at configuration and shall not be changed because functionality is requiring that fixed timing (e.g. filters).
Variable cyclic:
Variable cyclic means that the cycle times are defined at configuration, but might be mode dependent and therefore vary during runtime.
On pre condition:
On pre condition means that no cycle time can be defined. The function will be called when conditions are fulfilled. Alternatively, the function may be called cyclically however the cycle time will be assigned dynamically during runtime by other modules.

8.5 Callback Definitions

8.5.1 Callbacks from Wakeup Sources

8.5.1.1 EcuM_CheckWakeup

See *8.6.4.4* EcuM_CheckWakeup (EcuM2929) for a description of the `EcuM_CheckWakeup` function.

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

8.5.1.2 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.	

⌋(BSW00359,BSW00360,BSW00440,BSW09098)

[EcuM1117] ⌈`EcuM_SetWakeupEvent` sets (OR-operation) all events passed as a bit set in the `sources` in parameter (EcuM_WakeupSourceType bitmask) in the internal pending wakeup events variable (see section 7.6.3 Internal Representation of Wakeup States).⌋()

[EcuM2707] ⌈`EcuM_SetWakeupEvent` shall start the wakeup validation timeout timer according to section 7.6.4.3 Wakeup Validation Timeout.⌋()

[EcuM2867] ⌈If Development Error Reporting is turned on and parameter "`sources`" contains an unknown (unconfigured) wakeup source, `EcuM_SetWakeupEvent` shall not update its internal variable and shall send the `ECUM_E_UNKNOWN_WAKEUP_SOURCE` error message to the DET module instead.⌋()

[EcuM2171] ⌈`EcuM_SetWakeupEvent` must be callable from interrupt context, from OS context and an OS-free context.⌋(BSW00333)

8.5.1.3 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 that have been validated
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	After wakeup, the ECU State Manager will stop the process during the WAKEUP VALIDATION state/sequence to wait for validation of the wakeup event.This API service is used to indicate to the ECU Manager module that the wakeup events indicated in the sources parameter have been validated.	

](BSW00359,BSW00360,BSW00440)

[EcuM4078] [`EcuM_ValidateWakeupEvent` sets (OR-operation) all events passed as a bit set in the `sources` in parameter (EcuM_WakeupSourceType bitmask) in the internal validated wakeup events variable (see section 7.6.3 Internal Representation of Wakeup States).]()

[EcuM4079] [`EcuMValidateWakeupEvent` shall invoke `BswM_EcuM_CurrentWakeup` with its `sources` parameter and state value ECUM_WKSTATUS_VALIDATED.]()

[EcuM2645] [`EcuM_ValidateWakeupEvent` shall invoke `ComM_EcuM_WakeUpIndication` for each wakeup event if the `EcuMComMChannelRef` parameter (see EcuM101_Conf) in the `EcuMGeneral` configuration container for the corresponding wakeup source is configured.]()

[EcuM2868] [If Development Error Reporting is turned on and the `sources` parameter contains an unknown (unconfigured) wakeup source, `EcuM_ValidateWakeupEvent` shall ignore the call and send the `ECUM_E_UNKNOWN_WAKEUP_SOURCE` error message to the DET module.]()

[EcuM2345] [`EcuM_ValidateWakeupEvent` shall be callable from interrupt context, from OS context, and an OS-free context.](BSW00333)

[EcuM2790] [`EcuM_ValidateWakeupEvent` shall return without effect for all sources except communication channels when called while the ECU Manager module is in the UP Phase (see section 7.1.2 UP Phase).]()

[EcuM2791] [`EcuM_ValidateWakeupEvent` shall have full effect in any ECU Phase for those sources that correspond to a communication channel (see EcuM2645),.]()

8.6 Callout Definitions

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

Since callouts are not ECU Manager module functions 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" occur. In this situation it is not possible to continue processing and the ECU must be stopped. The integrator may choose the modality how the ECU is stopped, i.e. reset, halt, restart, safe state etc.	

⌋()

The ECU Manager module can invoke `EcuM_ErrorHook`: in all phases

Class of `EcuM_ErrorHook:` Mandatory

EcuM_ErrorHook is integration code and the integrator is free to define additional individual error codes to be passed as the `reason` parameter. These codes shall not conflict with the development and production error codes as defined in Table 1 and Table 7 nor with the standard error codes, i.e. E_OK, E_NOT_OK, etc.

8.6.2 Callouts from the STARTUP Phase

8.6.2.1 EcuM_AL_SetProgrammableInterrupts

[EcuM4085][

Service name:	EcuM_AL_SetProgrammableInterrupts
Syntax:	`void EcuM_AL_SetProgrammableInterrupts( void )`
Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	EcuM_AL_SetProgrammableInterrupts shall set the interrupts on ECUs with programmable interrupts.

]()

8.6.2.2 EcuM_AL_DriverInitZero

[EcuM2905] [

Service name:	EcuM_AL_DriverInitZero
Syntax:	`void EcuM_AL_DriverInitZero( void )`
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.

]()

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

The ECU Manager module configuration tool must generate a default implementation of the `EcuM_AL_DriverInitZero` callout (EcuM2905) from the sequence of modules defined in the `EcuMDriverInitListZero` configuration container (see EcuM114_Conf). See also EcuM2559 and EcuM2730.

8.6.2.3 EcuM_DeterminePbConfiguration

[EcuM2906] [

Service name:	EcuM_DeterminePbConfiguration
Syntax:	`EcuM_ConfigType* EcuM_DeterminePbConfiguration(`

	void)	
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.	

]()

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

Content is written manually.

8.6.2.4 EcuM_AL_DriverInitOne

[EcuM2907] [

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.	

]()

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

The ECU Manager module configuration tool must generate a default implementation of the `EcuM_AL_DriverInitOne` callout from the sequence of modules defined in the `EcuMDriverInitListOne` configuration container (see EcuM111_Conf). See also 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.

8.6.3 Callouts from the SHUTDOWN Phase

8.6.3.1 EcuM_OnGoOffOne

[EcuM2916] [

Service name:	EcuM_OnGoOffOne

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

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.

]()

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

8.6.3.2 EcuM_OnGoOffTwo

[EcuM2917] [

Service name:	EcuM_OnGoOffTwo

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

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.

]()

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

8.6.3.3 EcuM_AL_SwitchOff

[EcuM2920] [

Service name:	EcuM_AL_SwitchOff

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

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.

]()

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

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

8.6.3.4 EcuM_AL_Reset

[EcuM4065][

Service name:	EcuM_AL_Reset	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	reset	Type of reset to be performed.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This callout shall take the code for resetting the ECU.	

]()

8.6.4 Callouts from the SLEEP Phase

8.6.4.1 EcuM_EnableWakeupSources

[EcuM2918] ⌈

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:	The ECU Manager Module calls EcuM_EnableWakeupSource to allow the system designer to notify wakeup sources defined in the wakeupSource bitfield that SLEEP will be entered and to adjust their source accordingly.

⌋()

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

[EcuM2546] ⌈The ECU Manager module shall derive the wakeup sources to be enabled (and used as the `wakeupSource` parameter) from the `EcuMWakeupSource` (see ECUM152_Conf) bitfield configured for the current sleep mode.⌋()

8.6.4.2 EcuM_GenerateRamHash

[EcuM2919] ⌈

Service name:	EcuM_GenerateRamHash

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

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

⌋()

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

8.6.4.3 EcuM_SleepActivity

[EcuM2928] ⌈

Service name:	EcuM_SleepActivity
Syntax:	`void EcuM_SleepActivity(` `  void`

	)
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.

]()

The ECU Manager module invokes `EcuM_SleepActivity` periodically during the Poll Sequence (see section 7.5.3 Activities in the Poll Sequence) if the MCU is not halted (i.e. clock is reduced).
Note: If called from the Poll sequence the EcuMcalls this callout functions in a blocking loop at maximum frequency. The callout implementation must ensure by other means if callout code shall be executed with a lower period. The integrator may choose any method to control this, e.g. with the help of OS counters, OS alarms, or Gpt timers.

8.6.4.4 EcuM_CheckWakeup

[EcuM2929] [

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.

]()

The ECU Manager module invokes `EcuM_CheckWakeup` periodically during the Poll Sequence (see section 7.5.3 Activities in the Poll Sequence) if the MCU is not halted, or when handling a wakeup interrupt.
Note: If called from the Poll sequence the EcuMcalls this callout functions in a blocking loop at maximum frequency. The callout implementation must ensure by other means if callout code shall be executed with a lower period. The integrator may choose any method to control this, e.g. with the help of OS counters, OS alarms, or Gpt timers.

[EcuM4080] ⌈The ECU Manager module shall derive the wakeup sources to be checked (and used as the `wakeupSource` parameter) from the `EcuMWakeupSource` (see ECUM152_Conf) bitfield configured for the current sleep mode. The integration code used for this callout must determine which wakeup sources must be checked.⌋()

8.6.4.5 EcuM_CheckRamHash

[EcuM2921] ⌈

Service name:	EcuM_CheckRamHash	

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

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. In case of applied multi core and existence of Satellite-EcuM(s): this API will be called by the Master-EcuM only.	

⌋()

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

8.6.4.6 EcuM_DisableWakeupSources

[EcuM2922] ⌈

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 ECU Manager Module calls EcuM_DisableWakeupSources to set the wakeup source(s) defined in the wakeupSource bitfield so that they are not able to wake the ECU up.	

]()

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

[EcuM4084] ⌈The ECU Manager module shall derive the wakeup sources to be checked (and used as the `wakeupSource` parameter) from the internal pending events variable (NOT operation). The integration code used for this callout must determine which wakeup sources must be checked.⌋()

8.6.4.7 EcuM_AL_DriverRestart

[EcuM2923] ⌈

Service name:	EcuM_AL_DriverRestart	
Syntax:	`void EcuM_AL_DriverRestart( 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 the wakeup case.	

]()

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

The ECU Manager module Configuration Tool shall generate a default implementation of the `EcuM_AL_DriverRestart` callout from the sequence of modules defined in the `EcuMDriverRestartList` configuration container (see EcuM115_Conf). See also EcuM2561, EcuM2559 and EcuM2730.

8.6.5 Callouts from the UP Phase

8.6.5.1 EcuM_StartWakeupSources

[EcuM2924] ⌈

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.

⌋()

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

8.6.5.2 EcuM_CheckValidation

[EcuM2925] ⌈

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().

⌋()

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

8.6.5.3 EcuM_StopWakeupSources

[EcuM2926] ⌈

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

](

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

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_EcuM_CurrentState	Function called by EcuM to indicate the current ECU Operation Mode.
BswM_EcuM_CurrentWakeup	Function called by EcuM to indicate the current state of a wakeup source.
BswM_Init	Initializes the BSW Mode Manager.
ComM_EcuM_WakeUpIndication	Notification of a wake up on the corresponding channel.
DisableAllInterrupts	--
EnableAllInterrupts	--
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	--
SchM_Deinit	Function for de-initialization of the SchM module.
SchM_Init	Function for initialization of the SchM module.
ShutdownOS	--
StartOS	--
WdgM_PerformReset	Instructs the Watchdog Manager to cause a watchdog reset.

Table 8 - 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.
CanTrcv_Init	Initializes the CanTrcv module.
Can_Init	This function initializes the module.
Det_Init	Service to initialize the Development Error Tracer.
Det_ReportError	Service to report development errors.
Dio_Init	Initializes the module.
EthTrcv_Init	Initializes the Ethernet Transceiver Driver
Eth_Init	Initializes the Ethernet Driver
Fls_Init	Initializes the Flash Driver.
FrTrcv_Init	This service initializes the FrTrcv.

Fr_Init	Initalizes the Fr.
GetCoreID	The function returns a unique core identifier.
GetEvent	--
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.
LinTrcv_Init	Initializes the Lin Transceiver Driver module.
Lin_Init	Initializes the LIN module.
Port_Init	Initializes the Port Driver module.
Pwm_Init	Service for PWM initialization.
SetEvent	--
ShutdownAllCores	After this service the OS on all AUTOSAR cores is shut down. Allowed at TASK level and ISR level and also internally by the OS. The function will never return. The function will force other cores into a shutdown.
Spi_Init	Service for SPI initialization.
StartCore	It is not supported to call this function after StartOS(). The function starts the core specified by the parameter CoreID. The OUT parameter allows the caller to check whether the operation was successful or not. If a core is started by means of this function StartOS shall be called on the core.
Wdg_Init	Initializes the module.

Table 9 - Optional Interfaces

]()

8.7.3 Configurable interfaces

There are no configurable interfaces.

8.8 API Parameter Checking

[EcuM3009] [If development error detection is enabled for this module, then all functions shall test input parameters and running conditions and use the following error codes in an adequate way:

- ECUM_E_UNINIT
- ECUM_E_SERVICE_DISABLED
- ECUM_E_NULL_POINTER
- ECUM_E_INVALID_PAR

Specific development errors are listed in the functions, where they apply.
](BSW00323)

9 Sequence Charts

9.1 State Sequences

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

- Figure 4: STARTUP Phase
- Figure 5 – StartPreOS Sequence
- Figure 6 – StartPostOS Sequence
- Figure 8 – SHUTDOWN Phase
- Figure 9 – OffPreOS Sequence
- Figure 10 – OffPostOS Sequence
- Figure 11 – SLEEP Phase
- Figure 12 – GoSleep Sequence
- Figure 13 – Halt Sequence
- Figure 14 – Poll Sequence
- Figure 15 - WakeupRestart Sequence
- Figure 17 – The WakeupValidation Sequence

9.2 Wakeup Sequences

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

9.2.1 GPT Wakeup Sequences

The General Purpose Timer (GPT) is one of the possible wake up sources. Usually the GPT is started before the ECU is put to sleep and the hardware timer causes an interrupt when it expires. The interrupt wakes the microcontroller, and executes the interrupt handler in the GPT module. It informs the ECU State Manager module that a GPT wake up has occurred. In order to distinguish different GPT channels that caused the wake up, the integrator can assign a different wake up source identifier to each GPT channel. Figure 37 shows the corresponding sequence of calls.

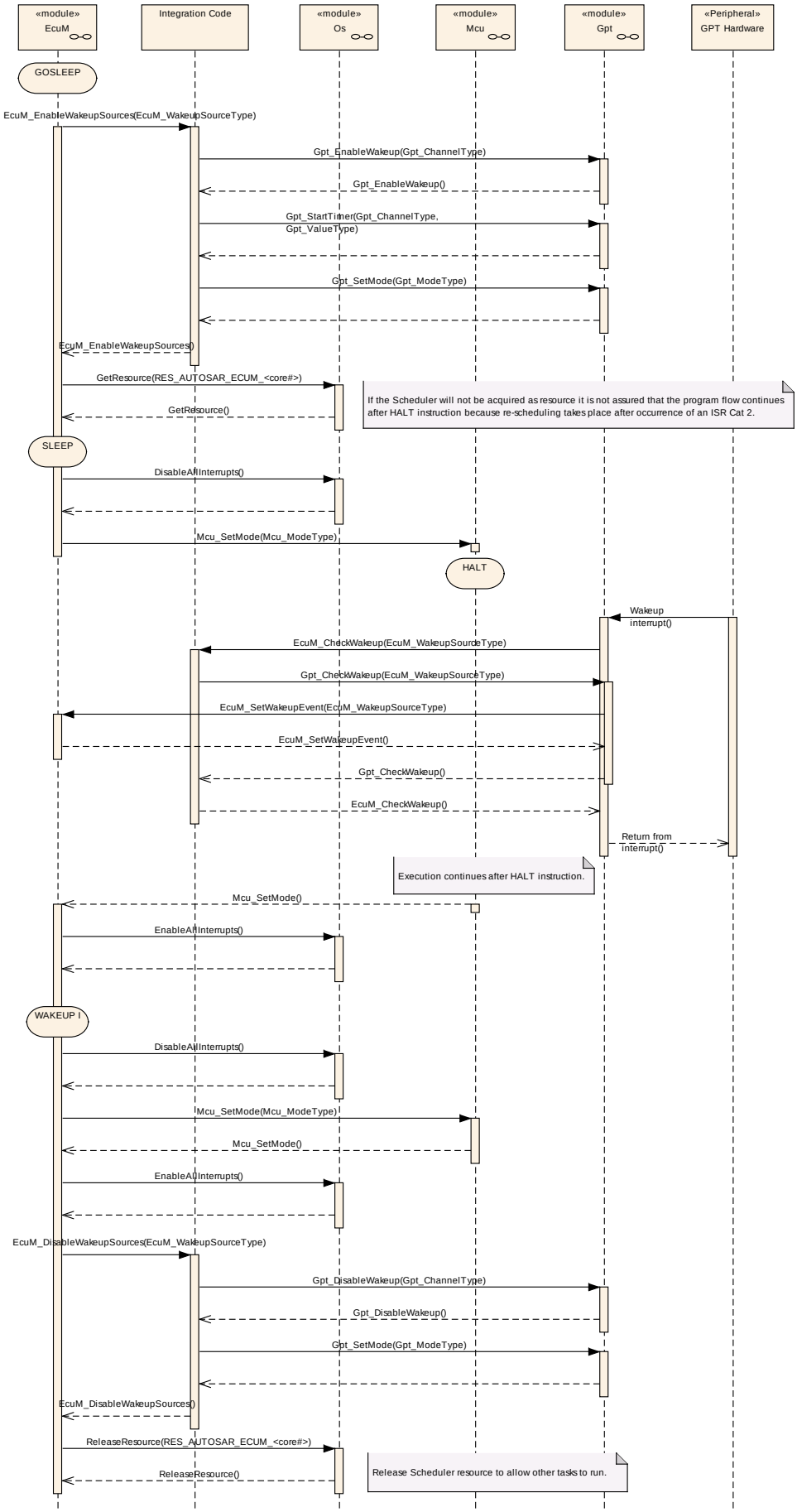
«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Gpt
«Peripheral»
GPT Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Gpt_EnableWakeup(Gpt_ChannelType)
Gpt_EnableWakeup()
Gpt_StartTimer(Gpt_ChannelType,
Gpt_ValueType)
Gpt_SetMode(Gpt_ModeType)
EcuM_EnableWakeupSources()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

Figure 37 – 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 38.

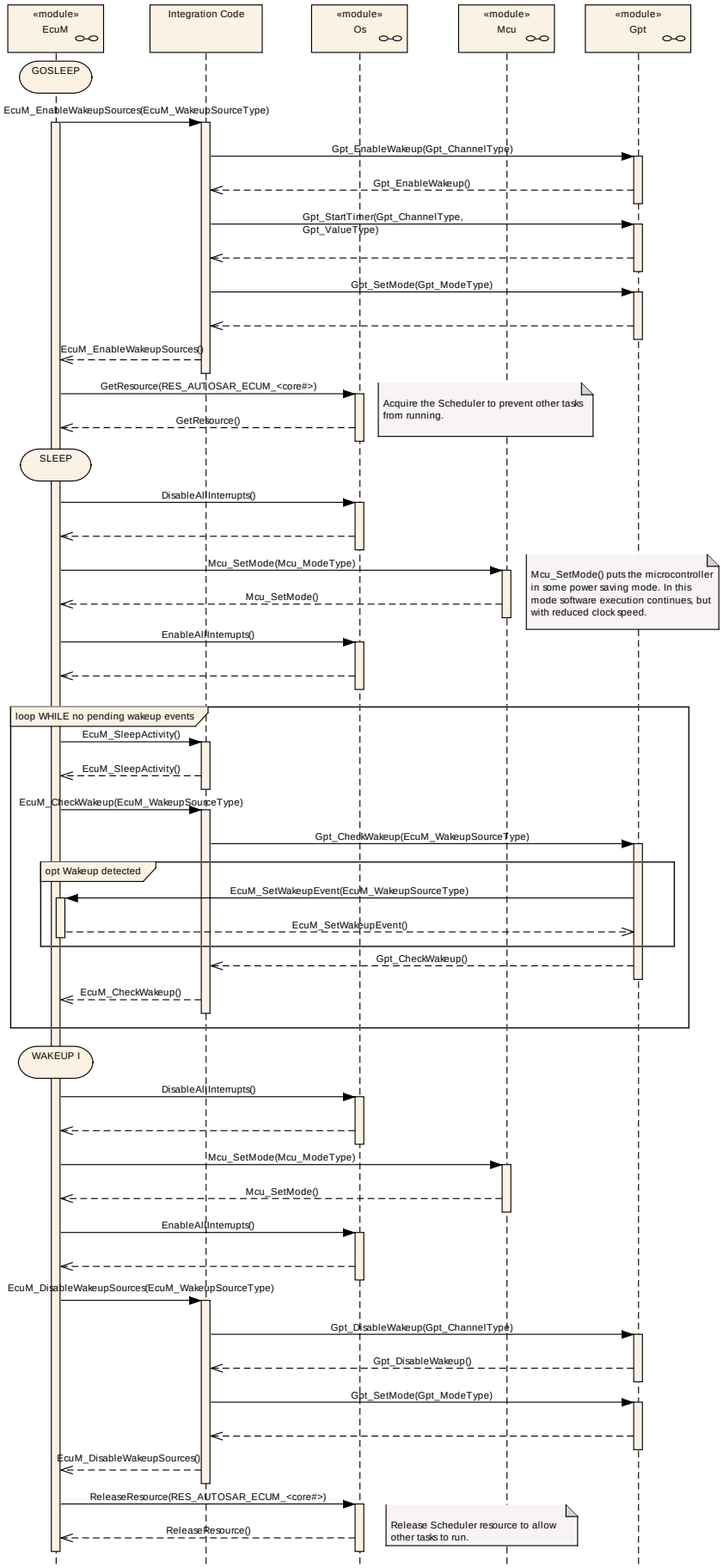
«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Gpt
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Gpt_EnableWakeup(Gpt_ChannelType)
Gpt_EnableWakeup()
Gpt_StartTimer(Gpt_ChannelType,
Gpt_ValueType)
Gpt_SetMode(Gpt_ModeType)
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_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

Figure 38 – GPT wake up by polling

9.2.2 ICU Wakeup Sequences

The Input Capture Unit (ICU) is another wake up source. In contrast to GPT, the ICU driver is not itself the wake up source. It is just the module that processes the wake up interrupt. Therefore, only the driver of the wake up source can tell if it was responsible for that wake up. This makes it necessary for EcuM_CheckWakeup (see EcuM2929) to ask the module that is the actual wake up source. In order to know which module to ask, the ICU has to pass the identifier of the wake up source to EcuM_CheckWakeup.

For shared interrupts the Integration Code may have to check multiple wake up sources within EcuM_CheckWakeup (see EcuM2929). To this end, the ICU has to pass the identifiers of all wake up sources that may have caused this interrupt to EcuM_CheckWakeup. Note that, EcuM_WakeupSourceType (see 8.2.4 EcuM_WakeupSourceType) contains one bit for each wake up source, so that multiple wake up sources can be passed in one call.

Figure 39 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 14 – Poll Sequence.

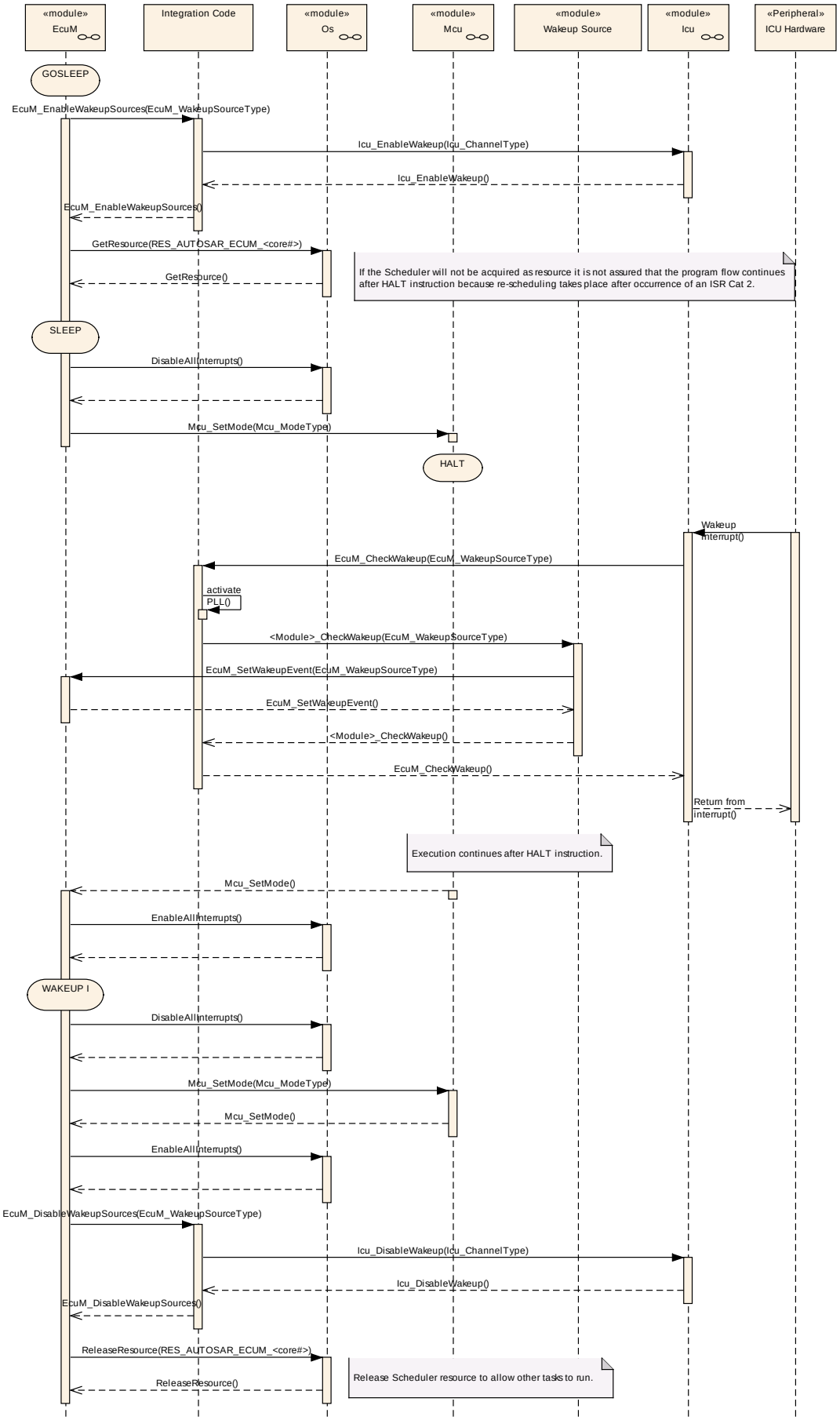
«module»
EcuM
Integration Code
«module»
Os
«module»
Mcu
«module»
Wakeup Source
«module»
Icu
«Peripheral»
ICU Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
Icu_EnableWakeup(Icu_ChannelType)
Icu_EnableWakeup()
EcuM_EnableWakeupSources()
GetResource(RES_AUTOSAR_ECUM_<core#>)
GetResource()
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurrence of an ISR Cat 2.
SLEEP
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
HALT
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
activate PLL()
<Module>_CheckWakeup(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
EcuM_SetWakeupEvent()
<Module>_CheckWakeup()
EcuM_CheckWakeup()
Return from interrupt()
Execution continues after HALT instruction.
Mcu_SetMode()
EnableAllInterrupts()
WAKEUP I
DisableAllInterrupts()
Mcu_SetMode(Mcu_ModeType)
Mcu_SetMode()
EnableAllInterrupts()
EcuM_DisableWakeupSources(EcuM_WakeupSourceType)
Icu_DisableWakeup(Icu_ChannelType)
Icu_DisableWakeup()
EcuM_DisableWakeupSources()
ReleaseResource(RES_AUTOSAR_ECUM_<core#>)
ReleaseResource()
Release Scheduler resource to allow other tasks to run.

Figure 39 – ICU wake up by interrupt

9.2.3 CAN Wakeup Sequences

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

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

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

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

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

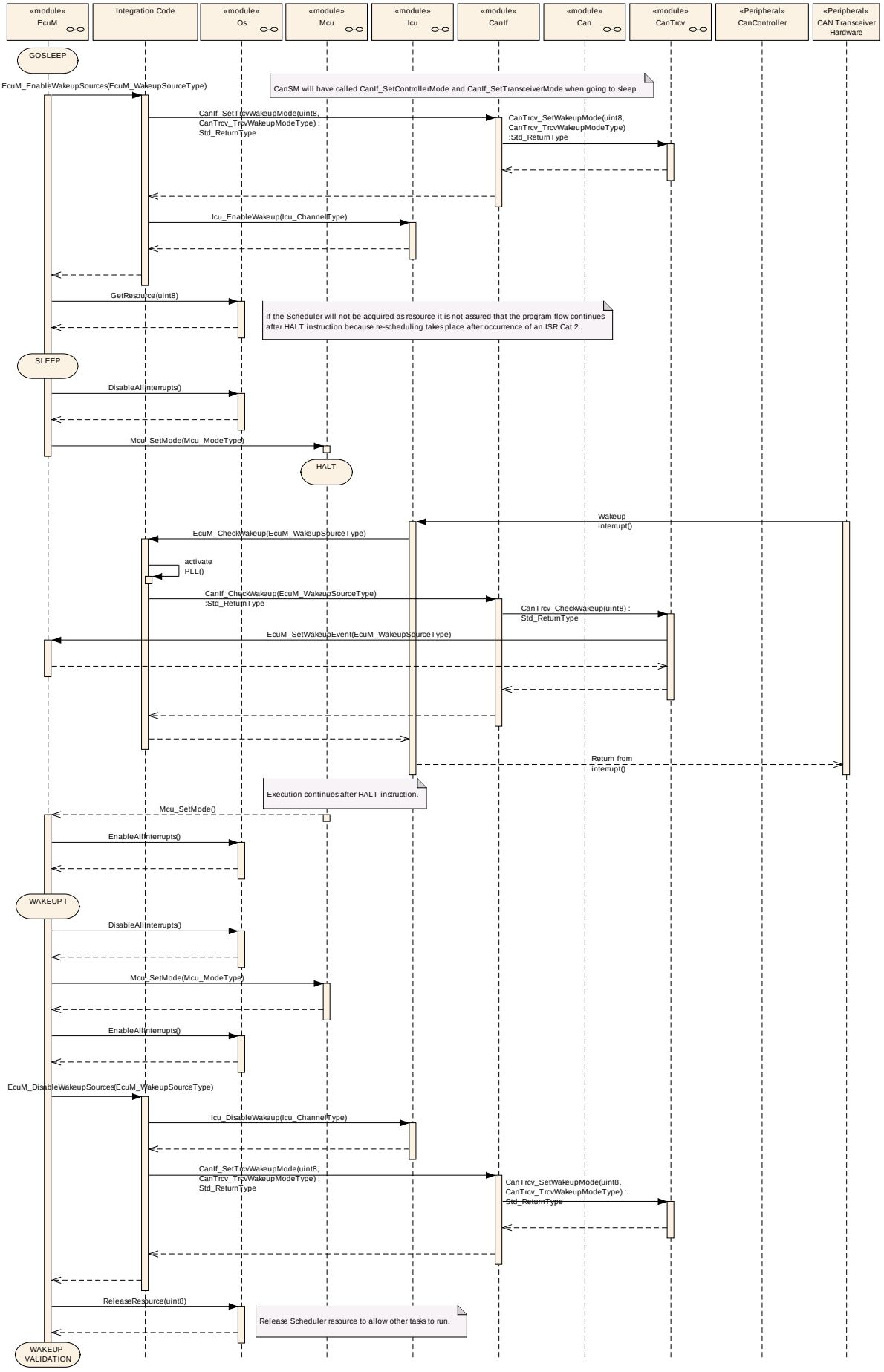
«module» EcuM
Integration Code
«module» Os
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode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

Figure 40 – CAN transceiver wake up by interrupt

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

A CAN controller wakeup by interrupt works similar to the GPT wakeup. Here the interrupt handler and the CheckWakeup functionality are both encapsulated in the CAN Driver module, as shown in Figure 41.

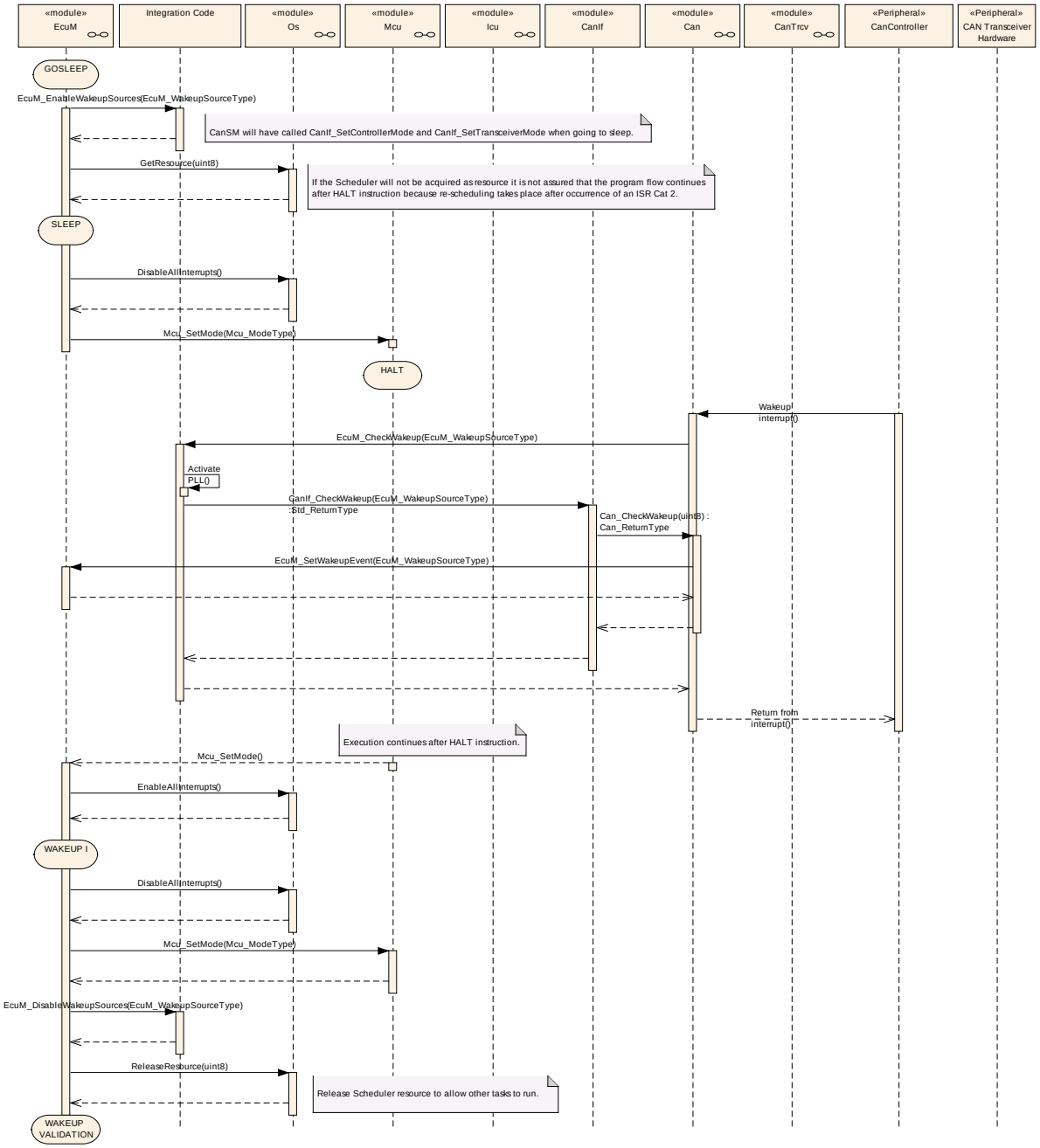

Figure 41 – CAN controller wake up by interrupt

Wake up by polling is possible both for CAN transceiver and controller. The ECU State Manager module will regularly check the CAN Interface module, which in turn asks either the CAN Driver module or the CAN Transceiver Driver module depending

on the wake up source parameter passed to the CAN Interface module, as shown in Figure 42.

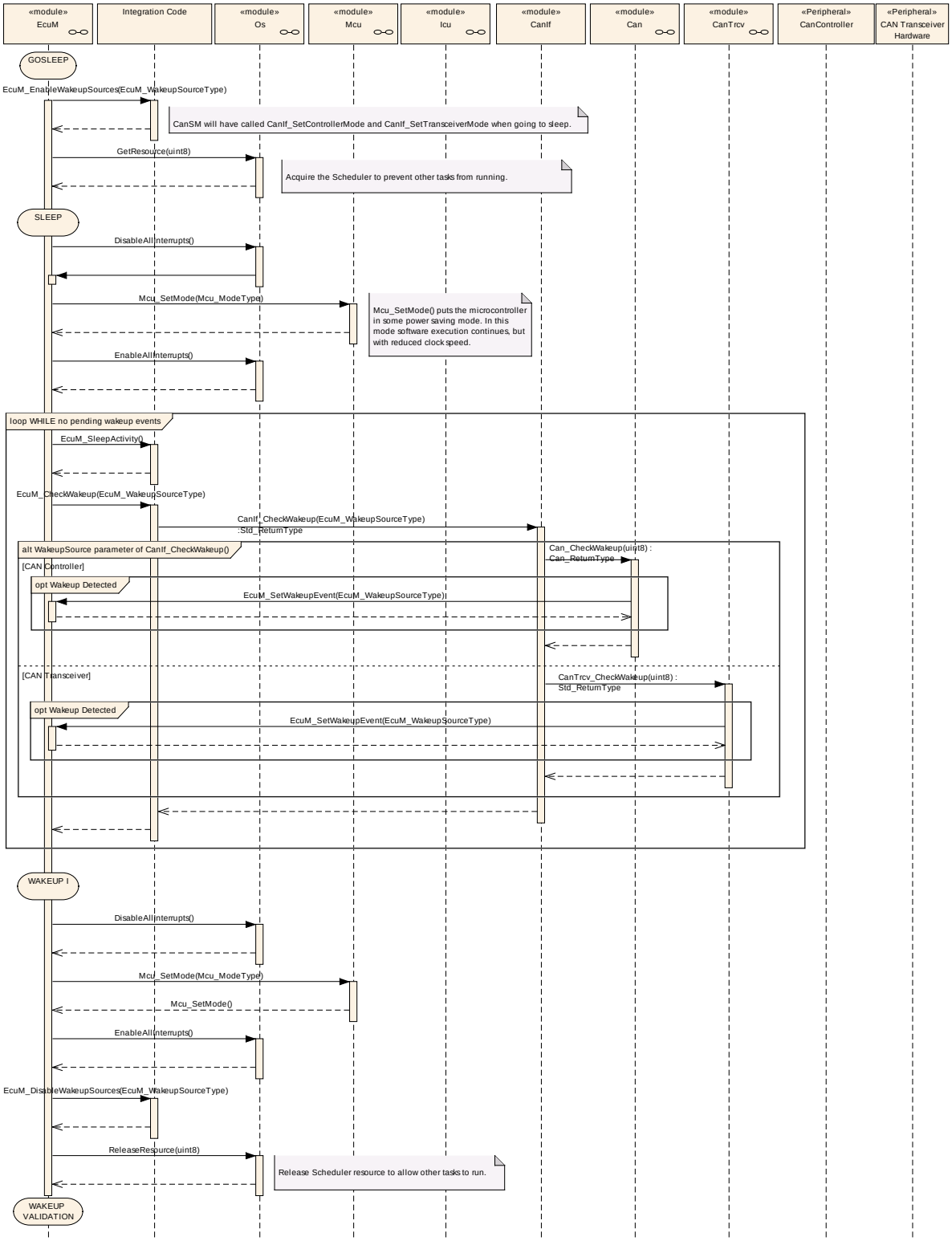

Figure 42 – CAN controller or transceiver wake up by polling

After the detection of a wake up event from the CAN transceiver or controller by either interrupt or polling, the wake up event can be validated (see EcuM2566). This

is done by switching on the corresponding CAN transceiver and controller in EcuM_StartWakeupSources (see EcuM2924). 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 CanIf are set to “Online”, it will forward CAN messages.
The CanIf recognizes the successful reception of at least one message and records it as a successful validation. During validation the ECU State Manager module regularly checks the CanIf in Integrator Code EcuM_CheckValidation (see EcuM2925).
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 (see EcuM2926) and go back to sleep.

The resulting sequence is shown in Figure 43.

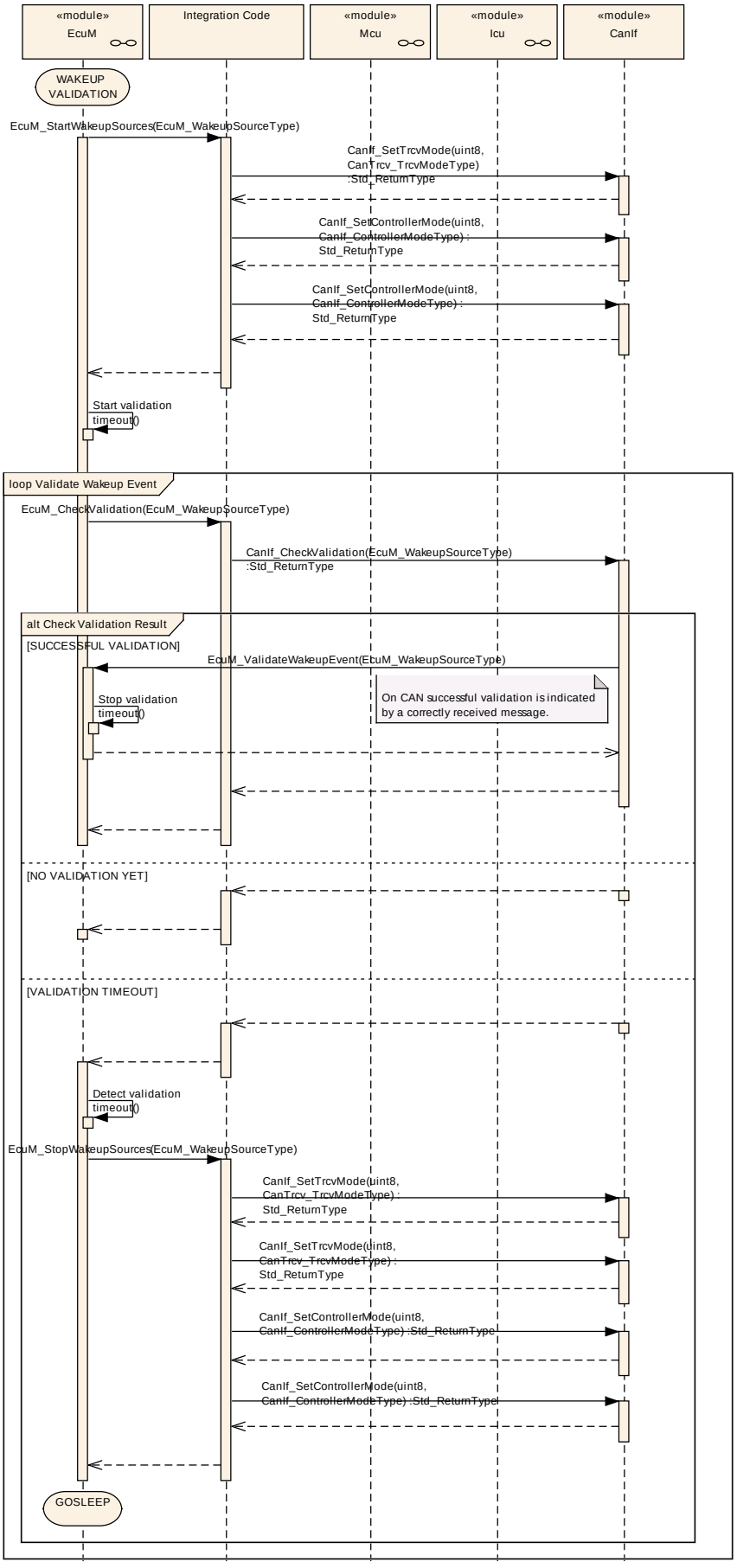
«module»
EcuM
Integration Code
«module»
Mcu
«module»
Icu
«module»
CanIf
WAKEUP VALIDATION
EcuM_StartWakeupSources(EcuM_WakeupSourceType)
CanIf_SetTrcvMode(uint8, CanTrcv_TrcvModeType) :Std_ReturnType
CanIf_SetControllerMode(uint8, CanIf_ControllerModeType) : Std_ReturnType
CanIf_SetControllerMode(uint8, CanIf_ControllerModeType) : Std_ReturnType
Start validation timeout()
loop Validate Wakeup Event
EcuM_CheckValidation(EcuM_WakeupSourceType)
CanIf_CheckValidation(EcuM_WakeupSourceType) :Std_ReturnType
alt Check Validation Result
[SUCCESSFUL VALIDATION]
EcuM_ValidateWakeupEvent(EcuM_WakeupSourceType)
Stop validation timeout()
On CAN successful validation is indicated by a correctly received message.
[NO VALIDATION YET]
[VALIDATION TIMEOUT]
Detect validation timeout()
EcuM_StopWakeupSources(EcuM_WakeupSourceType)
CanIf_SetTrcvMode(uint8, CanTrcv_TrcvModeType) : Std_ReturnType
CanIf_SetTrcvMode(uint8, CanTrcv_TrcvModeType) : Std_ReturnType
CanIf_SetControllerMode(uint8, CanIf_ControllerModeType) :Std_ReturnType
CanIf_SetControllerMode(uint8, CanIf_ControllerModeType) :Std_ReturnType
GOSLEEP

Figure 43 – CAN wake up validation

9.2.4 LIN Wakeup Sequences

Figure 44 shows the LIN transceiver wakeup via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.

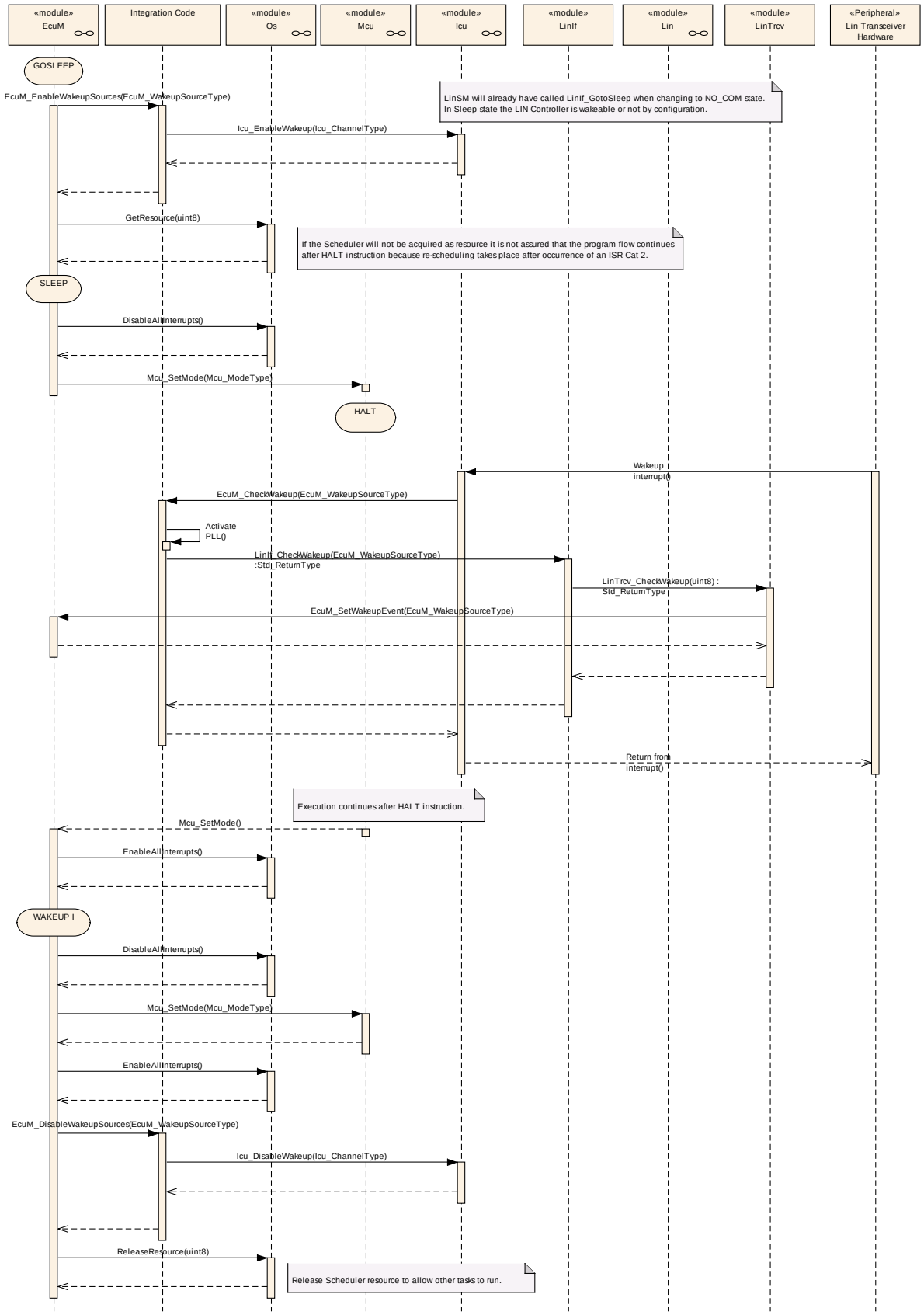

Figure 44 – LIN transceiver wake up by interrupt

As shown in Figure 46, 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.

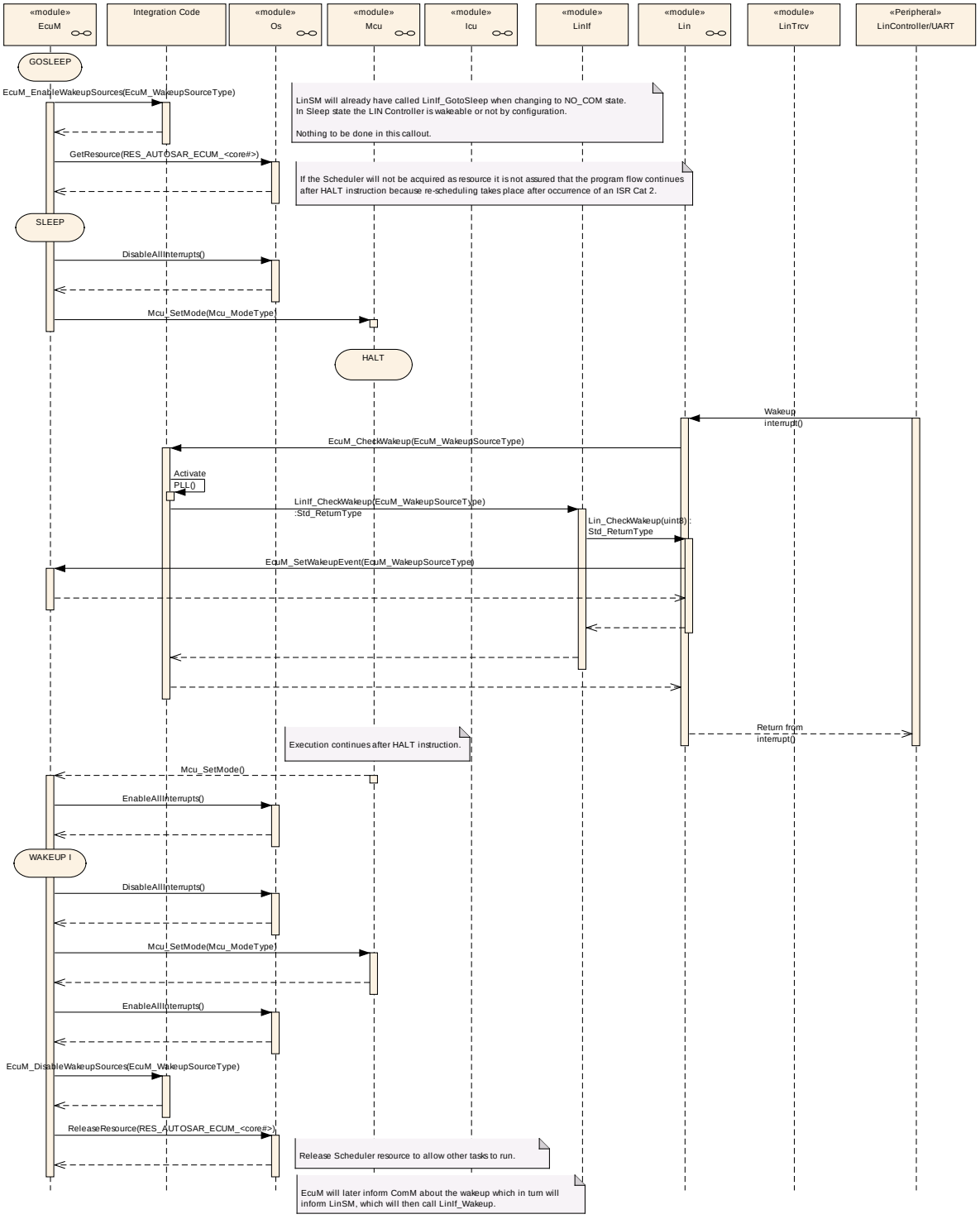

Figure 45 – LIN controller wake up by interrupt

Wake up by polling is possible for LIN transceiver and controller. The ECU State Manager module will regularly check the LIN Interface module, which in turn asks either the LIN Driver module or the LIN Transceiver Driver module, as shown in Figure 46.

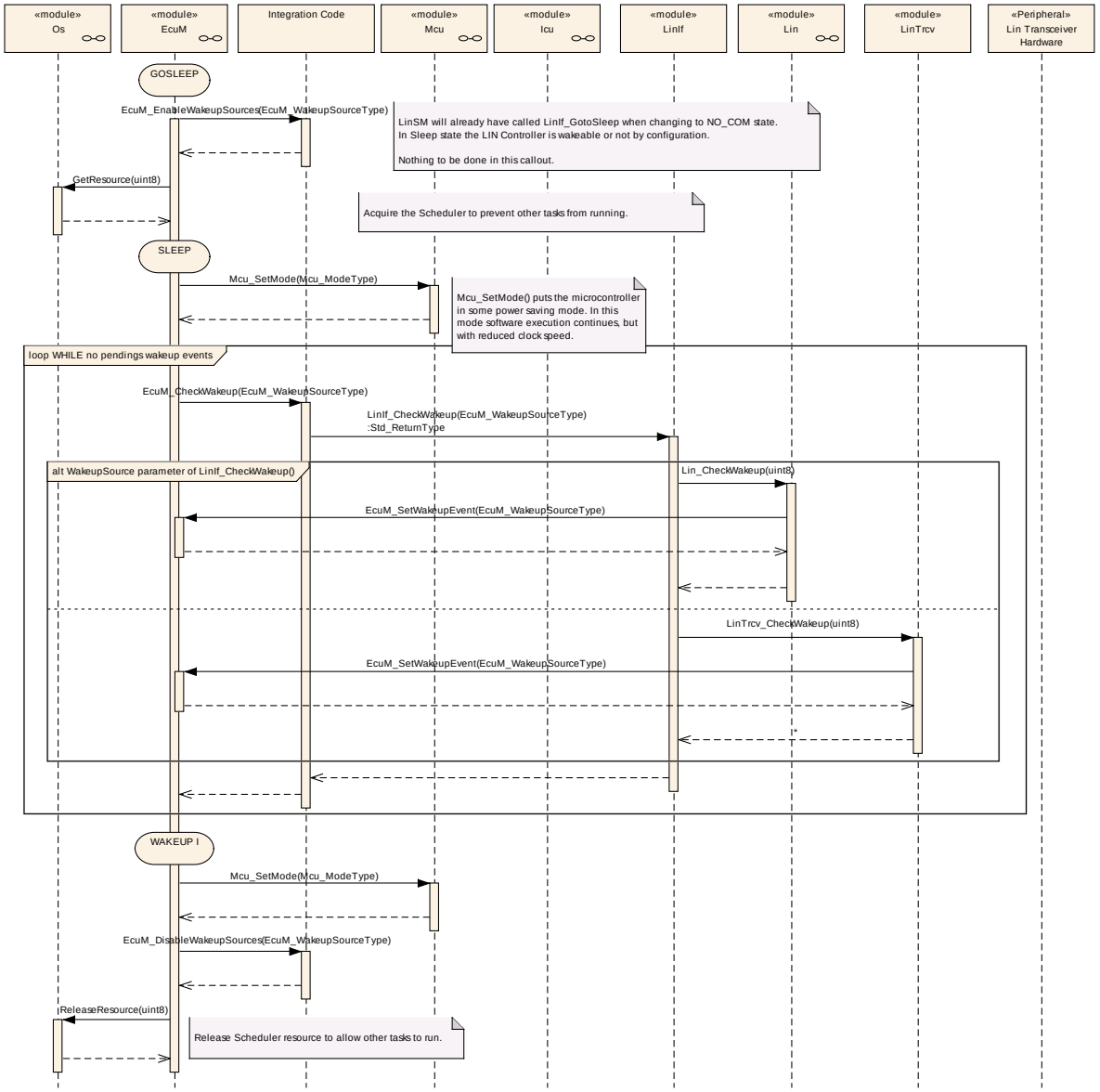

Figure 46 – LIN controller or transceiver wake up by polling

Note that LIN does not require wakeup validation.

9.2.5 FlexRay Wakeup Sequences

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

Figure 47 shows the FlexRay transceiver wakeup via interrupt. The interrupt is usually handled by the ICU Driver as described in Chapter 9.2.2.

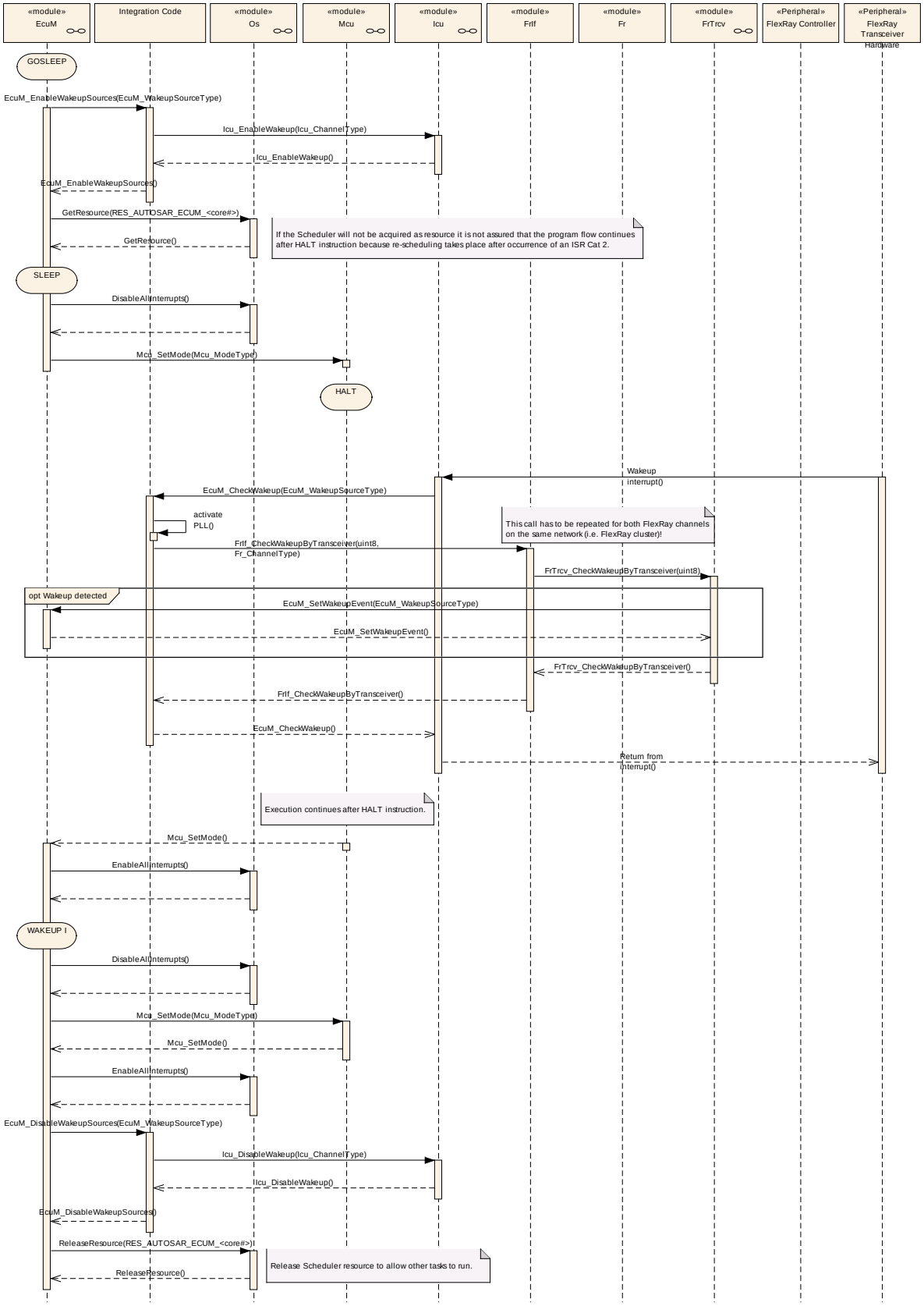

Figure 47 – FlexRay transceiver wake up by interrupt

Note that in EcuM_CheckWakeup (see EcuM2929) there need to be two separate calls to FrIf_WakeupByTransceiver, one for each FlexRay channel.

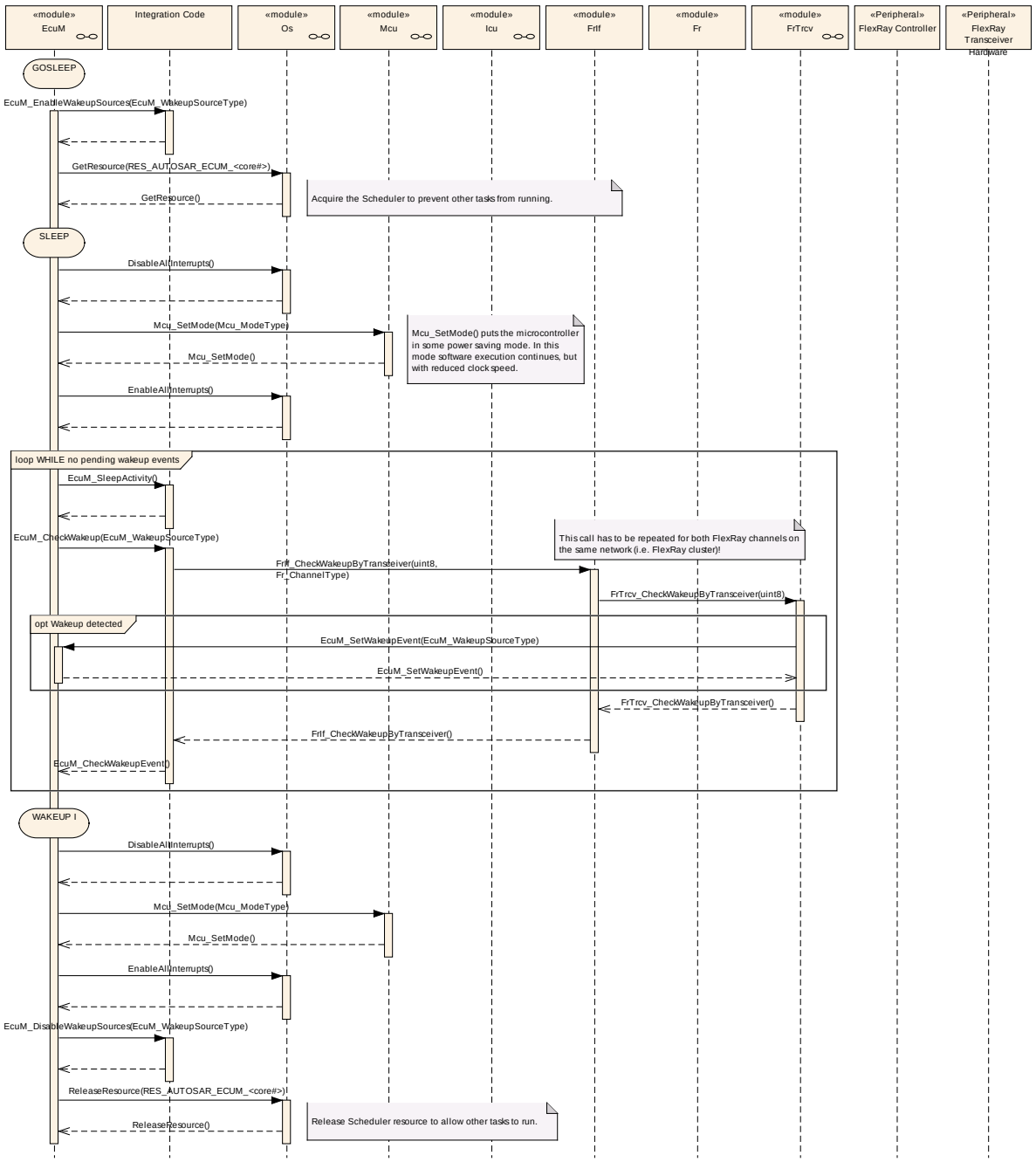

Figure 48 – FlexRay transceiver wake up by polling

9.2.6 TCP/IP Wakeup Sequences

With TCP/IP there can be no wake up from the bus. There is a wake up line connected to the ICU. All TCP/IP wake ups are therefore handled as normal ICU wake ups. Refer to section 9.2.1 GPT Wakeup Sequences for details.

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapters 10.2 and 10.3 specify the structure (containers) and the parameters of the module ECU Manager.

Chapter 10.4 specifies published information of the module ECU State Manager.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [5]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

[EcuM3007] ⌈The ECU State Manager shall support the configuration variant VARIANT-POST-BUILD: This configuration variant contains a mix of pre-compile time, link time and post-build time parameters ⌋(BSW00344, BSW00404,BSW00405,BSW00345)
The configuration class of each parameter is defined in chapters 10.2 and 10.3.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- all configuration parameters are kept in containers.

- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
<Reference a valid (sub)container by its name, e.g.,CanController>	*<Specifies the possible number of instances of the referenced container and its contained configuration parameters.* *Possible values:* *<multiplicity>* *<min_multiplicity..max_multiplicity>* *>*	*<Describe the scope of the referenced sub-container if known or mark it as "- -".* *The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network.* *Possible values of scope :* *instance, module, ECU, network>* *<Describe the dependencies with respect to the scope if known ot mark it as "- -".>*

Possible configuration times

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	Description
x	The configuration parameter shall be of configuration class Pre-compile time.
--	The configuration parameter shall never be of configuration class Pre-compile time.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	Description
x	The configuration parameter shall be of configuration class Link time.
--	The configuration parameter shall never be of configuration class Link time.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	Description
x	The configuration parameter shall be of configuration class Post Build and no specific implementation is required.
L	Loadable - the configuration parameter shall be of configuration class Post Build and only one configuration parameter set resides in the ECU.
M	Multiple - the configuration parameter shall be of configuration class Post Build and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class Post Build.

10.2 Common Containers and configuration parameters

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

The following containers contain various references to initialization structures of BSW modules. NULL shall be a valid reference meaning 'no configuration data available' but only if the implementation of the initialized BSW module supports this.

10.2.1 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.
EcuMFixedGeneral	0..1	This container holds the general, pre-compile configuration parameters for the EcuMFixed. Only applicable if EcuMFixed is implemented.
EcuMFlexGeneral	0..1	This container holds the general, pre-compile configuration parameters for the EcuMFlex. Only applicable if EcuMFlex is implemented.
EcuMGeneral	1	This container holds the general, pre-compile configuration parameters.

10.2.2 EcuMGeneral

SWS Item	**ECUM116_Conf :**
Container Name	EcuMGeneral
Description	This container holds the general, pre-compile configuration parameters.
Configuration Parameters	

SWS Item	**ECUM108_Conf :**
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM117_Conf :**		
Name	EcuMIncludeDem {ECUM_INCLUDE_DEM}		
Description	If enabled, the according BSW module will be included to the ECU State Manager.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM118_Conf :**		
Name	EcuMIncludeDet {ECUM_INCLUDE_DET}		
Description	If defined, the according BSW module will be initialized by the ECU State Manager		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM121_Conf :**		
Name	EcuMMainFunctionPeriod {ECUM_MAIN_FUNCTION_PERIOD}		
Description	This parameter defines the schedule period of EcuM_MainFunction. Unit: [s]		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency	dependency: EcuM2594		

SWS Item	**ECUM149_Conf :**		
Name	EcuMVersionInfoApi		
Description	Switches the version info API on or off		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	

Scope / Dependency	

No Included Containers

10.2.3 EcuMConfiguration

SWS Item	ECUM103_Conf :
Container Name	EcuMConfiguration{EcuM_Configuration} [Multi Config Container]
Description	This container contains the configuration (parameters) of the ECU State Manager.
Configuration Parameters	

Included Containers		
Container Name	Multiplicity	Scope / Dependency
EcuMCommonConfiguration	1	This container contains the common configuration (parameters) of the ECU State Manager.
EcuMFixedConfiguration	0..1	This container contains the configuration (parameters) of the EcuMFixed. Only applicable if EcuMFixed is implemented.
EcuMFlexConfiguration	0..1	This container contains the configuration (parameters) of the EcuMFlex. Only applicable if EcuMFlex is implemented.

10.2.4 EcuMCommonConfiguration

SWS Item	ECUM181_Conf :
Container Name	EcuMCommonConfiguration
Description	This container contains the common configuration (parameters) of the ECU State Manager.
Configuration Parameters	

SWS Item	ECUM102_Conf :		
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	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	--		
ConfigurationClass	Pre-compile time	--	
	Link time	X	VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUM104_Conf :		
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	**ECUM183_Conf :**		
Name	EcuMOSResource		
Description	This parameter is a reference to a OS ressource which is used to bring the ECU into sleep mode. In case of multi core each core shall have an own OsResource.		
Multiplicity	1..*		
Type	Reference to [OsResource]		
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.
EcuMDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in this container and can be extended by vendor specific error references.
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.
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
EcuMSleepMode	1..*	These containers describe the configured sleep modes. The names of these containers specify the symbolic names of the different sleep modes.
EcuMWakeupSource	1..*	These containers describe the configured wakeup sources.

10.2.5 EcuMDefaultShutdownTarget

SWS Item	**ECUM105_Conf :**
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	**ECUM107_Conf :**		
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	EcucEnumerationParamDef		
Range	EcuMStateOff	Corresponds to ECUM_STATE_OFF in EcuM_StateType.	
	EcuMStateReset	Corresponds to ECUM_STATE_RESET in EcuM_StateType. This literal is only be applicable for EcuMFlex.	
	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	**ECUM205_Conf :**		
Name	EcuMDefaultResetModeRef		
Description	If EcuMDefaultShutdownTarget is EcuMStateReset, this parameter selects the default reset mode. Otherwise this parameter may be ignored.		
Multiplicity	0..1		
Type	Reference to [EcuMResetMode]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**ECUM106_Conf :**		
Name	EcuMDefaultSleepModeRef		
Description	If EcuMDefaultShutdownTarget is EcuMStateSleep, this parameter selects the default sleep mode. Otherwise this parameter may be ignored.		
Multiplicity	0..1		
Type	Reference to [EcuMSleepMode]		
ConfigurationClass	***Pre-compile time***	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.6 EcuMDemEventParameterRefs

SWS Item	**ECUM160_Conf :**
Container Name	EcuMDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in this container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**ECUM162_Conf :**		
Name	ECUM_E_ALL_RUN_REQUESTS_KILLED		
Description	Reference to the DemEventParameter which shall be issued when the error "ECUM_E_ALL_RUN_REQUESTS_KILLED" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM163_Conf :**		
Name	ECUM_E_CONFIGURATION_DATA_INCONSISTENT		
Description	Reference to the DemEventParameter which shall be issued when the error "ECUM_E_CONFIGURATION_DATA_INCONSISTENT" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM161_Conf :**		
Name	ECUM_E_RAM_CHECK_FAILED		
Description	Reference to the DemEventParameter which shall be issued when the error "ECUM_E_RAM_CHECK_FAILED" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.7 EcuMDriverInitListOne

SWS Item	**ECUM111_Conf :**
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..*	These containers describe the entries in a driver init list.

10.2.8 EcuMDriverInitListZero

SWS Item	**ECUM114_Conf :**
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..*	These containers describe the entries in a driver init list.

10.2.9 EcuMDriverRestartList

SWS Item	**ECUM115_Conf :**
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..*	These containers describe the entries in a driver init list.

10.2.10 EcuMDriverInitItem

SWS Item	**ECUM110_Conf :**
Container Name	EcuMDriverInitItem
Description	These containers describe the entries in a driver init list.
Configuration Parameters	

SWS Item	**ECUM123_Conf :**		
Name	EcuMModuleID {ModuleID}		
Description	Short name of the module to be initialized, e.g. Mcu, Gpt etc.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM124_Conf :**		
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) and in case of multiple instantiation an uint8 value to identify the instance of the module(<MSN>_CtrlIdx) shall be passed as arguments.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.11 EcuMSleepMode

SWS Item	**ECUM131_Conf :**
Container Name	EcuMSleepMode
Description	These containers describe the configured sleep modes. The names of these containers specify the symbolic names of the different sleep modes.
Configuration Parameters	

SWS Item	**ECUM132_Conf :**	
Name	EcuMSleepModeId	
Description	This ID identifies this sleep mode in services like EcuM_SelectShutdownTarget.	
Multiplicity	1	
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)	
Range	0 .. 255	

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

SWS Item	**ECUM136_Conf :**		
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	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM133_Conf :**		
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	**ECUM152_Conf :**		
Name	EcuMWakeupSourceMask		
Description	These parameters are references to the wakeup sources 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

10.2.12 EcuMWakeupSource

SWS Item	**ECUM150_Conf :**
Container Name	EcuMWakeupSource{EcuM_WakupSource}
Description	These containers describe the configured wakeup sources.
Configuration Parameters	

SWS Item	**ECUM148_Conf :**
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. When the timeout is not instantiated, there is no validation routine and the ECU Manager shall not validate the wakeup source.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM151_Conf :**		
Name	EcuMWakeupSourceId {WakeupSourceName}		
Description	This parameter defines the identifier of this wakeup source.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 31		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM153_Conf :**		
Name	EcuMWakeupSourcePolling		
Description	This parameter describes if the wakeup source needs polling.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM101_Conf :**		
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			

SWS Item	**ECUM128_Conf :**		
Name	EcuMResetReasonRef {ResetReason}		
Description	This parameter describes the mapping of reset reasons detected by the MCU driver into wakeup		

	sources.		
Multiplicity	1		
Type	Reference to [McuResetReasonConf]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3 EcuM-Flex Containers and configuration parameters

10.3.1 EcuMFlexGeneral

SWS Item	**ECUM168_Conf :**
Container Name	EcuMFlexGeneral
Description	This container holds the general, pre-compile configuration parameters for the EcuMFlex. Only applicable if EcuMFlex is implemented.
Configuration Parameters	

SWS Item	**ECUM199_Conf :**		
Name	EcuMAlarmClockPresent {ECUM_ALARM_CLOCK_PRESENT}		
Description	This flag indicates whether the optional AlarmClock feature is present.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM196_Conf :**		
Name	EcuMEnableDefBehaviour {ECUM_DEF_BEHAVIOUR_ENABLED}		
Description	Switches the defensive behaviour on or off.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM171_Conf :**		
Name	EcuMResetLoopDetection {ECUM_RESET_LOOP_DETECTION}		
Description	If false, no reset loop detection is performed.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	ECUM200_Conf :		
Name	EcuMAlarmWakeupSource		
Description	This parameter describes the reference to the EcuMWakeupSource being used for the EcuM AlarmClock.		
Multiplicity	0..1		
Type	Reference to [EcuMWakeupSource]		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.2 EcuMFlexConfiguration

SWS Item	ECUM167_Conf :
Container Name	EcuMFlexConfiguration
Description	This container contains the configuration (parameters) of the EcuMFlex. Only applicable if EcuMFlex is implemented.
Configuration Parameters	

SWS Item	ECUM182_Conf :		
Name	EcuMFlexModuleConfigurationRef {InitConfiguration}		
Description	These parameters contain references to the init structure of the corresponding BSW module.		
Multiplicity	0..*		
Type	Choice reference to [AdcConfigSet , CanConfigSet , DemConfigSet , FlsConfigSet , GptChannelConfigSet , IcuConfigSet , LinGlobalConfig , McuModuleConfiguration , PortConfigSet , PwmChannelConfigSet , SpiDriver , WdgMConfigSet , WdgSettingsConfig]		
ConfigurationClass	Pre-compile time	--	
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	ECUM204_Conf :		
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	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
EcuMAlarmClock	0..*	These containers describe the configured alarm clocks. The name of these conatiners allows giving a symbolic name to one alarm clock.
EcuMFlexUserConfig	1..*	These containers describe the identifiers that are needed to

		refer to a software component or another appropriate entity in the system which uses the EcuMFlex Interfaces.
EcuMGoDownAllowedUsers	0..1	This container describes the collection of allowed users which are allowed to call the EcuM_GoDown API.
EcuMResetMode	1..*	These containers describe the configured reset modes. The name of these containers allows one of the following symbolic names to be given to the different reset modes: - ECUM_RESET_MCU - ECUM_RESET_WDGM - ECUM_RESET_IO.
EcuMSetClockAllowedUsers	0..1	This container describes the collection of allowed users which are allowed to call the EcuM_SetClock API.
EcuMShutdownCause	1..*	These containers describe the configured shut down or reset causes. The name of these containers allows to give one of the following symbolic names to the diffenrent shut down causes: - ECUM_CAUSE_ECU_STATE - ECU state machine entered a state for shutdown, - ECUM_CAUSE_WDGM - WdgM detected failure, - ECUM_CAUSE_DCM - Dcm requests shutdown (split into UDS services?), - and values from configuration.
EcuMShutdownTarget	1..*	These containers describe the configured shut down targets. The name of these containers allows to give symbolic names to the different shut down targets.

10.3.3 EcuMAlarmClock

SWS Item	**ECUM184_Conf :**
Container Name	EcuMAlarmClock
Description	These containers describe the configured alarm clocks. The name of these conatiners allows giving a symbolic name to one alarm clock.
Configuration Parameters	

SWS Item	**ECUM186_Conf :**		
Name	EcuMAlarmClockId		
Description	This ID identifies this alarmclock.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM188_Conf :**		
Name	EcuMAlarmClockTimeOut		
Description	This parameter allows to define a timout for this alarm clock.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM195_Conf :**		
Name	EcuMAlarmClockUser {AlarmClockUser}		
Description	This parameter allows an alarm to be assigned to a user.		
Multiplicity	1		
Type	Reference to [EcuMFlexUserConfig]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.4 EcuMFlexUserConfig

SWS Item	**ECUM201_Conf :**
Container Name	EcuMFlexUserConfig{EcuM_Flexed_User}
Description	These containers describe the identifiers that are needed to refer to a software component or another appropriate entity in the system which uses the EcuMFlex Interfaces.
Configuration Parameters	

SWS Item	**ECUM146_Conf :**		
Name	EcuMFlexUser {User}		
Description	Parameter used to identify one user.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**ECUM203_Conf :**		
Name	EcuMFlexEcucPartitionRef		
Description	Denotes in which "EcucPartition" the user of the EcuM is executed.		
Multiplicity	0..1		
Type	Reference to [EcucPartition]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-POST-BUILD
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.5 EcuMGoDownAllowedUsers

SWS Item	**ECUM206_Conf :**
Container Name	EcuMGoDownAllowedUsers
Description	This container describes the collection of allowed users which are

	allowed to call the EcuM_GoDown API.
Configuration Parameters	

SWS Item	**ECUM207_Conf :**		
Name	EcuMGoDownAllowedUserRef		
Description	These parameters describe the references to the users which are allowed to call the EcuM_GoDown API.		
Multiplicity	1..*		
Type	Reference to [EcuMFlexUserConfig]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.6 EcuMSetClockAllowedUsers

SWS Item	**ECUM197_Conf :**
Container Name	EcuMSetClockAllowedUsers
Description	This container describes the collection of allowed users which are allowed to call the EcuM_SetClock API.
Configuration Parameters	

SWS Item	**ECUM198_Conf :**		
Name	EcuMSetClockAllowedUserRef		
Description	These parameters describe the references to the users which are allowed to call the EcuM_SetClock API.		
Multiplicity	1..*		
Type	Reference to [EcuMFlexUserConfig]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.7 EcuMResetMode

SWS Item	**ECUM172_Conf :**
Container Name	EcuMResetMode
Description	These containers describe the configured reset modes. The name of these containers allows one of the following symbolic names to be given to the different reset modes: - ECUM_RESET_MCU - ECUM_RESET_WDGM - ECUM_RESET_IO.
Configuration Parameters	

SWS Item	**ECUM173_Conf :**
Name	EcuMResetModeId
Description	This ID identifies this reset mode in services like EcuM_SelectShutdownTarget.
Multiplicity	1
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)

Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.8 EcuMShutdownCause

SWS Item	**ECUM175_Conf :**
Container Name	EcuMShutdownCause
Description	These containers describe the configured shut down or reset causes. The name of these containers allows to give one of the following symbolic names to the diffenrent shut down causes: - ECUM_CAUSE_ECU_STATE - ECU state machine entered a state for shutdown, - ECUM_CAUSE_WDGM - WdgM detected failure, - ECUM_CAUSE_DCM - Dcm requests shutdown (split into UDS services?), - and values from configuration.
Configuration Parameters	

SWS Item	**ECUM176_Conf :**		
Name	EcuMShutdownCauseId		
Description	This ID identifies this shut down cause.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3.9 EcuMShutdownTarget

SWS Item	**ECUM178_Conf :**
Container Name	EcuMShutdownTarget
Description	These containers describe the configured shut down targets. The name of these containers allows to give symbolic names to the different shut down targets.
Configuration Parameters	

SWS Item	**ECUM179_Conf :**
Name	EcuMShutdownTargetId
Description	This ID identifies this shut down target in services like EcuM_SelectShutdownTarget.
Multiplicity	1
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)

Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.4 Published Information

[EcuM001_PI:⌈The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1]. ⌋(BSW00402)

Additional module-specific published parameters are listed below if applicable.

11 Not applicable requirements

[EcuM9999][These requirements are not applicable to this specification.](BSW159,BSW167,BSW00406,BSW00437,BSW168,BSW00426,BSW00427,BSW00432,BSW00434,BSW00417,BSW00422,BSW161,BSW162,BSW005,BSW00415,BSW00325,BSW164,BSW00326,BSW160,BSW00453,BSW00413,BSW00347,BSW00307,BSW00450,BSW00410,BSW00314,BSW00348,BSW00353,BSW00361,BSW00439,BSW00449,BSW00308,BSW00309,BSW00330,BSW00446,BSW010,BSW00341,BSW00334,BSW09116,BSW09102,BSW09017,BSW09101,BSW09195,BSW09197,BSW09198,BSW09237)