

# **Star Shine**

---

**None**

*Jian.Jiang*

*Copyright © 2023 Jian.Jiang*

## Table of contents

---

1. 星星世界	3
2. Mode Manger	4
2.1 EcuM	4
2.2 BswM	92
3. information safety	125
3.1 SecOC	125
4. 关于	151
4.1 发行说明	151
4.2 为MkDocs做贡献	152
4.3 许可	153

## 1. 星星世界

---



## 2. Mode Manger

---

### 2.1 EcuM

---

#### 2.1.1 EcuM

##### 概述

1. 初始化和取消初始化操作系统、SchM和BswM以及一些基本软件驱动程序模块
2. ECU配置休眠和关机
3. 管理ECU上的所有唤醒事件
4. EcuM提供唤醒验证协议，以区分“真实”唤醒事件和“不稳定”唤醒事件
5. 分步骤启动
6. 多核ECU: 在ECU的所有核心上协调启动、关机、休眠和唤醒

##### 特殊名称

术语	描述
被动唤醒	由连接的总线引起的唤醒，而不是计时器或传感器活动等内部事件
关机目标	ECU必须在进入休眠状态、断电或重置之前关闭。因此，SLEEP、OFF和RESET是有效的停机目标。通过选择关闭目标，应用程序可以将其对下一次关闭后ECU行为的期望传达给ECU管理器模块。
唤醒事件	引起唤醒的物理事件。CAN消息或切换IO线可以是唤醒事件。类似地，内部SW表示（例如中断）也可以称为唤醒事件
唤醒原因	唤醒原因是唤醒事件，它是上次唤醒的实际原因。
唤醒源	处理唤醒事件的外围设备或ECU部件称为唤醒源。

##### 限制

关断目标OFF只能使用ECU特殊硬件（例如电源保持电路）来实现。如果此硬件不可用，则此规范建议改为发出重置。但是，允许其他默认行为。

##### 依赖其他模块

###### MCU

初始化MCU

###### 具有唤醒功能的外网设备

唤醒源必须由驱动程序处理和封装。

驱动程序必须调用EcuM\_SetWakeupEvent以通知EcuM模块检测到挂起的唤醒事件。驱动程序不仅必须在ECU在睡眠阶段等待唤醒事件时调用EcuM\_SetWakeupEvent,还必须在驱动程序初始化阶段以及正常运行期间EcuM\_MainFunction运行时调用。

驱动程序必须提供显示函数才能将唤醒源置于睡眠状态。此功能应将唤醒源置于节能惰性运行模式，并重新启动唤醒通知机制。

如果唤醒源能够生成虚假事件1，则

- 驱动程序
- 使用驱动程序的软件堆栈
- 其他的合适BSW模块

必须为唤醒事件提供验证标注或调用ECU管理器模块的验证函数。如果不需要验证，则此要求不适用于相应的唤醒源。

## 操作系统

ECU 管理器模块启动 AUTOSAR 操作系统并关闭它。ECU 管理器模块定义了协议在操作系统启动之前如何处理控制，以及在操作系统关闭后如何处理控制。

## BSW调度器

ECU 管理器模块初始化 BSW 调度程序，ECU 管理器模块还包含EcuM\_MainFunction，该模块计划定期评估唤醒请求并更新闹钟。

## BSWM

BSW 模式管理器只能在模式管理运行后管理 ECU 状态机 – 即在初始化 SchM 之后，直到 SchM 被取消初始化或停止。当 BSW 模式管理器不运行时，ECU 管理器模块将控制 ECU。

- ECU 管理器模块在 ECU 启动后立即获得控制权，并在初始化 SchM 和 BswM 后将控制权移交给 BSW 模式管理器。
- BswM 将 ECU 的控制权传回 ECU 管理器模块，以锁定操作系统并处理唤醒事件。
- BswM 还会在操作系统在关机时停止之前立即将控制权传回 ECU 管理器。
- 验证唤醒源时，ECU Manager 模块通过模式切换请求指示唤醒源状态对 BswM 的更改。

## 软件组件

ECU 管理器模块处理以下 ECU 范围的属性：

- Shutdown targets.

## 2.1.2 Function

### 功能描述

- 启动阶段一直持续到模式管理设施运行为止。基本上，启动阶段包括启动模式管理所需的最少活动： 初始化低级驱动程序，启动操作系统以及初始化BSW调度程序和BSW模式管理器模块。类似地，关断阶段与启动阶段相反，启动阶段是模式管理被取消初始化的。
- UP阶段包含默认模式，以防使用ECU模式处理。这些模式之间的转换是通过 ECU状态管理器模块和BSW模式管理器模块 之间的合作完成的。

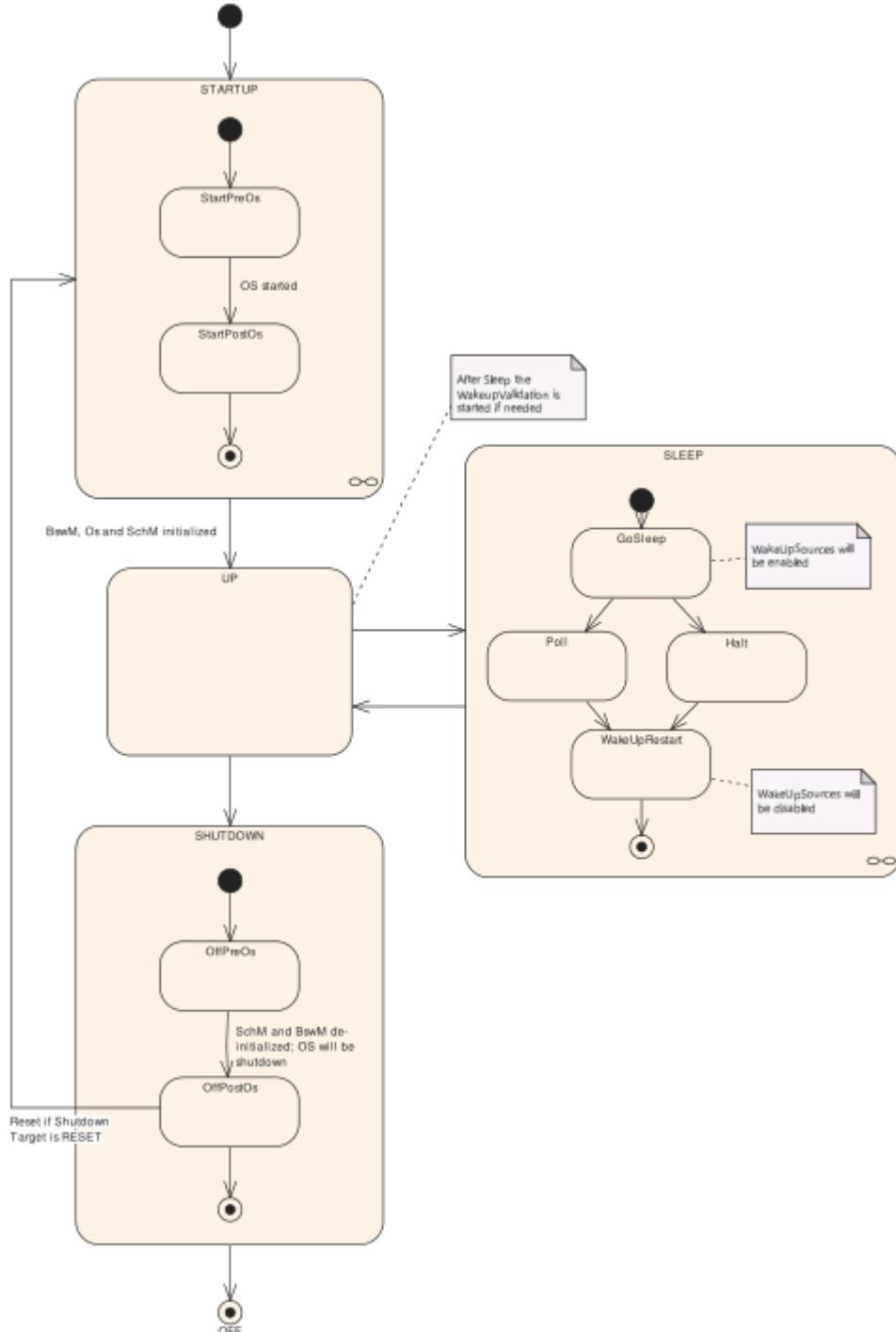


Figure 7.1: Phases of the ECU Manager

不同阶段的功能：

1. 启动阶段：目的是将基本软件模块初始化到通用模式管理设施正常运行的程度。:smile:
2. UP阶段：当 BSW 调度程序启动并调用BswM\_Init时，UP 阶段开始。BSW和SWC运行直到它们准备好关闭或者休眠ECU.
3. SHUTDOWN阶段： The SHUTDOWN phase handles the controlled shutdown of basic software modules and finally results in the selected shutdown target OFF or RESET .
4. SLEEP阶段： 在该阶段节能，一般，不执行代码，但仍提供电源。（EcuM根据预期或意外的唤醒事件唤醒ECU，但是应该忽略意外的唤醒事件，所以 EcuM提供了一个协议（指定了处理唤醒源的驱动程序和ECUM之间的协助过程）来验证唤醒事件）
5. OFF阶段：ECU在断电时进入关闭状态。ECU在这种状态下可以唤醒，但仅适用于具有集成电源控制的唤醒源。

#### STARTUP 阶段

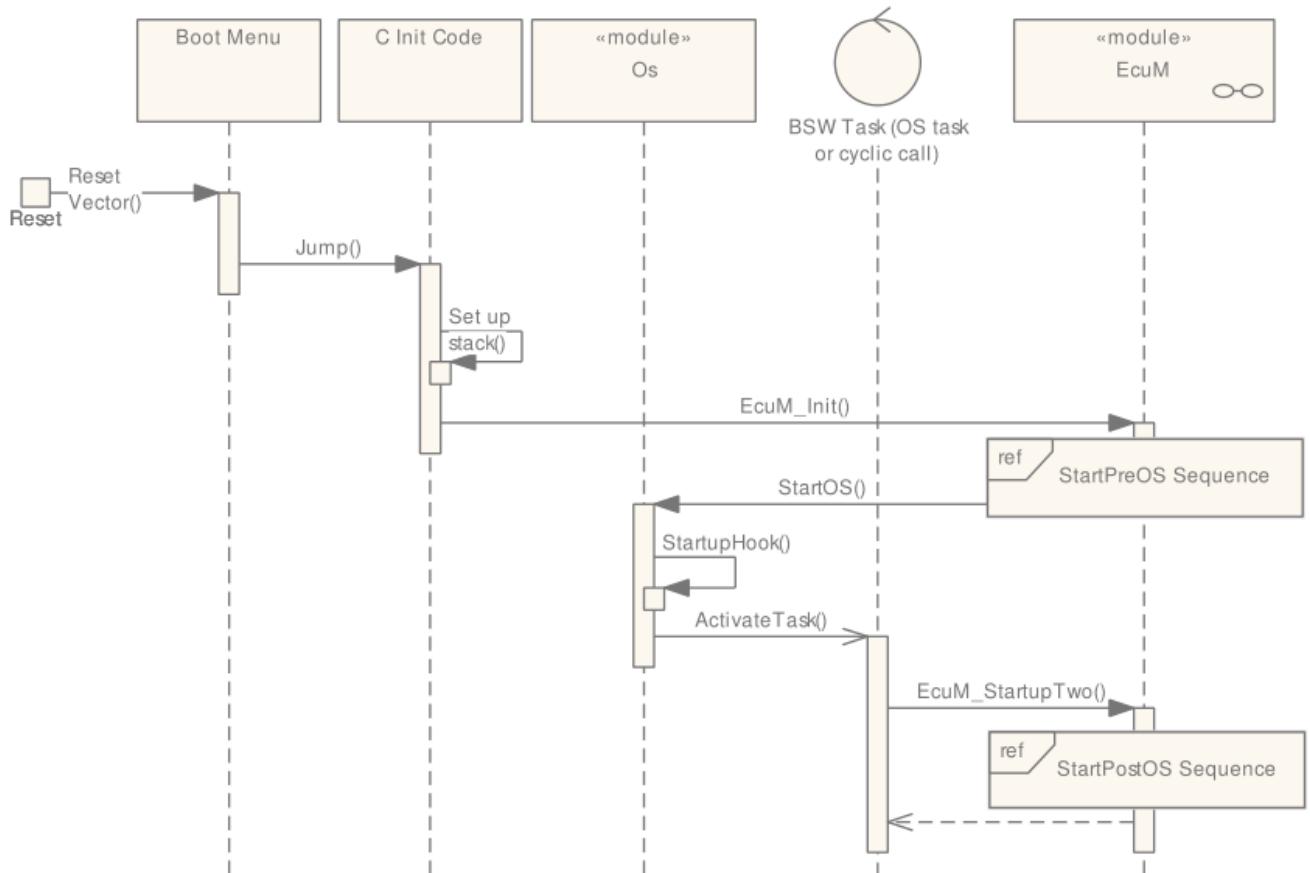


Figure 7.3: STARTUP Phase

## StartPreOS序列中的活动

表StartPreOS Sequence显示StartPreOS序列中的活动及其在EcuM\_Init中执行的顺序：

StartPreOS Sequence		
Initialization Activity	Comment	Opt.
Callout EcuM_AL_SetProgrammableInterrupts	在具有可编程中断优先级的ECU上，必须在启动操作系统之前设置这些优先级。	yes
Callout EcuM_AL_DriverInitZero	初始化块0此调用只能初始化不使用生成后配置参数的BSW模块。调用不仅可以包含驱动程序初始化，还可以包含任何类型的preOS、低级初始化代码	yes
Callout EcuM_DeterminePbConfiguration	此调用预期将返回一个指向完全初始化的EcuM_ConfigType结构的指针，该结构包含ECU管理器模块和所有其他BSW模块的构建后配置数据。	no
Check consistency of configuration data	如果检查失败，则调用EcuM_ErrorHook。	no
Call EcuMA DriverInitOne	调用不仅可以包含驱动程序初始化，还可以包含任何类型的预操作系统、低级初始化代码	yes
Get reset reason	重置原因源自对Mcu_GetResetReason的调用以及通过EcuM_WakeupSource配置容器定义的映射	no
Select default shutdown target	ECU Manager 模块应使用配置的默认关机目标（EcuMDefaultShutdownTarget）调用EcuM_GetValidatedWakeupEvents。	no
Call EcuM LoopDetection	如果启用了循环检测，则每次启动时都会调用此调用	yes
Start OS	启动AUTOSAR OS	no

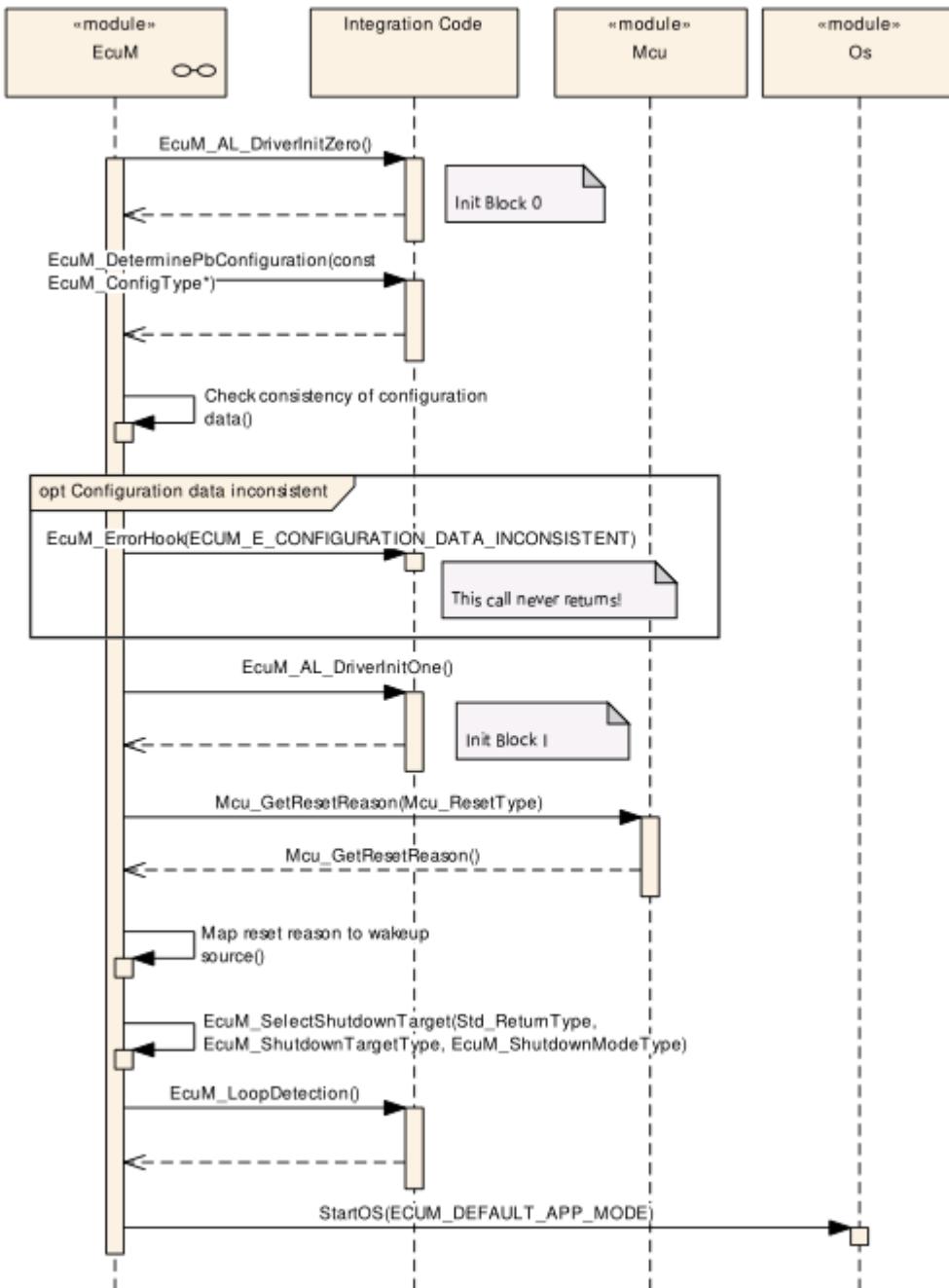
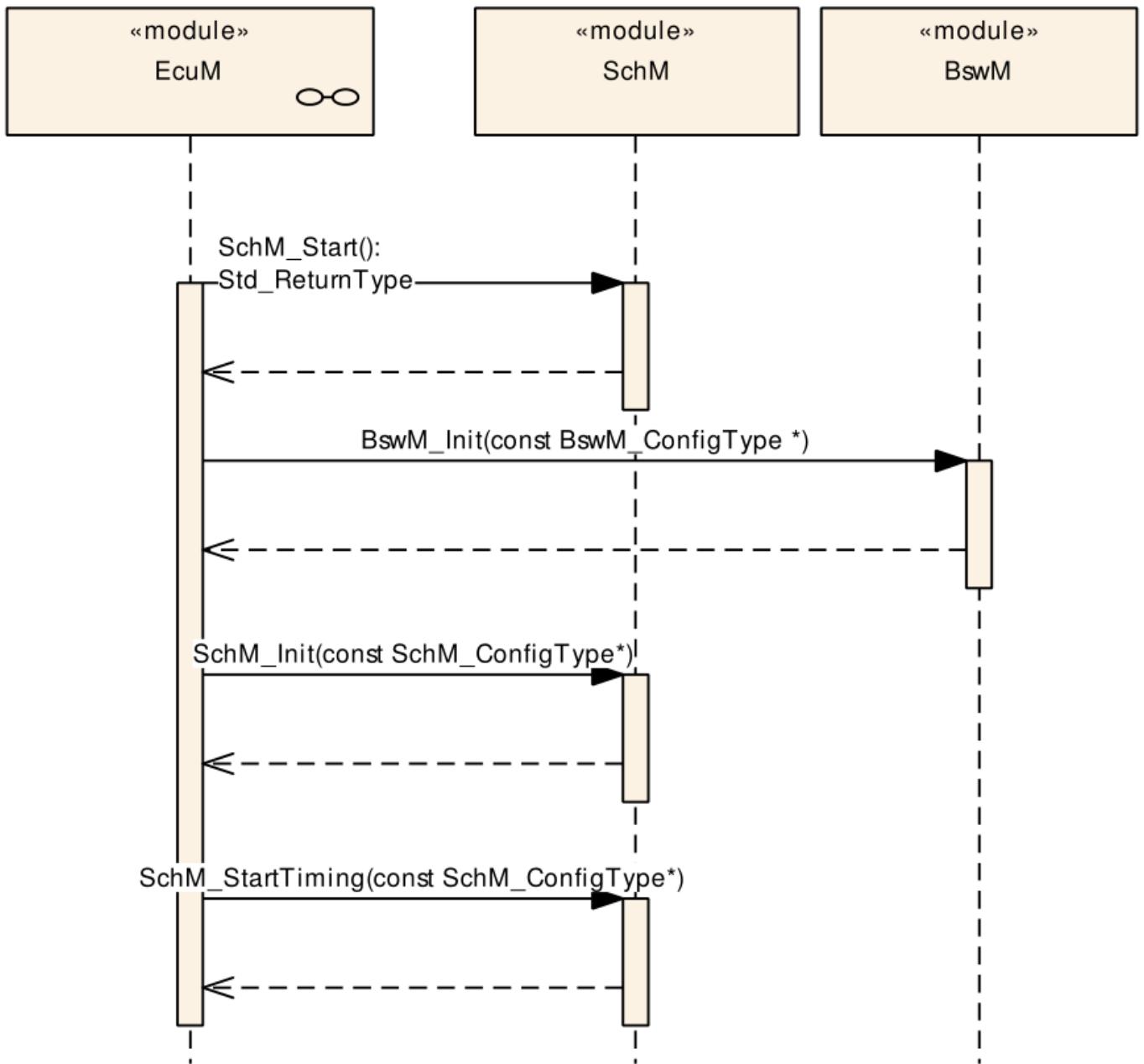


Figure 7.4: StartPreOS Sequence

StartPostOS序列中的活动

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

当通过EcuM\_StartupTwo 功能激活时，ECU管理器模块应执行StartPostOS序列中的操作：

**Figure 7.5: StartPostOS Sequence**

## SHUTDOWN 阶段

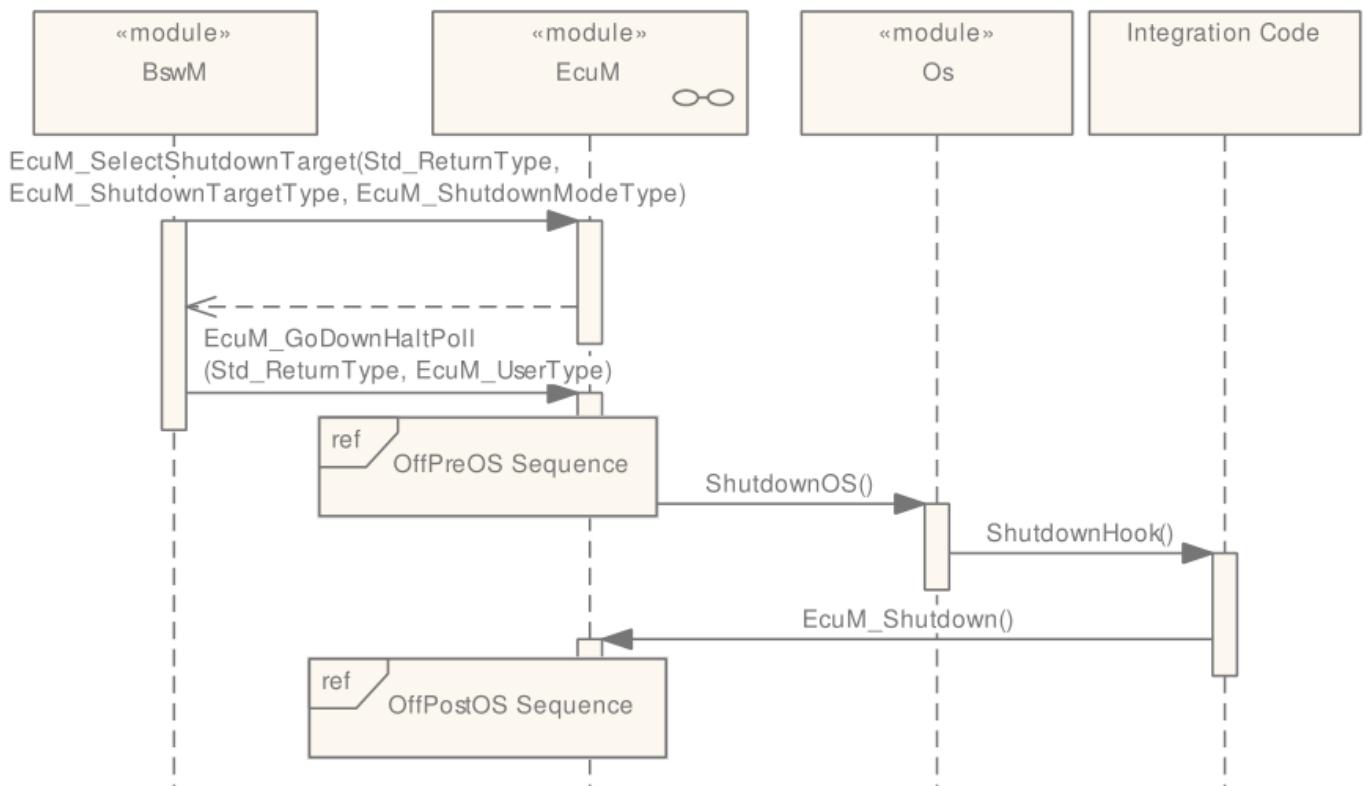
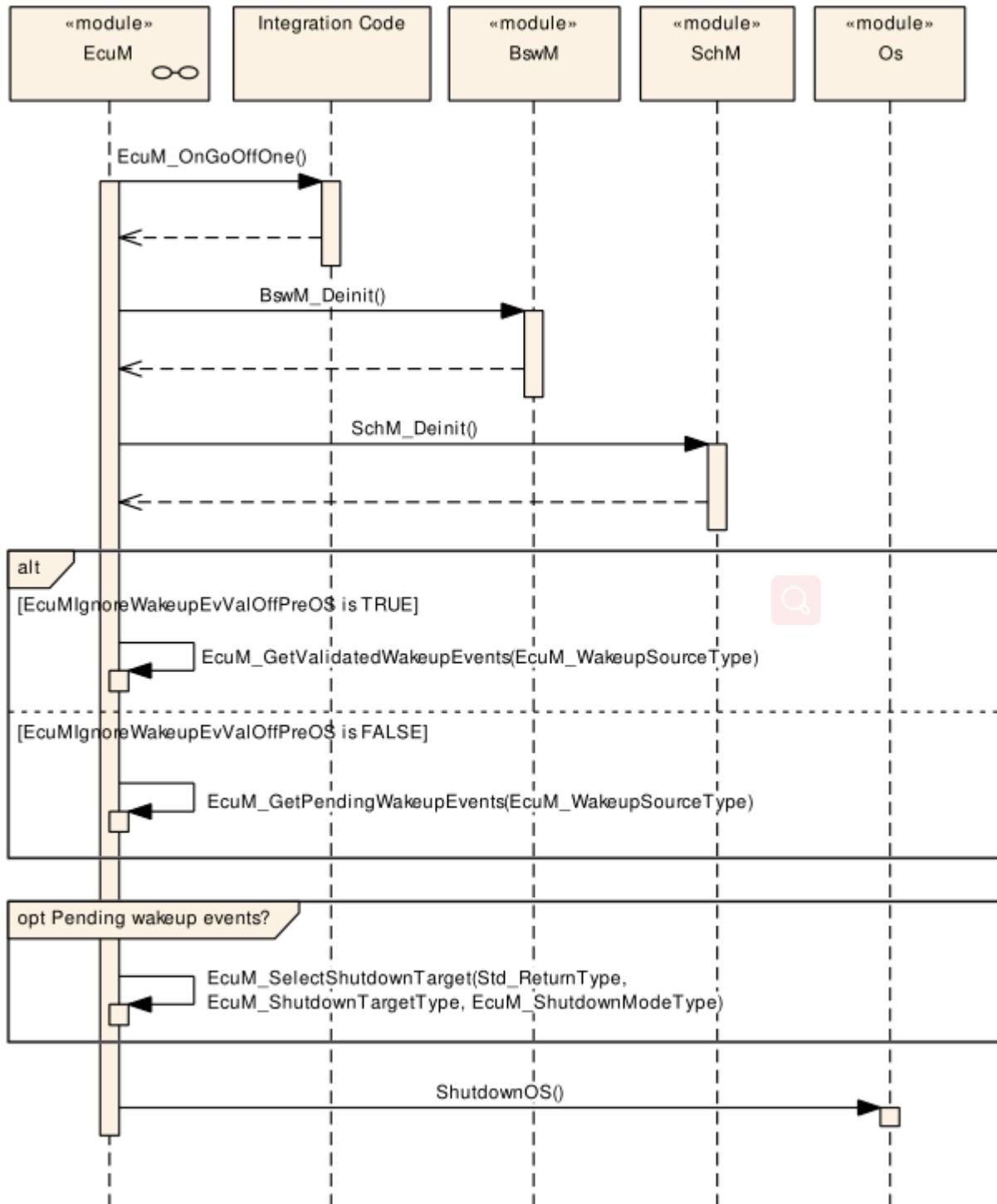


Figure 7.7: SHUTDOWN Phase

## OffPreOS序列中的活动

OffPreOS Sequence			
Shutdown Activity	Comment	Opt.	
De-init BSW Mode Manager		no	
De-init BSW Scheduler		no	
检查唤醒事件。所有挂起的唤醒事件或仅在关机期间验证的唤醒事件都将被考虑在关机期间，具体取决于 EcuMIgnoreWakeupEvValOffPreOS 的配置。	Purpose is to detect wakeup events that occurred during shutdown	no	
如果唤醒事件处于挂起状态，请将 RESET 设置为关机目标（将使用 EcuMDefaultResetModeRef 的默认复位模式）	仅当检测到挂起的唤醒事件以允许立即启动时，才应执行此操作	no	
ShutdownOS	Last operation in this OS task	no	

**Figure 7.8: OffPreOS Sequence**

OffPostOS序列中的活动

**OffPostOS Sequence**

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

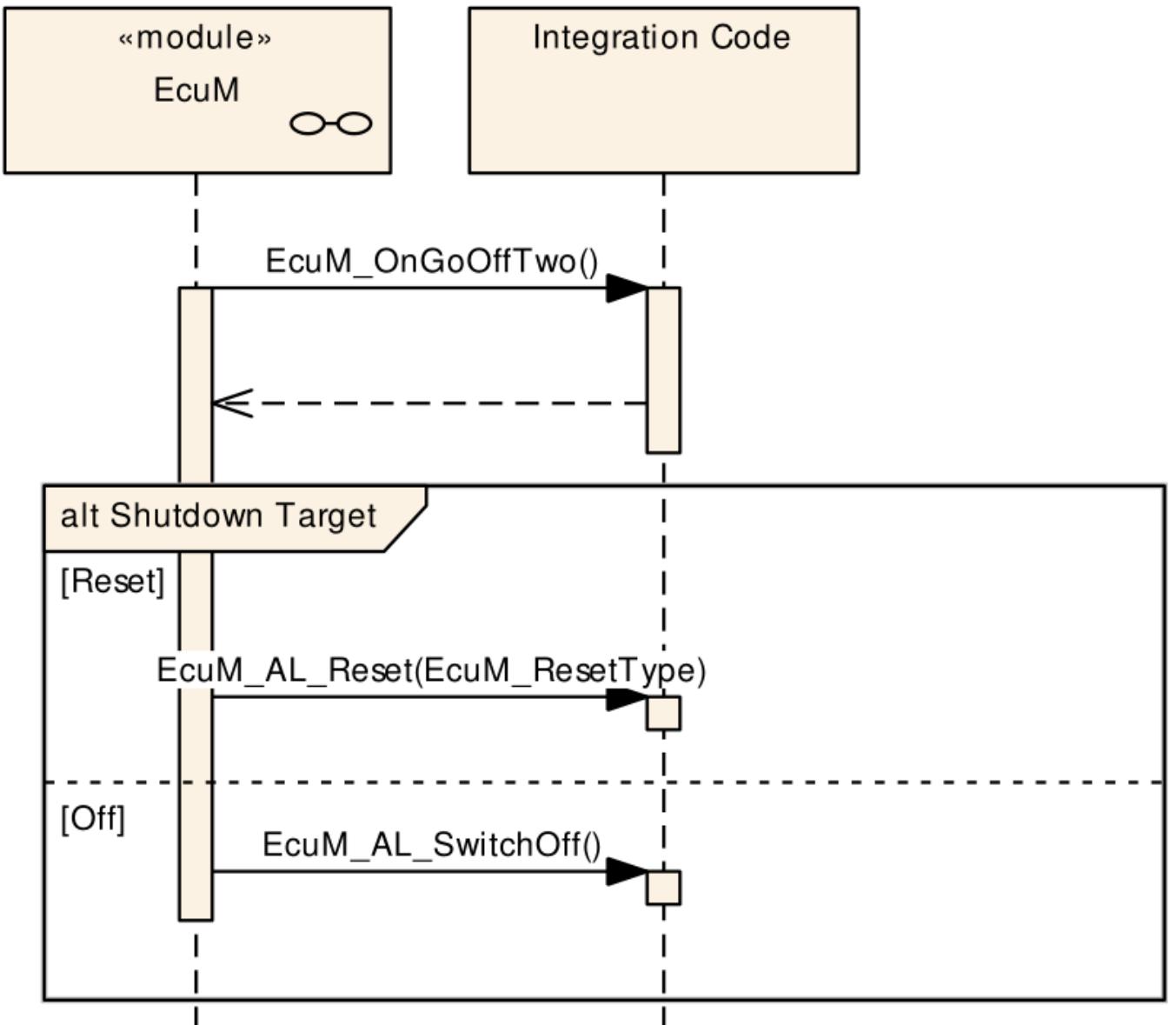


Figure 7.9: OffPostOS Sequence

## SLEEP 阶段

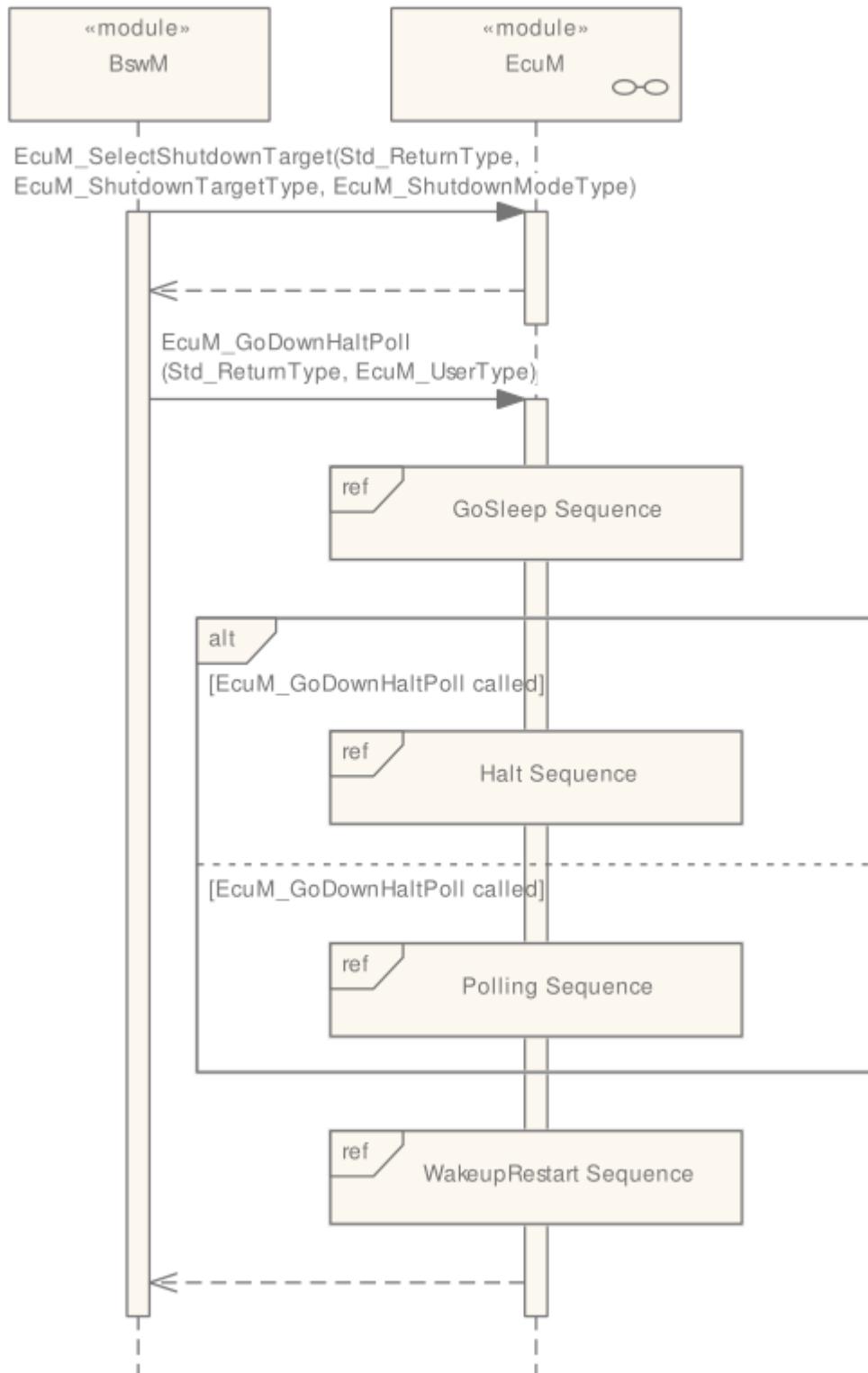


Figure 7.10: SLEEP Phase

GoSleep序列中的活动

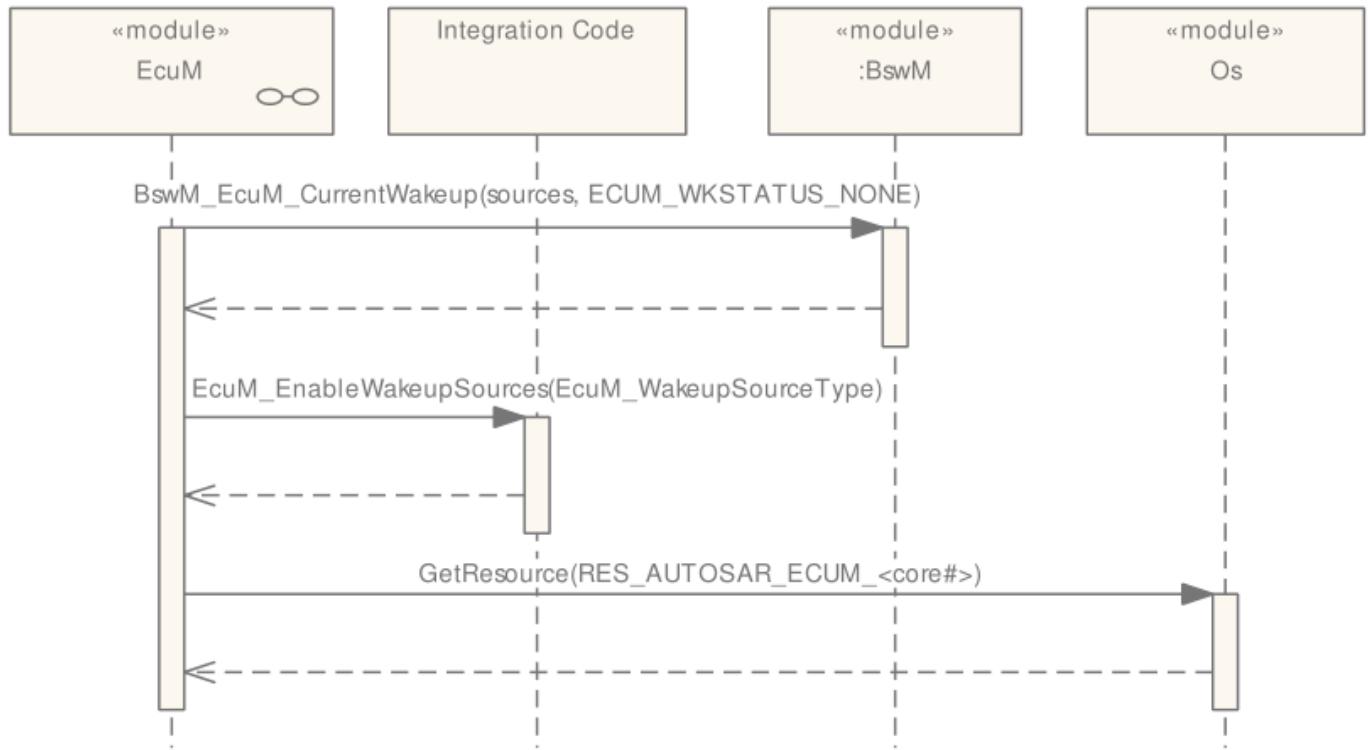


Figure 7.11: GoSleep Sequence

暂停序列中的活动

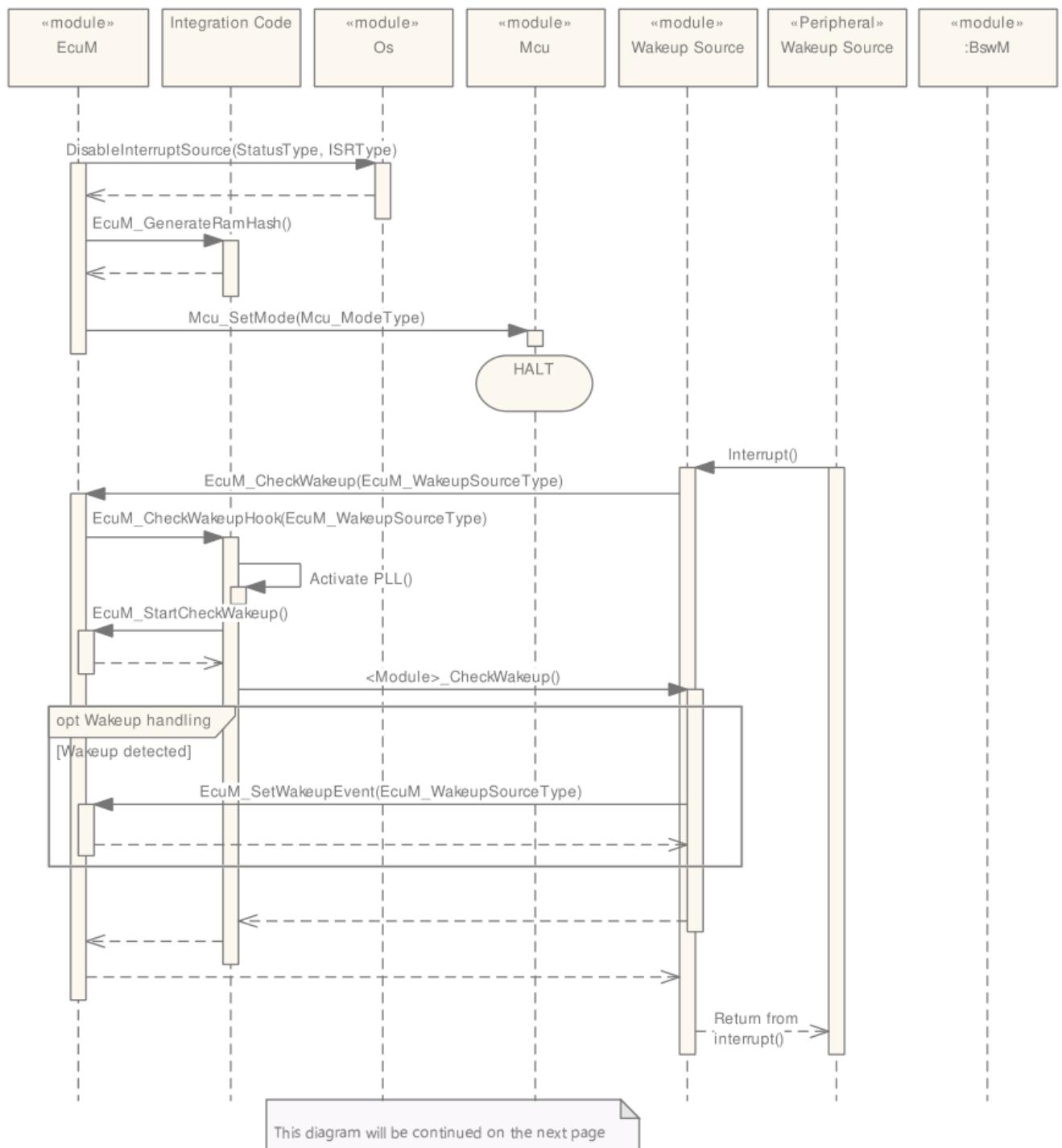


Figure 7.12: Halt Sequence

## 轮询序列中的活动

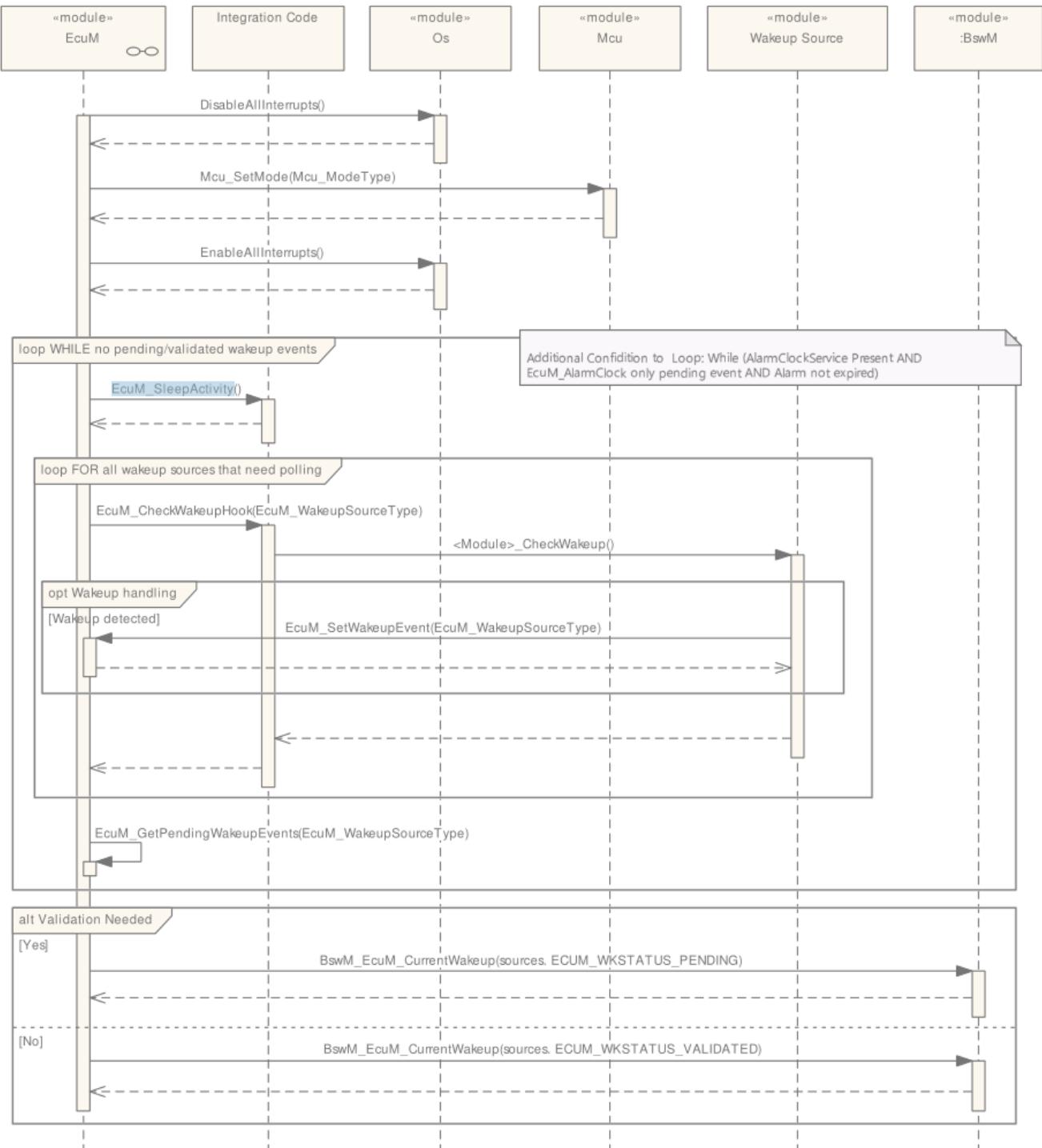


Figure 7.13: Poll Sequence

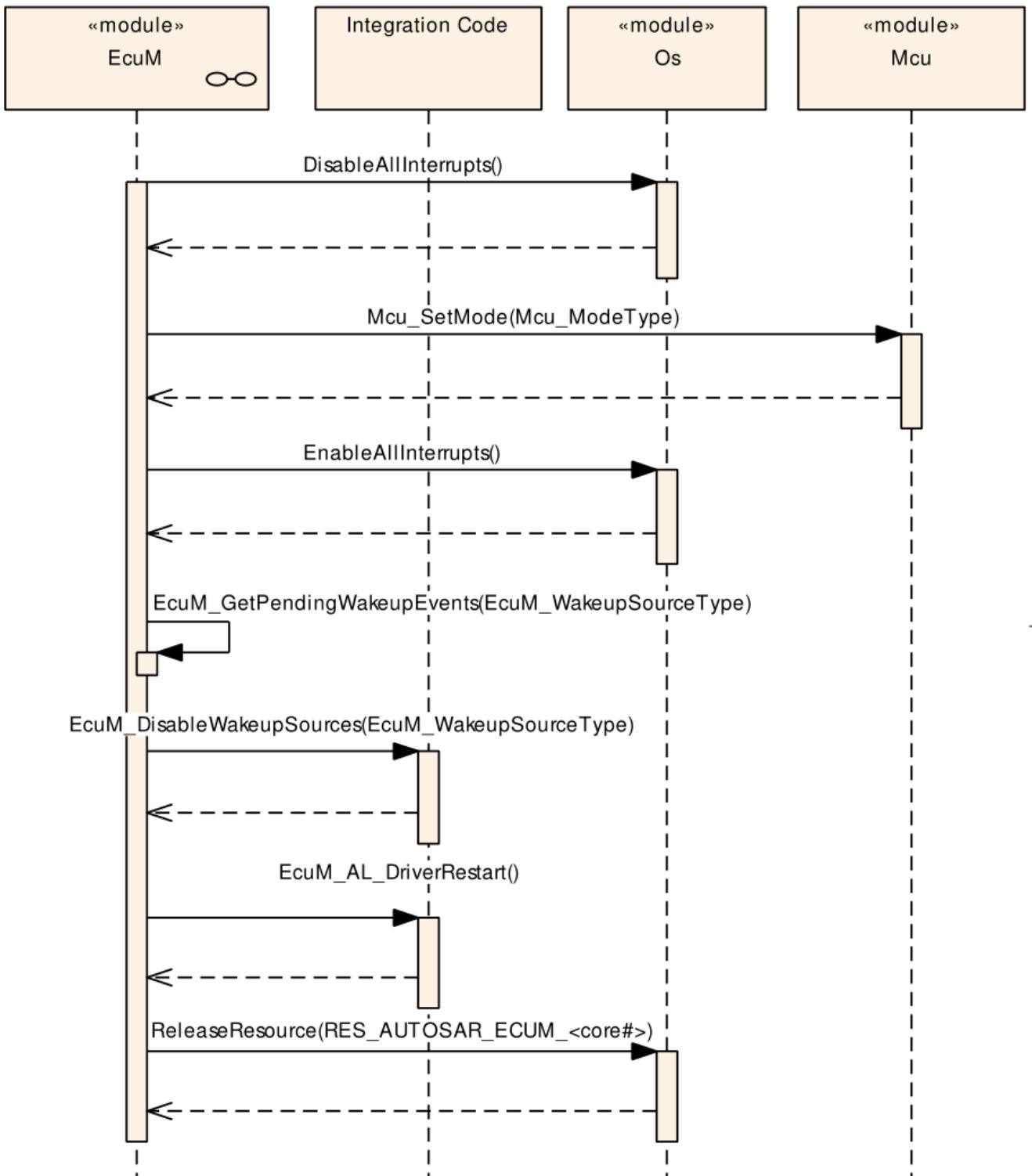
## Leaving Halt or Poll

- 如果在**ECU**处于停止或轮询状态时发生唤醒事件（例如切换唤醒线路，**CAN**总线上的通信等），则**ECU管理器模块**应通过执行唤醒重启序列重新获得控制权并退出睡眠阶段。
- 可以调用 **ISR** 来处理唤醒事件，但这取决于硬件和驱动程序实现。

如果在**ECU**处于停止或轮询状态时发生异常事件（硬件复位或电源循环），**ECU管理器模块**应在**STARTUP**阶段重新启动**ECU**。

## 唤醒重启序列中的活动

WakeupRestart*		
Wakeup Activity	Comment	Opt.
Restore MCU normal mode	选定的MCU模式在配置参数EcuMNormalMcuModeRef中配置	
Get the pending wakeup sources		
Callout EcuM_DisableWakeupSources	禁用当前挂起的唤醒源，但保留其他唤醒源，以便以后可以唤醒。	
Callout EcuM_AL_DriverRestart	初始化需要重新启动的驱动程序	
Unlock Scheduler	从此时开始，所有其他任务可能会再次运行。	

**Figure 7.14: WakeupRestart Sequence**

UP 阶段

在UP阶段，EcuM\_MainFunction定期执行，它有三个主要功能：

- 检查唤醒源是否已唤醒，并在必要时启动唤醒验证
- 更新闹钟计时器
- 仲裁RUN和POST\_RUN请求和释放。

### 唤醒源状态处理

唤醒源不仅在唤醒期间处理，而且与所有其他 EcuM 活动并行连续处理。此功能在EcuM\_MainFunction中运行，通过模式请求与ECU管理的其余部分完全分离。

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

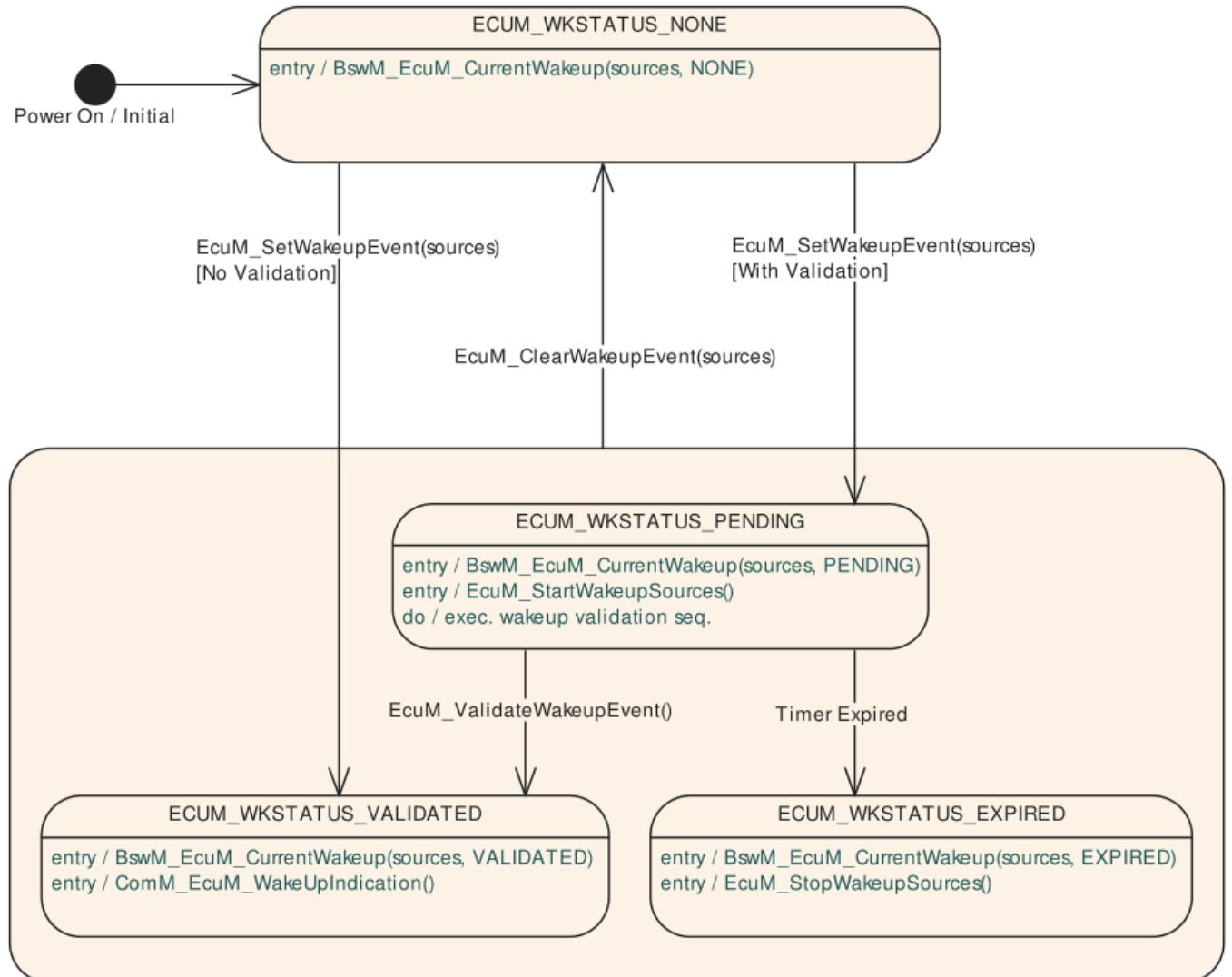


Figure 7.15: Wakeup Source States

当ECU管理器操作导致唤醒源的状态发生变化时，ECU管理器模块应向BswM发出模式请求，将唤醒源的模式更改为新的唤醒源状态。

当ECU管理器模块处于UP阶段时，唤醒事件通常不会触发状态更改。但是，它们会触发停止和轮询子阶段的结束。然后，ECU管理器模块自动执行唤醒重启序列，然后返回到 UP 阶段。

总结：每个挂起的事件都是独立验证的（如果已配置），EcuM 将结果作为模式请求发布到 BswM，这反过来可以触发 EcuM 中的状态更改。

唤醒验证序列中的活动

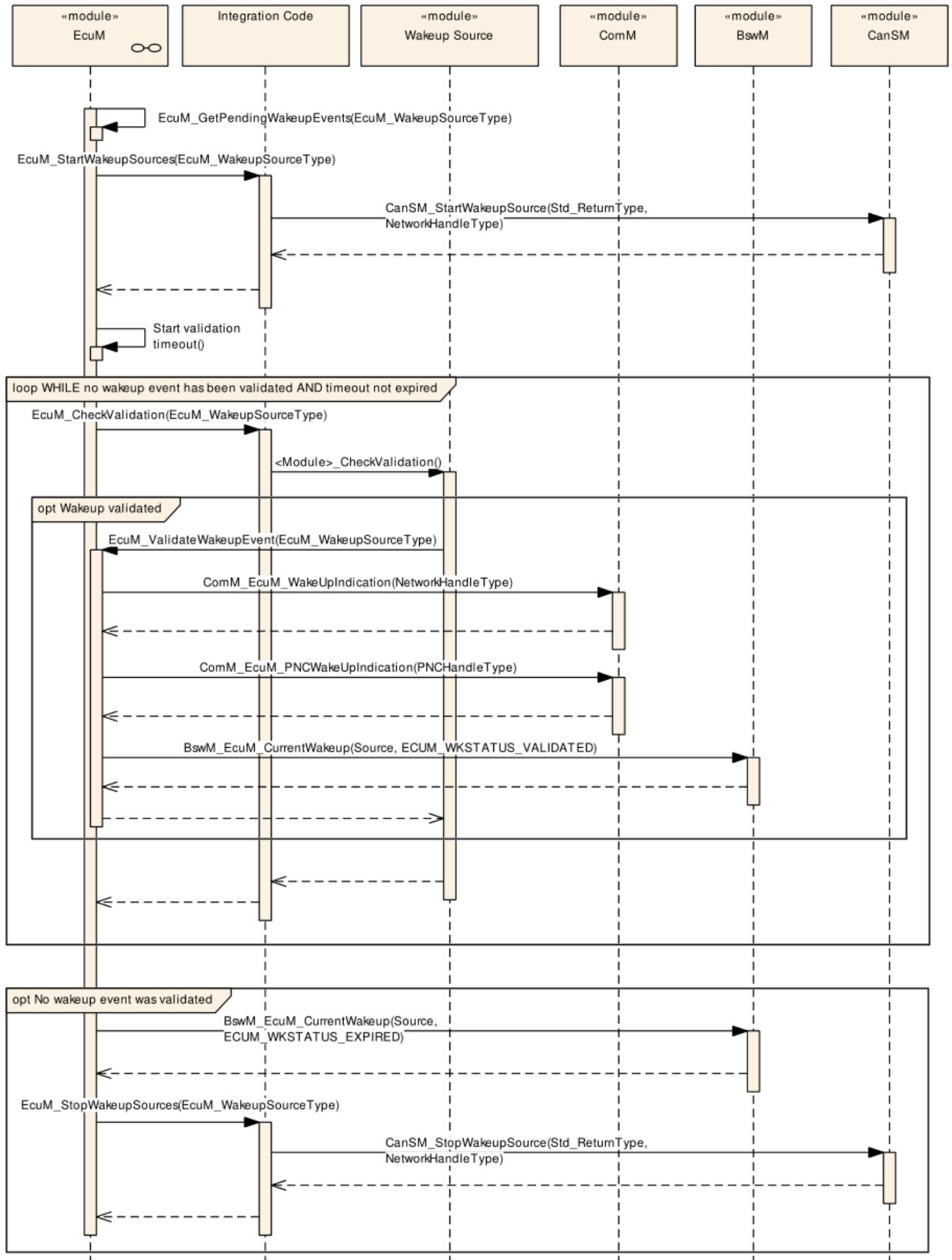


Figure 7.16: The WakeupValidation Sequence

## 通信信道唤醒

如果通信信道上发生唤醒，则相应的总线收发器驱动程序必须通过调用 `EcuM_SetWakeupEvent` 功能来通知ECU管理器模块。

## 唤醒源与ECU管理器的交互

当发生唤醒事件时，相应的驱动程序应通知ECU管理器模块唤醒。此通知最可能的方式是：

- 退出“停止”或“轮询”序列后。在这种情况下，ECU Manager 模块调用 `EcuM_AL_DriverRestart` 来重新初始化相关驱动程序，从而有机会扫描其硬件，例如挂起的唤醒中断。
- 如果唤醒源实际上处于睡眠模式，则驱动程序必须自主扫描唤醒事件；通过轮询或等待中断。

## 唤醒验证超时

ECU 管理器模块应提供单个唤醒验证超时计时器或每个唤醒源一个计时器。

## 具有唤醒源的驱动程序的要求

检测到唤醒事件时，驱动程序必须调用 `EcuM_SetWakeupEvent` 一次，并提供一个 `EcuM_WakeupSourceType` 参数，用于标识配置中指定的唤醒源。

ECU 管理器模块应检测在驱动程序初始化之前发生的唤醒，无论是从停止/轮询还是从关闭。

## SHUTDOWN TARGETS

“关闭标志”是一个描述性术语，用于描述未执行代码的所有状态 ECU。它们被称为关闭标志，因为它们是状态机在离开 UP 阶段时将驱动到的目标状态。以下状态是关闭标志：

- Off
- Sleep
- Reset

### Sleep

1. 在睡眠阶段不会错过任何唤醒事件。如果在进入睡眠序列中发生了唤醒事件，则不应输入停止或轮询序列。
2. ECU管理器模块可以定义一组可配置的睡眠模式（参见 `EcuMSleepMode`），其中每个模式本身都是一个关断目标。
3. ECU 管理器模块应允许将 MCU 休眠模式映射到 ECU 休眠模式，从而允许将它们作为关断目标寻址。
4. 关机目标睡眠应将所有内核置于睡眠状态。

### Reset

1. ECU 管理器模块应定义一组可配置的复位模式（参见 `EcuMResetMode` 和 `EcuM_ResetType`），其中每种模式本身都是一个关断目标。该集将最低限度地包含以下目标：
  - `Mcu_PerformReset`
  - `WdgM_PerformReset`
  - Toggle I/O Pin via DIO / SPI
2. ECU管理器模块应允许为复位目标定义别名
3. ECU 管理器模块应定义一组可配置的复位原因（请参阅 `EcuMSshutdownCause` 和 `EcuM_ShutdownCauseType`）。该集应至少包含以下目标：
  - ECU状态机进入停机状态
  - WdgM检测到故障
  - DCM请求关闭I
  - 重置时间。
4. ECU管理器模块应为BSW模块和SW-C提供设施
  - 记录停机原因
  - 获取一组最近的关机原因

**ALARM CLOCK**

ECU 管理器模块维护一个主闹钟，其值决定了 ECU 被唤醒的时间。此外，ECU管理器管理一个内部时钟，即EcuM时钟，用于与主报警进行比较。

请注意，警报唤醒机制仅与 SLEEP 阶段相关。SWC 和 BSW 模块可以在 UP 阶段（并且仅在 UP 阶段）设置和检索报警值，但是，这将在 SLEEP 阶段得到遵守。

与可以使用通用ECU管理器模块工具实现的其他定时/唤醒机制相比，闹钟服务在计时器到期之前不会启动唤醒重启序列。当ECU模块检测到其定时器引起唤醒事件时，它会增加其定时器并立即返回睡眠状态，除非时钟时间超过闹钟时间。

1. 当闹钟服务存在时（参见 `EcuMAlarmClockPresent`），EcuM 管理器模块应维护一个 EcuM 时钟，其时间应为自电池连接以来的时间（以秒为单位）。
2. EcuM时钟应跟踪UP和SLEEP阶段的时间。
3. 硬件允许时，ECU重置不应重置EcuM时钟时间。
4. 应有一个且仅一个唤醒源分配给 EcuM 时钟（请参阅 `EcuMAlarmWakeupSource`）。

**Alarm Clocks and Users**

SW-C和BSW模块可以各自维护一个闹钟（用户闹钟）。每个用户闹钟（参见`EcuMAlarmClock`）都与一个`EcuMAlarmClockUser`相关联，该用户标识相应的SW-C或BSW模块。

1. 每个EcuM用户最多应有一个用户闹钟。
2. EcuM用户不得设置其他用户闹钟的值。
3. ECU 管理器模块应始终将主闹钟值设置为最早的用户闹钟值的值。
4. 只有授权的 EcuM 用户才能设置 EcuM 时钟时间（请参阅 `EcuMSetClockAllowedUsers`）。

**EcuM Clock Time**

如果底层硬件机制是基于滴答的，ECUM应相应地“纠正”时间。

**EcuM Clock Time in the UP Phase**

`EcuM_MainFunction`在UP阶段递增EcuM时钟。它使用标准操作系统机制（警报/计数器）来推导其时间。请注意计数器与 EcuM 时间之间的粒度差异，以秒为单位

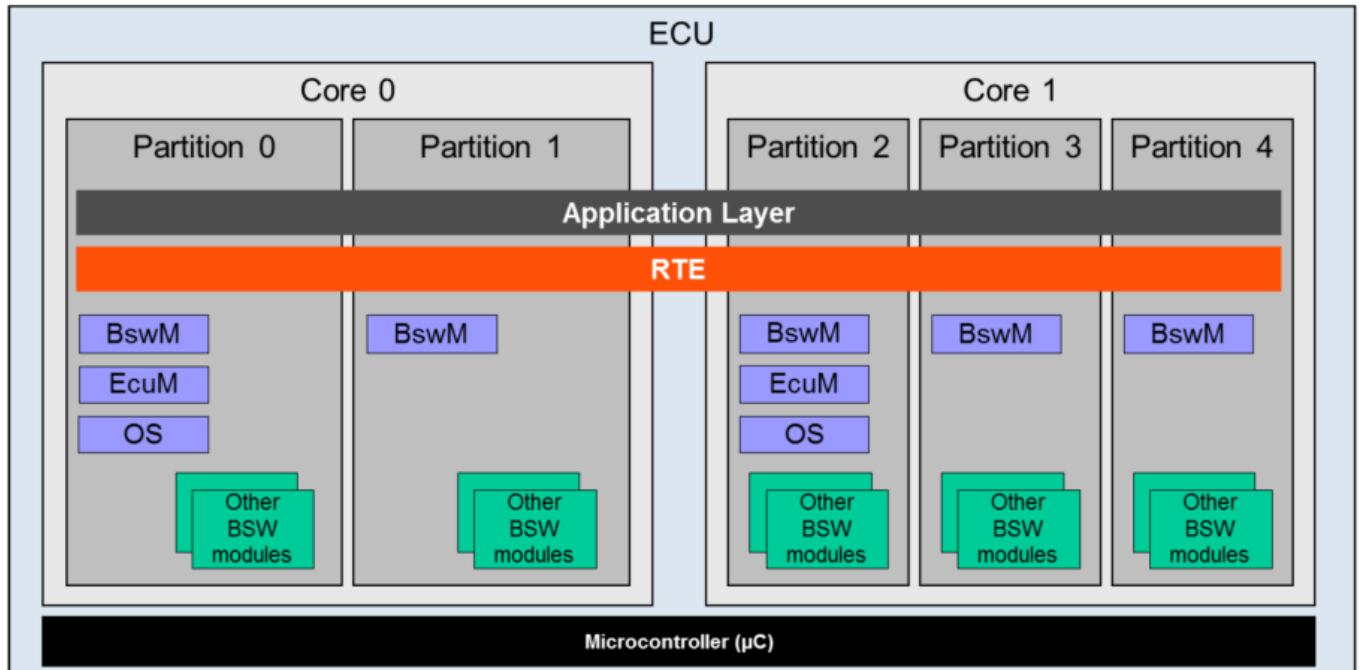
**EcuM Clock Time in the Sleep Phase**

有两种替代方法可以在睡眠期间增加 EcuM 时钟，具体取决于所选的睡眠模式（`EcuMSleepModeSuspend` 参数）在停止序列中（请参阅 7.5.2 停止序列中的活动），必须将 GPT 驱动程序放入`GPT_MODE_SLEEP`，以仅配置时基所需的计时器通道。它还要求 GPT 使用 `Gpt_EnableWakeup` API 启用基于计时器的唤醒通道。最好将`Gpt_StartTimer` API 设置为 1 秒，但如果无法达到此值，则需要更频繁地唤醒 EcuM 以累积多个计时器唤醒，直到累积 1 秒以增加时钟值。

1. 当离开睡眠状态时，ECU管理器模块将中止任何活动用户闹钟和主闹钟。这意味着时钟感应和其他事件引起的唤醒都将导致清除所有警报。
2. 用户警报和主警报应在启动前操作系统序列、唤醒重启序列和关闭前操作系统序列中取消

**MULTICORE**

BSW 模块可以分布在不同的分区上，因此可以分布在不同的内核上。一些BSW模块作为BswM必须包含在每个分区中。操作系统或 EcuM 等其他模块已包含在每个内核的一个分区中。



**Figure 7.17: Partitions inside an ECU**

在多核架构中，EcuM必须以每核一个实例的方式分布。

有一个指定的主核心，引导加载程序通过EcuM\_Init启动主 Ecu M。主 EcuM 启动一些驱动程序，确定构建后配置，并使用所有附属 EcuM 启动所有剩余内核。

现在，每个 EcuM 启动核心本地操作系统和所有核心本地 BswM（每个分区中正好驻留一个 BswM）。

#### Master Core

有一个明确的主核心。主内核是哪个内核，由引导加载器确定。主核心的EcuM作为第一个BSW模块启动并执行初始化操作。

然后是使用所有其他 EcuM 启动所有其他内核。

当这些启动时，它与每个卫星EcuM一起初始化核心本地操作系统和BswM。

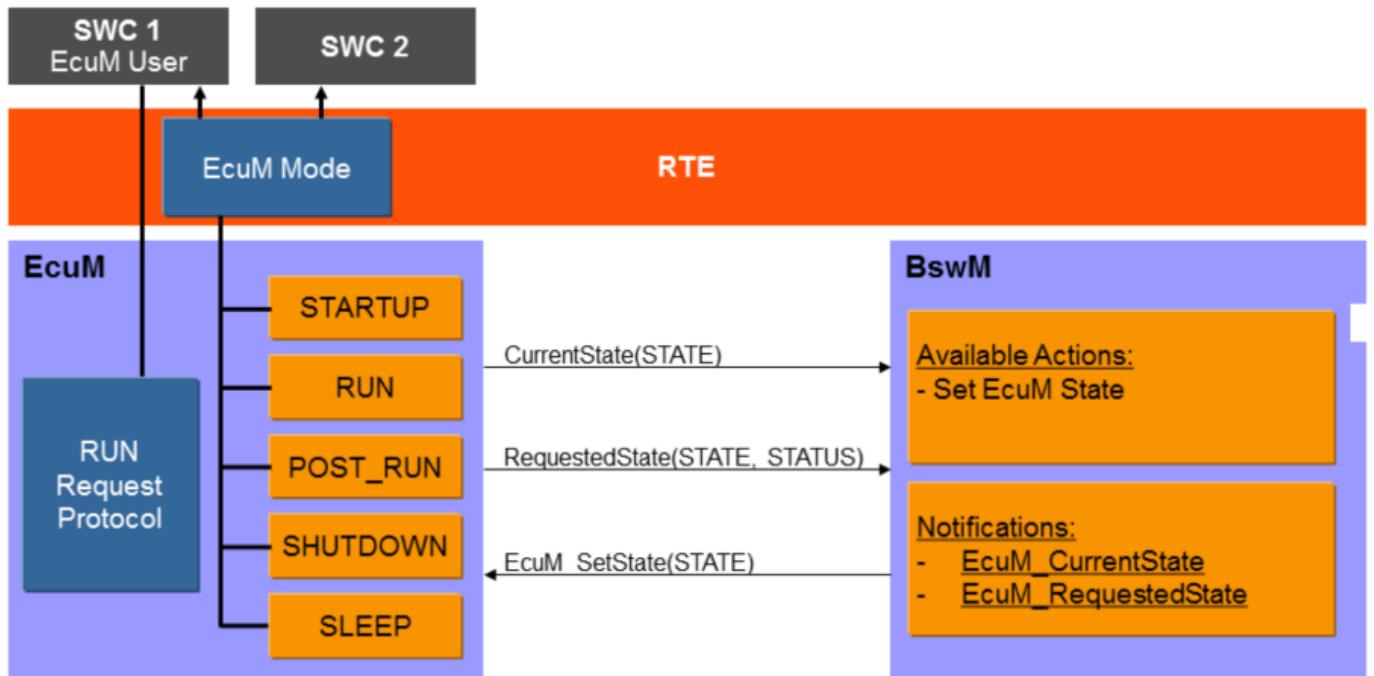
#### Slave Core

在每个从属核心上，必须运行一颗卫星EcuM。如果一个核心包含多个分区，则每个核心只能存在于 EcuM 上。

#### ECUM模式处理

ECU 状态管理器为 SW-C 提供接口，以可选地请求和释放模式 RUN 和 POST\_RUN。

EcuMFlex 仲裁 SW-C 发出的请求和发布，并将结果传播到 BswM。EcuM 和 BswM 之间的合作是必要的，因为只有 BswM 才能决定何时可以转换到不同的模式。由于 EcuM 没有自己的状态机，因此 EcuM 依赖于 BswM 进行的状态转换。因此 EcuM 不请求状态。此外，它通知 BswM 所有请求的当前仲裁。并且当RTE已经执行完属于某个模式的所有Runnable时通知BswM。



**Figure 7.35: Architectural Components of ECU Mode Handling**

SWS\_EcUM\_04115: 当 EcuModeHandling 配置为 true 时, 应应用 ECU 模式处理。

SWS\_EcUM\_04116: 当BswM通过 `EcuM_SetState` 设置EcUM的状态时, EcUM应向RTE指示相应的模式。

SWS\_EcUM\_04117:当最后一个 RUN 请求被释放时, ECU 状态管理器模块应使用 API `BswM_EcuM_RequestedState(ECUM_STATE_RUN, ECUM_RUNSTATUS_RELEASED)` 将此指示给 BswM。

如果 SW-C 在 POST\_RUN 期间需要运行后活动（例如关机准备）, 那么它必须在释放 RUN 请求之前请求 POST\_RUN。否则无法保证此 SW-C 将有机会运行其 POST\_RUN 代码。

SWS\_EcUM\_04118: 当ECU状态管理器未处于SWC请求的状态时, 应使用 `BswM_EcuM_RequestedState` API通知BswM请求的状态。

POST\_RUN 状态为 SW-C 提供后运行阶段, 并允许它们保存重要数据或关闭外围设备。

SWS\_EcUM\_04144: 当收到第一个 RUN 或 POST\_RUN 请求时, ECU 状态管理器模块应使用 `BswM_EcuM_RequestedState(ECUM_STATE_RUN, ECUM_RUNSTATUS_REQUESTED)` 将此指示给 BswM。

SWS\_EcUM\_04119: 当最后一个 POST\_RUN 请求被释放时, ECU 状态管理器模块应使用 API `BswM_EcuM_RequestedState(ECUM_STATE_POST_RUN, ECUM_RUNSTATUS_RELEASED)` 将此指示给 BswM。

#### Hint

为了防止ECU模式的模式机实例滞后以及状态EcUM和RTE异相, EcUM可以使用确认反馈来通知模式切换。

**Note**

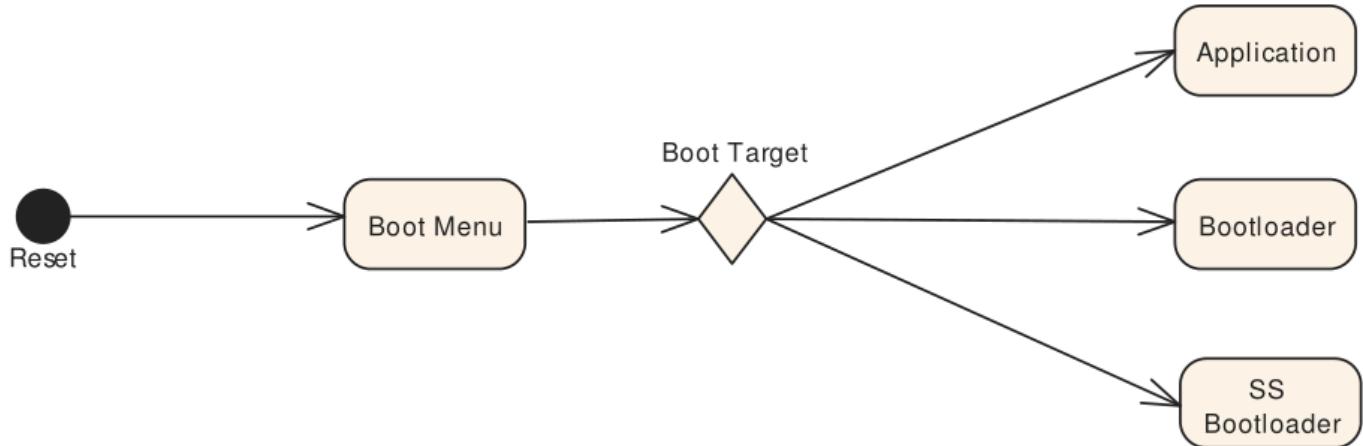
EcuM只请求RUN和POST\_RUN之间的模式，睡眠模式必须由BswM设置，因为EcuM没有关于何时可以进入该模式的信息。

State	Description
STARTUP	初始值。当调用 Rte_Start() 时由 Rte 设置。
RUN	一旦所有必要的BSW模块初始化，BswM切换到此模式。
POST_RUN	当没有RUN请求可用时，EcuM请求POST_RUN。
SLEEP	当没有RUN和POST_RUN请求可用且关机目标设置为SLEEP时，EcuM请求SLEEP模式
SHUTDOWN	当没有 RUN 和 POST_RUN 请求可用且关闭目标设置为 SHUTDOWN 时，EcuM 请求关闭模式。

SWS\_EcuM\_04143: EcuM 应通过调用接口 `BswM_EcuM_CurrentState(EcuM_StateType State)` 通知 BswM 当前状态。当 RTE 通过确认端口给出反馈时，EcuM 应设置一个新状态。

**ADVANCED TOPICS****与Bootloader的关系**

Bootloader不是AUTOSAR的一部分。然而，应用程序需要一个接口来激活引导加载程序。为此，提供了两个函数：`EcuM_SelectBootTarget` 和 `EcuM_GetBootTarget`。



**Figure 7.36: Selection of Boot Targets**

Bootloader、系统供应商的bootloader和应用程序是独立的程序镜像，在很多情况下甚至可以单独刷写。从一个图像到另一个图像的唯一方法是通过重置。启动菜单将根据所选的启动目标分支到一个或另一个映像。

**Relation to Complex Drivers**

如果复杂的驱动程序处理唤醒源，它必须遵循本文档中指定的处理唤醒事件的协议。

**启动和关闭期间的错误处理**

SWS\_EcuM\_02980: ECU管理器模块应忽略初始化期间发生的所有类型的错误，例如init函数返回的值

初始化是一个配置问题（请参阅 `EcuMDriverInitListZero`、`EcuMDiverInitListOne` 和 `EcuMDdriverRestartList`），因此无法标准化。

BSW模块负责按照SWS的规定，直接向DEM模块或DET模块报告初始化过程中发生的错误。ECU管理器模块不报告错误。BSW模块还负责采取任何特殊措施，以应对初始化过程中发生的错误。

## 2.1.3 API

---

### API specification

### Type definitions

EcuM\_ConfigType

<b>Name</b>	EcuM_ConfigType				
<b>Kind</b>	Structure				
<b>Elements</b>	-				
	<b>Type</b>	—			
	<b>Comment</b>	The content of this structure depends on the post-build configuration of EcuM.			
<b>Description</b>	A pointer to such a structure shall be provided to the ECU State Manager initialization routine for configuration.				
<b>Available via</b>	EcuM.h				

EcuM\_RunStatusType

<b>Name</b>	EcuM_RunStatusType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Range</b>	ECUM_RUNSTATUS_UNKNOWN	0	Unknown status. Init Value.
	ECUM_RUNSTATUS_REQUESTED	1	Status requested from EcuM
	ECUM_RUNSTATUS_RELEASED	2	Status released from EcuM.
<b>Description</b>	Result of the Run Request Protocol sent to BswM		
<b>Available via</b>	EcuM.h		

## Ecum\_WakeupSourceType

<b>Name</b>	Ecum_WakeupSourceType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint32		
<b>Range</b>	ECUM_WKSOURCE_POWER	0x01	Power cycle (bit 0)
	ECUM_WKSOURCE_RESET(default)	0x02	Hardware reset (bit 1)  If the Mcu driver cannot distinguish between a power cycle and a reset reason, then this shall be the default wakeup source
	ECUM_WKSOURCE_INTERNAL_RESET	0x04	internal reset of μC (bit 2) The internal reset typically only resets the uC core but not peripherals or memory controllers The exact behavior is hardware specific. This source may also indicate an unhandled exception
	ECUM_WKSOURCE_INTERNAL_WDG	0x08	Reset by internal watchdog (bit 3)
	ECUM_WKSOURCE_EXTERNAL_WDG	0x10	Reset by external watchdog (bit 4), if detection supported by hardware
<b>Description</b>	Ecum_WakeupSourceType 定义了一个具有 5 个预定义位置的位域（请参阅范围）。位域为每个唤醒源提供一个位。  在 WAKEUP 中，清除的所有位都表示没有已知的唤醒源。  在 STARTUP 中，所有清除的位都表示不知道重新启动或重置的原因。在这种情况下，应假定 ECUM_WKSOURCE_RESET。		
<b>Available via</b>	Ecum.h		

## EcuM\_WakeupStatusType

<b>Name</b>	EcuM_WakeupStatusType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Range</b>	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
<b>Description</b>	The type describes the possible states of a wakeup source.		
<b>Available via</b>	EcuM.h		

## EcuM\_ResetType

<b>Name</b>	EcuM_ResetType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Range</b>	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.
<b>Description</b>	This type describes the reset mechanisms supported by the ECU State Manager. It can be extended by configuration.		
<b>Available via</b>	EcuM.h		

## EcuM\_StateType

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

## FUNCTION DEFINITIONS

EcuM\_GetVersionInfo

<b>Service Name</b>	EcuM_GetVersionInfo	
<b>Syntax</b>	<pre>void EcuM_GetVersionInfo (     Std_VersionInfoType* versioninfo )</pre>	
<b>Service ID [hex]</b>	0x00	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	versioninfo	Pointer to where to store the version information of this module.
<b>Return value</b>	None	
<b>Description</b>	Returns the version information of this module.	
<b>Available via</b>	EcuM.h	

EcuM\_GoDownHaltPoll

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

## EcuM\_Init

<b>Service Name</b>	EcuM_Init
<b>Syntax</b>	<pre>void EcuM_Init (     void )</pre>
<b>Service ID [hex]</b>	0x01
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	Initializes the ECU state manager and carries out the startup procedure. The function will never return (it calls StartOS)
<b>Available via</b>	EcuM.h

## EcuM\_StartupTwo

<b>Service Name</b>	EcuM_StartupTwo
<b>Syntax</b>	<pre>void EcuM_StartupTwo (     void )</pre>
<b>Service ID [hex]</b>	0x1a
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	This function implements the STARTUP II state.
<b>Available via</b>	EcuM.h

## EcuM\_Shutdown

<b>Service Name</b>	EcuM_Shutdown
<b>Syntax</b>	<pre>void EcuM_Shutdown (     void )</pre>
<b>Service ID [hex]</b>	0x02
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	Typically called from the shutdown hook, this function takes over execution control and will carry out GO OFF II activities.
<b>Available via</b>	EcuM.h

**EcuM\_SetState**

Service Name	EcuM_SetState	
Syntax	void EcuM_SetState( EcuM_StateType state )	
Service ID [hex]	0x2b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	state	State indicated by BswM.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	BswM调用的通知状态切换的函数。	
Available via	EcuM.h	

**EcuM\_RequestRUN**

Service Name	EcuM_RequestRUN	
<b>Syntax</b>	Std_ReturnType EcuM_RequestRUN ( EcuM_UserType user )	
<b>Service ID [hex]</b>	0x03	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant for different users	
<b>Parameters (in)</b>	user	ID of the entity requesting the RUN state.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: The request was accepted by EcuM. E_NOT_OK: The request was not accepted by EcuM
<b>Description</b>	Places a request for the RUN state. Requests can be placed by every user made known to the state manager at configuration time.	
<b>Available via</b>	EcuM.h	

## EcuM\_ReleaseRUN

<b>Service Name</b>	EcuM_ReleaseRUN	
<b>Syntax</b>	Std_ReturnType EcuM_ReleaseRUN ( EcuM_UserType user )	
<b>Service ID [hex]</b>	0x04	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	user	ID of the entity releasing the RUN state.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: The release request was accepted by EcuM E_NOT_OK: The release request was not accepted by EcuM
<b>Description</b>	Releases a RUN request previously done with a call to EcuM_RequestRUN. The service is intended for implementing AUTOSAR ports.	
<b>Available via</b>	EcuM.h	

## EcuM\_RequestPOST\_RUN

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

## EcuM\_ReleasePOST\_RUN

<b>Service Name</b>	EcuM_ReleasePOST_RUN	
<b>Syntax</b>	Std_ReturnType EcuM_ReleasePOST_RUN ( EcuM_UserType user )	
<b>Service ID [hex]</b>	0x0b	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	user	ID of the entity releasing the POST RUN state.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: The release request was accepted by EcuM E_NOT_OK: The release request was not accepted by EcuM
<b>Description</b>	Releases a POST RUN request previously done with a call to EcuM_RequestPOST_RUN. The service is intended for implementing AUTOSAR ports.	
<b>Available via</b>	EcuM.h	

## EcuM\_SelectShutdownTarget

<b>Service Name</b>	EcuM_SelectShutdownTarget	
<b>Syntax</b>	Std_ReturnType EcuM_SelectShutdownTarget ( EcuM_ShutdownTargetType shutdownTarget, EcuM_ShutdownModeType shutdownMode )	
<b>Service ID [hex]</b>	0x06	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	shutdownTarget	The selected shutdown target.
	shutdownMode	The identifier of a sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or a reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) as defined by configuration.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: The new shutdown target was set E_NOT_OK: The new shutdown target was not set
<b>Description</b>	EcuM_SelectShutdownTarget selects the shutdown target. EcuM_SelectShutdownTarget is part of the ECU Manager Module port interface.	
<b>Available via</b>	EcuM.h	

## EcuM\_GetShutdownTarget

<b>Service Name</b>	EcuM_GetShutdownTarget	
<b>Syntax</b>	<pre>Std_ReturnType EcuM_GetShutdownTarget (   EcuM_ShutdownTargetType* shutdownTarget,   EcuM_ShutdownModeType* shutdownMode )</pre>	
<b>Service ID [hex]</b>	0x09	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	shutdownTarget	One of these values is returned: ECUM_SHUTDOWN_TARGET_SLEEP ECUM_SHUTDOWN_TARGET_RESET ECUM_SHUTDOWN_TARGET_OFF
	shutdownMode	If the out parameter "shutdownTarget" is ECUM_SHUTDOWN_TARGET_SLEEP, sleepMode tells which of the configured sleep modes was actually chosen. If "shutdownTarget" is ECUM_SHUTDOWN_TARGET_RESET, sleepMode tells which of the configured reset modes was actually chosen.
<b>Return value</b>	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service has failed, e.g. due to NULL pointer being passed
<b>Description</b>	EcuM_GetShutdownTarget returns the currently selected shutdown target as set by EcuM_SelectShutdownTarget. EcuM_GetShutdownTarget is part of the ECU Manager Module port interface.	
<b>Available via</b>	EcuM.h	

## EcuM\_GetLastShutdownTarget

<b>Service Name</b>	<b>EcuM_GetLastShutdownTarget</b>	
<b>Syntax</b>	<pre>Std_ReturnType EcuM_GetLastShutdownTarget (     Ecu_shutdownTargetType     shutdownTarget     Ecu_shutdownModeType      shutdownMode )</pre>	
<b>Service ID</b>	0x08	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters</b>	None	
<b>Parameters (out)</b>	shutdownTarget	返回以下值之一：ECUM_SHUTDOWN_TARGET_SLEEP ECUM_SHUTDOWN_TARGET_RESET ECUM_SHUTDOWN_TARGET_OFF
	shutdownMode	如果输出参数“shutdownTarget”为 ECUM_SHUTDOWN_TARGET_SLEEP，则sleepMode告诉实际 选择了哪些配置的休眠模式。如果“shutdownTarget” 是ECUM_SHUTDOWN_TARGET_RESET，则sleepMode告诉实 际选择了哪些配置的重置模式
<b>Return value</b>	Std_ReturnType	E_OK: 服务已成功 E_NOT_OK: 服务已失败，例如，由于 传递了NULL指针
<b>Description</b>	EcuM_GetLastShutdownTarget返回上一个关闭进程的关闭目标。  EcuM_GetLastShutdownTarget是ECU管理器模块端口接口的一部分	
<b>Available via</b>	EcuM.h	

## EcuM\_SelectShutdownCause

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

## EcuM\_GetShutdownCause

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

## EcuM\_CheckWakeup

<b>Service Name</b>	EcuM_CheckWakeup	
<b>Syntax</b>	<pre>void EcuM_CheckWakeup (     EcuM_WakeupSourceType wakeupSource )</pre>	
<b>Service ID [hex]</b>	0x49	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	wakeupSource	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This function can be called to check the given wakeup sources. It will pass the argument to the integrator function EcuM_CheckWakeupHook. It can 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.	
<b>Available via</b>	EcuM.h	

## EcuM\_GetPendingWakeupEvents

<b>Service Name</b>	EcuM_GetPendingWakeupEvents	
<b>Syntax</b>	<pre>EcuM_WakeupSourceType EcuM_GetPendingWakeupEvents (     void )</pre>	
<b>Service ID [hex]</b>	0x0d	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non-Reentrant, Non-Interruptible	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	EcuM_WakeupSource Type	All wakeup events
<b>Description</b>	Gets pending wakeup events.	
<b>Available via</b>	EcuM.h	

## EcuM\_ClearWakeupEvent

<b>Service Name</b>	EcuM_ClearWakeupEvent	
<b>Syntax</b>	<pre>void EcuM_ClearWakeupEvent (     EcuM_WakeupSourceType sources )</pre>	
<b>Service ID [hex]</b>	0x16	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non-Reentrant, Non-Interruptible	
<b>Parameters (in)</b>	sources	Events to be cleared
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Clears wakeup events.	
<b>Available via</b>	EcuM.h	

## EcuM\_GetValidatedWakeupEvents

<b>Service Name</b>	EcuM_GetValidatedWakeupEvents	
<b>Syntax</b>	<pre>EcuM_WakeupSourceType EcuM_GetValidatedWakeupEvents (     void )</pre>	
<b>Service ID [hex]</b>	0x15	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non-Reentrant, Non-Interruptible	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	EcuM_WakeupSource Type	All wakeup events
<b>Description</b>	Gets validated wakeup events.	
<b>Available via</b>	EcuM.h	

## EcuM\_GetExpiredWakeupEvents

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

## EcuM\_SetRelWakeupAlarm

<b>Service Name</b>	EcuM_SetRelWakeupAlarm	
<b>Syntax</b>	<pre>Std_ReturnType EcuM_SetRelWakeupAlarm (     EcuM_UserType user,     EcuM_TimeType time )</pre>	
<b>Service ID [hex]</b>	0x22	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	user	The user that wants to set the wakeup alarm.
	time	Relative time from now in seconds.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed ECUM_E_EARLIER_ACTIVE: An earlier alarm is already set
<b>Description</b>	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.	
<b>Available via</b>	EcuM.h	

## EcuM\_SetAbsWakeupAlarm

<b>Service Name</b>	EcuM_SetAbsWakeupAlarm	
<b>Syntax</b>	<pre>Std_ReturnType EcuM_SetAbsWakeupAlarm (     EcuM_UserType user,     EcuM_TimeType time )</pre>	
<b>Service ID [hex]</b>	0x23	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	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.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	<p>E_OK: The service has succeeded          E_NOT_OK: The service failed          ECUM_E_EARLIER_ACTIVE: An earlier alarm is already set          ECUM_E_PAST: The given point in time has already passed</p>
<b>Description</b>	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.	
<b>Available via</b>	EcuM.h	

## EcuM\_AbortWakeupAlarm

<b>Service Name</b>	EcuM_AbortWakeupAlarm	
<b>Syntax</b>	<pre>Std_ReturnType EcuM_AbortWakeupAlarm (     EcuM_UserType user )</pre>	
<b>Service ID [hex]</b>	0x24	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	user	The user that wants to cancel the wakeup alarm.
<b>Parameters</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	<p>E_OK: 服务已成功          E_NOT_OK: 服务失败          ECUM_E_NOT_ACTIVE: 未找到所属报警</p>
<b>Description</b>	EcuM_AbortWakeupAlarm中止此用户先前设置的唤醒警报。 EcuM_AbortWakeupAlarm是ECU管理器模块端口接口的一部分。	
<b>Available via</b>	EcuM.h	

## EcuM\_GetCurrentTime

<b>Service Name</b>	EcuM_GetCurrentTime	
<b>Syntax</b>	Std_ReturnType EcuM_GetCurrentTime ( EcuM_TimeType* time )	
<b>Service ID [hex]</b>	0x25	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	time	Absolute time in seconds since battery connect.
<b>Return value</b>	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: time points to NULL or the module is not initialized
<b>Description</b>	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.	
<b>Available via</b>	EcuM.h	

## EcuM\_GetWakeupTime

<b>Service Name</b>	EcuM_GetWakeupTime	
<b>Syntax</b>	Std_ReturnType EcuM_GetWakeupTime ( EcuM_TimeType* time )	
<b>Service ID [hex]</b>	0x26	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters</b>	None	
<b>Parameters (out)</b>	time	Absolute time in seconds for next wakeup. 0xFFFFFFFF means
<b>Return value</b>	Std_ReturnType	E_OK: 服务已成功 E_NOT_OK: 时间指向NULL或模块未初始化
<b>Description</b>	EcuM_GetWakeupTime返回主闹钟的当前值（所有用户闹钟的最小绝对时间）。EcuM_GetWakeupTime是ECU管理器模块端口接口的一部分。	
<b>Available via</b>	EcuM.h	

## EcuM\_SetClock

<b>Service Name</b>	EcuM_SetClock	
<b>Syntax</b>	<pre>Std_ReturnType EcuM_SetClock (     EcuM_UserType user,     EcuM_TimeType time )</pre>	
<b>Service ID [hex]</b>	0x27	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	user	User that wants to set the clock
	time	Absolute time in seconds since battery connect.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	Std_ReturnType	E_OK: The service has succeeded E_NOT_OK: The service failed
<b>Description</b>	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.	
<b>Available via</b>	EcuM.h	

## EcuM\_SelectBootTarget

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

## Ecum\_GetBootTarget

<b>Service Name</b>	<b>EcuM_GetBootTarget</b>	
<b>Syntax</b>	<pre>Std_ReturnType EcuM_GetBootTarget(     EcuM_Boot targetType * target )</pre>	
<b>Service ID</b>	0x13	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters</b>	None	
<b>Parameters</b>	target	The currently selected boot target.
<b>Return value</b>	Std_ReturnType	E_OK: The service always succeeds.
<b>Description</b>	EcuM_GetBootTarget返回当前引导目标-请参阅EcuM_SelectBootTarget。EcuM_GetBootTarget是ECU管理器模块端口接口的一部分。	
<b>Available via</b>	<b>EcuM.h</b>	

## Ecum\_SetWakeupEvent

<b>Service Name</b>	Ecum_SetWakeupEvent	
<b>Syntax</b>	<pre>void Ecum_SetWakeupEvent (     Ecum_WakeupSourceType sources )</pre>	
<b>Service ID [hex]</b>	0x0c	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non-Reentrant, Non-Interruptible	
<b>Parameters (in)</b>	sources	Value to be set
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Sets the wakeup event.	
<b>Available via</b>	<b>EcuM.h</b>	

## EcuM\_ValidateWakeupEvent

<b>Service Name</b>	EcuM_ValidateWakeupEvent	
<b>Syntax</b>	<pre>void EcuM_ValidateWakeupEvent (     EcuM_WakeupSourceType sources )</pre>	
<b>Service ID [hex]</b>	0x14	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	sources	Events that have been validated
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	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.	
<b>Available via</b>	EcuM.h	

## EcuM\_ErrorHook

<b>Service Name</b>	EcuM_ErrorHook	
<b>Syntax</b>	<pre>void EcuM_ErrorHook (     uint16 reason )</pre>	
<b>Service ID [hex]</b>	0x30	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	reason	Reason for calling the error hook
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	The ECU State Manager will call the error hook if fatal errors 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.	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_AL\_SetProgrammableInterrupts

<b>Service Name</b>	EcuM_AL_SetProgrammableInterrupts
<b>Syntax</b>	<pre>void EcuM_AL_SetProgrammableInterrupts (     void )</pre>
<b>Service ID [hex]</b>	0x4A
<b>Sync/Async</b>	Asynchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	If the configuration parameter EcuMSetProgrammableInterrupts is set to true, this callout EcuM_AL_SetProgrammableInterrupts is executed and shall set the interrupts on ECUs with programmable interrupts.
<b>Available via</b>	EcuM_externals.h

## EcuM\_AL\_DriverInitZero

<b>Service Name</b>	EcuM_AL_DriverInitZero
<b>Syntax</b>	<pre>void EcuM_AL_DriverInitZero (     void )</pre>
<b>Service ID [hex]</b>	0x31
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	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.
<b>Available via</b>	EcuM_externals.h

## EcuM\_DeterminePbConfiguration

<b>Service Name</b>	EcuM_DeterminePbConfiguration	
<b>Syntax</b>	<pre>const EcuM_ConfigType* EcuM_DeterminePbConfiguration (     void )</pre>	
<b>Service ID [hex]</b>	0x32	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	<code>const EcuM_ConfigType*</code>	Pointer to the EcuM post-build configuration which contains pointers to all other BSW module post-build configurations.
<b>Description</b>	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 configurations.	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_AL\_DriverInitOne

<b>Service Name</b>	EcuM_AL_DriverInitOne
<b>Syntax</b>	<pre>void EcuM_AL_DriverInitOne (     void )</pre>
<b>Service ID</b>	0x33
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters</b>	None
<b>Parameters</b>	None
<b>Return value</b>	None
<b>Description</b>	该调用应提供驱动器初始化和其他硬件相关的启动活动，以防电源复位。
<b>Available via</b>	EcuM_Externals.h

## EcuM\_LoopDetection

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

## EcuM\_OnGoOffOne

<b>Service Name</b>	EcuM_OnGoOffOne
<b>Syntax</b>	<pre>void EcuM_OnGoOffOne (     void )</pre>
<b>Service ID [hex]</b>	0x3C
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	This call allows the system designer to notify that the GO OFF I state is about to be entered.
<b>Available via</b>	EcuM_Externals.h

## EcuM\_OnGoOffTwo

<b>Service Name</b>	EcuM_OnGoOffTwo
<b>Syntax</b>	<pre>void EcuM_OnGoOffTwo (     void )</pre>
<b>Service ID [hex]</b>	0x3D
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	This call allows the system designer to notify that the GO OFF II state is about to be entered.
<b>Available via</b>	EcuM_Externals.h

## EcuM\_AL\_SwitchOff

<b>Service Name</b>	EcuM_AL_SwitchOff
<b>Syntax</b>	<pre>void EcuM_AL_SwitchOff (     void )</pre>
<b>Service ID [hex]</b>	0x3E
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	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.
<b>Available via</b>	EcuM_Externals.h

## EcuM\_AL\_Reset

<b>Service Name</b>	EcuM_AL_Reset	
<b>Syntax</b>	<pre>void EcuM_AL_Reset (     EcuM_ResetType reset )</pre>	
<b>Service ID [hex]</b>	0x4C	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	reset	Type of reset to be performed.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This callout shall take the code for resetting the ECU.	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_EnableWakeupSources

<b>Service Name</b>	EcuM_EnableWakeupSources	
<b>Syntax</b>	<pre>void EcuM_EnableWakeupSources (     EcuM_WakeupSourceType wakeupSource )</pre>	
<b>Service ID [hex]</b>	0x3F	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	wakeupSource	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	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.	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_GenerateRamHash

<b>Service Name</b>	<b>EcuM_GenerateRamHash</b>
<b>Syntax</b>	<pre>void EcuM_GenerateRamHash (     void )</pre>
<b>Service ID [hex]</b>	0x40
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	see <a href="#">EcuM_CheckRamHash</a>
<b>Available via</b>	<a href="#">EcuM_Externals.h</a>

## EcuM\_SleepActivity

<b>Service Name</b>	EcuM_SleepActivity
<b>Syntax</b>	<pre>void EcuM_SleepActivity (     void )</pre>
<b>Service ID [hex]</b>	0x41
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	This callout is invoked periodically in all reduced clock sleep modes. It is explicitly 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.
<b>Available via</b>	<a href="#">EcuM_Externals.h</a>

## EcuM\_StartCheckWakeups

<b>Service Name</b>	EcuM_StartCheckWakeups	
<b>Syntax</b>	<pre>void EcuM_StartCheckWakeups (     EcuM_WakeupSourceType WakeupSource )</pre>	
<b>Service ID [hex]</b>	0x00	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	WakeupSource	For this wakeup source the corresponding CheckWakeupsTimer shall be started.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This API is called by the ECU Firmware to start the CheckWakeupsTimer for the corresponding WakeupSource. If EcuMCheckWakeupsTimeout > 0 the CheckWakeupsTimer for the Wakeup Source is started. If EcuMCheckWakeupsTimeout <= 0 the API call is ignored by the EcuM.	
<b>Available via</b>	EcuM_externals.h	

## EcuM\_CheckWakeupsHook

<b>Service Name</b>	EcuM_CheckWakeupsHook	
<b>Syntax</b>	<pre>void EcuM_CheckWakeupsHook (     EcuM_WakeupSourceType wakeupSource )</pre>	
<b>Service ID [hex]</b>	0x42	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	wakeupSource	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This callout is called by the EcuM to poll a wakeup source.	
<b>Available via</b>	EcuM_externals.h	

## EcuM\_CheckRamHash

<b>Service Name</b>	EcuM_CheckRamHash	
<b>Syntax</b>	<pre>uint8 EcuM_CheckRamHash (     void )</pre>	
<b>Service ID [hex]</b>	0x43	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	uint8	0: RAM integrity test failed else: RAM integrity test passed
<b>Description</b>	<p>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.</p>	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_DisableWakeupSources

<b>Service Name</b>	EcuM_DisableWakeupSources	
<b>Syntax</b>	<pre>void EcuM_DisableWakeupSources (     EcuM_WakeupSourceType wakeupSource )</pre>	
<b>Service ID [hex]</b>	0x44	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	wakeupSource	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	<p>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.</p>	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_AL\_DriverRestart

<b>Service Name</b>	EcuM_AL_DriverRestart
<b>Syntax</b>	<pre>void EcuM_AL_DriverRestart (     void )</pre>
<b>Service ID [hex]</b>	0x45
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	This callout shall provide driver initialization and other hardware-related startup activities in the wakeup case.
<b>Available via</b>	EcuM_Externals.h

## EcuM\_StartWakeupSources

<b>Service Name</b>	EcuM_StartWakeupSources	
<b>Syntax</b>	<pre>void EcuM_StartWakeupSources (     EcuM_WakeupSourceType wakeupSource )</pre>	
<b>Service ID [hex]</b>	0x46	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	wakeupSource	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	The callout shall start the given wakeup source(s) so that they are ready to perform wakeup validation.	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_CheckValidation

<b>Service Name</b>	<b>EcuM_CheckValidation</b>	
<b>Syntax</b>	<pre>void EcuM_CheckValidation (     EcuM_WakeupSourceType wakeupSource )</pre>	
<b>Service ID</b>	0x47	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	wakeupSource	
<b>Parameters</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	EcuM调用此调用以验证唤醒源。如果检测到有效唤醒，则应通过EcuM_ValidateWakeupEvent () 向EcuM报告。	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_StopWakeupSources

<b>Service Name</b>	EcuM_StopWakeupSources	
<b>Syntax</b>	<pre>void EcuM_StopWakeupSources (     EcuM_WakeupSourceType wakeupSource )</pre>	
<b>Service ID [hex]</b>	0x48	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Non Reentrant	
<b>Parameters (in)</b>	wakeupSource	-
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	The callout shall stop the given wakeup source(s) after unsuccessful wakeup validation.	
<b>Available via</b>	EcuM_Externals.h	

## EcuM\_MainFunction

<b>Service Name</b>	EcuM_MainFunction	
<b>Syntax</b>	<pre>void EcuM_MainFunction (     void )</pre>	
<b>Service ID [hex]</b>	0x18	
<b>Description</b>	The purpose of this service is to implement all activities of the ECU State Manager while the OS is up and running.	
<b>Available via</b>	SchM_EcuM.h	

Callbacks from the STARTUP phase

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

Ports and Port Interface for EcuM\_ShutdownTarget Interface

EcuM\_ShutdownTarget客户端-服务器接口允许 SW-C 选择一个关闭目标，该目标将在下一个关闭阶段得到遵守。但是，请注意，ECU 管理器模块不提供端口接口来允许 SW-C 启动关断。

Service Interfaces

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

<b>Operation</b>	GetLastShutdownTarget																
<b>Comment</b>	Returns the shutdown target of the previous shutdown																
<b>Mapped to AP</b>	<a href="#">EcuM_GetLastShutdownTarget</a>																
<b>Variation</b>																	
	<b>shutdownTarget</b> <table border="1"> <tr> <td>Type</td> <td><a href="#">EcuM_ShutdownTargetType</a></td> </tr> <tr> <td>Direction</td> <td>OUT</td> </tr> <tr> <td>Comment</td> <td>The shutdown target of the previous shutdown</td> </tr> <tr> <td>Variation</td> <td>—</td> </tr> </table> <b>shutdownMode</b> <table border="1"> <tr> <td>Type</td> <td><a href="#">EcuM_ShutdownMode_Type</a></td> </tr> <tr> <td>Direction</td> <td>OUT</td> </tr> <tr> <td>Comment</td> <td>The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the</td> </tr> <tr> <td>Variation</td> <td>—</td> </tr> </table>	Type	<a href="#">EcuM_ShutdownTargetType</a>	Direction	OUT	Comment	The shutdown target of the previous shutdown	Variation	—	Type	<a href="#">EcuM_ShutdownMode_Type</a>	Direction	OUT	Comment	The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the	Variation	—
Type	<a href="#">EcuM_ShutdownTargetType</a>																
Direction	OUT																
Comment	The shutdown target of the previous shutdown																
Variation	—																
Type	<a href="#">EcuM_ShutdownMode_Type</a>																
Direction	OUT																
Comment	The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the																
Variation	—																
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>																

<b>Operation</b>	GetShutdownCause								
<b>Comment</b>	Returns the selected shutdown cause as set by the operation SelectShutdownCause.								
<b>Mapped to API</b>	<a href="#">EcuM_GetShutdownCause</a>								
<b>Variation</b>	—								
<b>Parameters</b>	<b>shutdownCause</b> <table border="1"> <tr> <td>Type</td> <td><a href="#">EcuM_ShutdownCauseType</a></td> </tr> <tr> <td>Direction</td> <td>OUT</td> </tr> <tr> <td>Comment</td> <td>The selected cause of the next shutdown</td> </tr> <tr> <td>Variation</td> <td>—</td> </tr> </table>	Type	<a href="#">EcuM_ShutdownCauseType</a>	Direction	OUT	Comment	The selected cause of the next shutdown	Variation	—
Type	<a href="#">EcuM_ShutdownCauseType</a>								
Direction	OUT								
Comment	The selected cause of the next shutdown								
Variation	—								
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>								

<b>Operation</b>	GetShutdownTarget																		
<b>Comment</b>	Returns the currently selected shutdown target for the next shutdown as set by the operation SelectShutdownTarget.																		
<b>Mapped to API</b>	<a href="#">EcuM_GetShutdownTarget</a>																		
<b>Variation</b>	–																		
<b>Parameters</b>	<p>shutdownTarget</p> <table> <tr> <td><b>Type</b></td><td><a href="#">EcuM_ShutdownTargetType</a></td></tr> <tr> <td><b>Direction</b></td><td>OUT</td></tr> <tr> <td><b>Comment</b></td><td>The shutdown target of the next shutdown</td></tr> <tr> <td><b>Variation</b></td><td>–</td></tr> <tr> <td>shutdownMode</td><td></td></tr> <tr> <td><b>Type</b></td><td><a href="#">EcuM_ShutdownModeType</a></td></tr> <tr> <td><b>Direction</b></td><td>OUT</td></tr> <tr> <td><b>Comment</b></td><td>The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the shutdown</td></tr> <tr> <td><b>Variation</b></td><td>–</td></tr> </table>	<b>Type</b>	<a href="#">EcuM_ShutdownTargetType</a>	<b>Direction</b>	OUT	<b>Comment</b>	The shutdown target of the next shutdown	<b>Variation</b>	–	shutdownMode		<b>Type</b>	<a href="#">EcuM_ShutdownModeType</a>	<b>Direction</b>	OUT	<b>Comment</b>	The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the shutdown	<b>Variation</b>	–
<b>Type</b>	<a href="#">EcuM_ShutdownTargetType</a>																		
<b>Direction</b>	OUT																		
<b>Comment</b>	The shutdown target of the next shutdown																		
<b>Variation</b>	–																		
shutdownMode																			
<b>Type</b>	<a href="#">EcuM_ShutdownModeType</a>																		
<b>Direction</b>	OUT																		
<b>Comment</b>	The sleep mode (if target is ECUM_SHUTDOWN_TARGET_SLEEP) or the reset mechanism (if target is ECUM_SHUTDOWN_TARGET_RESET) of the shutdown																		
<b>Variation</b>	–																		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>																		

<b>Operation</b>	SelectShutdownCause								
<b>Comment</b>	–								
<b>Mapped to API</b>	<a href="#">EcuM_SelectShutdownCause</a>								
<b>Variation</b>	–								
<b>Parameters</b>	<p>shutdownCause</p> <table> <tr> <td><b>Type</b></td><td><a href="#">EcuM_ShutdownCauseType</a></td></tr> <tr> <td><b>Direction</b></td><td>IN</td></tr> <tr> <td><b>Comment</b></td><td>The selected shutdown cause</td></tr> <tr> <td><b>Variation</b></td><td>–</td></tr> </table>	<b>Type</b>	<a href="#">EcuM_ShutdownCauseType</a>	<b>Direction</b>	IN	<b>Comment</b>	The selected shutdown cause	<b>Variation</b>	–
<b>Type</b>	<a href="#">EcuM_ShutdownCauseType</a>								
<b>Direction</b>	IN								
<b>Comment</b>	The selected shutdown cause								
<b>Variation</b>	–								
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>								

<b>Operation</b>	SelectShutdownTarget																
<b>Comment</b>	The SW-C selects the cause corresponding to the next shutdown target																
<b>Mapped to API</b>	<a href="#">EcuM_SelectShutdownTarget</a>																
<b>Variation</b>	–																
<b>Parameters</b>	<p>shutdownTarget</p> <table> <tr> <td><b>Type</b></td><td><a href="#">EcuM_ShutdownTargetType</a></td></tr> <tr> <td><b>Direction</b></td><td>IN</td></tr> <tr> <td><b>Comment</b></td><td>The selected shutdown cause</td></tr> <tr> <td><b>Variation</b></td><td>–</td></tr> </table> <p>shutdownMode</p> <table> <tr> <td><b>Type</b></td><td><a href="#">EcuM_ShutdownModeType</a></td></tr> <tr> <td><b>Direction</b></td><td>IN</td></tr> <tr> <td><b>Comment</b></td><td>The identifier of a sleep mode (if shutdownTarget is ECUM_SHUTDOWN_TARGET_SLEEP) or a reset mechanism (if shutdownTarget is ECUM_SHUTDOWN_TARGET_RESET) as defined by configuration.</td></tr> <tr> <td><b>Variation</b></td><td>–</td></tr> </table>	<b>Type</b>	<a href="#">EcuM_ShutdownTargetType</a>	<b>Direction</b>	IN	<b>Comment</b>	The selected shutdown cause	<b>Variation</b>	–	<b>Type</b>	<a href="#">EcuM_ShutdownModeType</a>	<b>Direction</b>	IN	<b>Comment</b>	The identifier of a sleep mode (if shutdownTarget is ECUM_SHUTDOWN_TARGET_SLEEP) or a reset mechanism (if shutdownTarget is ECUM_SHUTDOWN_TARGET_RESET) as defined by configuration.	<b>Variation</b>	–
<b>Type</b>	<a href="#">EcuM_ShutdownTargetType</a>																
<b>Direction</b>	IN																
<b>Comment</b>	The selected shutdown cause																
<b>Variation</b>	–																
<b>Type</b>	<a href="#">EcuM_ShutdownModeType</a>																
<b>Direction</b>	IN																
<b>Comment</b>	The identifier of a sleep mode (if shutdownTarget is ECUM_SHUTDOWN_TARGET_SLEEP) or a reset mechanism (if shutdownTarget is ECUM_SHUTDOWN_TARGET_RESET) as defined by configuration.																
<b>Variation</b>	–																
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>																

Port Interface for EcuM\_BootTarget Interface

想要选择引导目标的SW-C必须需要客户端-服务器接口EcuM\_BootT target。

Service Interfaces

Name	<a href="#">EcuM_BootTarget</a>		
<b>Comment</b>	A SW-C that wants to select a boot target must use the client-server interface <a href="#">EcuM_Boot</a>		
<b>IsService</b>	true		
<b>Variation</b>			
<b>Possible Errors</b>	0	<a href="#">E_OK</a>	Operation successful
	1	<a href="#">E_NOT_OK</a>	Operation failed

<b>Operation</b>	GetBootTarget										
<b>Comment</b>	Returns the current boot target										
<b>Mapped to API</b>	<a href="#">EcuM_GetBootTarget</a>										
<b>Variation</b>	–										
<b>Parameters</b>	<p>target</p> <table> <tr> <td><b>Type</b></td><td><a href="#">EcuM_BootTargetType</a></td></tr> <tr> <td><b>Direction</b></td><td>OUT</td></tr> <tr> <td><b>Comment</b></td><td>The currently selected boot target</td></tr> <tr> <td><b>Variation</b></td><td>–</td></tr> </table>			<b>Type</b>	<a href="#">EcuM_BootTargetType</a>	<b>Direction</b>	OUT	<b>Comment</b>	The currently selected boot target	<b>Variation</b>	–
<b>Type</b>	<a href="#">EcuM_BootTargetType</a>										
<b>Direction</b>	OUT										
<b>Comment</b>	The currently selected boot target										
<b>Variation</b>	–										
<b>Possible Errors</b>	<a href="#">E_OK</a>										

<b>Operation</b>	SelectBootTarget
<b>Comment</b>	Selects a boot target
<b>Mapped to API</b>	<a href="#">EcuM_SelectBootTarget</a>
<b>Variation</b>	-
<b>Parameters</b>	target
	<b>Type</b> <a href="#">EcuM_BootTargetType</a>
	<b>Direction</b> IN
	<b>Comment</b> The selected boot target
	<b>Variation</b> -
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>

## Port Interface for EcuM\_AlarmClock Interface

想要使用闹钟的 SW-C 必须要求客户端-服务器接口EcuM\_AlarmClock。EcuM\_AlarmClock 接口使用端口定义的参数值来标识管理其闹钟的用户。

## Service Interfaces

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

<b>Operation</b>	AbortWakeupAlarm		
<b>Comment</b>	Aborts the wakeup alarm previously set by this user		
<b>Mapped to API</b>	<a href="#">EcuM_AbortWakeupAlarm</a>		
<b>Variation</b>	-		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a> <a href="#">ECUM_E_NOT_ACTIVE</a>		

<b>Operation</b>	SetAbsWakeupAlarm		
<b>Comment</b>	Sets the user's wakeup alarm to an absolute point in time		
<b>Mapped to API</b>	<a href="#">EcuM_SetAbsWakeupAlarm</a>		
<b>Variation</b>	–		
<b>Parameters</b>	time <b>Type</b> <a href="#">EcuM_TimeType</a> <b>Direction</b> IN <b>Comment</b> Absolute time in seconds. Note that, absolute alarms use knowledge of the current time <b>Variation</b> –		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a> <a href="#">ECUM_E_EARLIER_ACTIVE</a> <a href="#">ECUM_E_PAST</a>		

<b>Operation</b>	SetClock		
<b>Comment</b>	Sets the EcuM clock time to the provided value		
<b>Mapped to API</b>	<a href="#">EcuM_SetClock</a>		
<b>Variation</b>	–		
<b>Parameters</b>	time <b>Type</b> <a href="#">EcuM_TimeType</a> <b>Direction</b> IN <b>Comment</b> Absolute time in seconds since battery connect <b>Variation</b> –		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>		

<b>Operation</b>	SetRelWakeupAlarm		
<b>Comment</b>	Sets a user's wakeup alarm relative to the current point in time		
<b>Mapped to API</b>	<a href="#">EcuM_SetRelWakeupAlarm</a>		
<b>Variation</b>	–		
<b>Parameters</b>	time <b>Type</b> <a href="#">EcuM_TimeType</a> <b>Direction</b> IN <b>Comment</b> Relative time from now in seconds <b>Variation</b> –		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a> <a href="#">ECUM_E_EARLIER_ACTIVE</a>		

Port Interface for EcuM\_Time Interface

想要使用EucM的时间功能的SW-C必须需要客户端-服务器接口EcuM\_time

Service Interfaces

<b>Name</b>	EcuM_Time		
<b>Comment</b>	—		
<b>IsService</b>	true		
<b>Variation</b>	—		
<b>Possible Errors</b>	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

<b>Operation</b>	GetCurrentTime		
<b>Comment</b>	返回EcuM时钟的当前值（即电池连接后的秒数）		
<b>Mapped to API</b>	<a href="#">EcuM_GetCurrentTime</a>		
<b>Variation</b>	—		
<b>Parameters</b>	time		
	Type	<a href="#">EcuM_TimeType</a>	
	Direction	OUT	
	Comment	Absolute time in seconds since battery connect	
	Variatio		
<b>Possible Errors</b>	E_OK E_NOT_OK		

<b>Operation</b>	GetWakeupTime		
<b>Comment</b>	Returns the current value of the master alarm clock (the minimum absolute time of all user alarm clocks)		
<b>Mapped to API</b>	<a href="#">EcuM_GetWakeupTime</a>		
<b>Variation</b>	—		
<b>Parameters</b>	time		
	Type	<a href="#">EcuM_TimeType</a>	
	Direction	OUT	
	Comment	Absolute time in seconds for next wakeup. 0xFFFFFFFF means no active alarm.	
	Variatio	—	
<b>Possible Errors</b>	E_OK E_NOT_OK		

Port Interface for EcuM\_StateRequest Interface

当容器 EcuMMModeHandling 可用时，ECU 状态管理器模块应为以下功能提供系统服务：

- requesting RUN
- releasing RUN
- requesting POST\_RUN
- releasing POST\_RUN

需要保持 ECU 活动状态或需要在 ECU 关闭之前执行任何操作的 SW-C 需要客户端-服务器接口 EcuM\_StateRequest。此接口使用端口定义的参数值来标识请求模式的用户。

Service Interfaces

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

<b>Operation</b>	ReleasePOSTRUN		
<b>Comment</b>	–		
<b>Mapped to API</b>	<a href="#">EcuM_ReleasePOST_RUN</a>		
<b>Variation</b>	–		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>		

<b>Operation</b>	ReleaseRUN		
<b>Comment</b>	–		
<b>Mapped to API</b>	<a href="#">EcuM_ReleaseRUN</a>		
<b>Variation</b>	–		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>		

<b>Operation</b>	RequestPOSTRUN		
<b>Comment</b>	–		
<b>Mapped to API</b>	<a href="#">EcuM_RequestPOST_RUN</a>		
<b>Variation</b>	–		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>		

<b>Operation</b>	RequestRUN		
<b>Comment</b>	–		
<b>Mapped to API</b>	<a href="#">EcuM_RequestRUN</a>		
<b>Variation</b>	–		
<b>Possible Errors</b>	<a href="#">E_OK</a> <a href="#">E_NOT_OK</a>		

Port Interface for EcuM\_CurrentMode Interface

ECU 状态管理器模块的模式端口应声明以下模式:

- STARTUP
- RUN
- POST\_RUN
- SLEEP
- SHUTDOWN

此定义是应用程序确实需要知道的ECU模式的简化视图。它不会以任何方式限制或限制如何定义应用程序模式。应用程序模式完全由应用程序本身处理。

发生模式更改时，应通过 RTE 模式端口将模式更改通知 SW-C。

该规范假定端口名为 currentMode，并且将使用 RTE 的直接 API。在这些条件下，通过调用 `ype Rte_Switch_currentMode currentMode (Rte_ModeT ype_EcuM_Mode mode) Rte_StatusT` 模式发出信号，其中模式是要通知的新模式。值范围由前面的要求指定。应忽略返回值。

想要收到模式更改通知的 SW-C 应要求模式开关接口 `EcuM_CurrentMode`。

#### Service Interfaces

<b>Name</b>	<code>EcuM_CurrentMode</code>		
<b>Comment</b>	Interface to read the current ECU mode		
<b>IsService</b>	true		
<b>Variation</b>	-		
<b>ModeGroup</b>	currentMode	<code>EcuM_Mode</code>	

#### Definition of the ECU Manager Service

请注意，这些定义只能在ECU配置期间完成（因为某些ECU管理器模块配置参数决定了ECU管理器模块服务提供的端口数）。另请注意，SW-C 的实现不依赖于这些定义。

在 AUTOSAR 系统中，RTE 上方和下方都有端口。ECU 管理器模块服务描述定义了提供给 RTE 的端口，使用此服务的每个软件的描述都必须包含“服务端口”，这些端口需要来自 RTE 的这些 ECU 管理器模块端口。

EcuM 提供以下端口：

<b>Name</b>	<code>ShutdownTarget_{UserName}</code>		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	<code>EcuM_ShutdownTarget</code>
<b>Description</b>	Provides an interface to SW-Cs to select a new shutdown target and query the current shutdown target.		
<b>Variation</b>	<code>UserName = {ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.SHORT-NAME)}</code>		

<b>Name</b>	<code>BootTarget_{UserName}</code>		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	<code>EcuM_BootTarget</code>
<b>Description</b>	Provides an interface to SW-Cs to select a new boot target and query the current boot target.		
<b>Variation</b>	<code>UserName = {ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.SHORT-NAME)}</code>		

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

<b>Name</b>	time		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	EcuM_Time
<b>Description</b>	Provides the EcuM's time service to SWCs		
<b>Variation</b>	-		

<b>Name</b>	StateRequest_{UserName}		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	EcuM_StateRequest
<b>Description</b>	Provides an interface to SWCs to request state changes of the ECU state. The port uses port-defined argument values to identify the user.		
<b>Port Defined Argument Value(s)</b>	Type	EcuM_UserType	
	Value	{ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.value)}	
<b>Variation</b>	UserName -{ecuc(EcuM/EcuMConfiguration/EcuMFlexConfiguration/EcuMFlexUserConfig/EcuMFlexUser.SHORT-NAME)}		

<b>Name</b>	currentMode		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	EcuM_CurrentMode
<b>Description</b>	-		
<b>Variation</b>	-		

EcuM提供以下类型:

<b>Name</b>	EcuM_UserType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Description</b>	Unique value for each user.		
<b>Variation</b>	-		
<b>Available via</b>	Rte_EcuM_Type.h		

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

<b>Name</b>	EcuM_BootTargetType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Range</b>	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
<b>Description</b>	This type represents the boot targets the ECU Manager module can be configured with. The default boot target is ECUM_BOOT_TARGET_OEM_BOOTLOADER.		
<b>Variation</b>	-		
<b>Available via</b>	Rte_EcuM_Type.h		

<b>Name</b>	EcuM_ShutdownCauseType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Range</b>	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
<b>Description</b>	This type describes the cause for a shutdown by the ECU State Manager. It can be extended by configuration.		
<b>Variation</b>	-		
<b>Available via</b>	Rte_EcuM_Type.h		

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

<b>Name</b>	EcuM_ShutdownTargetType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint8		
<b>Range</b>	ECUM_SHUTDOWN_TARGET_SLEEP	0x0	-
	ECUM_SHUTDOWN_TARGET_RESET	0x1	-
	ECUM_SHUTDOWN_TARGET_OFF	0x2	-
<b>Description</b>	-		
<b>Variation</b>	-		
<b>Available via</b>	Rte_EcuM_Type.h		

对于多核ECU，可以在一个或多个内核上提供EcuM AUTOSAR服务（标准化AUTOSAR接口）。

对于多核ECU，其他BSW模块使用的EcuM C-API接口（标准化接口）应在运行EcuM的每个分区中提供。

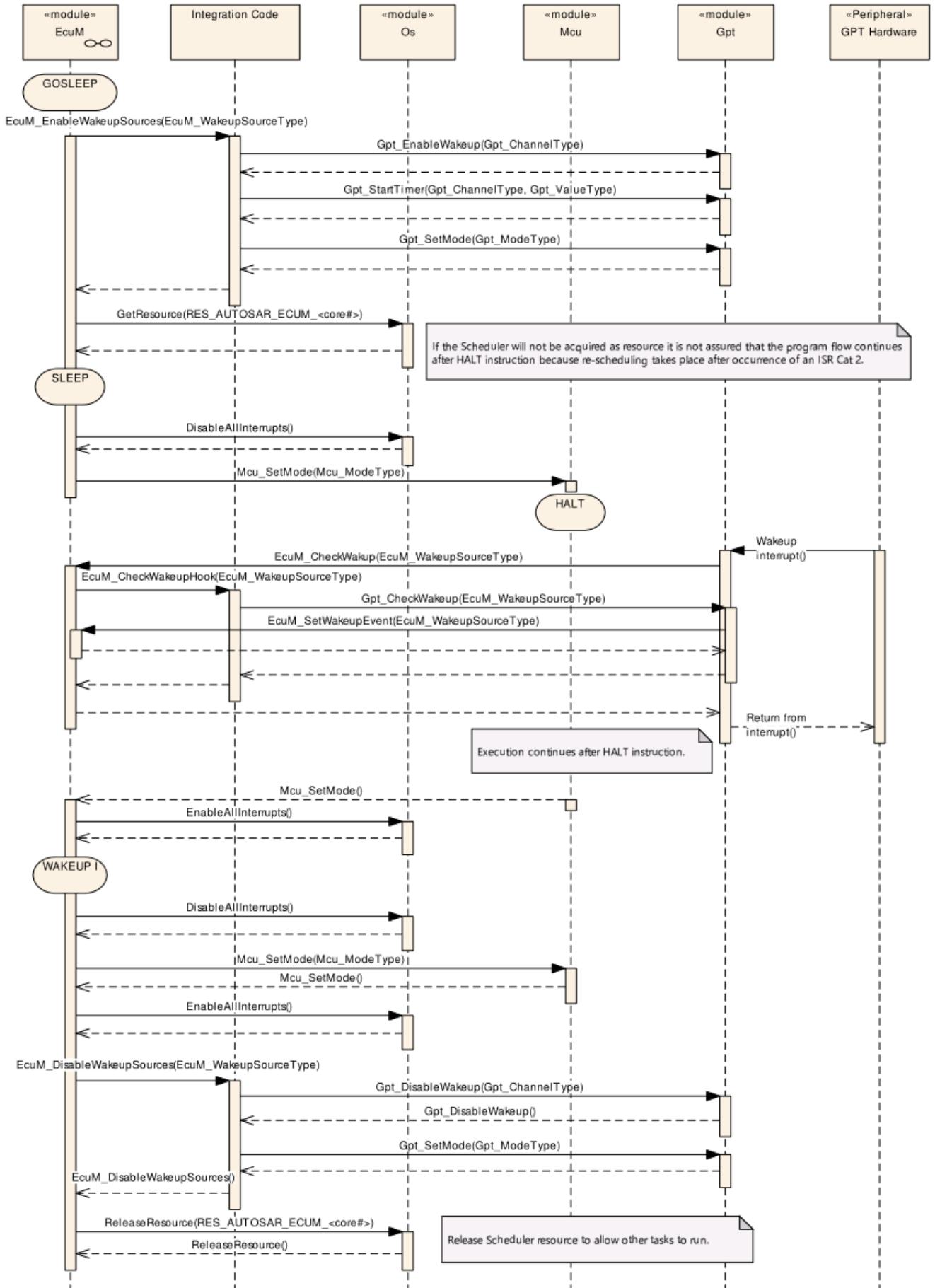
其他 BSW 模块用于与 EcuM 通信的 C-API 接口由每个 EcuM 实例提供，因为每个 EcuM 实例都可以执行一些独立的操作。如果 BSW 模块想要使用 EcuM，但位于不包含自己的 EcuM 实例的分区。这些模块可以使用 SchM 函数来跨越分区边界。

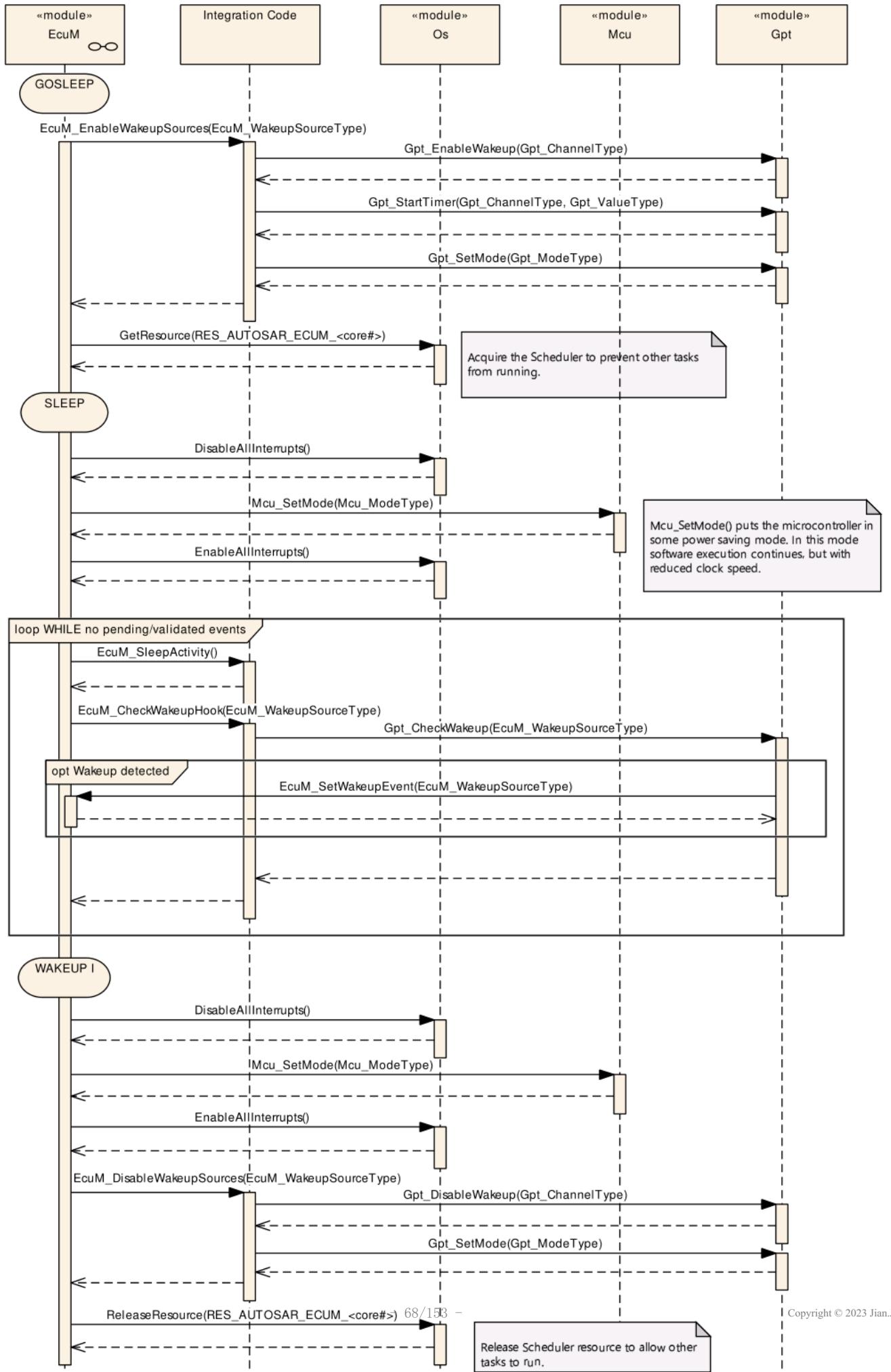
## 2.1.4 Sequence Charts

### Sequence Charts

#### GPT WAKEUP SEQUENCES

通用计时器（GPT）是可能的唤醒源之一。通常，GPT 在 ECU 进入睡眠状态之前启动，硬件计时器在到期时会导致中断。中断唤醒微控制器，并在 GPT 模块中执行中断处理程序。它通知 ECU 状态管理器模块发生了 GPT 唤醒。为了区分导致唤醒的不同 GPT 通道，集成商可以为每个 GPT 通道分配不同的唤醒源标识符。图 9.1 显示了相应的调用顺序。

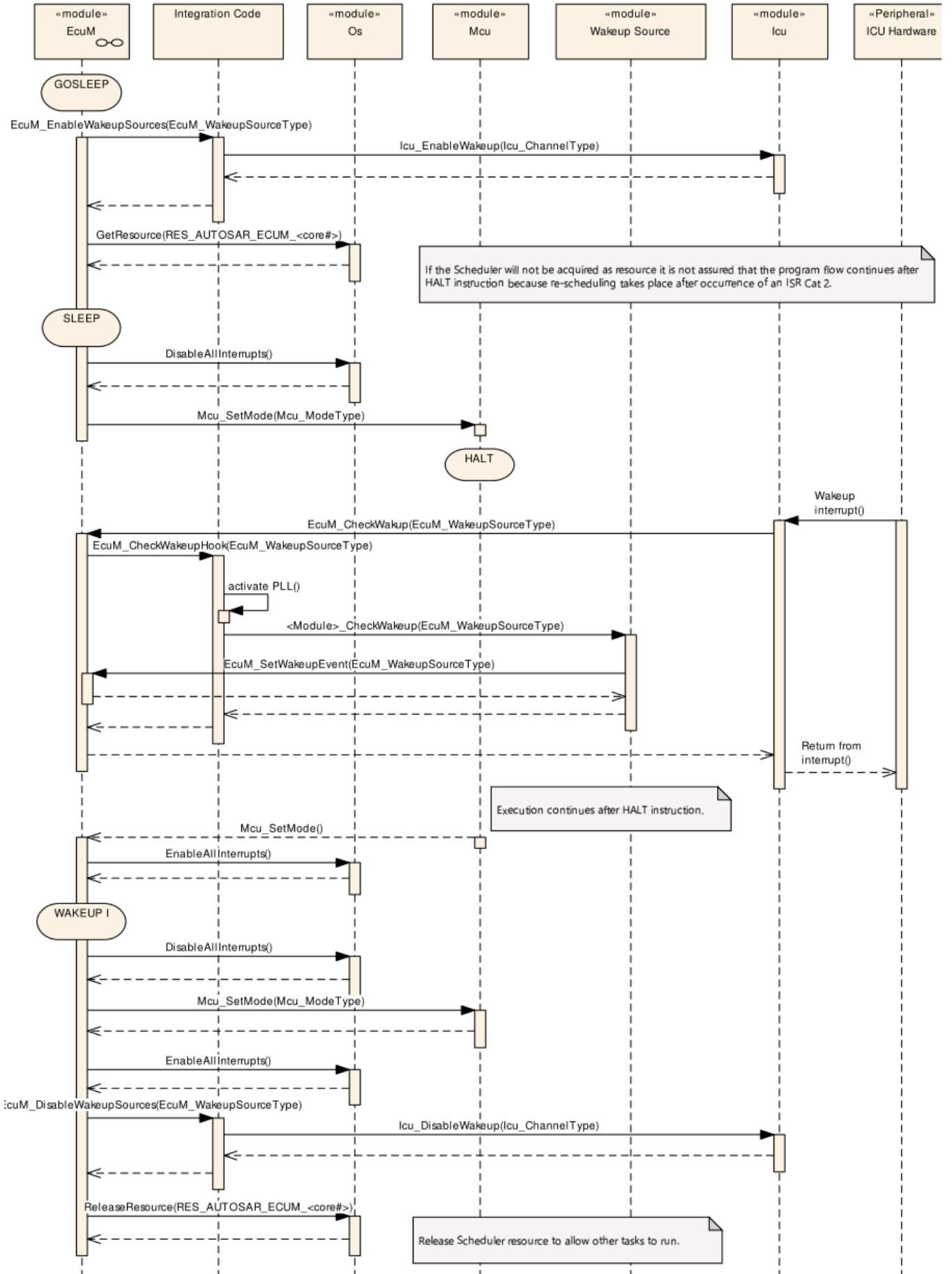
**Figure 9.1: GPT wake up by interrupt**



#### ICU WAKEUP SEQUENCES

输入捕获单元（ICU）是另一个唤醒源。与GPT相反，ICU驱动程序本身并不是唤醒源。它只是处理唤醒中断的模块。因此，只有唤醒源的驱动程序才能判断它是否负责该唤醒。这使得EcuM\_CheckWakeupHook必须询问作为实际唤醒源的模块。为了知道要询问哪个模块，ICU必须将唤醒源的标识符传递给EcuM\_CheckWakeup。对于共享中断，集成代码可能必须检查EcuM\_CheckWakeupHook内的多个唤醒源。为此，ICU必须传递可能导致此中断EcuM\_CheckWakeup的所有唤醒源的标识符。请注意，EcuM\_WakeupSourceType（请参阅 8.2.3 EcuM\_WakeupSourceType）包含每个唤醒源的一个位，因此可以在一次调用中传递多个唤醒源。

图 9.3 显示了生成的调用序列。由于 ICU 只负责处理唤醒中断，因此轮询 ICU 是不明智的。为了轮询，必须直接检查唤醒源，如图 38 所示。

**Figure 9.3: ICU wake up by interrupt**

#### CAN WAKEUP SEQUENCES

在CAN上，收发器或通信控制器可以使用中断或轮询来检测唤醒。唤醒源标识符应在收发器和控制器之间共享，因为ECU状态管理器模块只需要知道已唤醒的网络并将其传递给通信管理器模块。

在中断或共享中断情况下，不清楚哪个特定的唤醒源（CAN控制器、CAN收发器、LIN控制器等）检测到唤醒。因此，集成器必须将派生的 EcuM\_CheckWakeups 唤醒源（wakeupSource），它可以代表共享中断或仅代表中断通道，分配给传递给 CanIf\_CheckWakeups（WakeupSource）的特定唤醒源。因此，这里的参数 wakeupSource 来自 EcuM\_CheckWakeups（）可能与 CanIf\_CheckWakeups 的 WakeupSource 不同，或者它们可以相等。这取决于硬件拓扑和EcuM\_CheckWakeupsHook集成器代码中的实现。

在中断或共享中断情况下，不清楚哪个特定的唤醒源（CAN控制器、CAN收发器、LIN控制器等）检测到唤醒。因此，集成器必须将 EcuM\_CheckWakeups(wakeupSource) 的派生的唤醒源分配给传递给特定唤醒源，它可以代表共享中断或仅代表中断通道，并传递给 CanIf\_CheckWakeups(WakeupSource)。因此，这里的参数 wakeupSource 来自 EcuM\_CheckWakeups（）可能与 CanIf\_CheckWakeups 的 WakeupSource 不同，或者它们可以相等。这取决于硬件拓扑和EcuM\_CheckWakeupsHook集成器代码中的实现。

在CanIf\_CheckWakeups(WakeupSource)期间，CAN接口模块(CanIf)将检查是否有任何设备(CAN通信控制器或收发器)配置了“WakeupSource”的值。如果是这种情况，则通过相应的设备驱动程序模块检查设备是否唤醒。如果设备检测到唤醒，设备驱动程序将通过EcuM\_SetWakeupsEvent(sources)通知 EcuM。参数“sources”在设备上设置为已配置的值。因此它被设置为调用CanIf\_CheckWakeups()时使用的值。

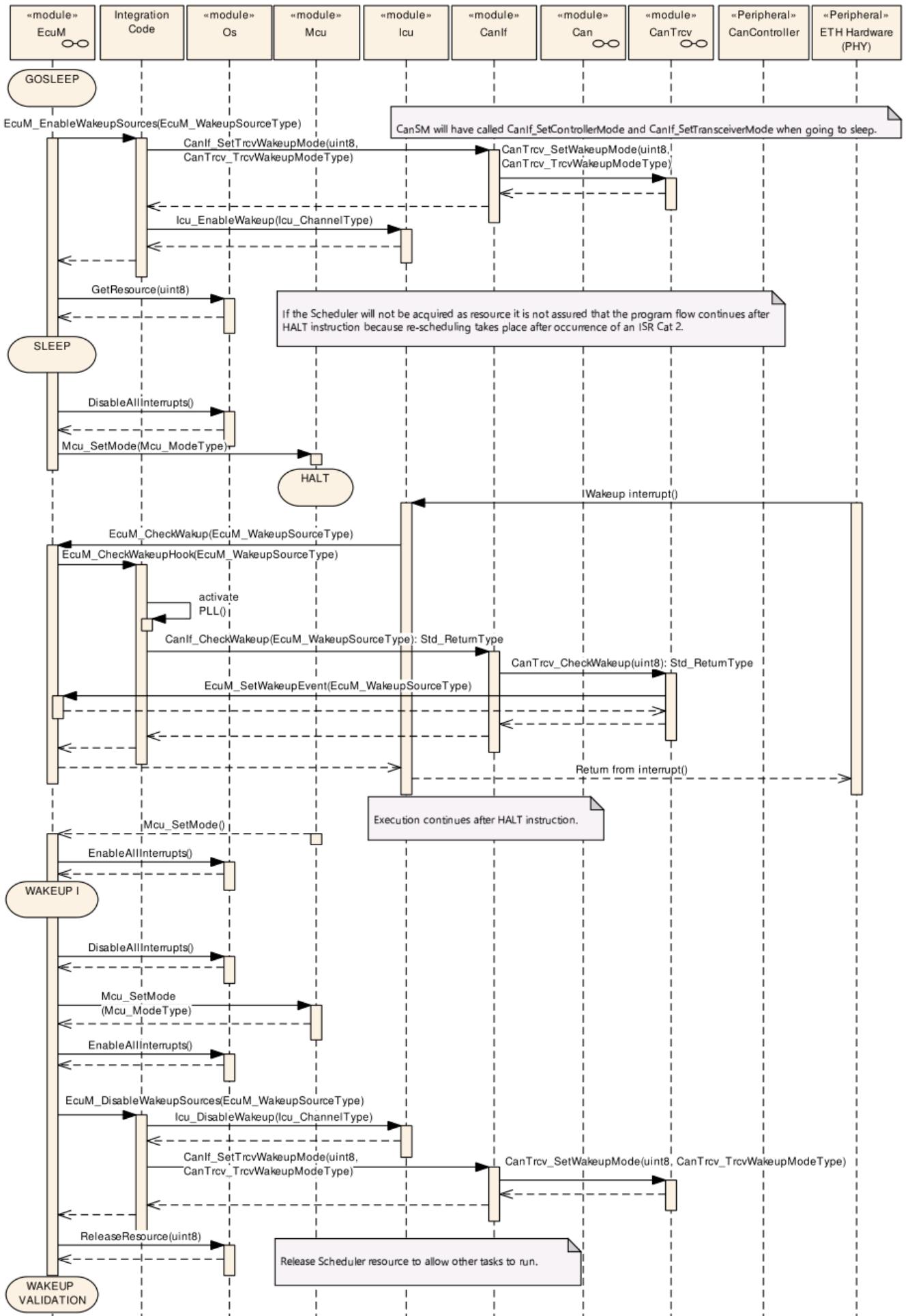


Figure 9.4: CAN transceiver wake up by interrupt

CAN控制器中断唤醒的工作方式类似于GPT唤醒。此处，中断处理程序和 CheckWakeup 功能都封装在 CAN 驱动程序模块中，如图 9.5 所示。

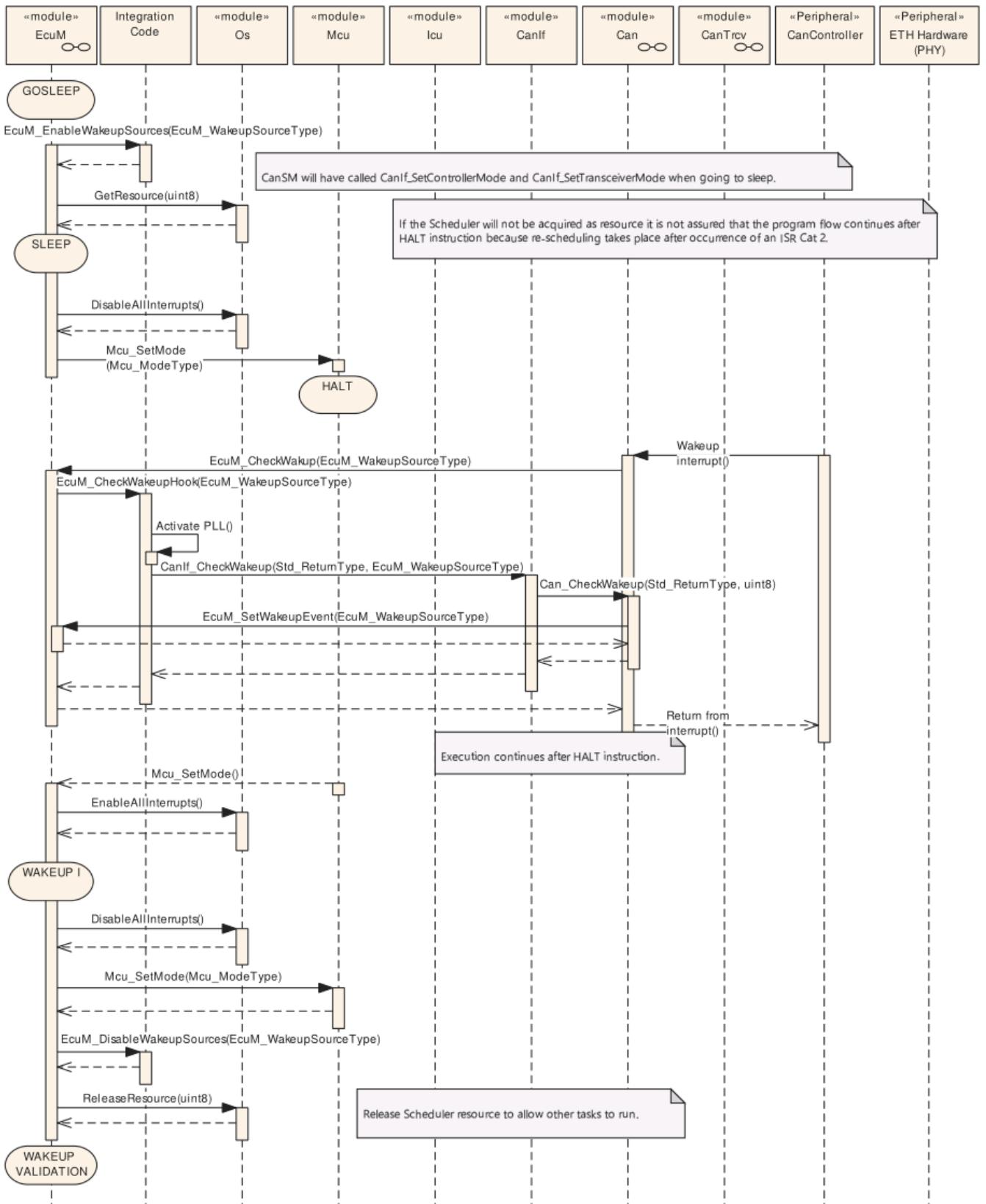


Figure 9.5: CAN controller wake up by interrupt

CAN收发器和控制器都可以通过轮询唤醒。ECU状态管理器模块将定期检查CAN接口模块，该模块根据传递给CAN接口模块的唤醒源参数询问CAN驱动程序模块或CAN收发器驱动程序模块，如图9.6所示。

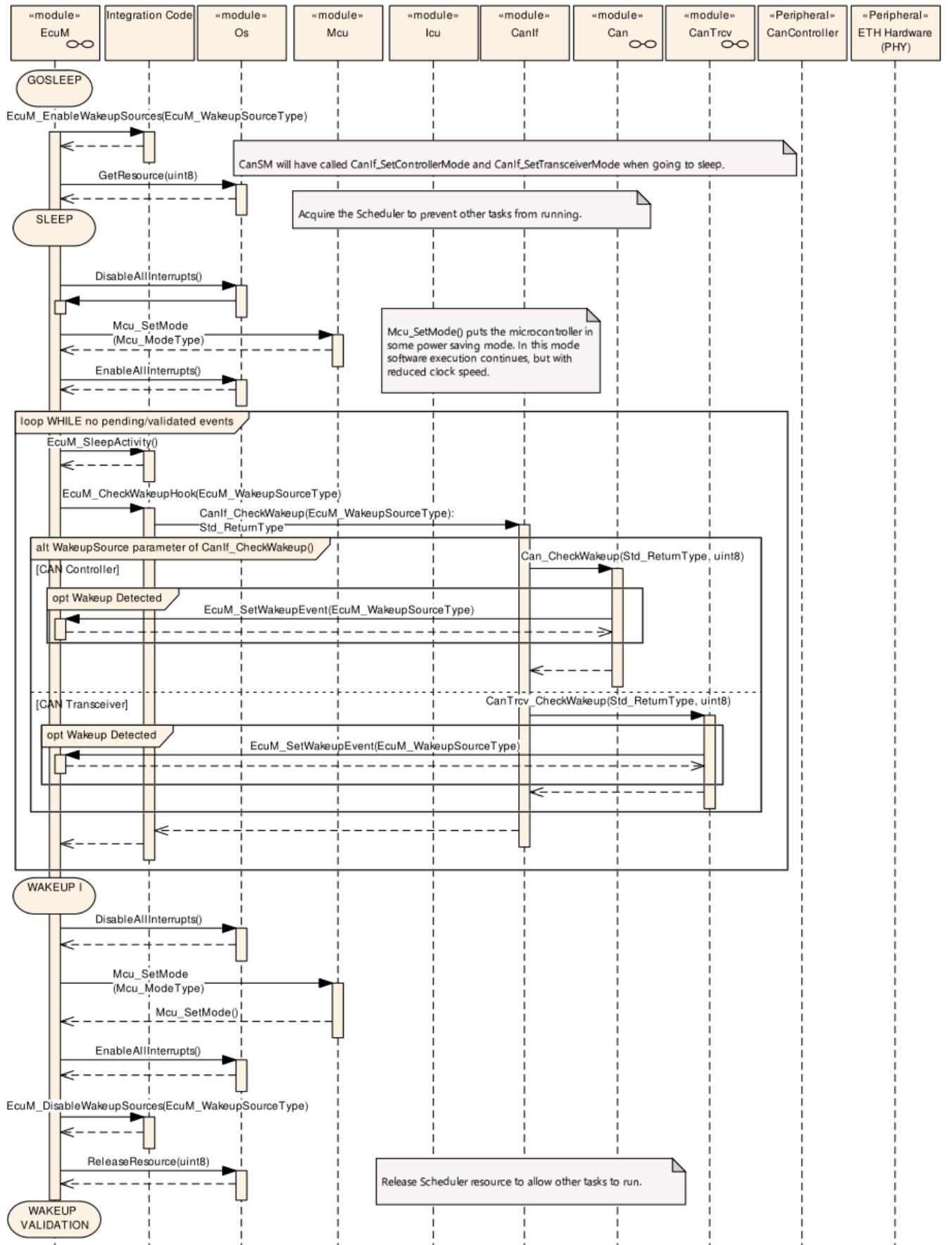
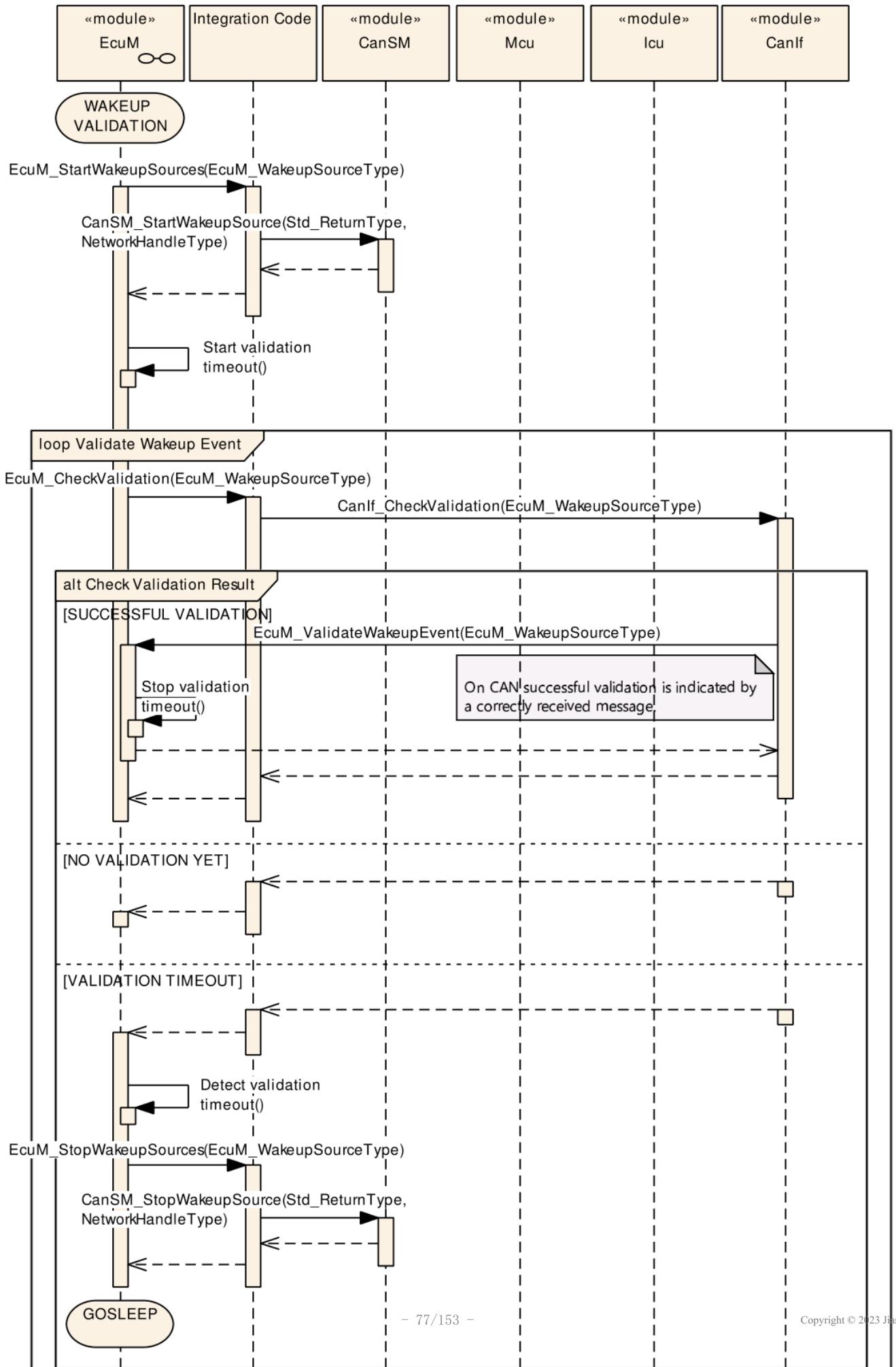


Figure 9.6: CAN controller or transceiver wake up by polling

通过中断或轮询检测到来自CAN收发器或控制器的唤醒事件后，可以验证唤醒事件。这是通过打开EcuM\_StartWakeupSources中相应的CAN收发器和控制器来完成的（参见[SWS\_EcuM\_02924]）。这取决于所使用的CAN收发器和控制器，集成器代码EcuM\_StartWakeupSource中的哪些函数调用是必要的。例如，在图 9.7 中，提到了启动和停止来自 CAN 状态管理器模块的唤醒源所需的函数调用。

CanIf 识别成功接收至少一条消息，并将其记录为成功验证。在验证期间，ECU状态管理器模块定期检查集成器代码EcuM\_CheckValidation中的CanIf ECU状态管理器模块在验证成功后，将通过通信管理器模块继续正常启动CAN网络。



**LIN WAKEUP SEQUENCES**

图9.8显示了LIN收发器通过中断唤醒。中断通常由ICU驱动程序处理，如9.2.2章所述。

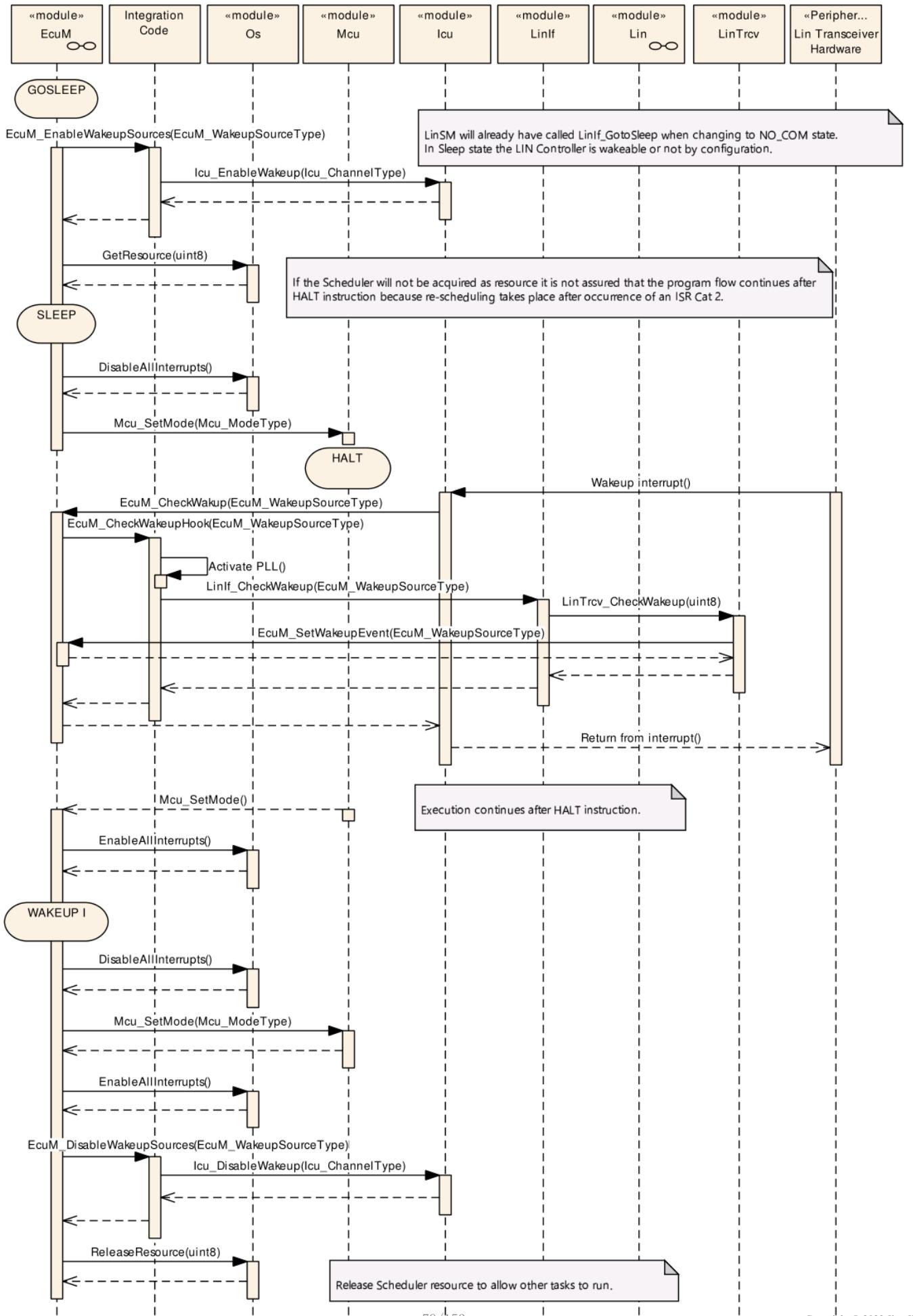


Figure 9.8: LIN transceiver wake up by interrupt

如图9.9所示，LIN控制器中断唤醒原理与CAN控制器中断唤醒原理类似。在这两种情况下，Driver模块封装了中断处理程序。

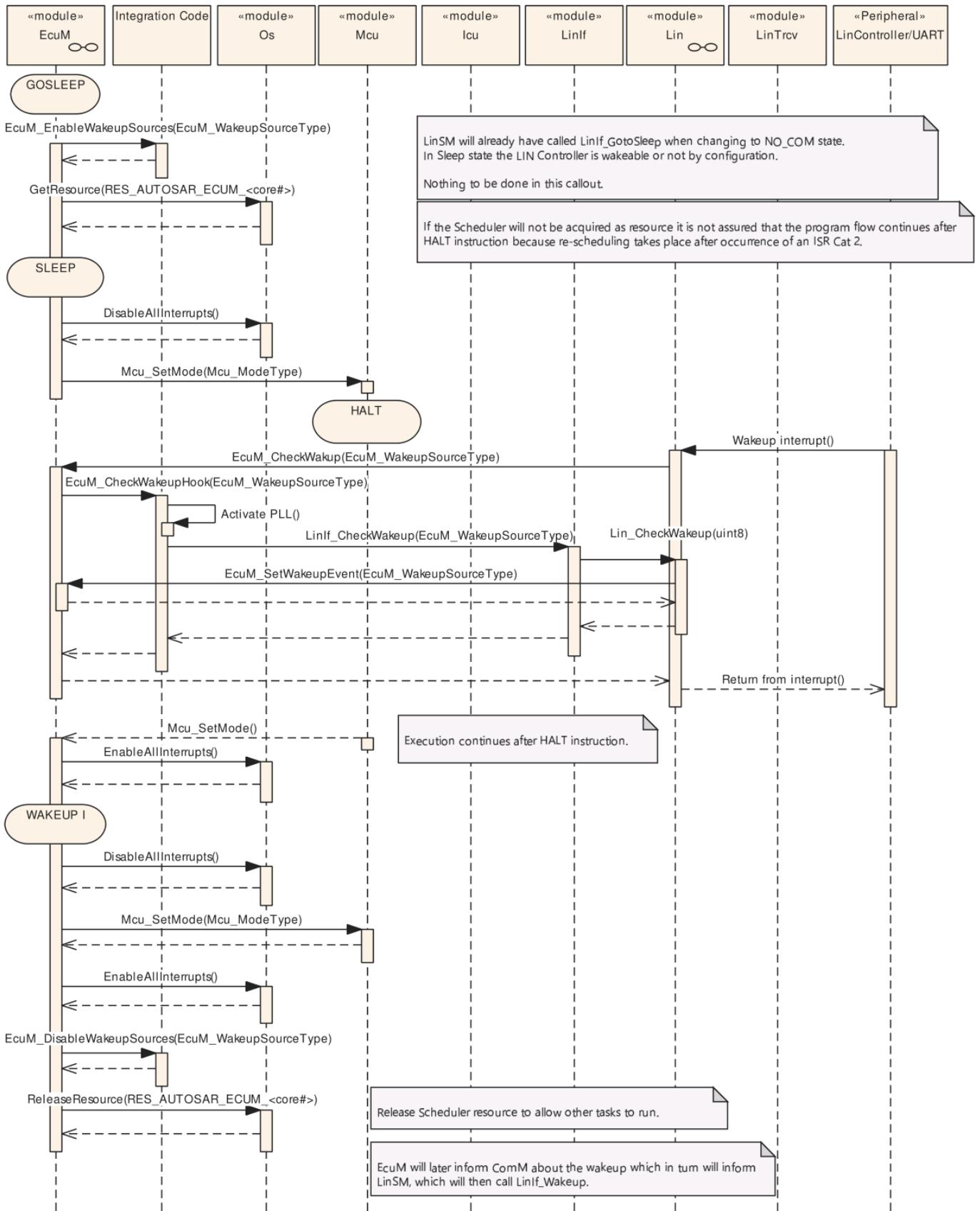
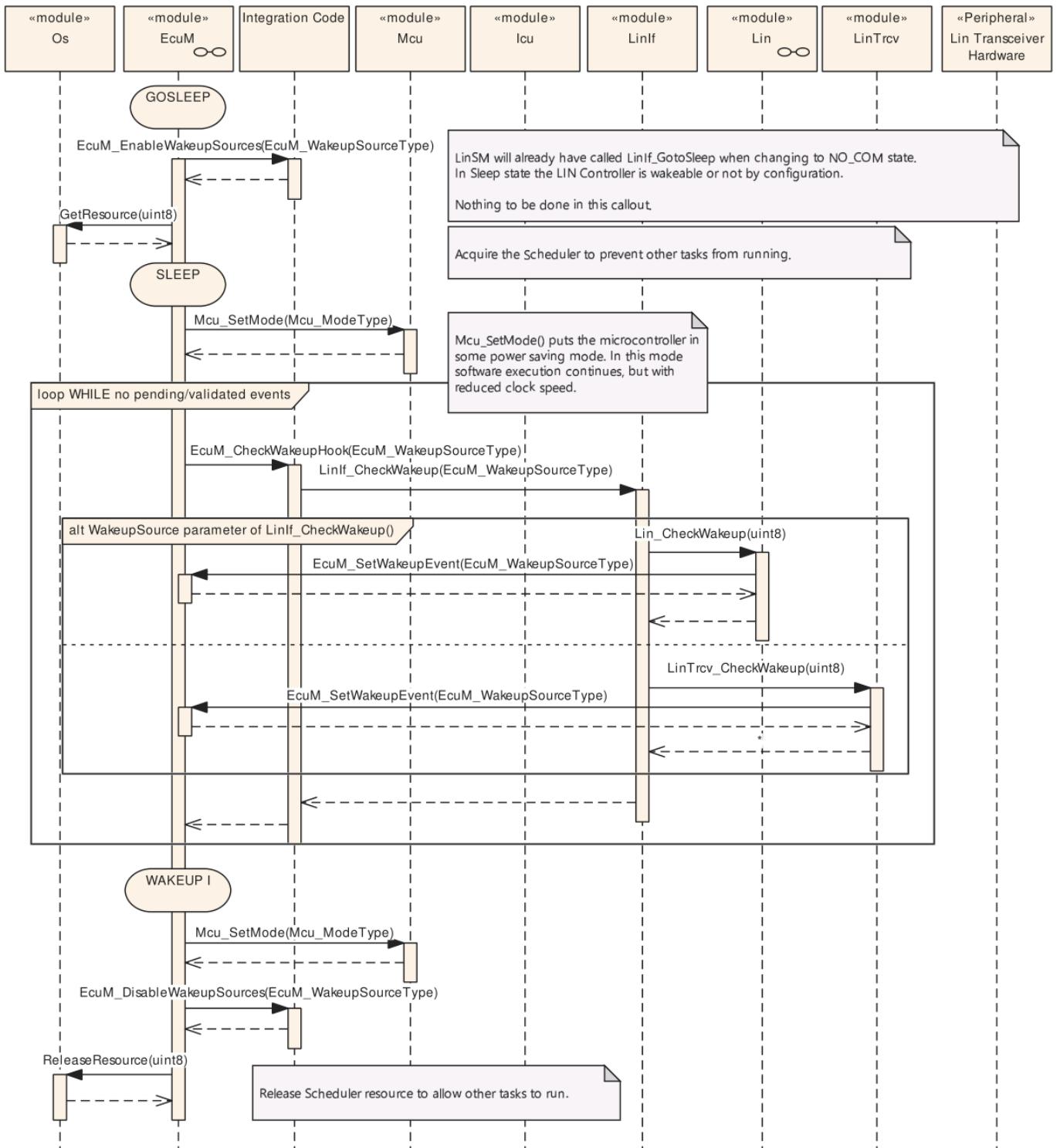


Figure 9.9: LIN controller wake up by interrupt

LIN收发器和控制器可以轮询唤醒。ECU状态管理器模块会定期检查LIN接口模块，而LIN接口模块会询问LIN驱动模块或LIN收发器驱动模块，如图9.10所示。



**Figure 9.10: LIN controller or transceiver wake up by polling**

#### FLEXRAY WAKEUP SEQUENCES

对于FlexRay，只有通过FlexRay收发器才能实现唤醒。FlexRay集群中有两个不同通道的收发器。它们被视为属于一个网络，因此，两个通道应该只配置一个唤醒源标识符。图9.11显示了FlexRay收发器通过中断唤醒。

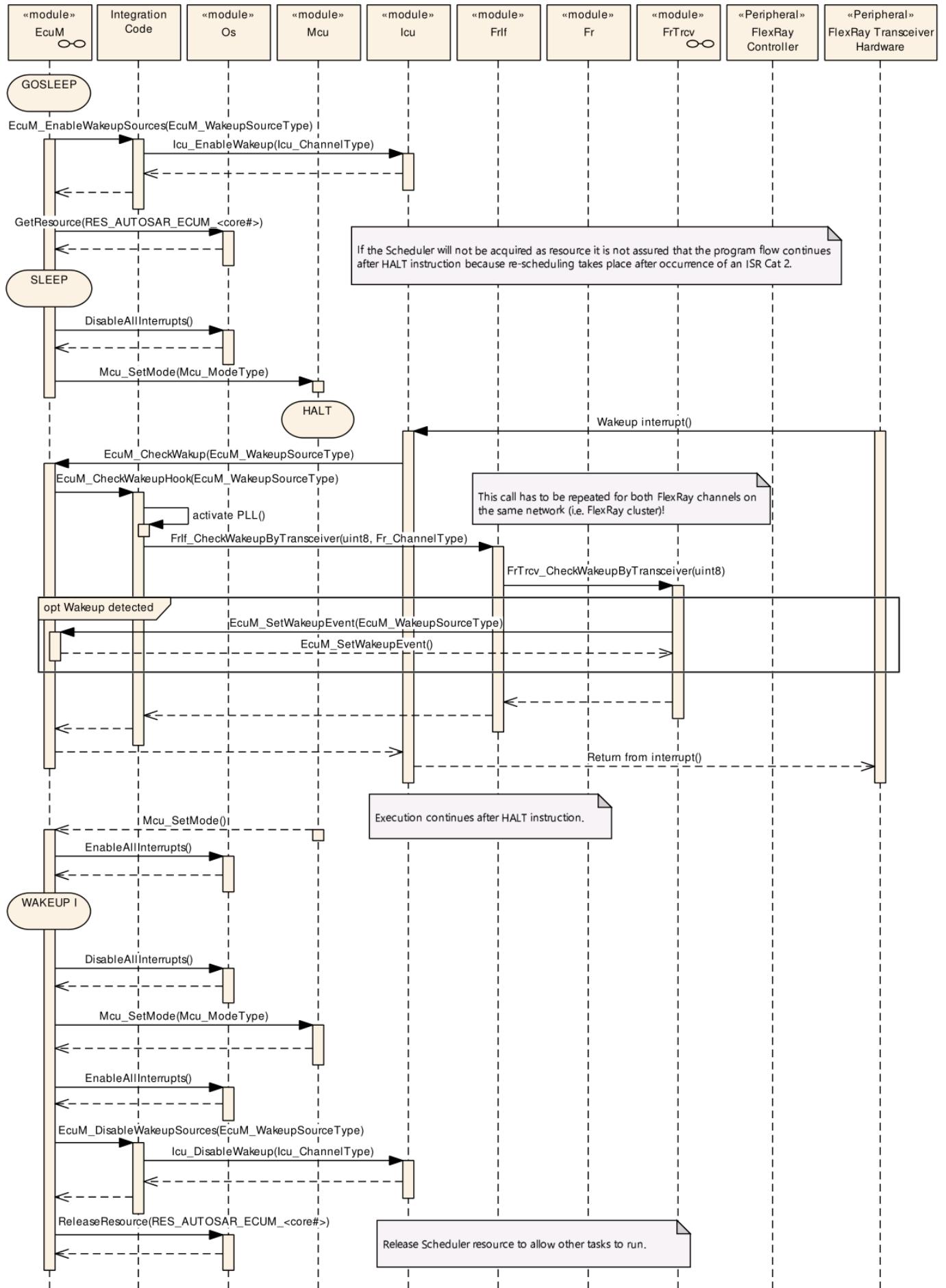


Figure 9.11: FlexRay transceiver wake up by interrupt

注意，在EcuMM\_CheckWakeupHook中需要对FrIf\_WakeupByTransceiver进行两个单独的调用，每个FlexRay通道一个。

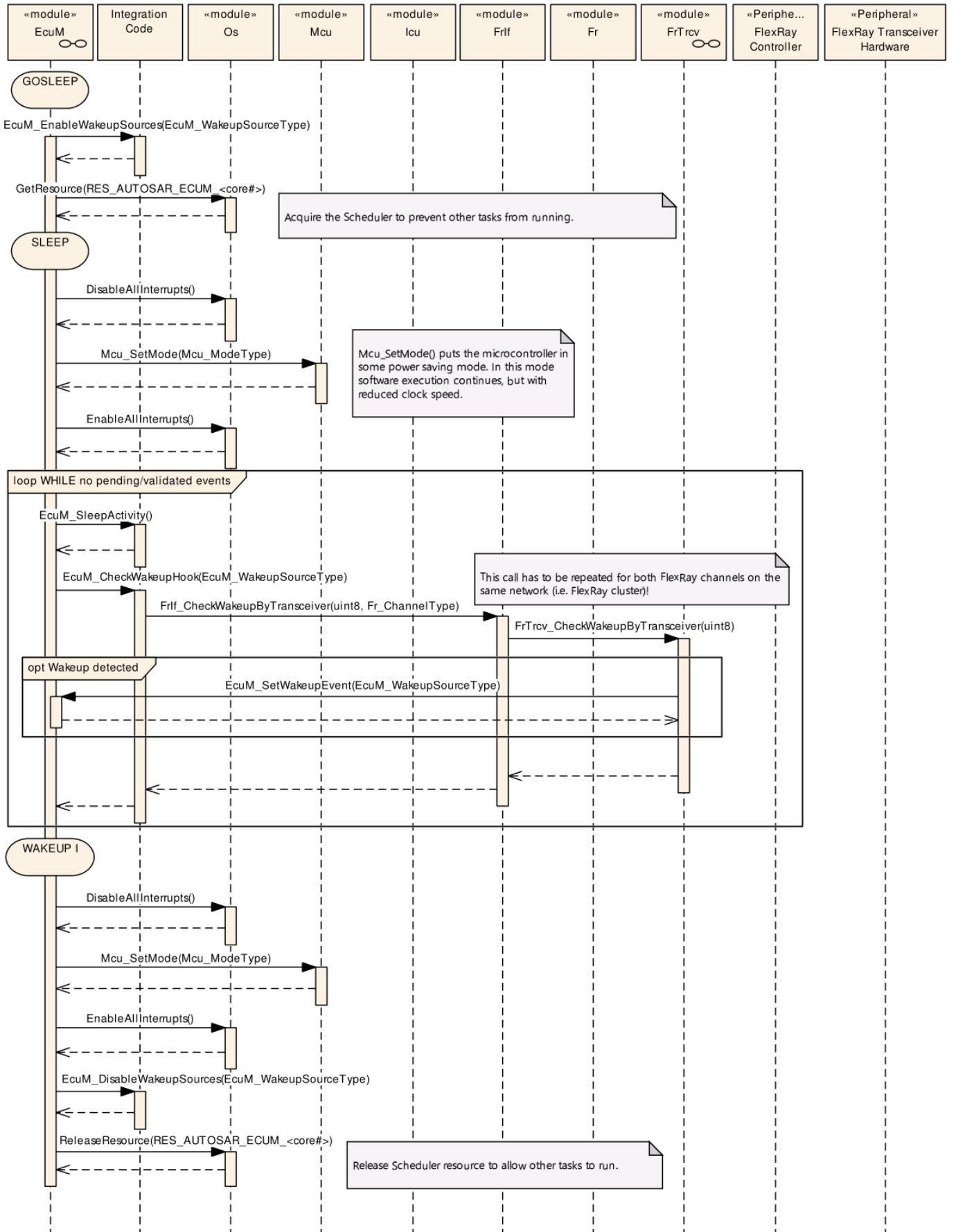


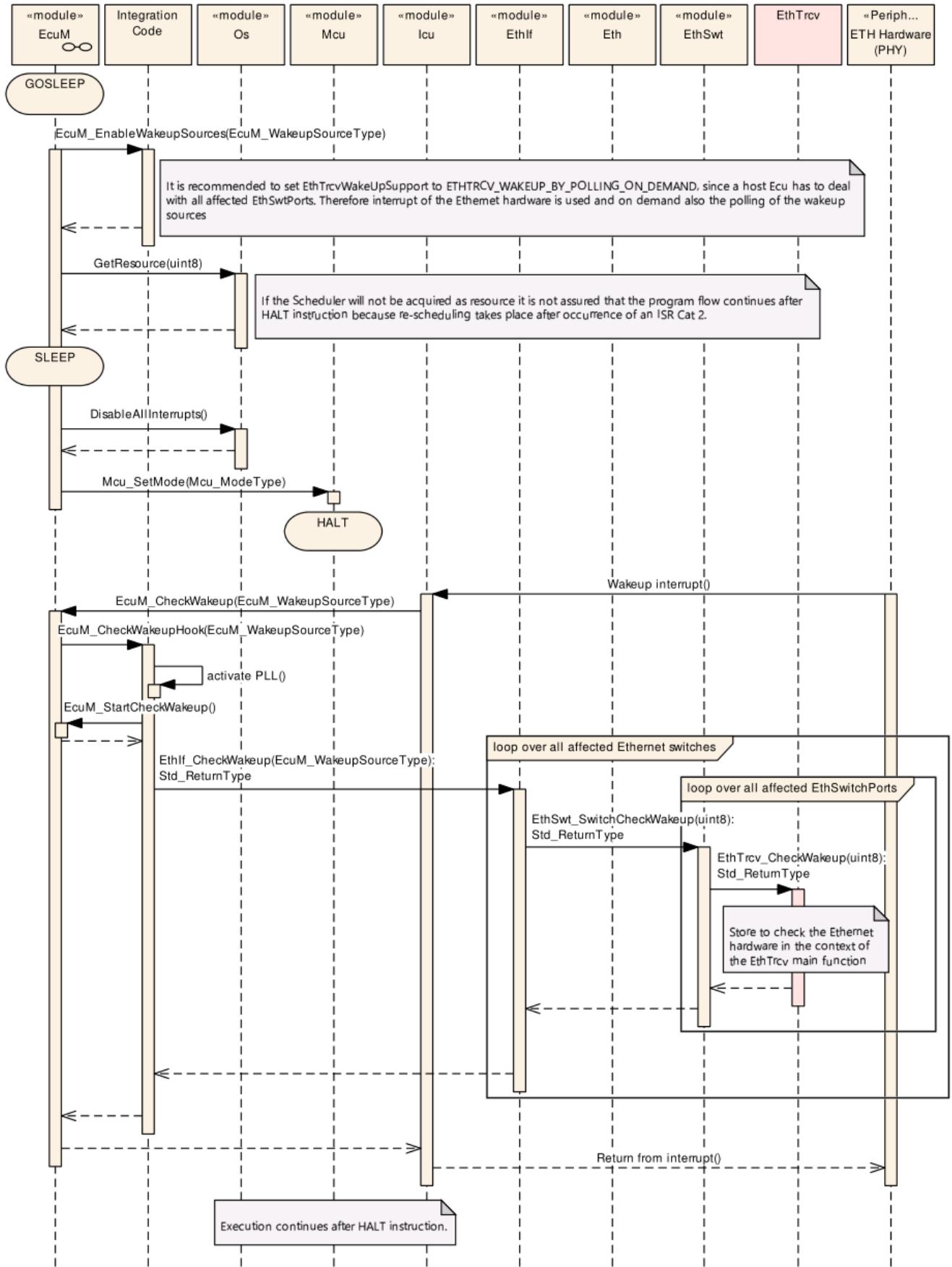
Figure 9.12: FlexRay transceiver wake up by polling

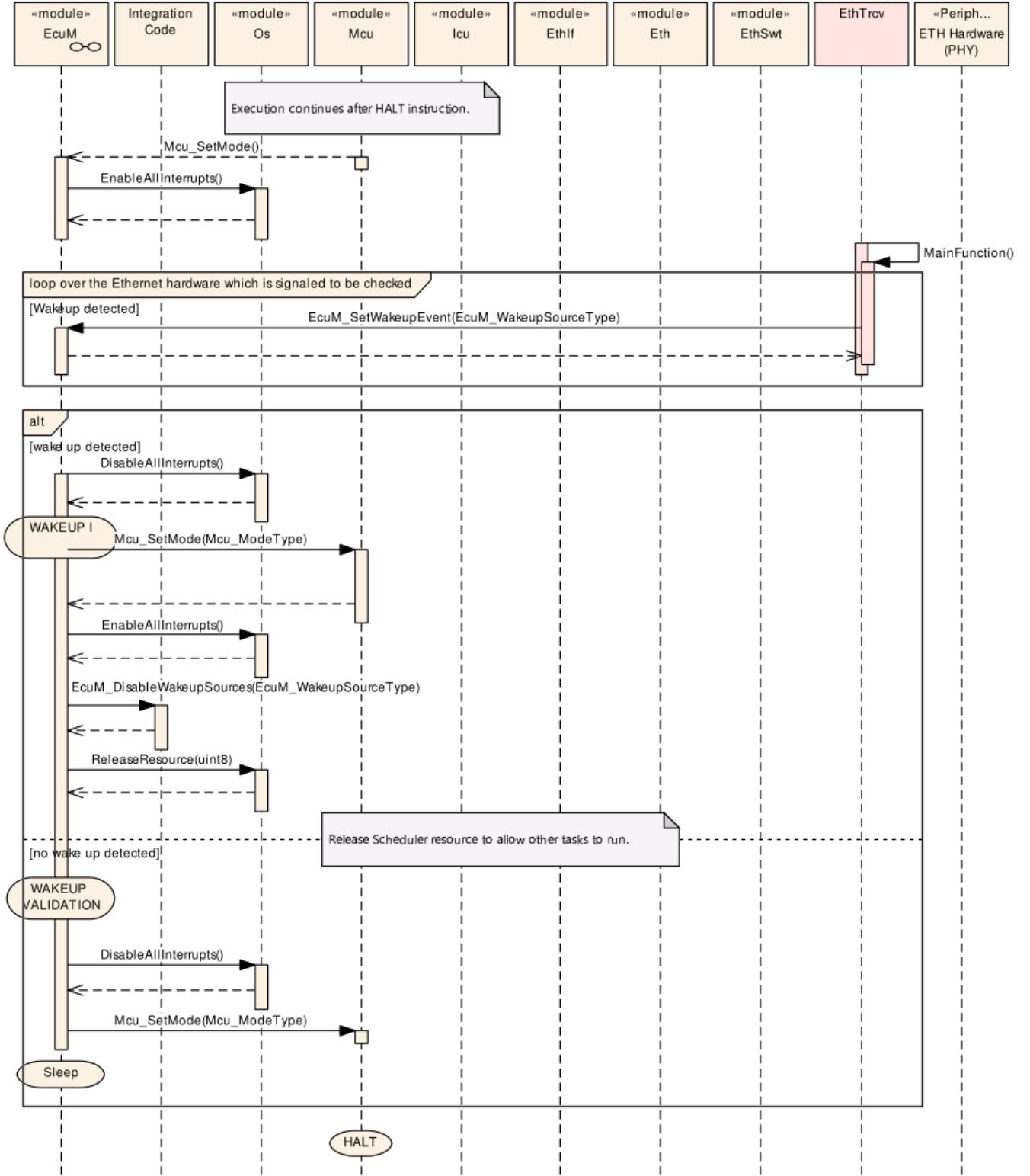
#### ETHERNET WAKEUP SEQUENCE

在具有OATC10兼容以太网硬件的以太网交换网络上，使用的以太网硬件(PHY)可以检测到唤醒。对于维护以太网交换机(主机ECU)的以太网ECU，建议使用按需轮询来检查以太网硬件通知的唤醒。因为检查所有受影响的EthSwtPort可能会花费时间，并且在中断中检查是不可接受的。因此，中断信号表明至少有一个以太网交换机端口检测到唤醒。在中断的上下文中，受影响的EthTrcv在EthTrcv\_MainFunction中被发出异步检查的信号。

每个EthTrcv应该有自己的唤醒源，以区分唤醒到达哪个EthSwtPort。如果EthSwtPort分配给相同的pnc，则可以共享唤醒源。

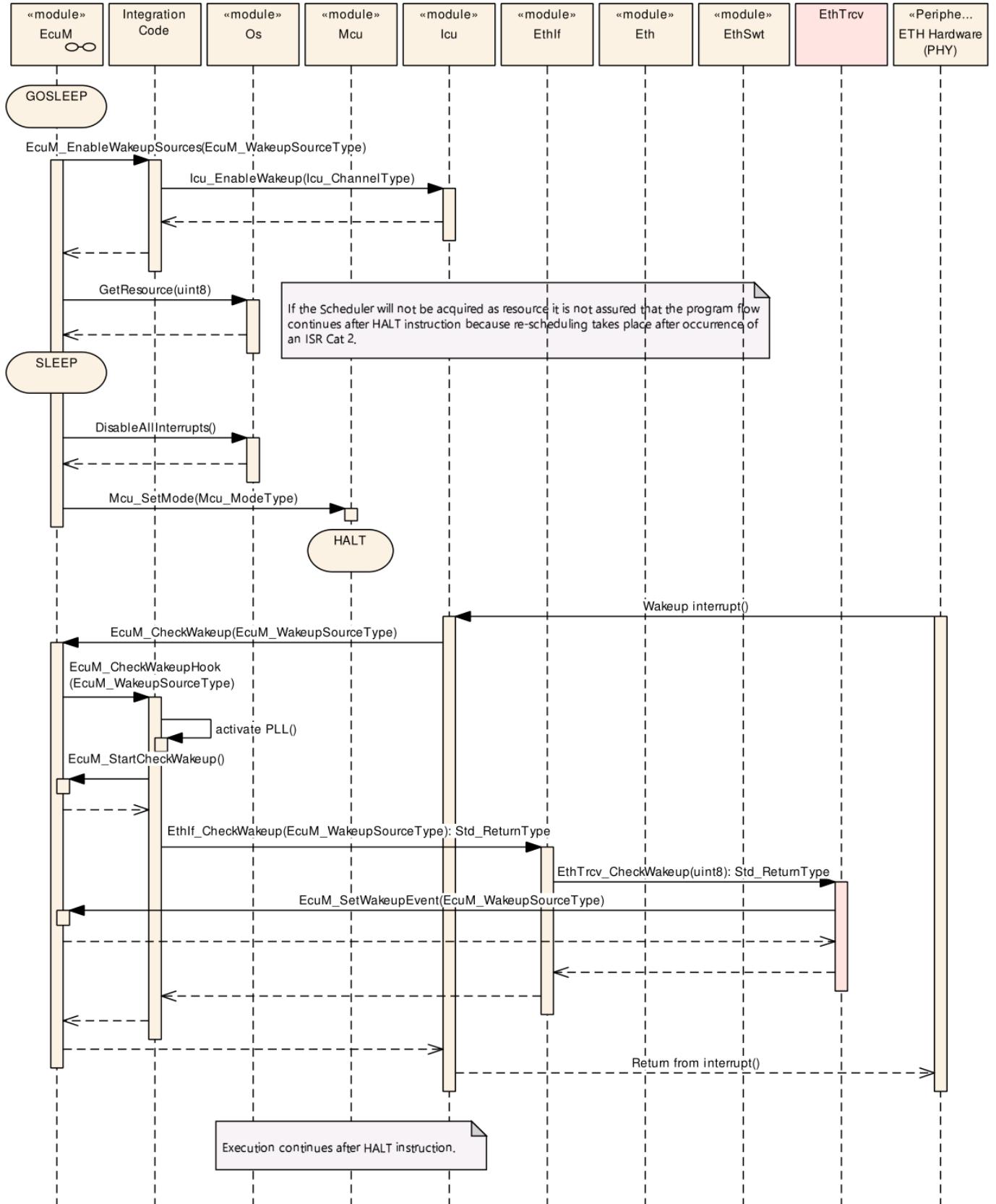
下面的以太网唤醒序列部分是可选的，因为没有“集成代码”的规范。因此，它是特定于实现的，例如在EcuM\_CheckWakeupHook期间调用EthIf来检查唤醒源。

**Figure 9.13: Passive wakeup of a host ecu (ECU that maintain a Ethernet switch) (part 1)**

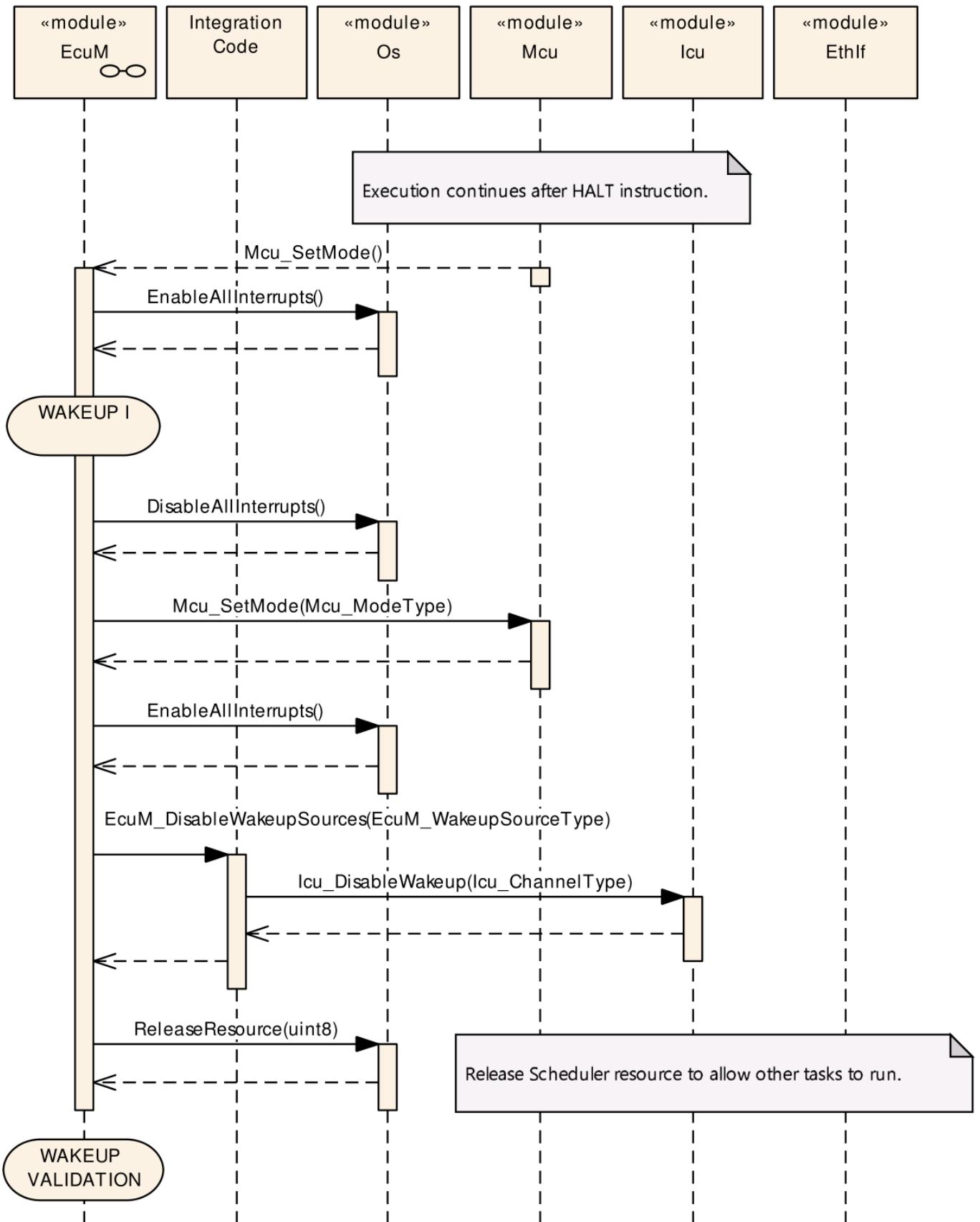


**Figure 9.14: Passive wakeup of a host ecu (ECU that maintain a Ethernet switch) (part 2)**

单个以太网ECU(不维护以太网交换机的ECU)可以选择如何通过中断或轮询来检测唤醒。与主机ECU的不同之处在于，它不需要检查大量的以太网交换机端口。



**Figure 9.15: Passive wakeup of a single ECU (ECU which do not maintain a Ethernet switch) (part 1)**

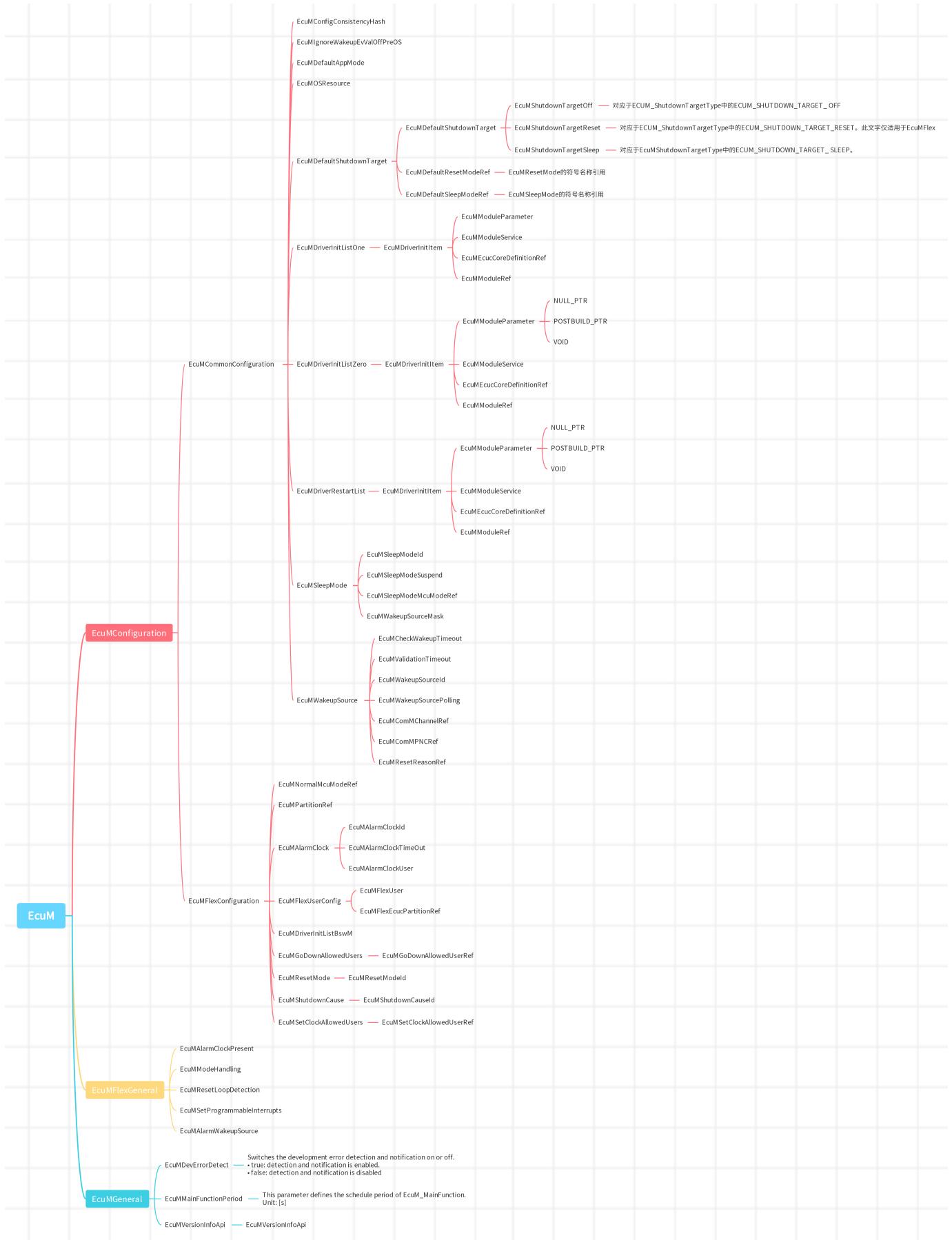


**Figure 9.16: Passive wakeup of a single ECU (ECU which do not maintain a Ethernet switch) (part 2)**

## 2.1.5 Configuration

---

### **Configuration specification**



## 2.2 BswM

### 2.2.1 BswM

#### 概述

BSW 模式管理器是实现驻留在 BSW 中的车辆模式管理和应用模式管理概念部分的模块。其职责是基于简单的规则对来自应用层 SW-C 或其他 BSW 模块的模式请求进行仲裁，并根据仲裁结果执行操作。

#### 特殊名称

术语	描述
BSW	Basic Software
BswM	BSW Mode Manager
BSWMD	Basic Software Module Description
CDD	Complex Drive
Dem	Diagnostic Event Manager

#### 限制

一个分区内最多可以使用一个 BSW 模式管理器实例。

#### 依赖其他模块

BSW 模式管理器具有与 AUTOSAR 架构中的许多 BSW 模块的接口。然而，这些接口中的大多数都是可选的，并根据每个 ECU 的需求使用。

##### RTE

BswM 通过 RTE 接收来自 SW-C 的模式请求。模式切换通知也通过 RTE 传播到 SW-C。

##### ECUM-FLEX

EcuM Flex 可以向 BswM 指示其唤醒源的状态。当使用 ECU 模式处理时，BswM 可以设置 EcuM - Flex 的状态，并根据 RUN 请求协议接收某些模式的状态。

##### COMM

源自通信的模式开关指示通过 BswM 进一步传播到 SW-C。

BswM 可以通过 ComMUsers 在 ComM 上请求通信模式。

##### COM

COM 中 I-PDU 组的处理由 BswM 执行。作为 IPDU 组启动/停止的一部分，可以将包含的信号值重置为其相应的初始化值。

BswM 处理 COM 中信号的截止时间监控的启用和禁用

BswM 还可以触发 I-PDU 的传输。

##### PDUR

BswM 可以在 PDU 路由器中启用和禁用 I-PDU 的路由组。

##### CANSM

源自 CanSM 的模式开关指示通过 BswM 进一步传播到 SW-C。

**LINSM**

BswM 协调 LinSM 中 LIN 计划表的切换，以及 COM 中相应 I-PDU 组的启动和停止。

源自 LinSM 的模式开关指示通过 BswM 进一步传播到 SW-C。

**FRSM**

源自 FrSM 的模式开关指示通过 BswM 进一步传播到 SW-C。

FlexRay 上“单插槽模式”的使用由FrSM根据BswM的请求控制。FlexRay 堆栈的发送功能可以通过调用 API FrSM\_SetEcuPassive由 BswM 通过 FrSM 进行控制。

**ETHSM**

源自 EthSM 的模式开关指示通过 BswM 进一步传播到 SW-C。

**DCM**

DCM 根据收到的诊断请求对 BswM 执行模式请求。

示例：DCM 可以请求“禁用正常通信”。在此模式下，BswM 将关闭相应的 I-PDU 组和 NM PDU。

**J1939DCM**

J1939Dcm将通信状态变化报告给BswM，以便进一步传播到sw - c。BswM通过 J1939Dcm\_SetState 改变J1939Dcm的状态。

**J1939RM**

BswM通过J1939Rm\_SetState改变J1939Rm的状态。

**NM INTERFACE**

BswM将使用Nm\_EnableCommunication和Nm\_DisableCommunication基于当前模式控制Nm通信。

示例：在“禁用正常通信”模式下，BswM需要禁用相应NM信道上的NM通信。

Nm使用BswM\_Nm\_CarWakeUpIndication指示车辆唤醒。

**NvM**

NvM模块通过注册为NvM回调的“集成代码”向BswM报告其块的状态。BswM的操作导致NvM在启动和关闭期间读取和写入所有块。

**OS**

BswM所需的操作系统特性是特定于实现的。

**SD**

BswM通过几个导出的API从Sd接收状态指示（示例见第8.3章）。来自Sd的这些状态指示可以配置为BswMModeRequestSources。

## 2.2.2 Function

### 功能描述

BSW模式管理器基本功能的操作可以描述为两个不同的任务：模式仲裁和模式控制。

模式仲裁部分启动模式切换，由从 SW-C 或其他 BSW 模块收到的模式请求和模式指示的基于规则的仲裁产生

模式控制部分通过执行包含其他BSW模块的模式切换操作的动作列表来执行模式切换。

BswM应该被视为一个模式管理框架模块，其中行为完全由其配置定义。

#### MODE ARBITRATION (仲裁)

BswM 执行的模式仲裁简单且基于规则。用于模式仲裁的规则在 BSW 模式管理器模块的配置中指定。

这些规则由平凡的布尔表达式组成，因此模式仲裁对运行时的影响很小

#### Arbitration Rules

规则是由一组模式请求条件组成的==逻辑表达式==。当输入 模式请求 和 模式指示发生更改时，或在执行 BswM 主函数期间，将评估规则。评估结果（True 或 False）用于决定相应模式控制操作列表的执行。

#### Mode Conditions(条件) and Logical Expressions (表达式) .

组成模式仲裁规则的逻辑表达式可以使用不同的运算符，例如AND、OR、XOR、NOT和NAND。 表达式中的每一项对应于一个模式请求条件。 如果模式条件引用 BswMModeRequestPort，则该条件将验证请求或指示的模式是否等于或不等于特定模式。 如果条件引用 BswMEventRequestPort，则条件将验证请求端口是 SET 还是 CLEAR。 BswMEventRequestPort 事件请求不同于模式请求，因为请求者不向 BswM 发送请求的模式/值，因此，没有模式条件供 BswM 评估。 相反，只有 BswM 评估事件的接收。 当请求者发送/调用事件时，BswMEventRequestPort 将处于 SET 状态。 BswM 随后可以通过执行 BswMClearEventRequest 操作将 BswMEventRequestPort 置于 CLEAR 状态。 图 7.1 显示了具有两个条件的示例规则。 规则和可用逻辑操作集被定义为第 10.2 章中描述的 ECU 配置的一部分。



**Figure 7.1: Pseudocode representation of an example rule with two conditions.**

当 BswMModeCondition 具有 BswMConditionType = BSWM\_EVENT\_IS\_SET 并引用 BswMEventRequestPort 时：

- 如果 BswMEventRequestPort 处于 SET 状态，则 BswMModeCondition 应评估为 TRUE
- 如果 BswMEventRequestPort 处于 CLEAR 状态，则 BswMModeCondition 应评估为 FALSE

当 BswMModeCondition 具有 BswMConditionType = BSWM\_EVENT\_IS\_CLEARED 并引用 BswMEventRequestPort 时：

- 如果 BswMEventRequestPort 处于 SET 状态，则 BswMModeCondition 应评估为 FALSE
- 如果 BswMEventRequestPort 处于 CLEAR 状态，则 BswMModeCondition 应评估为 TRUE

当 BswM 在配置的 BswMEventRequestPort 上接收到事件时（例如，ComM 调用 BswM\_ComM\_InitiateReset），BswMEventRequestPort 应置于 SET 状态。

当在 BswMEventRequestPort 上执行 BswMClearEventRequest 动作时，BswMEventRequestPort 应置于 CLEAR 状态。

**Requirements of Mode Arbitration**

如上所述，BswM 接受模式请求和模式指示作为模式仲裁的输入。模式请求通常源自应用程序 SW-C，但也可能源自其他 BSW 模块，例如 DCM。模式指示总是由其他 BSW 模块发出，例如不同的总线特定状态管理器和 EcuM。在本文档中，通用术语模式仲裁请求对应于模式指示或模式请求。

BswM 应根据传入的模式请求执行模式仲裁。

BswM 应根据传入模式指示执行模式仲裁。

BswM 应根据事件请求以及事件请求的清除执行模式仲裁。



BswM 以相同的方式处理所有模式仲裁请求（请求和指示）。它们通过在 BswMModeRequestSource 配置容器中选择相应的模式条件类型来配置。

BswM 应使用配置的规则执行模式仲裁

模式仲裁规则应使用模块配置参数进行配置。

不允许 BswM 使用先前仲裁规则评估的结果作为逻辑表达式的输入。



禁止使用规则评估的结果作为其他规则评估的输入。它在很大程度上满足于 BswM 配置容器的现有结构，因为逻辑表达式的可配置输入不包括先前规则评估的结果。

作为评估 BswM 仲裁规则的结果而调用的动作只能在动作列表的上下文中调用。

BswM 应根据传入的模式切换通知执行模式仲裁。

**Immediate and Deferred Operation**

有两种不同的方式来安排模式仲裁的处理。它要么在模式请求/指示的上下文中==立即完成==，要么==延迟（循环）==到 BswM 的主要功能。

“立即” 请求在调用者的上下文中执行。系统集成商有责任确保操作列表的执行不会危及系统性能或一致性。

“立即” 请求在调用者的上下文中执行。系统集成商有责任确保操作列表的执行不会危及系统性能或一致性。

应该可以将 BswM 配置为在收到模式仲裁请求后立即执行模式仲裁。这是通过将 BswMRequestProcessing 配置参数（在 BswMModeRequestPort 容器内）设置为 BSWM\_IMMEDIATE 来配置的。

只有使用特定立即模式条件的模式仲裁规则才应由 BswM 在该特定模式请求/指示的上下文中进行评估。

应该（也）可以将模式仲裁推迟到执行 BswM 的主要功能之前。这是通过将 BswMRequestProcessing 配置参数（在 BswMModeRequestPort 容器内）设置为 BSWM\_DEFERRED 来配置的。

应该可以将 BswM 配置为在设置事件时立即执行模式仲裁。这是通过将 BswMEventRequestProcessing 配置参数（在 BswMEventRequestPort 容器内）设置为 BSWM\_IMMEDIATE 来配置的。

应该（也）可以将模式仲裁推迟到执行 BswM 的主要功能之前。这是通过将 BswMEventRequestProcessing 配置参数（在 BswMEventRequestPort 容器内）设置为 BSWM\_DEFERRED 来配置的。

在每次执行 BswM 的主要功能期间，应评估使用至少一个延迟模式条件的所有规则。

BswM 应推迟在其主要功能处理期间收到的模式仲裁请求，直到它完成。任何此类延迟的 IMMEDIATE 请求应在 BswM 主函数退出之前直接处理。任何此类延迟的 DEFERRED 请求应在下一个后续 BswM 主函数中处理。

BswM 应推迟在处理 IMMEDIATE 请求期间收到的模式仲裁请求，直到它完成。任何此类推迟的 IMMEDIATE 请求应在处理原始 IMMEDIATE 请求后直接处理。任何此类推迟的 DEFERRED 请求都应在下一个后续 BswM 主函数中处理。

BswM 实现可以选择使用保护机制（例如独占区域），以保证操作或 BswM 主要功能的执行不会被任何其他任务（例如更高优先级的任务）打断。

端口“更新”的术语说明：任何模式请求端口都有关联的值\状态。更新一个端口意味着改变它的值\状态。

BswM 应在仲裁实际发生之前直接更新立即模式请求端口的值，而不是在模式请求端口被触发时更新。

当模式请求端口被触发时，BswM 将更新 DEFERRED 模式请求端口的值。

#### Arbitration(仲裁) Behavior(行为) after Initialization

初始化后 BswM 模式仲裁的行为由配置容器 BswMModeInitValue 控制。该参数可以为配置中的每个 BswMModeRequestPort 配置一次。

如果容器 BswMModeInitValue 不存在或 ModeRequest 还没有初始值，则 BswM 应将相应的模式条件视为未定义并且不会将其用于模式仲裁，直到相应的模式仲裁请求已被第一次更新。

BswM 应仅仲裁在其逻辑表达式中不包含任何未定义模式条件的规则。

每个BswMModeRequestPort初始化后的初始值可以由配置容器BswMModeInitValue控制。

在定义 BswMModeInitValue 的情况下，BswM 应在 BswM 初始化时使用 BswMBswModeInitValue 或 BswMCompuScaleModeValue 初始化相应的 BswMModeRequestSource。BswM 应拒绝包含单个 BswMModeInitValue 的 BswMBswModeInitValue 和 BswMCompuScaleModeValue 的配置。该初始化值应用于仲裁规则，直到相应的模式仲裁请求已更新，例如 BswM\_RequestMode 的每次调用都应更新 GenericRequest 模式。

#### Note

the Rte and SchM modes always have an intial value

在 BswM 初始化时，所有 BswMEventRequestPort 都应初始化为 CLEAR 状态。

#### MODE CONTROL(控制)

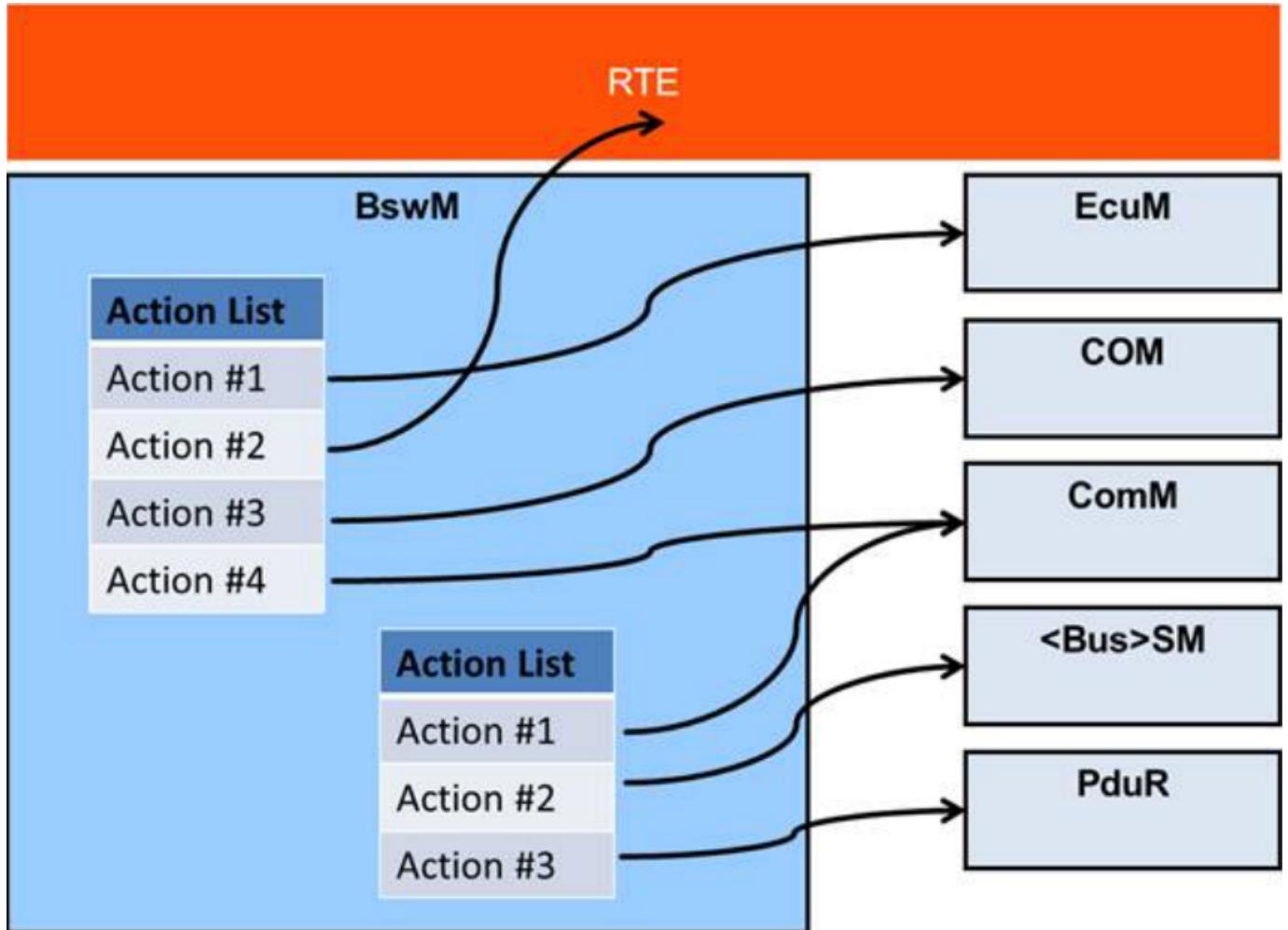
BswM 的模式控制部分根据模式仲裁的结果执行所有必需的操作。这是使用动作列表完成的。动作列表是 BswM 在模式仲裁触发时执行的有序动作列表。

动作列表中的动作可以分为三种类型：

1. 调用其他 BSW 模块或 RTE。
2. 链接到要包含在执行中的==其他操作列表==。
3. 模式仲裁规则。当相应的动作列表被执行时，这些规则将被评估。这样，就得到了规则的层次结构。

BswM 不需要存储或响应任何 BSW 模块特定返回值对其执行的操作。因此，BSW 中的不同状态管理器将它们的当前状态指示给 BswM，以用作模式仲裁的输入。

但是，如果返回错误 (E\_NOT\_OK)，则 BswM 可以发出 Det Runtime Error 和/或取消当前正在执行的操作列表。



**Figure 7.2: Example showing two action lists**

如图7.2所示，BswM可以包含多个Action List，一个Action List可以容纳多个action。为了减少动作列表的总数，应该可以将它们级联起来。动作列表的元素可以是具体动作或对另一个动作列表的引用，或者如上所述，模式仲裁要执行的规则。应该有一个标志连接到每个动作列表条目，说明它的类型（动作/参考/规则）。激活具有具体操作的列表的方式与激活具有引用的列表甚至混合列表的方式之间应该没有区别。

## Mode Processing Cycle

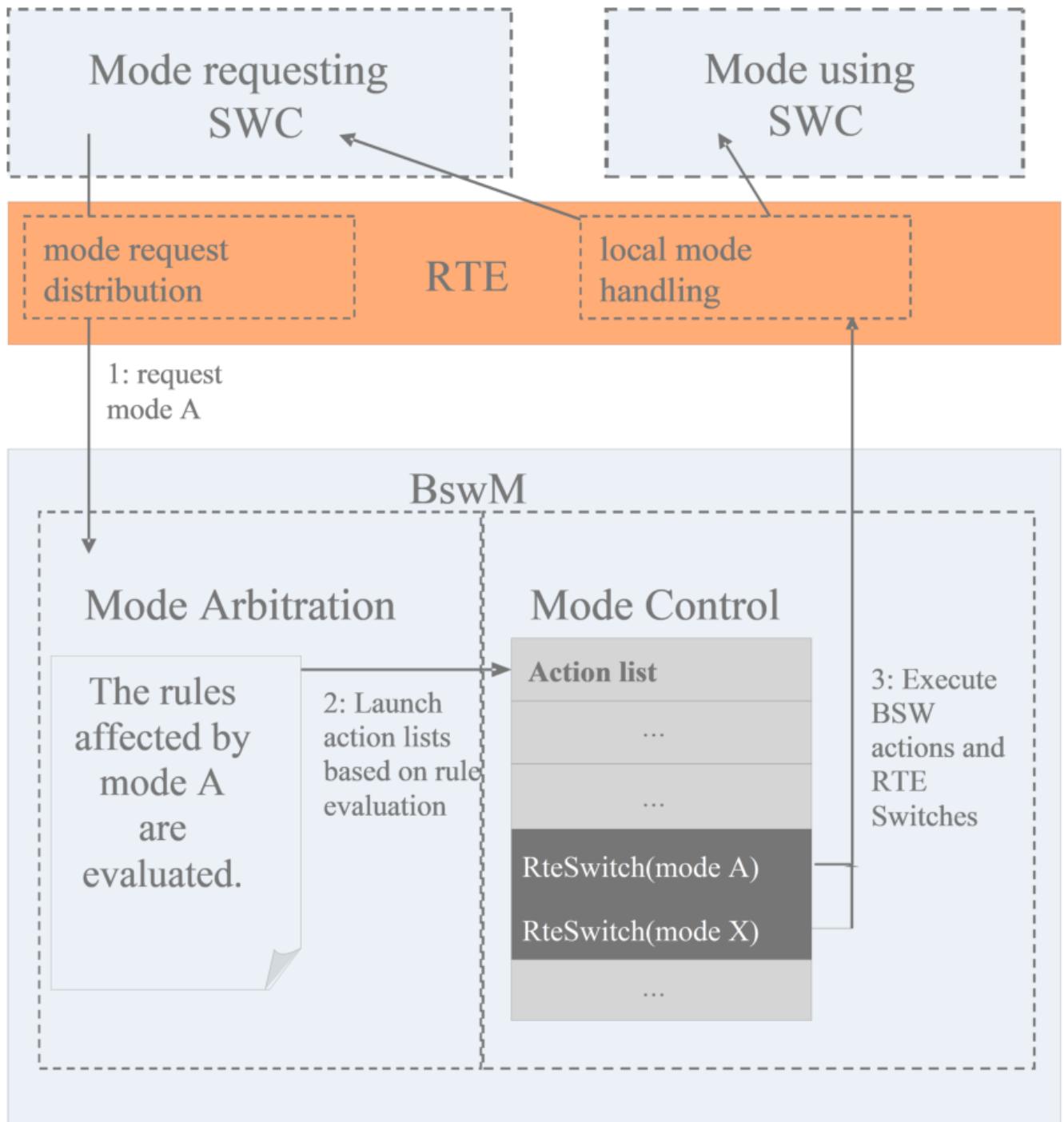


Figure 7.3: Mode Processing Cycle

1. 模式请求器 SW-C 通过其发送端口请求模式 A。RTE 分发请求，BswM 通过其接收端口接收请求。
2. BswM 评估其规则作为接收到的模式仲裁请求的结果，或者在 BswM 主要功能的执行期间循环评估。
3. 根据选择的执行方法执行相应的动作列表（参见“触发和条件动作列表”部分）。
4. 在执行动作列表时，BswM 可能会发出一个或多个对 RTE 开关 API [6] 的调用，作为通知受影响的 SW-C 仲裁结果的动作。任何 SW-C，尤其是模式请求者都可以注册以接收模式切换指示。

**Note**

注意，模式请求者只能从本地 BswM 接收模式切换指示；对于本地代理 SW-C 发出的来自不同 ECU 的请求也是如此。

**Requirements(要求) on Mode Control**

BswM 应通过作为模式仲裁中规则评估结果执行的动作列表来执行模式控制。

对于模式仲裁的每条规则，BswM 应能够根据规则评估为 True 或 False 执行不同的操作列表。

动作列表包含 BswM 应以有序方式执行的一组动作。

动作列表可能包含指向 BswM 应包含在执行中的其他动作列表的链接。

动作列表还可以包括模式仲裁规则的链接，BswM 应在当前动作列表的执行范围内评估这些规则。

如果使用级联动作列表（即使用对其他规则或动作列表的引用），动作列表结构最多可包含七（7）个分层级别。

注意：此限制的目的是使 BswM 实现和生成器工具的测试成为可能。该限制必须由生成器工具检查。

与在模式仲裁请求的上下文中评估的规则关联的动作列表应在模式仲裁触发时立即由 BswM 执行，而不是延迟到主函数执行。

基本原理(Rationale)：这允许在必要时对模式请求进行非常短的延迟。

如果顶层动作列表在模式仲裁期间被多个规则触发，这将导致在模式控制期间执行动作列表的单个触发器。

顶级动作列表是由顶级规则（即不嵌套在动作列表中的规则）直接执行的动作列表，并且不嵌套在另一个动作列表中。[SWS\_BswM\_00223] 仅适用于顶级操作列表。[SWS\_BswM\_00223] 不适用于嵌套规则和嵌套操作列表，因为它们在父操作列表中的顺序是用户定义的，应予以遵守。

如果在模式控制期间要执行多个顶级动作列表，则执行顺序应从==最高==的 BswMActionListPriority 开始，并继续到最低的。在 BswMActionListPriority 相同的情况下，执行顺序是==任意的

对于不是顶级动作列表的动作列表，BswMActionListPriority 将被忽略。

没有 BswMActionListPriority 的动作列表应被解释为具有等于 0 的 BswMActionListPriority

BswM 应拒绝 BswMActionList 包含具有相同值 BswMActionListItemIndexes 的 BswMActionListItems 的配置

当执行 BswMActionList 时：BswM 应从具有最低值 BswMActionListItemIndex 的 BswMActionListItem 开始。随后的 BswMActionListItems 应按其 BswMActionListItemIndex 的递增顺序执行。

在动作列表中，配置的 BswMActionListItemIndexes 不一定需要连续或从零开始。BswM 将开始执行具有最低索引的操作列表项，并继续执行具有最高索引的操作列表项。如果索引有“间隙”（即不连续），这些间隙将被忽略。

**Triggered and Conditional action lists**

有两种方法可以基于规则的评估来执行动作列表。要么在每次使用相应的结果评估规则时执行，要么仅在评估结果与之前的评估发生变化时执行。使用 BswMActionListExecution 参数（在 BswMActionList 容器内）配置动作列表的执行方法。

但是，对于不被规则直接引用的嵌套动作列表，BswMActionListExecution 参数（例如 BSWM\_CONDITION 或 BSWM\_TRIGGER）没有意义，并且不会影响嵌套动作列表的执行方式。每当执行其父动作列表时，相应地执行这样的嵌套动作列表（即不被规则直接引用）。

如果为触发执行配置了 True 动作列表，则 BswM 应仅在相应规则的评估从 False 变为 True 时执行它。

如果为触发执行配置了 False 动作列表，则 BswM 应仅在相应规则的评估从 True 变为 False 时执行它。

如果为条件执行配置了 True 动作列表，则 BswM 应在每次相应规则被评估为 True 时执行它。

如果 False 动作列表被配置为条件执行，BswM 将在每次相应规则被评估为 False 时执行它。

如果动作返回 E\_NOT\_OK 并且相应的 BswMAbortOnFail 配置参数设置为“true”，则 BswM 将中止动作列表的执行。

### Available Actions

可在动作列表中使用的一组动作是预定义的。这样做的原因是简化 ECU 配置和 BswM 配置代码的生成。

BswM 应能够执行由配置容器 BswMAvailableActions 定义的预定义操作。

BswM 应能够调用 AUTOSAR BSW 中的任何函数，即使它不在 BswMAvailableActions 中定义的标准化操作中。

BswM 应能够调用用户定义的函数

用户定义函数的参数及其值应在 ECU 配置时使用 BswMUserCallout 配置容器定义。

### Behavior of Mode Control after Initialization

BswM 初始化后模式控制的行为由 BswMRuleInitState 参数（在 BswMRule 容器内）配置。它定义了“先前的评估结果”，用于在初始化后第一次评估规则后决定执行什么动作列表时使用。配置参数 BswMActionListExecution（在 BswMActionList 容器内）也会影响初始化后的动作列表执行。

BswMRuleInitState	BswMActionListExecution	Rule evaluated to true	Rule evaluated to false
BSWM_UNDEFINED	BSWM_TRIGGER	Execute "true" action list	Execute "false" action list
BSWM_TRUE	BSWM_TRIGGER	Do nothing	Execute "false" action list
BSWM_FALSE	BSWM_TRIGGER	Execute "true" action list	Do nothing
BSWM_UNDEFINED	BSWM_CONDITION	Execute "true" action list	Execute "false" action list
BSWM_TRUE	BSWM_CONDITION	Execute "true" action list	Execute "false" action list
BSWM_FALSE	BSWM_CONDITION	Execute "true" action list	Execute "false" action list

### Waiting Functionality

有时需要延迟特定操作或等待进一步的模式控制。为此，将定时器处理添加到 BswM。

计时器始终由作为 BswMModeRequestSource 的 BswMTimer 和控制此 BswMTimer 的相应操作（请参阅 BswMTimerControl）组成，即计时器只能在操作 BswMTimerControl-> BswMModeRequestSource/BsMTimer 的上下文中控制。BswMTimer 的值（例如 BSWM\_TIMER\_STOPPED、BSWM\_TIMER\_STARTED、BSWM\_TIMER\_EXPIRED）可以通过 BswM 中配置的其他规则进行评估，以触发操作列表。没有外部接口来控制或操纵定时器。

每个 BswMTimer 都应在初始化期间停止（BSWM\_TIMER\_STOPPED）。

动作 BswMTimerAction BSWM\_TIMER\_START 应使用相应的定时器值（参考 BswMTimerValue）重新加载参考的 BswMTimer（通过 BswMTimerRef）并将定时器的模式更改为 BSWM\_TIMER\_STARTED。

#### Note

计时器只能通过 BswMTimerAction 操作重新加载（不可能自动重新加载）

BSWM\_TIMER\_STARTED 模式下的每个 BswMTimer 应在 BSWM\_MainFunction 期间递减计时器（按 BSWM\_MainFunction 的循环时间）。

#### Note

BswMTimer 分辨率是 BswM\_MainFunction 周期的倍数。此外，BswMTimer 的准确性取决于 BswM\_MainFunction 的准确性。

如果处于模式 BSWM\_TIMER\_STARTED 的 BswMTimer 到期，其模式应更改为 BSWM\_TIMER\_EXPIRED，然后 BswMTimer 模式应在同一 BswM\_MainFunction 周期中仲裁。

动作 BswMTimerAction BSWM\_TIMER\_STOP 应立即停止引用的 BswMTimer（通过 BswMTimerRef）并将其模式更改为 BSWM\_TIMER\_STOPPED。

BswM 应忽略与 BswMTimer 关联的 BswMRequestProcessing（例如，IMMEDIATE、DEFERRED）配置。BswM 应始终将 BswMTimer 的处理视为已延迟；BswMTimer 在 BswM 主函数期间被仲裁。

### Note

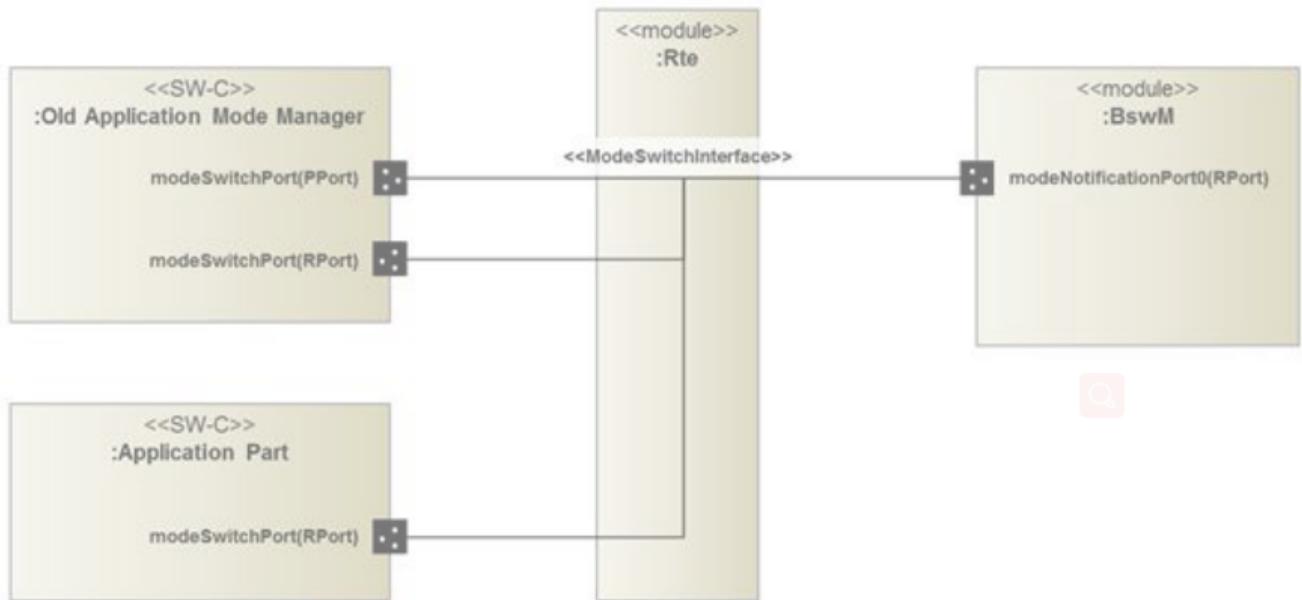
BswM\_TIMER\_EXPIRED 模式下的 BswMTimer 不会被 BswM 自动设置为 BSWM\_TIMER\_STOPPED。 用户需要配置一个动作，以便将 BswMTimer 从 BSWM\_TIMER\_EXPIRED 转换到另一种模式。 如果没有配置将 BswMTimer 转换出 BSWM\_TIMER\_EXPIRED 模式的操作，则 BswMTimer 将在接下来的 BswM 主函数周期中继续被仲裁为 BSWM\_TIMER\_EXPIRED。

### Multi Partition Support

对于多个 BswM 实例，每个 BswM 实例将根据自己的配置集生成自己单独的服务组件描述。 集成商需要将这些单独的服务组件分配给相应的分区。

BswM 存在于==每个分区==中，具有特定于分区的配置（每个分区的 BswMConfig 的==单独实例==）。 包含的动作列表在本地分区执行。

### BswM Interfaces and Ports



**Figure 7.5: Connections between SW-C based Application Mode Manager, Application Parts and the BSW Mode Manager**

### Mode Request Ports

BSW 模式管理器必须使用在 SW-C 的上下文中定义的接口声明一个接收器端口：

```
RequirePort AppModeRequestInterface modeRequestPort_{ArbName}_{ReqName};
```

要读取当前请求的模式，BSW 模式管理器实现必须调用：

```
Rte_Read_modeRequestPort_{ArbName}_{ReqName}_requestedMode( &<variable> );
```

### Mode Switch Ports

与模式请求一样，BSW 模式管理器仅引用在其为模式开关提供端口的相应 SW-C 描述的上下文中定义的模式开关接口。 对于上面的例子，模式开关的声明是：

```
ProvidePort AppModeInterface modeSwitchPort_{ModConName}_{SwitchName};
```

配置参数BswMModeSwitchInterfaceRef引用此模式切换接口。

要切换当前活动模式，BSW 模式管理器实现必须将以下调用之一插入其操作列表：

```
Rte_Switch_modeSwitchPort_{ModConName}_{SwitchName}_currentMode( <new_mode> );
```

```
SchM_Switch_modeSwitchPort_{ModConName}_{SwitchName}_currentMode( <new_mode> );
```

### Notifications of Mode Switches

除了模式请求之外，当前活动的模式也可以用作模式仲裁的输入。对于应用程序和车辆模式，BSW 模式管理员需要注册为模式用户。然后它通过模式切换端口接收有关模式更改的通知。对于上面的例子，模式通知的声明是：

```
RequirePort AppModeInterface modeNotificationPort_{ArbName}_{ModeName};
```

要读取当前活动模式，BSW 模式管理器实现必须调用以下函数之一：

```
Rte_Mode_modeNotificationPort_{ArbName}_{ModeName}_currentMode(&<variable> );
```

```
SchM_Mode_modeNotificationPort_{ArbName}_{ModeName}_currentMode( &<variable> );
```

如果配置了增强型 Rte\_Mode 或 SchM\_Mode，BSW 模式管理器实现必须调用以下函数之一：

```
Rte_Mode_modeNotificationPort_{ArbName}_{ModeName}_currentMode(&<variable>, &<previousmode>, &<nextmode> );
```

```
SchM_Mode_modeNotificationPort_{ArbName}_{ModeName}_currentMode( &<variable>, &<previousmode>, &<nextmode> );
```

## 2.2.3 API

---

### API specification

#### TYPE DEFINITIONS

BswM\_ConfigType

<b>Name</b>	BswM_ConfigType				
<b>Kind</b>	Structure				
<b>Elements</b>	-				
	<b>Type</b>	-			
	<b>Comment</b>	The contents of this structure depends on the configuration variant.			
<b>Description</b>	This structure contains all post-build configurable parameters of the BSW Mode Manager. A pointer to this structure is passed to the BSW Mode Manager initialization function for configuration.				
<b>Available via</b>	BswM.h				

BswM\_ModeType

<b>Name</b>	BswM_ModeType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint16		
<b>Range</b>	0-65535	-	-
<b>Description</b>	This type identifies the modes that can be requested by BswM Users.		
<b>Available via</b>	BswM.h		

BswM\_UserType

<b>Name</b>	BswM_UserType		
<b>Kind</b>	Type		
<b>Derived from</b>	uint16		
<b>Range</b>	0-65535	-	-
<b>Description</b>	This type identifies a BswM User that makes mode requests to the BswM.		
<b>Available via</b>	BswM.h		

## FUNCTION DEFINITIONS

BswM\_BswMPartitionRestarted

<b>Service Name</b>	BswM_BswMPartitionRestarted
<b>Syntax</b>	<pre>void BswM_BswMPartitionRestarted (     void )</pre>
<b>Service ID [hex]</b>	0x1e
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	Function called by Restart Task if the partition containing the BswM has been restarted.
<b>Available via</b>	BswM.h

BswM\_CanSM\_CurrentState

<b>Service Name</b>	BswM_CanSM_CurrentState	
<b>Syntax</b>	<pre>void BswM_CanSM_CurrentState (     NetworkHandleType Network,     CanSM_BswMCurrentStateType CurrentState )</pre>	
<b>Service ID [hex]</b>	0x05	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The CAN channel that the indicated state corresponds to.
	CurrentState	The current state of the CAN channel.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by CanSM to indicate its current state.	
<b>Available via</b>	BswM_CanSM.h	

## BswM\_ComM\_CurrentMode

<b>Service Name</b>	<b>BswM_ComM_CurrentMode</b>	
<b>Syntax</b>	<pre>void BswM_ComM_CurrentMode (     NetworkHandleType Network,     ComM_ModeType RequestedMode )</pre>	
<b>Service ID [hex]</b>	0x0e	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The ComM communication channel that the indicated state corresponds to.
	RequestedMode	The current state of the ComM communication channel
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by ComM to indicate the current communication mode of a ComM channel.	
<b>Available via</b>	BswM_ComM.h	

## BswM\_ComM\_CurrentPNCMode

<b>Service Name</b>	<b>BswM_ComM_CurrentPNCMode</b>	
<b>Syntax</b>	<pre>void BswM_ComM_CurrentPNCMode (     PNCHandleType PNC,     ComM_PncModeType CurrentPncMode )</pre>	
<b>Service ID [hex]</b>	0x15	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	PNC	The handle of the PNC for which the current state is reported.
	CurrentPncMode	The current mode of the PNC.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function is called by ComM to indicate the current mode of the PNC.	
<b>Available via</b>	BswM_ComM.h	

## BswM\_ComM\_InitiateReset

<b>Service Name</b>	BswM_ComM_InitiateReset
<b>Syntax</b>	<pre>void BswM_ComM_InitiateReset (     void )</pre>
<b>Service ID [hex]</b>	0x22
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	Function is called by ComM to signal a shutdown.
<b>Available via</b>	BswM_ComM.h

## BswM\_Dcm\_ApplicationUpdated

<b>Service Name</b>	BswM_Dcm_ApplicationUpdated
<b>Syntax</b>	<pre>void BswM_Dcm_ApplicationUpdated (     void )</pre>
<b>Service ID [hex]</b>	0x14
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	This function is called by the DCM in order to report an updated application.
<b>Available via</b>	BswM_Dcm.h

## BswM\_Dcm\_CommunicationMode\_CurrentState

<b>Service Name</b>	BswM_Dcm_CommunicationMode_CurrentState	
<b>Syntax</b>	<pre>void BswM_Dcm_CommunicationMode_CurrentState (     NetworkHandleType Network,     Dcm_CommunicationModeType RequestedMode )</pre>	
<b>Service ID [hex]</b>	0x06	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The communication channel that the diagnostic mode corresponds to.
	RequestedMode	The requested diagnostic communication mode.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by DCM to inform the BswM about the current state of the communication mode.	
<b>Available via</b>	BswM_Dcm.h	

## BswM\_Deinit

<b>Service Name</b>	BswM_Deinit
<b>Syntax</b>	<pre>void BswM_Deinit (     void )</pre>
<b>Service ID [hex]</b>	0x04
<b>Sync/Async</b>	Synchronous
<b>Reentrancy</b>	Non Reentrant
<b>Parameters (in)</b>	None
<b>Parameters (inout)</b>	None
<b>Parameters (out)</b>	None
<b>Return value</b>	None
<b>Description</b>	Deinitializes the BSW Mode Manager.
<b>Available via</b>	BswM.h

## BswM\_EcuM\_CurrentState

<b>Service Name</b>	BswM_EcuM_CurrentState	
<b>Syntax</b>	<pre>void BswM_EcuM_CurrentState (     EcuM_StateType CurrentState )</pre>	
<b>Service ID [hex]</b>	0x28	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	CurrentState	The requested ECU Operation Mode
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called EcuM to indicate the current ECU Operation Mode.	
<b>Available via</b>	BswM_EcuM.h	

## BswM\_EcuM\_CurrentWakeup

<b>Service Name</b>	BswM_EcuM_CurrentWakeup	
<b>Syntax</b>	<pre>void BswM_EcuM_CurrentWakeup (     EcuM_WakeupSourceType source,     EcuM_WakeupStatusType state )</pre>	
<b>Service ID [hex]</b>	0x10	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	source	Wakeup source(s) that changed state.
	state	The new state of the wakeup source(s)
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by EcuM to indicate the current state of a wakeup source.	
<b>Available via</b>	BswM_EcuM.h	

## BswM\_EcuM\_RequestState

<b>Service Name</b>	<b>BswM_EcuM_RequestState</b>	
<b>Syntax</b>	<pre>void BswM_EcuM_RequestState (     EcuM_StateType State,     EcuM_RunStatusType CurrentState )</pre>	
<b>Service ID [hex]</b>	0x29	
<b>Sync/Async</b>	<b>Synchronous</b>	
<b>Reentrancy</b>	<b>Reentrant</b>	
<b>Parameters (in)</b>	State	The requested state by EcuM
	CurrentState	Result of the Run Request Protocol
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by EcuM notify about current Status of the Run Request Protocol.	
<b>Available via</b>	BswM_EcuM.h	

## BswM\_EthIf\_PortGroupLinkStateChg

<b>Service Name</b>	<b>BswM_EthIf_PortGroupLinkStateChg</b>	
<b>Syntax</b>	<pre>void BswM_EthIf_PortGroupLinkStateChg (     EthIf_SwitchPortGroupIdxType PortGroupIdx,     EthTrcv_LinkstateType PortGroupState )</pre>	
<b>Service ID [hex]</b>	0x26	
<b>Sync/Async</b>	<b>Synchronous</b>	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	PortGroupIdx	The port group index in the context of the Ethernet Interface
	PortGroupState	The state of the port group. State is derived from the physical link of the Ethernet Transceiver:ETHTRCV_LINK_STATE_DOWN==Port group has link down. ETHTRCV_LINK_STATE_ACTIVE==Port group has link up.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by EthIf to indicate the link state change of a certain Ethernet switch port group.	
<b>Available via</b>	BswM_EthIf.h	

## BswM\_EthSM\_CurrentState

<b>Service Name</b>	BswM_EthSM_CurrentState	
<b>Syntax</b>	<pre>void BswM_EthSM_CurrentState (     NetworkHandleType Network,     EthSM_NetworkModeStateType CurrentState )</pre>	
<b>Service ID [hex]</b>	0x0d	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The Ethernet channel that the indicated state corresponds to.
	CurrentState	The current state of the Ethernet channel.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by EthSM to indicate its current state.	
<b>Available via</b>	BswM_EthSM.h	

## BswM\_FrSM\_CurrentState

<b>Service Name</b>	BswM_FrSM_CurrentState	
<b>Syntax</b>	<pre>void BswM_FrSM_CurrentState (     NetworkHandleType Network,     FrSM_BswM_StateType CurrentState )</pre>	
<b>Service ID [hex]</b>	0x0c	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The FlexRay cluster that the indicated state corresponds to.
	CurrentState	The current state of the FlexRay cluster.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by FrSM to indicate its current state.	
<b>Available via</b>	BswM_FrSM.h	

## BswM\_GetVersionInfo

<b>Service Name</b>	<b>BswM_GetVersionInfo</b>	
<b>Syntax</b>	<pre>void BswM_GetversionInfo(     Std_VersionInfoType* VersionInfo )</pre>	
<b>Service ID [hex]</b>	0x01	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	None	
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	VersionInfo	Pointer to where to store the version information of the module.
<b>Return value</b>	None	
<b>Description</b>	Returns the version information of this module.	
<b>Available via</b>	BswM.h	

## BswM\_Init

<b>Service Name</b>	<b>BswM_Init</b>	
<b>Syntax</b>	<pre>void BswM_Init (     const BswM_ConfigType * ConfigPtr )</pre>	
<b>Service ID [hex]</b>	0x00	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Conditionally Reentrant	
<b>Parameters (in)</b>	ConfigPtr	Pointer to post-build configuration data
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Initializes the BSW Mode Manager.	
<b>Available via</b>	BswM.h	

## BswM\_J1939DcmBroadcastStatus

<b>Service Name</b>	BswM_J1939DcmBroadcastStatus	
<b>Syntax</b>	<pre>void BswM_J1939DcmBroadcastStatus (     uint16 NetworkMask )</pre>	
<b>Service ID [hex]</b>	0x1b	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	NetworkMask	Mask containing one bit for each available network. The bit position within this mask corresponds to the ComMChannel.ComMChannelId for the communication channel (so ComMChannelID 0 is represented by bit 0). The meaning for each bit is: 1: Network enabled, 0: Network disabled. Note: only the first 16 communication channel IDs can be supported by this API.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This API tells the BswM the desired communication status of the available networks. The status will typically be activated via COM I-PDU group switches.	
<b>Available via</b>	BswM_J1939Dcm.h	

## BswM\_J1939Nm\_StateChangeNotification

<b>Service Name</b>	BswM_J1939Nm_StateChangeNotification	
<b>Syntax</b>	<pre>void BswM_J1939Nm_StateChangeNotification (     NetworkHandleType Network,     uint8 Node,     Nm_StateType NmState )</pre>	
<b>Service ID [hex]</b>	0x18	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	Identification of the J1939 channel
	Node	Identification of the J1939 node
	NmState	Current (new)state of the J1939 node
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Notification of current J1939Nm state after state changes	
<b>Available via</b>	BswM_J1939Nm.h	

## BswM\_LinSM\_CurrentSchedule

<b>Service Name</b>	BswM_LinSM_CurrentSchedule	
<b>Syntax</b>	<pre>void BswM_LinSM_CurrentSchedule (     NetworkHandleType Network,     LinIf_SchHandleType CurrentSchedule )</pre>	
<b>Service ID [hex]</b>	0x0a	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The LIN channel that the schedule table switch have occurred on.
	CurrentSchedule	The currently active schedule table of the LIN channel.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by LinSM to indicate the currently active schedule table for a specific LIN channel.	
<b>Available via</b>	BswM_LinSM.h	

## BswM\_LinSM\_CurrentState

<b>Service Name</b>	BswM_LinSM_CurrentState	
<b>Syntax</b>	<pre>void BswM_LinSM_CurrentState (     NetworkHandleType Network,     LinSM_ModeType CurrentState )</pre>	
<b>Service ID [hex]</b>	0x09	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The LIN channel that the indicated state corresponds to.
	CurrentState	The current state of the LIN channel.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by LinSM to indicate its current state.	
<b>Available via</b>	BswM_LinSM.h	

## BswM\_LinTp\_RequestMode

<b>Service Name</b>	BswM_LinTp_RequestMode	
<b>Syntax</b>	<pre>void BswM_LinTp_RequestMode (     NetworkHandleType Network,     LinTp_Mode LinTpRequestedMode )</pre>	
<b>Service ID [hex]</b>	0x0b	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	The LIN channel that the LinTp mode request relates to.
	LinTpRequestedMode	The requested LIN TP mode.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by LinTP to request a mode for the corresponding LIN channel. The LinTp_Mode correlates to the LIN schedule table that should be used.	
<b>Available via</b>	BswM_LinTp.h	

## BswM\_Nm\_CarWakeUpIndication

<b>Service Name</b>	BswM_Nm_CarWakeUpIndication	
<b>Syntax</b>	<pre>void BswM_Nm_CarWakeUpIndication (     NetworkHandleType Network )</pre>	
<b>Service ID [hex]</b>	0x24	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	Identification of the Nm-Channel
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by Nm to indicate a CarWakeUp.	
<b>Available via</b>	BswM_Nm.h	

## BswM\_Nm\_StateChangeNotification

<b>Service Name</b>	BswM_Nm_StateChangeNotification	
<b>Syntax</b>	<pre>void BswM_Nm_StateChangeNotification (     NetworkHandleType Network,     Nm_StateType currentState )</pre>	
<b>Service ID [hex]</b>	0x27	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Network	Identification of the Nm-channel
	currentState	Current (new) state of the Nm-channel
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Notification of current Nm state after state changes.	
<b>Available via</b>	BswM_Nm.h	

## BswM\_NvM\_CurrentBlockMode

<b>Service Name</b>	BswM_NvM_CurrentBlockMode	
<b>Syntax</b>	<pre>void BswM_NvM_CurrentBlockMode (     NvM_BlockIdType Block,     NvM_RequestResultType CurrentBlockMode )</pre>	
<b>Service ID [hex]</b>	0x16	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Block	The Block that the new NvM Mode corresponds to.
	CurrentBlockMode	The current block mode of the NvM block.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by NvM to indicate the current block mode of an NvM block.	
<b>Available via</b>	BswM_NvM.h	

## BswM\_NvM\_CurrentJobMode

Service Name	BswM_NvM_CurrentJobMode	
Syntax	<pre>void BswM NvM CurrentJobMode (     NvM MultiBlockRequestType     MultiBlockRequest,     NvM RequestResultType  CurrentJobMode )</pre>	
Service ID [hex]	0x17	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	MultiBlockRequest	Indicates which multi block service this callback refers to
	CurrentJobMode	Current state of the multi block job indicated by parameter MultiBlockRequest
Parameters (inout)	None	
	None	
Return value	None	
	Function called by NvM to inform the BswM about the current state of a multi block job.	
Available via	BswM_NvM.h	

## BswM\_RequestMode

Service Name	BswM_RequestMode	
Syntax	<pre>void BswM_RequestMode (     BswM_UserType requesting_user,     BswM_ModeType requested_mode )</pre>	
Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	requesting_user	The user that requests the mode
	requested_mode	The requested mode.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Generic function call to request modes. This function shall only be used by other BSW modules that does not have a specific mode request interface.	
Available via	BswM.h	

## BswM\_Sd\_ClientServiceCurrentState

<b>Service Name</b>	BswM_Sd_ClientServiceCurrentState	
<b>Syntax</b>	<pre>void BswM_Sd_ClientServiceCurrentState (     uint16 SdClientServiceHandleId,     Sd_ClientServiceCurrentStateType CurrentClientState )</pre>	
<b>Service ID [hex]</b>	0x1f	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	SdClientServiceHandleId	HandleId to identify the ClientService
	CurrentClientState	Current state of the ClientService
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by Service Discovery to indicate current state of the Client Service (available/down).	
<b>Available via</b>	BswM_Sd.h	

## BswM\_Sd\_ConsumedEventGroupCurrentState

<b>Service Name</b>	BswM_Sd_ConsumedEventGroupCurrentState	
<b>Syntax</b>	<pre>void BswM_Sd_ConsumedEventGroupCurrentState (     uint16 SdConsumedEventGroupHandleId,     Sd_ConsumedEventGroupCurrentStateType ConsumedEventGroupState )</pre>	
<b>Service ID [hex]</b>	0x21	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	SdConsumedEventGroup HandleId	HandleId to identify the Consumed Eventgroup
	ConsumedEventGroup State	Status of the Consumed Eventgroup
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Function called by Service Discovery to indicate current status of the Consumed Eventgroup (available/down).	
<b>Available via</b>	BswM_Sd.h	

## BswM\_Sd\_EventHandlerCurrentState

Service Name	BswM_Sd_EventHandlerCurrentState	
Syntax	<pre>void BswM_Sd_EventHandlerCurrentstate (     uint16 SdEventHandlerHandleId,     Sd_EventHandlerCurrentStateType EventHandlerStatus )</pre>	
Service ID [hex]	0x20	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	SdEventHandlerHandleId	HandleId to identify the EventHandler
	EventHandlerStatus	Status of the EventHandler
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Function called by Service Discovery to indicate current status of the EventHandler (requested/released).	
Available via	BswM_Sd.h	

## BswM\_SoAd\_SoConModeChg

Service Name	BswM_SoAd_SoConModeChg	
Syntax	<pre>void BswM_SoAd_SoConModeChg (     SoAd_SoConIdType SoConId,     SoAd_SoConModeType State )</pre>	
Service ID [hex]	0x2a	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different SoConIds. Non reentrant for the same SoConId.	
Parameters (in)	SoConId	The socket connection index.
	State	The state of the SoAd socket connection.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Function called by SoAd to notify state changes of a socket connection.	
Available via	BswM_SoAd.h	

## SCHEDULED FUNCTIONS

## BswM\_MainFunction

<b>Service Name</b>	BswM_MainFunction
<b>Syntax</b>	void BswM_MainFunction ( void )
<b>Service ID [hex]</b>	0x03
<b>Description</b>	Main function of the BswM
<b>Available via</b>	SchM_BswM.h

## SERVICE INTERFACES

## Ports

## BswM\_modeNotificationPort

<b>Name</b>	modeNotificationPort_{ArbName}_{ModeName}
<b>Kind</b>	RequiredPort
<b>Interface-Ref</b>	{ecuc(BswM/BswMConfig/BswMArbitration/BswMMModeRequestPort/BswMMModeRequestSource/BswMSwcModeNotification.BswMSwcModeNotificationModeDeclarationGroupPrototypeRef)}.parent
<b>Description</b>	
<b>Variation</b>	ArbName = {ecuc(BswM/BswMConfig/BswMArbitration.SHORT-NAME)} ModeName = {ecuc(BswM/BswMConfig/BswMArbitration/BswMMModeRequestPort/BswMMModeRequestSource/BswMSwcModeNotification.SHORT-NAME)}

## BswM\_modeRequestPort

<b>Name</b>	modeRequestPort_{ArbName}_{ReqName}
<b>Kind</b>	RequiredPort
<b>Interface-Ref</b>	{ecuc(BswM/BswMConfig/BswMArbitration/BswMMModeRequestPort.BswMMModeRequestSource.BswMSwcModeRequest.BswMSwcModeRequestVariableDataPrototypeRef)}.parent
<b>Description</b>	–
<b>Variation</b>	ArbName = {ecuc(BswM/BswMConfig/BswMArbitration.SHORT-NAME)} ReqName = {ecuc(BswM/BswMConfig/BswMArbitration/BswMMModeRequestPort.SHORT-NAME)}

## BswM\_modeSwitchPort

<b>Name</b>	modeSwitchPort_{ModConName}_{SwitchName}
<b>Kind</b>	ProvidedPort
<b>Interface-Ref</b>	{ecuc(BswM/BswMConfig/BswMMModeControl/BswMSwitchPort.BswMMModeSwitchInterfaceRef)}
<b>Description</b>	–
<b>Variation</b>	{ecuc(BswM/BswMConfig/BswMMModeControl/BswMSwitchPort.BswMMModeSwitchInterfaceRef} != NULL ModConName = {ecuc(BswM/BswMConfig/BswMMModeControl.SHORT-NAME)} SwitchName = {ecuc(BswM/BswMConfig/BswMMModeControl/BswMSwitchPort.SHORT-NAME)}

## API TO REQUEST PORT MAPPINGS

API	Request Port	API/ Config-parameter pairs
BswM_BswMPartitionRestarted	BswMPartitionRestarted	
BswM_CanSM_CurrentState	BswMCanSMIndication	Network / BswMCanSMChan- nelRef
BswM_ComM_CurrentMode	BswMComMIndication	Network / BswMComMChannel- Ref
BswM_ComM_CurrentPNCMode	BswMComMPncRequest	PNC /BswMComMPncRef
BswM_ComM_InitiateReset	BswMComMInitiateReset	
BswM_Dcm_ApplicationUpdated	BswMDcmApplicationUpdated Indication	
BswM_Dcm_CommunicationMode_CurrentState	BswMDcmComModeRequest	Network / BswMDcmComM ChannelRef
BswM_EcuM_CurrentState	BswMEcuMIndication	
BswM_EcuM_CurrentWakeups	BswMEcuMWakeupSource	source / BswMEcuMWakeupSr- cRef
BswM_EcuM_RequestedState	BswMEcuMRUNRequestIndica- tion	State / BswMEcuMRUNRequest ProtocolPort
BswM_Ethlf_PortGroupLinkStateChg	BswMEthlfPortGroupLinkState- Chg	PortGroupIdx/ BswMEthlf SwitchPortGroupRef
BswM_EthSM_CurrentState	BswMEthSMIndication	Network /BswMEthSMChannel- Ref
BswM_FrSM_CurrentState	BswMFrSMIndication	Network / BswMFrSMChannel- Ref
BswM_J1939DcmBroadcastStatus	BswMJ1939DcmBroadcast Status	NetworkMask / BswMJ1939 DcmChannelRef
BswM_J1939Nm_StateChangeNotification	BswMJ1939NmIndication	Network / BswMJ1939NmChan- nelRef,Node / BswMJ1939Nm NodeRef
BswM_LinSM_CurrentSchedule	BswMLinScheduleIndication	Network / BswMLinSMChannel Ref
BswM_LinSM_CurrentState	BswMLinSMIndication	Network / BswMLinSMChannel Ref
BswM_LinTp_RequestMode	BswMLinTpModeRequest	Network / BswMLinTpChannel Ref
BswM_Nm_CarWakeUpIndication	BswMNmCarWakeUpIndication	
BswM_Nm_StateChangeNotification	BswMNmStateChangeNotifica- tion	Network / BswMNmChannelRef
BswM_NvM_CurrentBlockMode	BswMNVMBRequest	Block/ BswMNVMBBlockRef
BswM_NvM_CurrentJobMode	BswMNVMJobModeIndication	MultiBlockRequest / BswMNvm Service
BswM_RequestMode	BswMGenericRequest	requesting user / BswMMode RequesterId
BswM_Sd_ClientServiceCurrentState	BswMSdClientServiceCurrent State	SdClientServiceHandleId BswMSdClientMethodsRef
BswM_Sd_ConsumedEventGroupCurrentState	BswMSdConsumedEventGroup CurrentState	SdConsumedEventGroupHan- dleId / BswMSdConsumedEvent GroupRef
BswM_Sd_EventHandlerCurrentState	BswMSdEventHandlerCurrent State	SdEventHandlerHandleId BswMSdEventHandlerRef

## 2.2.4 Sequence Charts

### Sequence diagrams

#### DEFERRED OPERATION OF BSWM

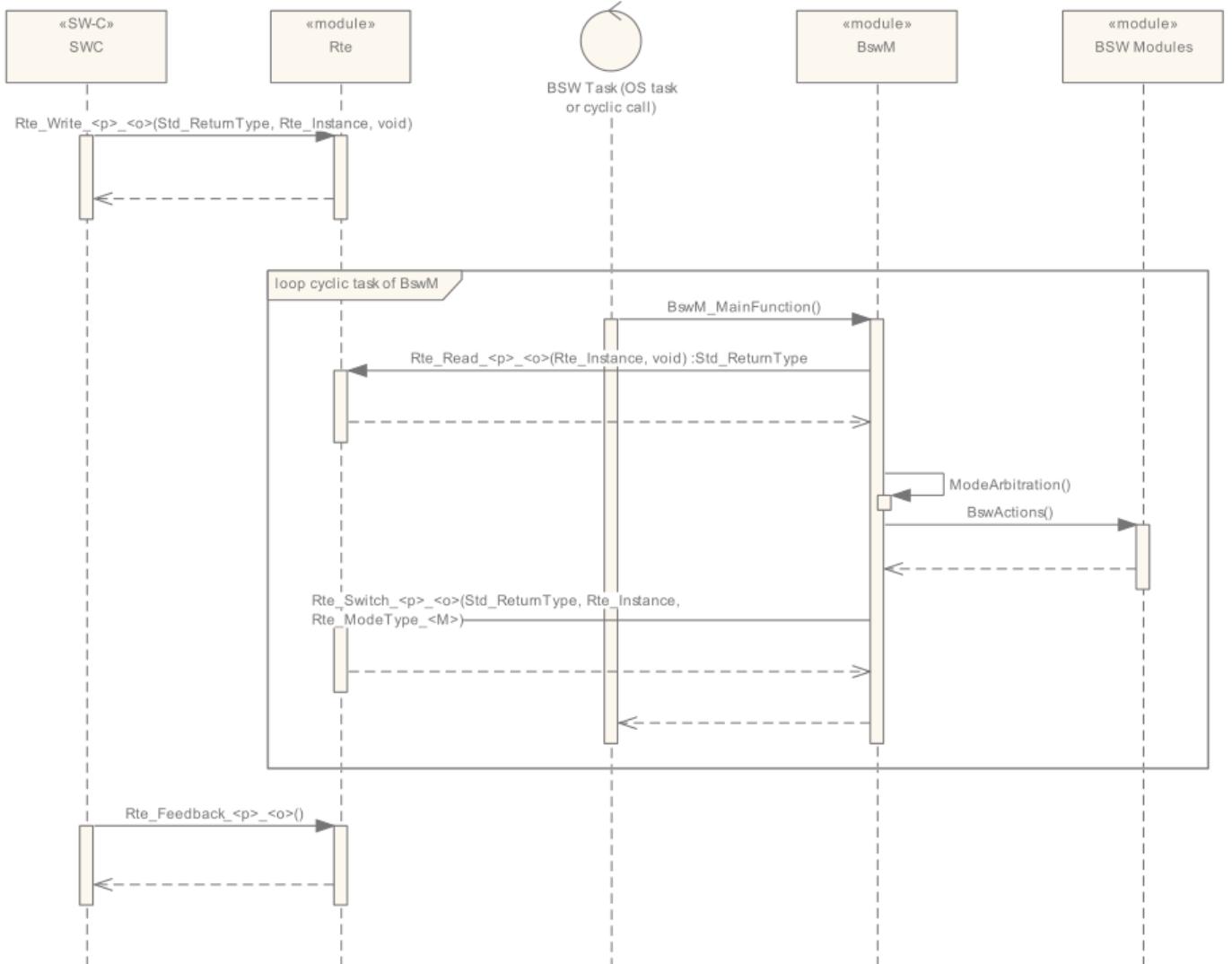


Figure 9.1: Deferred operation of BswM

## IMMEDIATE OPERATION OF BSWM

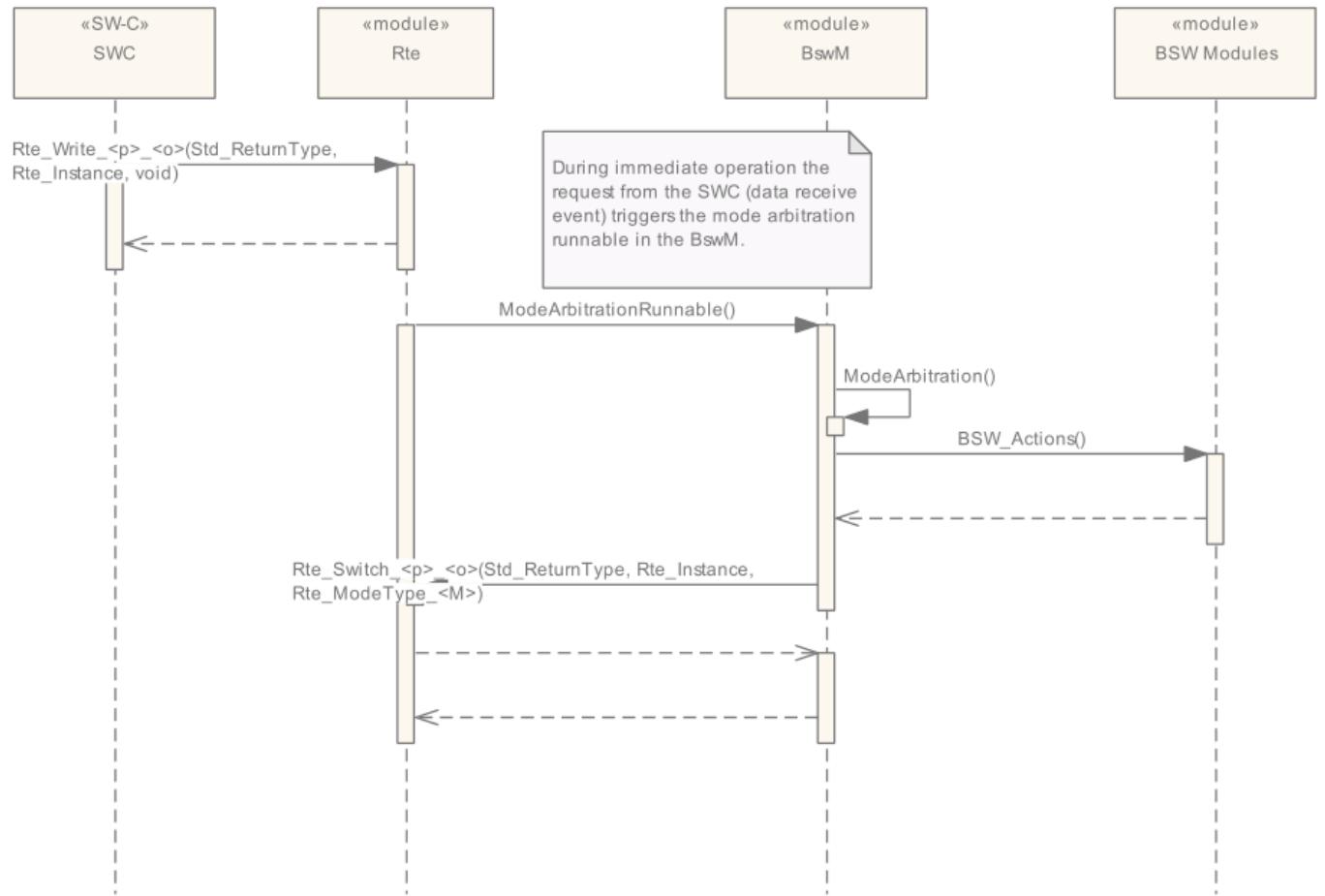
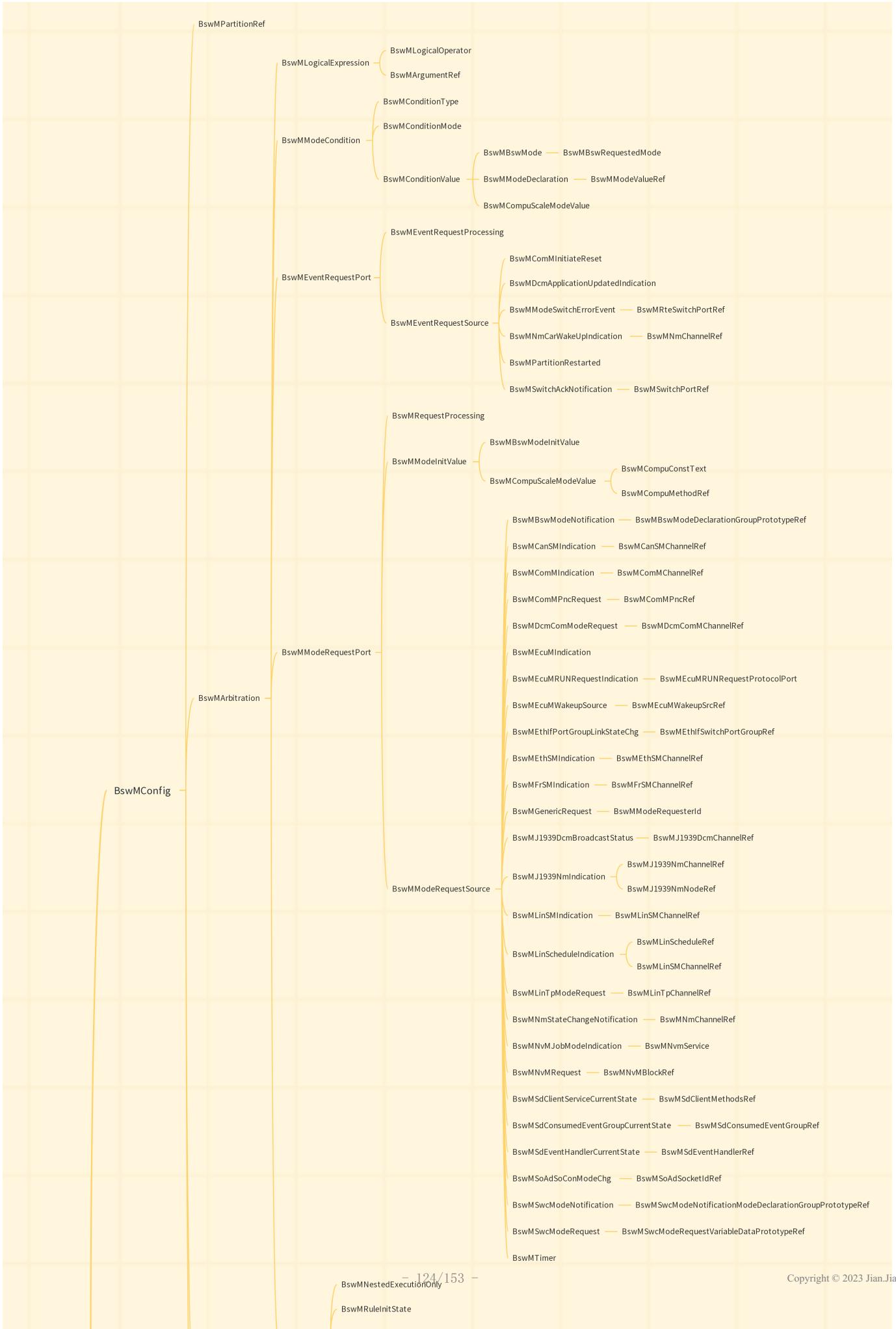


Figure 9.2: Immediate operation of BswM

## 2.2.5 Configuration

---

### Configuration specification



## 3. information safety

### 3.1 SecOC

#### 3.1.1 SecOC

##### 概述

SecOC 模块旨在为 PDU 级别的关键数据提供资源高效且可行的身份验证机制。身份验证机制应与当前的 AUTOSAR 通信系统无缝集成。对资源消耗的影响应尽可能小，以便允许作为遗留系统的附加保护。该规范基于以下假设：主要使用消息身份验证代码（MAC）的对称身份验证方法。它们使用比非对称方法小得多的密钥实现了相同级别的安全性，并且可以在软件和硬件中紧凑高效地实施。但是，该规范提供了必要的抽象级别，以便可以使用对称方法和非对称身份验证方法。

SecOC模块集成在AUTOSAR PduR级别上。图1显示了作为Autosar通信堆栈一部分的SecOC模块的集成。

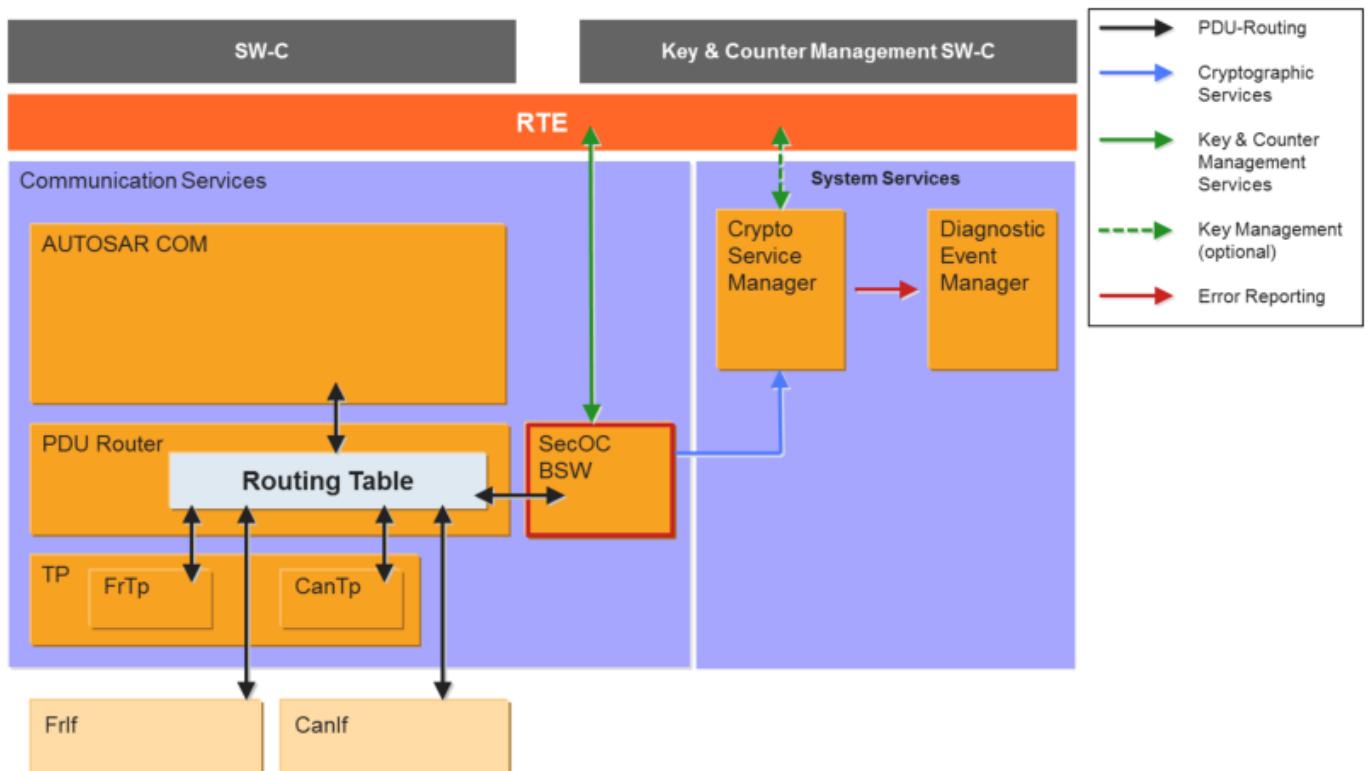


Figure 1: Integration of the SecOC BSW

在此设置中，PduR 负责将传入和传出的安全相关 IPDU 路由到 SecOC 模块。然后，SecOC 模块应添加或处理安全相关信息，并将结果以 I-PDU 的形式传播回 PduR。PduR 然后负责进一步路由 I-PDU。此外，SecOC 模块利用 CSM 提供的加密服务并与 Rte 交互以允许密钥和计数器管理。SecOC 模块应支持 PduR 支持的所有类型的通信范例和原则，尤其是多播通信、传输协议和 PduR 网关。以下部分提供了 SecOC 接口、功能和配置的详细规范。

## 特殊名称

术语	描述
CSM	AUTOSAR 加密服务管理器 (The AUTOSAR Crypto Service Manager)
SecOC	安全车载通信 (Secure Onboard Communication)
MAC	消息验证代码 (Message Authentication Code)
FV	新鲜度值 (Freshness Value)
FM	新鲜度管理 (Freshness Manager)
Authentic I-PDU	真实 I-PDU 是任意 AUTOSAR I-PDU，其内容在网络传输期间通过安全 I-PDU 得到保护。 安全内容包括完整的 I-PDU 或 I-PDU 的一部分。
Authentication	身份验证是一种与身份相关的服务。该功能适用于实体和信息本身。进行沟通的双方应相互确认。通过通道传递的信息应根据来源、来源日期、数据内容、发送时间等进行认证。出于这些原因，密码学的这一方面通常分为两大类：实体认证和数据来源认证。数据源身份验证隐式地提供数据完整性（因为如果消息被修改，则源已更改）。
Authentication Information	认证信息由新鲜度值（或其一部分）和认证者（或其部分）组成。认证信息是 SecOC 为实现安全 I-PDU 而添加的附加信息
Authenticator	Authenticator 是用于提供消息身份验证的数据。通常，术语消息认证码 (MAC) 用于对称方法，而术语签名或数字签名指具有不同财产和约束的非对称方法。
Data integrity	数据完整性是指自数据由授权来源创建、传输或存储以来，数据未以未经授权的方式进行更改的属性。为了确保数据的完整性，应该有能力检测未经授权方的数据操作。数据操作包括插入、删除和替换等操作
Data origin authentication	数据源身份验证是一种身份验证类型，其中一方被确认为过去某个时间（通常未指定）创建的指定数据的（原始）来源。根据定义，数据源身份验证包括数据完整性。
Distinction unilateral/bilateral authentication	在单方认证中，一方证明身份。请求方甚至没有通过证明允许请求身份验证的程度的身份验证。在双边认证中，请求者至少也被认证（见下文）以证明请求的特权。基于上述双边身份验证，有一种有效且更安全的方法可以对两个端点进行身份验证。除了接收方（在第一条消息中）最初请求的身份验证（在第二条消息中），发送方还请求身份验证。接收方发送第三条消息，提供发送方请求的身份验证。这只是三个消息（与四个和两个单方面消息形成对比）。
Entity authentication	实体认证是一个过程，通过该过程，一方（通过获取确凿证据）确信协议中涉及的第二方的身份，并且第二方实际参与了（即，在获取证据时或之前）。
Message authentication	消息认证是一个与数据源认证类似的术语。它提供关于原始消息源的数据源身份验证（和数据完整性，但没有唯一性和及时性保证）。
Secured I-PDU	安全 I-PDU 是 AUTOSAR I-PDU，其包含由附加认证信息补充的真实 I-PDU 的有效载荷。
Transaction authentication	事务认证表示消息认证得到增强，以额外提供数据的唯一性和及时性保证（从而防止无法检测的消息重放）。

## 限制

### 适用于汽车领域

SecOC 模块用于需要安全通信的所有 ECU。

SecOC 模块未指定用于 MOST 和 LIN 通信网络。由于未特别支持 MOST，因此多媒体和远程信息处理汽车领域的适用性可能受到限制。

### SOMEIP T P CONSTRAINTS

SecOC 模块只能用于保护整个 SomeIpTp 消息，不能用于保护 SomeIpTp 消息的各个段。

允许在传输侧执行以下模块顺序：

```
SecOC -> PduR -> SomeIpTp
```

不允许在变速器侧执行以下模块顺序：

```
SomeIpTp -> PduR -> SecOC
```

SecOC无法用于保护SomeIpTp单个消息段的主要原因如下：

- SomeIpTp 需要调用 SomeIpTp\_TriggerTransmit 来创建 SomeIpTp 标头。
- SecOC不支持上层通过TriggerTransmit提供数据。

## 依赖其他模块

### 对PdUR的依赖性

SecOC模块取决于PdUR的API和功能。它提供PDU路由器所需的上层和下层API功能，即

- 通信接口模块的API,
- 传输协议模块的API,
- 使用传输协议模块的上层模块的API,
- 处理来自通信接口模块的I-PDU的上层模块的API。

为了向PdUR提供安全处理的结果，SecOC模块需要PdUR的相应API功能。

### 对CSM的依赖性

SecOC 模块依赖于 CSM 模块在 AUTOSAR 中提供的密码算法。 SecOC 模块需要 API 函数来生成和验证加密签名或消息验证码，即

- MAC生成接口（Csm\_MacGenerate），
- MAC验证接口（Csm\_MacVerify），
- 签名生成接口（Csm\_SignatureGenerate），
- 签名验证接口（Csm\_SignatureVerify），

### RTE的依赖性

SecOC 模块提供了一个具有管理功能的API。此 API 包含以下由 RTE 作为服务接口提供的 API 函数。

- SecOC\_VerificationStatus
- SecOC\_VerifyStatusOverride.
- SecOC\_VerificationStatusIndication

Rte 包括 BSW-Scheduler。 SecOC 模块依赖于 BSWScheduler 在 SecOCMainFunctionPeriodRx 和 SecOCMainFunctionPeriodTx 中配置的周期内调用函数 SecOC\_MainFunctionRx 和 SecOC\_MainFunctionTx。

### 3.1.2 Function

#### 功能描述

敏感数据的身份验证和完整性保护对于保护车辆系统的正确和安全功能是必要的——这确保接收到的数据来自正确的 ECU 并具有正确的值。

SecOC 模块旨在为 PDU 级别的敏感数据提供资源高效且可行的身份验证机制。本规范中提出的方法通常支持使用 对称 和 非对称 方法进行 真实性和完整性 保护。这两种方法大致针对相同的目标，并在概念上表现出主要相似之处，但由于底层原语的不同技术特性，也存在一些差异。另外，Authenticator 常用的术语也不同。通常，术语消息认证码（MAC）用于对称方法，而术语签名或数字签名指的是具有不同属性和约束的非对称方法。

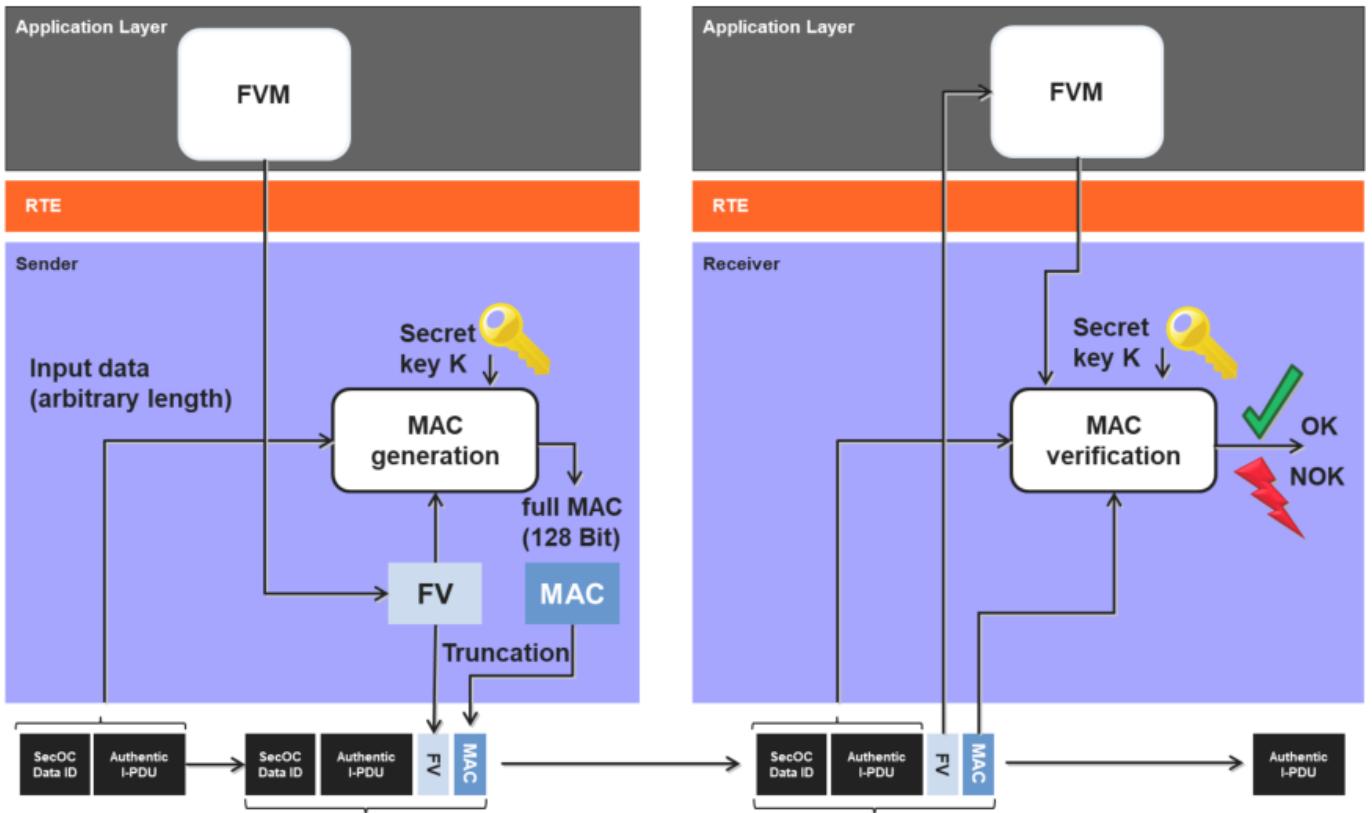
#### 安全解决方案规范

本文档中描述的 SecOC 模块提供了验证车辆架构内 ECU 之间基于 PDU 的通信的 真实性和 新鲜度 所需的功能。该方法要求 发送 ECU 和接收 ECU 都实现一个 SecOC 模块。两个 SecOC 模块都集成在一起，在发送方和接收方提供上层和下层 PduR API。两侧的 SecOC 模块一般都与 PduR 模块进行交互。

为了提供消息新鲜度，发送方和接收方的 SecOC 模块从 外部新鲜度管理器 获取每个 唯一 可识别的安全 I-PDU 的 新鲜度，即每个安全通信链路。

在发送方，SecOC 模块通过向传出的 Authentic(真实的) I-PDU 添加 认证信息 来创建 Secured (安全、可靠) I-PDU。认证信息包括 认证符 (例如消息认证码) 和可选的新鲜度值。无论新鲜度值是否包含在安全 I-PDU 有效载荷中，新鲜度值都会在验证器的生成过程中被考虑。当使用 Freshness (新鲜) Counter 而不是 Timestamp 时，Freshness Manager 应在向接收方提供身份验证信息之前增加 Freshness Counter。

在接收方，SecOC 模块通过验证发送方 SecOC 模块附加的 认证信息 来检查 Authentic I-PDU 的新鲜度和真实性。为了验证 Authentic I-PDU 的真实性和新鲜度，提供给接收端 SecOC 的 Secured I-PDU 应该与发送端 SecOC 提供的 Secured I-PDU 相同，接收端 SecOC 应该知道 Freshness Value 在创建 Authenticator 期间由发送方 SecOC 使用。



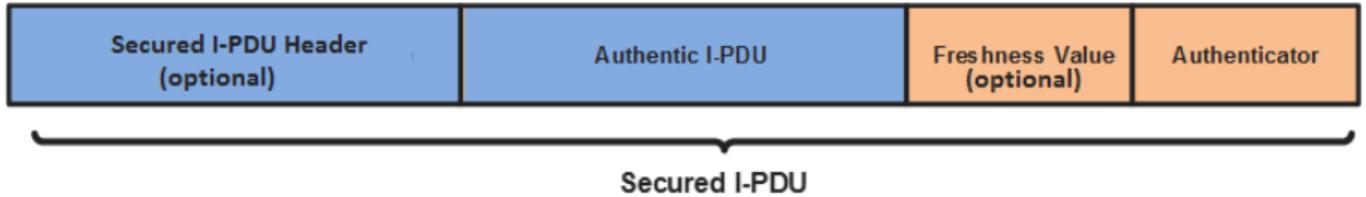
MAC: Message Authentication Code  
FV: Freshness Value  
FVM: FV Manager

Figure 2: Simplified View of Message Authentication and Freshness Verification flow

### 安全解决方案的基本实体

Authentic（真实） I-PDU and Secured（安全） I-PDU

Secured I-PDU 的有效载荷由 Authentic I-PDU 和 Authenticator（例如消息认证码）组成。 安全 I-PDU 的有效载荷可以选择性地包括用于创建验证器的新鲜度值（例如 MAC）。 安全 I-PDU 中内容的结构顺序符合图 3。



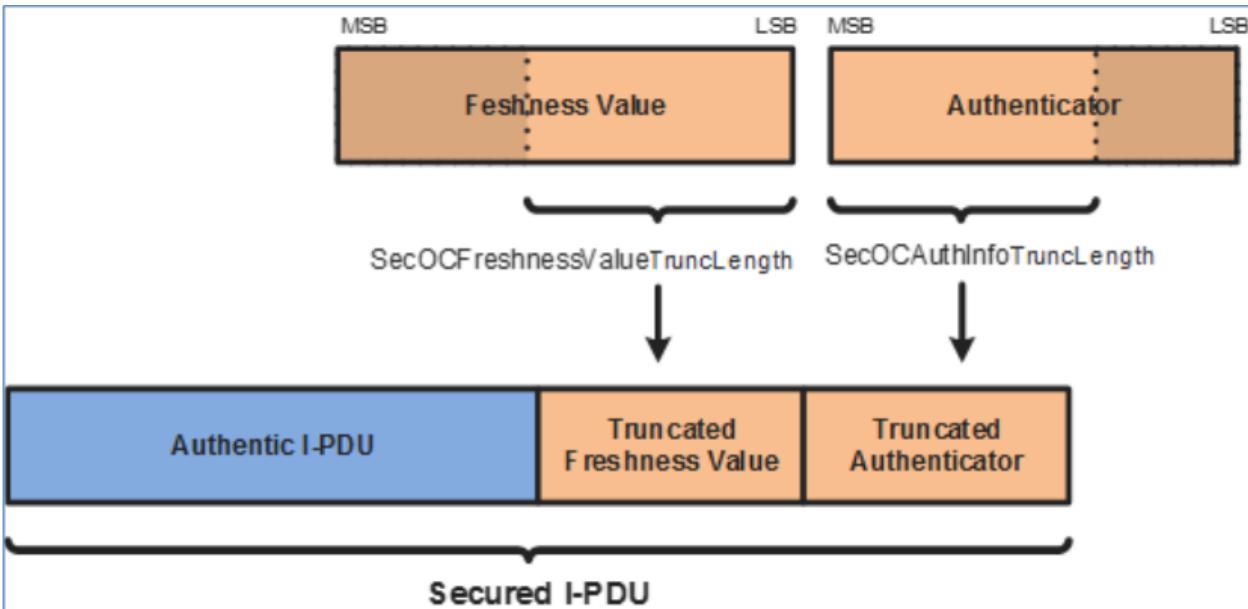
安全I-PDU内的真实I-PDU的长度、新鲜度值和鉴权器可以从一个唯一不可定义的安全I-PDU变化到另一个。

认证器（例如 MAC）是指使用密钥、安全 I-PDU 的 数据标识符 、 真实有效负载 和 新鲜度值 生成的唯一 认证数据字符串 。 Authenticator（认证器）提供高水平的可信度，即 Authentic I-PDU 中的数据是由合法来源生成的，并在预定的时间提供给接收 ECU。

根据用于生成 Authenticator 的身份验证算法（参数 SecOCTxAuthServiceConfigRef 或 SecOCRxAuthServiceConfigRef），可能会截断由身份验证算法生成的结果 Authenticator（例如，在 MAC 的情况下）。 当消息有效载荷的长度有限并且没有足够的空间来包含完整的 Authenticator 时，可能需要截断。

Secured I-PDU 中包含的 Authenticator 长度（参数 SecOCAuthInfoTruncLength）特定于可唯一识别的 Secured I-PDU。 这允许通过为每个安全 I-PDU 提供 MAC 截断长度的细粒度配置来提供跨系统的灵活性（即，两个独立的唯一安全 I-PDU 可能具有不同的验证码长度，包括在安全 I-PDU 的有效载荷中）。

如果截断是可能的，则 Authenticator 应该只被截断到认证算法生成的结果 Authenticator 的最高有效位。 图 5 显示了根据参数 SecOCFreshnessValueTruncLength 和 SecOCAuthInfoTruncLength 截断验证器和新鲜度值的示例。



**Figure 4: An example of Secured I-PDU contents with truncated Freshness Counter and truncated Authenticator (without Secured I-PDU Header)**

#### Note

对于静态参与者的资源约束嵌入用例，我们建议使用消息认证码（MAC）作为认证的基础（例如，基于AES[19]的CMAC[16]，具有足够的密钥长度）。

 Note

在使用 MAC 的情况下，可以仅传输和比较 MAC 的一部分。这称为 MAC 截断。然而，这至少对于伪造单个 MAC 会导致较低的安全级别。虽然我们建议始终使用至少 128 位的密钥长度，但 MAC 截断可能是有益的。当然，必须仔细选择每个用例的 MAC 的实际长度。对于一些指导，我们参考 [16] 的附录 A。一般来说，64 位及以上的 MAC 大小被认为可以提供足够的保护来抵御 NIST 的猜测攻击。根据用例，不同的 MAC 大小可能是合适的，但这需要安全专家仔细判断。

SWS\_SecOC\_00011：直接或间接传输到通信链路另一侧的所有 SecOC 数据（例如，新鲜度值、认证器、数据标识符、SecOC 消息链路数据等）应按照大端字节顺序进行编码，以便每个 SecOC 模块以相同的方式解释数据。

SWS\_SecOC\_00261：安全 I-PDU 报头应以字节表示真实 I-PDU 的长度。标头的长度应可通过参数 SecOCAuthPduHeaderLength 进行配置。

 Note

SecOC 支持在单独的消息（安全 PDU 集合）和安全 I-PDU 报头中结合使用身份验证数据。SecOC 还涵盖动态长度的真实 I-PDU。

## Authenticator(认证器)覆盖的数据

计算 Authenticator 的数据包括安全 I-PDU 的数据标识符（参数 SecOCDataId）、真实 I-PDU 数据和完整新鲜度值。它们分别连接在一起构成位数组，该位数组被传递到身份验证算法中以进行身份验证器生成/验证。

\[ DataToAuthenticator = Data Identifier | secured part of the Authentic I-PDU | Complete Freshness Value \]

 Note

“|”表示串联

## 新鲜度值

每个安全 I-PDU 配置有至少一个新鲜度值。新鲜度值是指用于确保安全 I-PDU 的新鲜度的单调计数器。这种单调计数器可以通过称为 Freshness counter 的单个消息计数器或称为 Freshness Timestamp 的时间戳值来实现。新鲜度值来自新鲜度管理器。

SWS\_SecOC\_00094：如果参数 SecOCFreshnessValueTruncLength 被配置为比实际新鲜度值更小的长度，SecOC 应仅包括新鲜度值的最低有效位直到安全 I-PDU 中的 SecOCFreshnessValueTruncLength。

如果参数 SecOCFreshnessValueTruncLength 配置为 0，则新鲜度值不应包含在受保护的 I-PDU 中。

 Note

包含在经过验证的消息有效负载中的完整新鲜度值的更多位数导致更大的窗口，在该窗口中接收器保持与发送器新鲜度值同步而不执行同步策略。

 Note

当在经过身份验证的消息有效负载中包含部分新鲜度值时，新鲜度值被称为两部分，最高有效位和最低有效位。安全 I-PDU 有效载荷中包含的计数器部分称为新鲜度值的最低有效位，而计数器的其余部分称为新鲜度值的最高有效位。

SWS\_SecOC\_00219：如果 SecOCUseAuthDataFreshness 设置为 TRUE，SecOC 将使用 Authentic I-PDU 的一部分作为新鲜度。在这种情况下，SecOCAuthDataFreshnessStartPosition 确定 Authentic I-PDU 内新鲜度的起始位置（以比特为单位），SecOCAuthDataFreshnessLen 确定其长度（以比特为单位）。

**Note**

这允许重用来自有效负载的现有新鲜度值，这些值保证在新鲜度时间戳的有效期内是唯一的，例如一个4位E2E计数器。在这种情况下，SecOC不需要生成任何额外的计数器值。

**Example**

如果 SecOCUseAuthDataFreshness 设置为 TRUE，SecOCAuthDataFreshnessStartPosition 设置为“11”，SecOCAuthDataFreshLen 设置为“4”，则将提取PDU的以下部分：

Byte index of the PDU	0	1	...
Start bit numbering scheme	7 6 5 4 3 2 1 0	15 14 13 12 11 10 9 8	...

For a PDU “AB CD” (hex), the authentic data freshness would be “1101” (bin).

SWS\_SecOC\_00220: 新鲜度管理器以字节数组的形式在接口函数中提供或接收新鲜度信息。新鲜度始终与数组中第一个字节的MSB对齐。新鲜度的第15位是第2字节的MSB，依此类推。新鲜度数组中未使用的位必须设置为0。相关的长度信息必须以位为单位。

**Example**

10位新鲜度“001101011”(bin)可以位于2字节数组中，并对应于值：“35 80”(十六进制)。长度值为10。

SWS\_SecOC\_00221: 如果对于PDU配置，SecOCQueryFreshnessValue=CFUNC 和 SecOCProvideTxTruncatedFreshnessValue=TRUE，则每当为各个PDU构造 DataToAuthenticator 时，SecOC都会调用接口函数 SecOC\_GetTxFreshnessTruncData。

SWS\_SecOC\_00222: 如果PDU配置的 SecOCQueryFreshnessValue=CFUNC 和 SecOCProvideTxTruncatedFreshnessValue=False，则每当为相应的PDU构造 DataToAuthenticator 时，SecOC都会调用接口函数 SecOC\_GetTxFreshness。

SWS\_SecOC\_00223: 如果对于PDU配置，SecOCQueryFreshnessValue=RTE AND SecOCProvideTxTruncatedFreshness Value=True，则每当为相应的PDU构造 DataToAuthenticator 时，SecOC都会调用服务操作 FreshnessManagement\_GetTxFreshnessTruncData。

SWS\_SecOC\_00224: 如果对于PDU配置，SecOCQueryFreshnessValue=RTE AND SecOCProvideTxTruncatedFreshnessValue=False，则每当为相应的PDU构造 DataToAuthenticator 时，SecOC都会调用服务操作 FreshnessManagement\_GetTxFreshness。

SWS\_SecOC\_00225: 对于排队到SecOC的每个传输请求，应维护认证构建计数器。

SWS\_SecOC\_00226: 在初始处理安全 I-PDU 的传输请求时，SecOC 应将认证构建计数器设置为 0。

SWS\_SecOC\_00227: 如果新鲜度函数的查询（例如 SecOC\_GetTxFreshness()）返回 E\_BUSY 或验证器的计算（例如 Csm\_MacGenerate()）返回 E\_BUSY、QUEUE\_FULL 或任何其他可恢复的错误，则验证构建计数器应递增。

**Note**

返回值 E\_NOT\_OK 不被视为可恢复错误。

SWS\_SecOC\_00228: 如果构建身份验证失败，且身份验证构建计数器尚未达到配置值 SecOCAuthenticationBuildAttempts，则应在下次调用Tx主函数时重试新鲜度尝试和身份验证器计算。

SWS\_SecOC\_00229: 如果认证建立计数器已经达到配置值 SecOCAuthenticationBuildAttempts，或者新鲜度函数的查询返回 E\_NOT\_OK，或者认证器的计算返回不可恢复的错误，例如返回 E\_NOT\_OK 或 KEY\_FAILURE，SecOC模块将使用 SecOCDefaultAuthenticationInformationPattern 来处理所有的如果发送 SecOCDefaultAuthenticationInformationPattern 由服务 SecOC\_SendDefaultAuthenticationInformation 启用，则新鲜度值和 Authenticator 字节用于构建身份验证信息。如果未启用发送 SecOCDefaultAuthenticationInformationPattern，SecOC 模块应从其内部缓冲区中删除 Authentic I-PDU 并取消传输请求。

### Example

SecOCFreshnessValueTxLength=4位 SecOCAuthInfoTxLength=20位 SecOCDefaultAuthenticatorValue=0xA5 安全PDU中生成的默认身份验证信息将是0x05（截短的新鲜度值）|0xA5 0xA5 0xA0（截短的身份验证器）。“|”表示串联。

SWS\_SecOC\_00230: 如果PDU配置的 SecOCQueryFreshnessValue=CFUNC 和 SecOCProvideTxTruncatedFreshnessValue=TRUE , SecOC将调用名为 SecOC\_GetTxFreshnessTruncData 的函数, 以获取TX消息的当前新鲜度。

SWS\_SecOC\_00231: 如果PDU配置的SecOCQueryFreshnessValue=CFUNC和SecOCProvideTxTruncatedFreshness Value=FALSE, SecOC将调用名为 SecOC\_GetTxFreshness的函数, 以获取TX消息的当前新鲜度。

SWS\_SecOC\_00232: 如果对于PDU配置, SecOCQueryFreshnessValue=CFUNC , SecOC调用具有 sws\_SecOC\_91005 中描述的签名的函数, 以指示已成功启动安全I-PDU进行传输。

### Note

消息出现在总线上后, 不会调用此函数。在成功向PduR发送请求之后, 调用该函数被认为是更安全的。

SWS\_SecOC\_00233: 如果 SecOCQueryFreshnessValue = RTE 对于 PDU 配置, SecOC 调用服务操作 FreshnessManagement\_SPduTxConfirmation 以指示安全 I-PDU 已成功启动传输。

SWS\_SecOC\_00234: 对于SecOC内的每个已处理的安全I-PDU, 应维护认证构建计数器和认证验证尝试计数器。

SWS\_SecOC\_00235: 在初始处理接收到的安全I-PDU时, 认证构建计数器和认证验证尝试计数器应设置为0。

SWS\_SecOC\_00236: 如果新鲜度函数的查询 (例如 SecOC\_GetRxFreshness() ) 返回 E\_BUSY, 则认证构建计数器应递增, 并且不应执行任何认证验证尝试。

SWS\_SecOC\_00237: 如果验证器的验证 (例如 csm\_MacVerify() ) 返回 E\_BUSY 、 QUEUE\_FULL 或任何其他可恢复的错误, 则验证构建计数器应递增。

### Note

返回值 E\_NOT\_OK 不被视为可恢复错误。

SWS\_SecOC\_00238: 如果身份验证构建尝试失败, 且身份验证构建计数器尚未达到配置值 SecOCAuthenticationBuildAttempts , 则应在下次调用Rx主函数时重试新鲜度尝试和身份验证。

SWS\_SecOC\_00239: 如果认证器的验证可以成功执行但验证失败 (例如 MAC 验证失败或密钥无效) , 则应增加认证验证尝试计数器并将认证构建计数器设置为 0。

 Note

重置身份验证构建计数器应防止过早中断身份验证过程，即使仍然可以进行身份验证验证尝试。

SWS\_SecOC\_00240: 如果认证构建计数器已达到配置值 `SecOCAuthenticationBuildAttempts`，SecOC模块应从其内部缓冲区中删除Authentic I-PDU，并丢弃接收到的消息。`VerificationResultType` 应设置为 `SECOC_AUTHENTICATIONBUILDFAILURE`。

如果使用 `SecOC_VerifyStatusOverride`，则根据 `overrideStatus` 值处理验证结果和I-PDU。

SWS\_SecOC\_00256: 如果新鲜度函数的查询返回 `E_NOT_OK`，SecOC模块应从其内部缓冲区中删除Authentic I-PDU，并丢弃接收到的消息。`VerificationResultType` 应设置为 `SECOC_FRESHNESSFAILURE`。

SWS\_SecOC\_00241: 如果认证验证尝试计数器已达到配置值 `SecOCAuthenticationVerifyAttempts`，或者认证器的验证返回了不可恢复的错误，例如返回 `E_NOT_OK` 或 `KEY_FAILURE`，SecOC模块应从其内部缓冲区中删除Authentic I-PDU，并应丢弃接收到的消息。`VerificationResultType` 应设置为 `SECOC_VERIFICATIONFAILURE`。

如果使用 `SecOC_VerifyStatusOverride`，则根据 `overrideStatus` 值处理验证结果和I-PDU。

SWS\_SecOC\_00242: 如果认证器验证成功，则 `VerificationResultType` 应设置为 `SECOC_VERIFICATIONSUCCESS`。

SWS\_SecOC\_00243: 新鲜度管理应使用验证状态调用功能（SWS\_SECCO\_00119）获取安全I-PDU的验证结果。此通知可以用作同步其他新鲜度尝试的示例，也可以用于计数器增量。

 Note

SecOC 允许覆盖状态（参见 SWS\_SECOC\_00142）。因此，如果新鲜度管理依赖状态标注，同时使用状态覆盖功能，则必须小心。这会导致新鲜度管理中的冲突，并可能导致不正确的 freshness 值。

SWS\_SecOC\_00244: 如果 PDU 配置的 `SecOCQueryFreshnessValue = RTE` AND `SecOCUseAuthDataFreshness = TRUE` 并且安全的 PDU 被完全接收，SecOC 调用 Rte 服务 `FreshnessManagement_GetRxFreshnessAuthData` 来查询当前新鲜度。根据配置 `SecOCAuthDataFreshnessStartPosition` 和 `SecOCAuthDataFreshnessLen` 的配置，接收到的 PDU 数据的一部分被传递到此服务操作。

SWS\_SecOC\_00245: 如果 PDU 配置的 `SecOCQueryFreshnessValue = RTE` AND `SecOCUseAuthDataFreshness = FALSE` 并且安全的 PDU 被完全接收，SecOC 调用 Rte 服务 `FreshnessManagement_GetRxFreshness` 来查询当前新鲜度。

SWS\_SecOC\_00246: 如果对于PDU配置 `SecOCQueryFreshnessValue = CFUNC` AND `secOCUseAuthDataFreshness = TRUE`，并且安全PDU被完全接收，SecOC调用接口函数 `SecOC_GetRxFreshnessAuthData` 查询当前新鲜度。根据配置 `SecOCAuthDataFreshnessStartPosition` 和 `SecOCAuthDataFreshnessLen` 的配置，接收到的 PDU 数据的一部分将传递给此函数。

SWS\_SecOC\_00247: 如果对于PDU配置，`SecOCQueryFreshnessValue=CFUNC` AND `secOCUseAuthDataFreshness=False`，并且已完全接收到受保护的 PDU，则SecOC调用接口函数 `SecOC_GetRxFreshness` 以查询当前的新鲜度。

SWS\_SecOC\_00248: 如果 Rx 新鲜度请求函数返回 `E_NOT_OK`，则 Authentic I-PDU 的验证被认为失败，并且该 PDU 的验证重试计数器应增加。如果认证尝试的次数已经达到 `SecOCAuthenticationVerifyAttempts`，SecOC 模块应该从它的内部缓冲区中移除 Authentic I-PDU。故障 `SECOC_E_FRESHNESS_FAILURE` 应报告给 DET 模块。

SWS\_SecOC\_00249: 如果对于PDU配置，`SecOCQueryFreshnessValue=CFUNC` 和 `SecOCUseAuthDataFreshness=True`，SecOC将查询名为 `SecOC_GetRxFreshnessAuthData` 的函数，以获取RX消息的当前新鲜度。

SWS\_SecOC\_00250: 如果PDU配置的 `SecOCQueryFreshnessValue=CFUNC` 和 `SecOCUseAuthDataFreshness=False`，SecOC将查询名为 `SecOC_GetRxFreshness` 的函数，以获取RX消息的当前新鲜度。

### I-PDU的认证

SWS\_SecOC\_00031: 安全I-PDU的创建以及真实I-PDU的认证包括以下六个步骤:

1. 准备安全I-PDU
2. 构造身份验证器的数据
3. 生成验证器
4. 构建安全I-PDU
5. 增量新鲜度计数器
6. 广播安全I-PDU

SWS\_SecOC\_00033: SecOC 模块应准备安全 I-PDU。在准备期间, SecOC 应分配必要的缓冲区来保存认证过程的中间结果和最终结果。

SWS\_SecOC\_00034: SecOC 模块应构造 `DataToAuthenticator`, 即用于计算 `Authenticator` 的数据。`DataToAuthenticator` 由 Data Id 的完整 16 位表示 (参数 `SecOCDataId`)、Authentic I-PDU 的安全部分和完整的 Freshness Value 对应于给定顺序的 `SecOCFreshnessValueID` 组成。为此, Data Id 和 Freshness Value 应以 Big Endian 字节顺序进行编码。

SWS\_SecOC\_00035: SecOC 模块应通过将 `DataToAuthenticator`、`DataToAuthenticator` 的长度传递到对应于 `SecOCTxAuthServiceConfigRef` 的认证算法中来生成认证器。

SWS\_SecOC\_00036: SecOC模块应将生成的Authenticator截断为 `SecOCAuthInfoTruncLength` 指定的位数。

SWS\_SecOC\_00037: SecOC 模块应通过将安全 I-PDU 报头 (可选)、新鲜度值 (可选) 和认证符添加到真实 I-PDU 来构建安全 I-PDU。

安全I-PDU的方案 (包括安全I-PDU中内容的结构顺序) 应符合以下要求:

```
$$ SecuredPDU = SecuredIPDUHeader (optional) | AuthenticIPDU | FreshnessValue [SecOCFreshnessValueTruncLength] (optional) | Authenticator  
[SecOCAuthInfoTruncLength] $$
```

#### Note

作为安全I-PDU的一部分包括的新鲜度计数器和认证器可以按照特定于安全I-PDU标识符的配置被截断。此外, Freshness Value可能是Authentic I-PDU的一部分

### I-PDU的验证

SWS\_SecOC\_00040: 安全I-PDU的验证包括以下六个步骤:

- 解析真实的 I-PDU、新鲜度值和验证器
- 从 Freshness Manager 获取新鲜度值
- 构造数据以进行身份验证
- 验证身份验证信息
- 向新鲜度管理发送确认
- 将 Authentic I-PDU 传递给上层

SWS\_SecOC\_00203: 如果使用 `SecOCRxSecuredPduCollection`, 则SecOC在收到构成安全I-PDU的真实I-PDU和加密I-PDU之前, 不得进行任何验证。只有在两者都收到后, SecOC才能尝试验证生成的安全I-PDU。如果使用 `SecOC_VerifyStatusOverride`, 则根据 `overrideStatus` 值处理验证结果和I-PDU。

#### Note

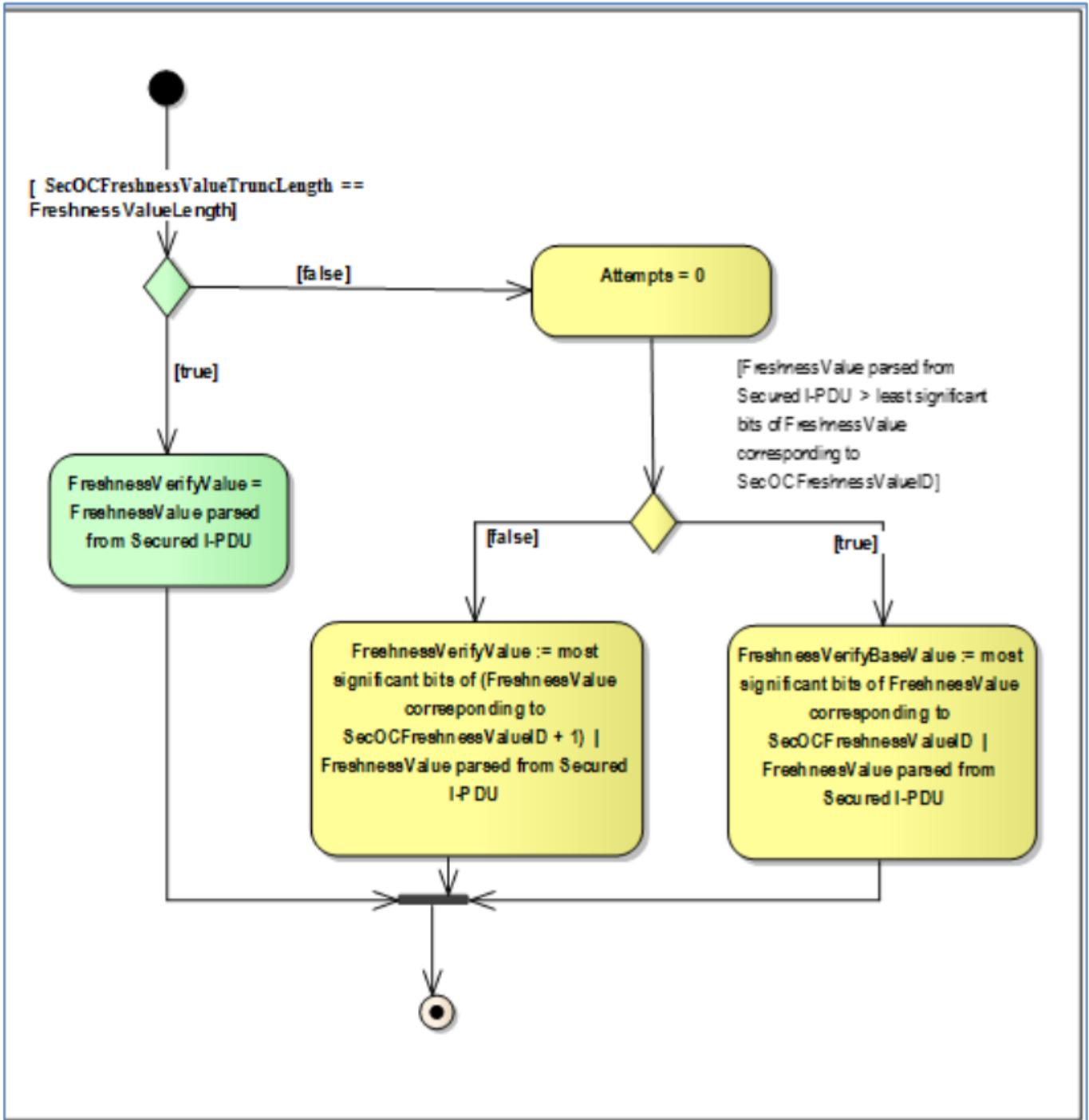
这适用于当 SecOC 从 PduR 接收到安全 I-PDU 时的所有情况, 这在使用 `SecOCRxSecuredPduCollection` 时发生在上述部分中。

SWS\_SecOC\_00211: 如果使用了 `SecOCRxSecuredPduCollection`, 那么 SecOC 将不会尝试验证安全 I-PDU, 直到它收到并缓冲了具有匹配消息链接器值的真实 I-PDU 和加密 I-PDU。如果使用 `SecOC_VerifyStatusOverride`, 则根据 `overrideStatus` 值处理验证结果和I-PDU。



如果 SecOCUseMessageLink 的多重性为 0，则意味着 SecOCMessageLinkLen 为 0，消息链接器值始终匹配。

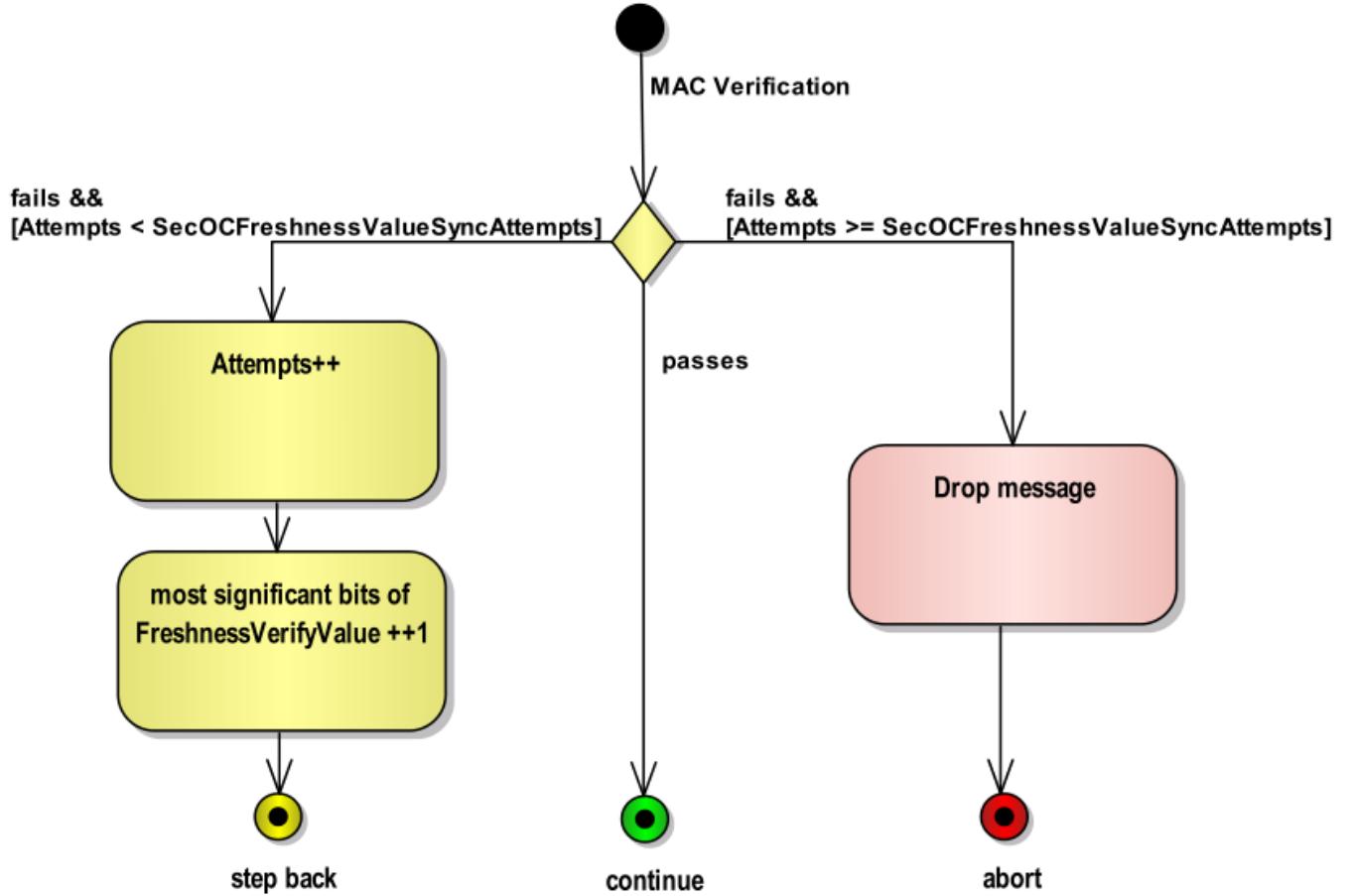
SWS\_SecOC\_00042: 收到安全的I-PDU后，SecOC应解析真实的I-PDU、新鲜度值和来自它的身份验证器。



**Figure 5: Construction of Freshness Value**

SWS\_SecOC\_00046: SecOC模块应构建用于计算接收方身份验证器（DataToAuthenticator）的数据。此数据由 SecOCDataId | AuthenticIPDU | FreshnessVerifyValue 组成

SWS\_SecOC\_00047: SecOC模块应通过将 DataToAuthenticator、DataToAuthenticator 的长度、从Secured I-PDU解析的Authenticator和 SecOCAuthInfoTruncLength 传递到对应于 SecOCRxAuthServiceConfigRef 的认证算法中来验证Authenticator。如果使用 SecOC\_VerifyStatusOverride，则根据overrideStatus值处理验证结果和I-PDU。



**Figure 6: Verification of MAC**

SWS\_SecOC\_00048: SecOC模块应报告相应安全Rx PDU的验证状态，如下所示：

如果 `SecOCRxPduProcessing / SecOCVerificationStatusPropagationMode` 设置为 `BOTH` 或 `FAILURE_ONLY`，则应根据当前配置通过调用函数 `SecOC_VerificationStatusCallout` 和 `SecOC_VerificationStatus` 接口提供验证状态。如果配置设置为 `NONE`，则不会提供报告。

#### Note

如果Freshness Manager需要安全PDU的状态（无论其是否成功验证），例如同步时间或计数器，则该状态应从SecOC提供的VerificationStatus服务中获取。

SWS\_SecOC\_00271: SecOC 模块应按如下方式报告相应安全 Rx-PDU 的验证状态：

如果 `SecOCRxPduProcessing / SecOCClientServerVerificationStatusPropagationMode` 设置为 `BOTH` 或 `FAILURE_ONLY`，则验证状态应根据其当前配置通过服务接口 `SecOC_VerificationStatusIndication` 提供。如果配置设置为 `NONE`，则不会提供报告

SWS\_SecOC\_00272: 如果配置项 `SecOCGeneral / SecOCPropagateOnlyFinalVerificationStatus` 设置为 `TRUE`，则只报告 `最终状态`。如果此项设置为 `FALSE`，则应根据 SWS\_SecOC\_00048 和 SWS\_SecOC\_00271 报告每个单独的验证状态（最后一次以及所有之前失败的验证状态）。

I-PDU的成功验证

SWS\_SecOC\_00050: 如果安全I-PDU验证成功或相应地设置了状态覆盖，SecOC模块应使用PduR的下层接口将真实I-PDU传递给上层通信模块。

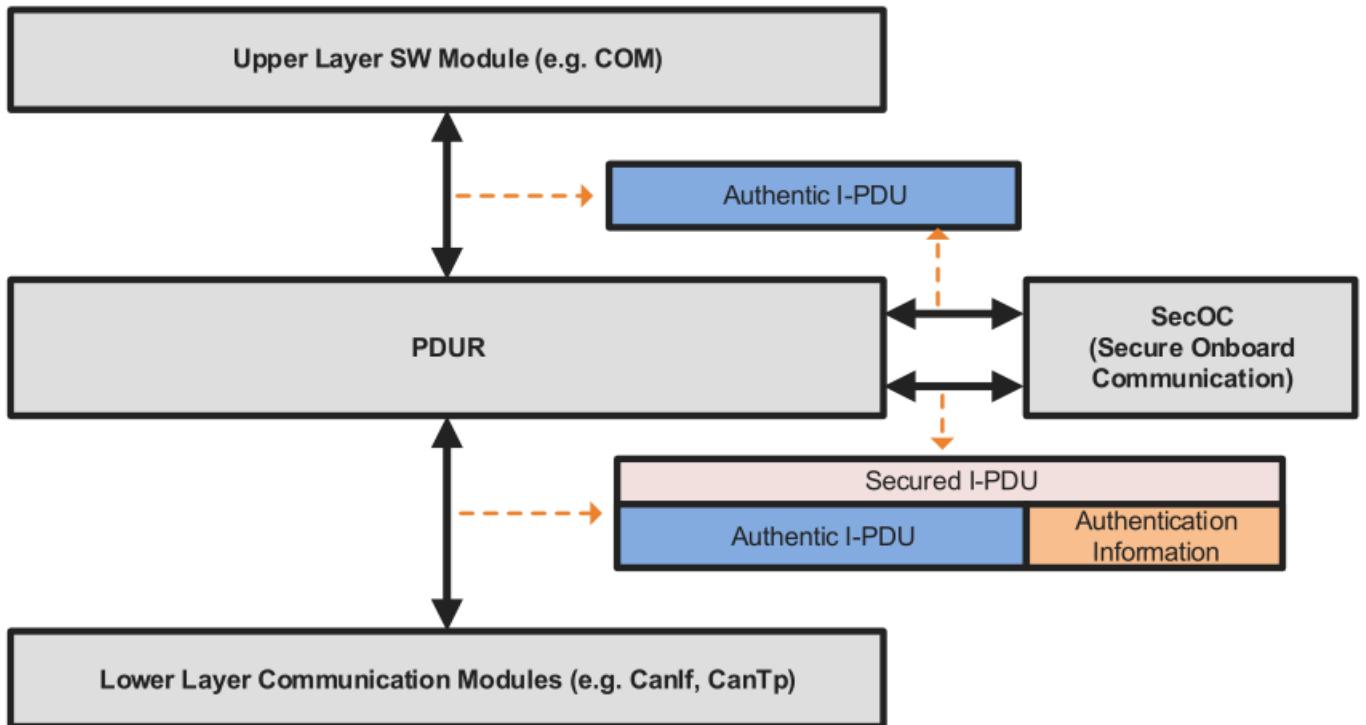
#### 不对称方法下的适应

尽管本文档因此使用了对称密码学中的术语和概念，但SecOC模块可以配置为同时使用对称和非对称加密算法。如果使用数字签名而非整个文档中描述的MAC方法的非对称方法，则必须进行一些调整：

1. 代替发送方和（所有）接收方之间的共享密钥，使用由公钥和密钥组成的密钥对。发送方使用秘密（或私有）密钥来生成签名，（所有）接收方使用相应的公钥来验证签名。私钥必须不可从公钥计算，并且接收方不能对其进行评估。
2. 为了验证消息，接收方需要访问签名生成算法的完整签名/输出。因此，MAC案例中提议的签名截断是不可能的。参数 `SecOCAuthInfoTruncLength` 必须设置为签名的 **完整长度**。
3. 签名验证使用与签名生成不同的算法。因此，接收方/验证者不是在接收方“重建”MAC并将其与接收到的（截断的）MAC进行比较，而是使用 `DataToAuthenticator`（包括完整计数器）和签名作为输入执行验证算法，并获得布尔值作为输出，确定验证是否通过或失败。

#### 与PDUR的关系

SecOC模块布置在AUTOSAR分层架构中的PDU路由器旁边；见图7。



**Figure 7: Transformation of an Authentic I-PDU in a Secured I-PDU by SecOC**

SWS\_SecOC\_00153: SecOC模块的实施应确保没有其他模块依赖于它，并且如果不需要SecOC模块，则可以在没有SecOC模块情况下构建系统。

SWS\_SecOC\_00212: SecOC应确保在真实PDU中接收的元数据在相应的安全PDU中保持不变，反之亦然。

#### 初始化

SecOC模块提供了一个初始化函数 (`SecOC_Init`)，如 SWS\_SecOC\_00106 中所定义。此函数初始化所有内部全局变量和缓冲区以存储 SecOC I-PDU 和所有中间结果。SecOC 的环境在调用 `SecOC` 模块的除 `SecOC_GetVersionInfo` 之外的任何其他函数之前应调用 `SecOC_Init`。实现者必须确保在开发模式下返回 `SecOC_E_UNINIT`，以防在模块初始化之前调用 API 函数。

对于通过SecOC模块的I-PDU数据传输路径，在SecOC模块内部分配了一个 **缓冲区**。这个缓冲区需要 **初始化**，因为它可能在它被下层通信模块的上层完全填充数据之前被传输。

SWS\_SecOC\_00054: 在 `SecOC_Init` 中，模块应初始化所有内部全局变量和 SecOC I-PDU 的缓冲区。

SWS\_SecOC\_00269: AUTOSAR SecOC 模块应使用配置参数 `SecOCTxPduUnusedAreasDefault` (ECUC\_SecOC\_00101) 确定的值填充传输的安全或传输的加密 Pdu 的未使用区域，例如 0xFF。

### 传出PDU的身份验证

术语“身份验证”描述了通过将身份验证信息添加到真实I-PDU来创建安全I-PDU。通常，与PduR模块的交互和Authentic I-PDU的认证按照以下方案组织：

1. 对于Authentic I-PDU的每一次传输请求，上层通信模块应通过 `PduR_<Up>Transmit` 调用PduR模块。
2. PduR 将该请求路由到 SecOC 模块并调用 `SecOC_[If|Tp]Transmit`
3. SecOC模块将Authentic I-PDU复制到自己的内存并返回。
4. 在其主功能的下一个调度呼叫期间，SecOC模块通过计算认证信息来创建安全I-PDU，并通过经由PduR模块通知相应的下层模块来启动安全I-PDU的传输。
5. 此后，SecOC模块扮演上层通信模块的角色，并因此服务于所有下层请求，以提供关于安全I-PDU的信息或复制其数据。
6. 最后，将安全I-PDU的成功或不成功传输的确认提供给上层通信模块，作为对真实I-PDU成功或不失败传输的确认

#### Note

对于每个Authentic I-PDU，上层通信模块的配置方式应确保其调用PduR模块，与直接传输请求的调用方式相同。在这种情况下，上层通过SecOC模块与TriggerTransmit和TP行为解耦。

为了启动Authentic I-PDU的传输，上层模块始终（与用于具体传输的总线接口无关）通过 `PduR_<Up>Transmit` 调用PduR模块。PduR将该请求路由到 SecOC模块，以便SecOC模块能够立即访问上层通信模块的缓冲器中的Authentic I-PDU。

SWS\_SecOC\_00252: SecOC模块应在开始传输相应的安全I-PDU之前，将完整的Authentic I-PDU复制到其内部存储器中。

#### Note

这意味着 Up 与 Lower PDU 接口的 IF/TP 配置之间没有依赖关系。

SWS\_SecOC\_00201: 如果使用 `SecOCTxSecuredPduCollection`，则 SecOC 应将 Secured(安全) I-PDU 作为两条消息进行传输：原始 Authentic(真实) I-PDU 和单独的 Cryptographic(加密) I-PDU。Cryptographic(加密) I-PDU 应包含 Secured(安全) I-PDU 的所有认证信息，因此 Authentic(真实) I-PDU 和 Cryptographic(加密) I-PDU 包含重构 Secured(安全) I-PDU 所需的所有信息。

SWS\_SecOC\_00202: SecOC应在同一主功能周期内传输真实I-PDU及其相应的加密I-PDU。

SWS\_SecOC\_00209: 如果使用 `SecOCTxSecuredPduCollection`，则 SecOC 应在加密 I-PDU 内重复一部分真实 I-PDU 作为消息链接器，并且加密 I-PDU 应构造为 `Cryptographic I-PDU =Authentication Data | Message Linker`

#### Note

“|”表示串联。

SWS\_SecOC\_00210: 如果使用 `SecOCUseMessageLink`，则 SecOC 应使用 Authentic I-PDU 内长度为 `SecOCMessageLinkLen` 位的位位置 `SecOCMessageLinkPos` 处的值作为消息链接器。

SWS\_SecOC\_00057: 根据 SWS\_SecOC\_00031 中描述的过程，SecOC 模块应提供足够的缓冲区容量来存储传入的 Authentic I-PDU、传出的 Secured I-PDU 和认证过程的所有中间数据。

SWS\_SecOC\_00146: SecOC模块应为真实I-PDU和安全I-PDU提供单独的缓冲区。

SWS\_SecOC\_00110: 来自上层通信模块的任何传输请求都应覆盖包含 Authentic I-PDU 的缓冲区，而不影响相应 Secured I-PDU 的缓冲区。

因此，可以处理真实I-PDU的上层更新，而不影响具有下层通信模块的安全I-PDU的正在进行的传输活动。

SWS\_SecOC\_00262: 对于 `secOCAuthPduHeaderLength > 0` 的 Tx Secured I-PDU，SecOC 模块应将 Secured I-PDU Header 添加到 Secured I-PDU 中，并在 Secured I-PDU 中添加 Authentic I-PDU 的长度，以处理动态 Authentic I -PDU。

### Note

这个 Header 的主要目的是指示 Freshness Value 和 Authenticator 在具有动态长度 Authentic I-PDU 的 Secured I-PDU 中的位置。一些不能选择任意长度 L-PDU 的总线（例如 CAN FD 和 FlexRay）也需要这个 Header，因为 Freshness Value 和 Authenticator 的位置并不总是在 Secured I-PDU 的末尾，因为下层模块（例如 CanIf 和 FrIf）可能会在 SecOC 处理后添加特定于总线的填充字节（然后 L-PDU 包含 带有填充的安全 I-PDU 将是：Secured I-PDU Header | Authentic I-PDU | Freshness Value | Authenticator | Bus-specific padding）。

#### 直接传输期间的身份验证

为了使用允许临时传输的总线接口（例如 CanIf）传输 Authentic I-PDU，PDU 路由器模块触发 SecOC 模块对 Authentic I-PDU 的传输操作。在这种情况下，SecOC 模块准备通过分配内部缓冲区容量并通过将真实 I-PDU 复制到本地缓冲区位置，在真实 I-PDU 的基础上创建安全 I-PDU。之后它从 `SecOC_[If|Tp]Transmit` 返回。

`SWS_SecOC_00058`: SecOC 模块应分配内部缓冲容量，以将 Authentic I-PDU 和认证信息存储在连续存储器位置。

Secured I-PDU 的实际创建是在下一次后续调用计划的 `main` 函数期间处理的。这包括根据 `SWS_SecOC_00031` 计算认证信息并将认证信息（即 Authenticator 和可能被截断的新鲜度值）连续添加到 Authentic I-PDU 后面的缓冲区位置。此后，SecOC 模块通过在 `PduR` 调用 `PduR_SecOCTransmit` 触发安全 I-PDU 到目标下层模块的传输。

`SWS_SecOC_00060`: 对于使用允许 ad-hoc 传输的总线接口（例如 CanIf）传输 Authentic I-PDU，SecOC 模块应根据 `SWS_SecOC_00031` 中指定的总体方法计算计划主函数中的 Authenticator。

`SWS_SecOC_00061`: 为了使用允许临时通信的总线接口（例如 CanIf）传输真实 I-PDU，SecOC 模块应在计划的主函数中创建安全 I-PDU。

`SWS_SecOC_00062`: SecOC 模块应提供完整的安全 I-PDU，以便通过触发 `PduR_SecOCTransmit` 进一步传输到目标下层模块。

`SWS_SecOC_00063`: 如果 PDU Router 模块通过调用 `SecOC_[If|Tp]TxConfirmation` 通知 SecOC 模块目标下层模块已经确认 Secured I-PDU 的传输或在传输过程中报告了错误，SecOC 模块应传递接收到的结果 通过调用 `PduR_SecOC[If|Tp]TxConfirmation` 将相应的 Authentic I-PDU 发送给上层模块。

`SWS_SecOC_00064`: 对于使用允许临时通信（例如 CanIf）的总线接口传输真实 I-PDU，如果为安全 I-PDU 调用 `SecOC_TxConfirmation`，SecOC 模块应释放包含安全 I-PDU 的缓冲区。

#### 触发传输期间的身份验证

### 3.1.3 API

#### API specification

### Type definitions

EcuM\_ConfigType

image-20230221172651518

EcuM\_RunStatusType

image-20230221172747835

EcuM\_WakeupSourceType

image-20230221173523580

EcuM\_WakeupStatusType

image-20230221173626689

EcuM\_ResetType

image-20230221173657000

EcuM\_StateType

image-20230221173722215

#### FUNCTION DEFINITIONS

EcuM\_GetVersionInfo

image-20230221175212308

EcuM\_GoDownHaltPoll

image-20230221175246784

EcuM\_Init

image-20230221175548687

EcuM\_StartupTwo

image-20230221175630866

EcuM\_Shutdown

image-20230221175915905

**EcuM\_SetState**

Service Name	EcuM SetState	
Syntax	void EcuM SetState( EcuM StateType state )	
Service ID [hex]	0x2b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	state	State indicated by BswM.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	BswM调用的通知状态切换的函数。	
Available via	EcuM.h	

**EcuM\_RequestRUN**

image-20230221180628703

**EcuM\_ReleaseRUN**

image-20230221181316351

**EcuM\_RequestPOST\_RUN**

image-20230221181420693

**EcuM\_ReleasePOST\_RUN**

image-20230221181621835

**EcuM\_SelectShutdownTarget**

image-20230221181650377

**EcuM\_GetShutdownTarget**

image-20230221181751580

**EcuM\_GetLastShutdownTarget**

image-20230221183159649

**EcuM\_SelectShutdownCause**

image-20230221183441489

**EcuM\_GetShutdownCause**

image-20230221183527146

**EcuM\_CheckWakeup**

image-20230221183605216

**EcuM\_GetPendingWakeupEvents**

image-20230221183710682

**EcuM\_ClearWakeupEvent**

image-20230221183754806

EcuM\_GetValidatedWakeupEvents

image-20230221183831113

EcuM\_GetExpiredWakeupEvents

image-20230221184434532

EcuM\_SetRelWakeupAlarm

image-20230222093425360

EcuM\_SetAbsWakeupAlarm

image-20230222093520226

EcuM\_AbortWakeupAlarm

image-20230222094514830

EcuM\_GetCurrentTime

image-20230222095022403

EcuM\_GetWakeupTime

image-20230222095800719

EcuM\_SetClock

image-20230222095853711

EcuM\_SelectBootTarget

image-20230222095956574

EcuM\_GetBootTarget

image-20230222100426533

EcuM\_SetWakeupEvent

image-20230222100509893

EcuM\_ValidateWakeupEvent

image-20230222100547006

EcuM\_ErrorHook

image-20230222100647200

EcuM\_AL\_SetProgrammableInterrupts

image-20230222100745858

EcuM\_AL\_DriverInitZero

image-20230222100824623

EcuM\_DeterminePbConfiguration

image-20230222100908498

EcuM\_AL\_DriverInitOne

image-20230222101414508

EcuM\_LoopDetection

image-20230222101455353

**EcuM\_OnGoOffOne**

image-20230222101536682

**EcuM\_OnGoOffTwo**

image-20230222101622153

**EcuM\_AL\_SwitchOff**

image-20230222101701603

**EcuM\_AL\_Reset**

image-20230222101739816

**EcuM\_EnableWakeupSources**

image-20230222101813564

**EcuM\_GenerateRamHash**

image-20230222102143939

**EcuM\_SleepActivity**

image-20230222102220368

**EcuM\_StartCheckWakeup**

image-20230222102255882

**EcuM\_CheckWakeupHook**

image-20230222102332348

**EcuM\_CheckRamHash**

image-20230222102407410

**EcuM\_DisableWakeupSources**

image-20230222102438792

**EcuM\_AL\_DriverRestart**

image-20230222102509396

**EcuM\_StartWakeupSources**

image-20230222102546055

**EcuM\_CheckValidation**

image-20230222102845827

**EcuM\_StopWakeupSources**

image-20230222102953360

**EcuM\_MainFunction**

image-20230222103016504

**Callbacks from the STARTUP phase**

image-20230222103310832

**Ports and Port Interface for EcuM\_ShutdownTarget Interface**

EcuM\_ShutdownTarget客户端-服务器接口允许 SW-C 选择一个关闭目标，该目标将在下一个关闭阶段得到遵守。但是，请注意，ECU 管理器模块不提供端口接口来允许 SW-C 启动关断。

**Service Interfaces**

image-20230222103801225

image-20230222104753099

image-20230222104821801

image-20230222104849173

image-20230222104909301

image-20230222104931051

**Port Interface for EcuM\_BootTarget Interface**

想要选择引导目标的 SW-C 必须需要客户端-服务器接口 EcuM\_BootT target。

**Service Interfaces**

image-20230222105431567

image-20230222105455324

image-20230222105511243

**Port Interface for EcuM\_AlarmClock Interface**

想要使用闹钟的 SW-C 必须要求客户端-服务器接口 EcuM\_AlarmClock。EcuM\_AlarmClock 接口使用端口定义的参数值来标识管理其闹钟的用户。

**Service Interfaces**

image-20230222105939055

image-20230222110000871

image-20230222110017636

image-20230222110040343

image-20230222110054805

**Port Interface for EcuM\_Time Interface**

想要使用 EcuM 的时间功能的 SW-C 必须需要客户端-服务器接口 EcuM\_time

**Service Interfaces**

image-20230222110217897

image-20230222110543898

image-20230222110641384

**Port Interface for EcuM\_StateRequest Interface**

当容器 EcuMModeHandling 可用时，ECU 状态管理器模块应为以下功能提供系统服务：

- requesting RUN
- releasing RUN
- requesting POST\_RUN
- releasing POST\_RUN

需要保持 ECU 活动状态或需要在 ECU 关闭之前执行任何操作的 SW-C 需要客户端-服务器接口 EcuM\_StateRequest。此接口使用端口定义的参数值来标识请求模式的用户。

**Service Interfaces**

image-20230222110943327

image-20230222110956054

image-20230222111012648

image-20230222111032366

image-20230222111045573

#### **Port Interface for EcuM\_CurrentMode Interface**

ECU 状态管理器模块的模式端口应声明以下模式:

- STARTUP
- RUN
- POST\_RUN
- SLEEP
- SHUTDOWN

此定义是应用程序确实需要知道的ECU模式的简化视图。它不会以任何方式限制或限制如何定义应用程序模式。应用程序模式完全由应用程序本身处理。

发生模式更改时，应通过 RTE 模式端口将模式更改通知 SW-C。

该规范假定端口名为 currentMode，并且将使用 RTE 的直接 API。在这些条件下，通过调用 `ype Rte_Switch_currentMode` (`Rte_ModeType_EcuM_Mode` 模式) `Rte_StatusT` 模式发出信号，其中模式是要通知的新模式。值范围由前面的要求指定。应忽略返回值。

想要收到模式更改通知的 SW-C 应要求模式开关接口 `EcuM_CurrentMode`。

#### **Service Interfaces**

image-20230222111759183

#### **Definition of the ECU Manager Service**

请注意，这些定义只能在ECU配置期间完成（因为某些ECU管理器模块配置参数决定了ECU管理器模块服务提供的端口数）。另请注意，SW-C 的实现不依赖于这些定义。

在 AUTOSAR 系统中，RTE 上方和下方都有端口。ECU 管理器模块服务描述定义了提供给 RTE 的端口，使用此服务的每个软件的描述都必须包含“服务端口”，这些端口需要来自 RTE 的这些 ECU 管理器模块端口。

EcuM 提供以下端口：

image-20230222112014589

image-20230222112117543

image-20230222112135316

image-20230222112151110

image-20230222112317215

image-20230222112350554

EcuM 提供以下类型：

image-20230222112426701

image-20230222112450582

image-20230222112809772

image-20230222112826040

image-20230222112843541

image-20230222112859015

对于多核ECU，可以在一个或多个内核上提供EcuM AUTOSAR服务（标准化AUTOSAR接口）。

对于多核ECU，其他BSW模块使用的EcuM C-API接口（标准化接口）应在运行EcuM的每个分区中提供。

其他 BSW 模块用于与 EcuM 通信的 C-API 接口由每个 EcuM 实例提供，因为每个 EcuM 实例都可以执行一些独立的操作。如果 BSW 模块想要使用 EcuM，但位于不包含自己的 EcuM 实例的分区。这些模块可以使用 SchM 函数来跨越分区边界。

### 3.1.4 Sequence Charts

#### Sequence Charts

##### GPT WAKEUP SEQUENCES

通用计时器（GPT）是可能的唤醒源之一。通常，GPT在ECU进入睡眠状态之前启动，硬件计时器在到期时会导致中断。中断唤醒微控制器，并在GPT模块中执行中断处理程序。它通知ECU状态管理器模块发生了GPT唤醒。为了区分导致唤醒的不同GPT通道，集成商可以为每个GPT通道分配不同的唤醒源标识符。图9.1显示了相应的调用顺序。

image-20230222133446346

image-20230222133559387

##### ICU WAKEUP SEQUENCES

输入捕获单元（ICU）是另一个唤醒源。与GPT相反，ICU驱动程序本身并不是唤醒源。它只是处理唤醒中断的模块。因此，只有唤醒源的驱动程序才能判断它是否负责该唤醒。这使得EcuM\_CheckWakeUpHook必须询问作为实际唤醒源的模块。为了知道要询问哪个模块，ICU必须将唤醒源的标识符传递给EcuM\_CheckWakeUp。对于共享中断，集成代码可能必须检查EcuM\_CheckWakeUpHook内的多个唤醒源。为此，ICU必须传递可能导致此中断EcuM\_CheckWakeUp的所有唤醒源的标识符。请注意，EcuM\_WakeupSourceType（请参阅8.2.3 EcuM\_WakeupSourceType）包含每个唤醒源的一个位，因此可以在一次调用中传递多个唤醒源。

图9.3显示了生成的调用序列。由于ICU只负责处理唤醒中断，因此轮询ICU是不明智的。为了轮询，必须直接检查唤醒源，如图38所示。

image-20230222134033181

##### CAN WAKEUP SEQUENCES

在CAN上，收发器或通信控制器可以使用中断或轮询来检测唤醒。唤醒源标识符应在收发器和控制器之间共享，因为ECU状态管理器模块只需要知道已唤醒的网络并将其传递给通信管理器模块。

在中断或共享中断情况下，不清楚哪个特定的唤醒源（CAN控制器、CAN收发器、LIN控制器等）检测到唤醒。因此，集成器必须将派生的EcuM\_CheckWakeUp唤醒源（wakeupSource），它可以代表共享中断或仅代表中断通道，分配给传递给CanIf\_CheckWakeUp（WakeupSource）的特定唤醒源。因此，这里的参数wakeupSource来自EcuM\_CheckWakeUp（）可能与CanIf\_CheckWakeUp的WakeupSource不同，或者它们可以相等。这取决于硬件拓扑和EcuM\_CheckWakeUpHook集成器代码中的实现。

在中断或共享中断情况下，不清楚哪个特定的唤醒源（CAN控制器、CAN收发器、LIN控制器等）检测到唤醒。因此，集成器必须将EcuM\_CheckWakeUp(wakeupSource)的派生的唤醒源分配给传递给特定唤醒源，它可以代表共享中断或仅代表中断通道，并传递给CanIf\_CheckWakeUp(WakeupSource)。因此，这里的参数wakeupSource来自EcuM\_CheckWakeUp（）可能与CanIf\_CheckWakeUp的WakeupSource不同，或者它们可以相等。这取决于硬件拓扑和EcuM\_CheckWakeUpHook集成器代码中的实现。

在CanIf\_CheckWakeUp(WakeupSource)期间，CAN接口模块(CanIf)将检查是否有任何设备(CAN通信控制器或收发器)配置了“WakeupSource”的值。如果是这种情况，则通过相应的设备驱动程序模块检查设备是否唤醒。如果设备检测到唤醒，设备驱动程序将通过EcuM\_SetWakeupEvent(sources)通知EcuM。参数“sources”在设备上设置为已配置的值。因此它被设置为调用CanIf\_CheckWakeUp()时使用的值。

image-20230222135606103

CAN控制器中断唤醒的工作方式类似于GPT唤醒。此处，中断处理程序和CheckWakeUp功能都封装在CAN驱动程序模块中，如图9.5所示。

image-20230222135917935

CAN收发器和控制器都可以通过轮询唤醒。ECU状态管理器模块将定期检查CAN接口模块，该模块根据传递给CAN接口模块的唤醒源参数询问CAN驱动程序模块或CAN收发器驱动程序模块，如图9.6所示。

image-20230222145440781

通过中断或轮询检测到来自CAN收发器或控制器的唤醒事件后，可以验证唤醒事件。这是通过打开EcuM\_StartWakeupSources中相应的CAN收发器和控制器来完成的（参见[SWS\_EcuM\_02924]）。这取决于所使用的CAN收发器和控制器，集成器代码EcuM\_StartWakeupSource中的哪些函数调用是必要的。例如，在图9.7中，提到了启动和停止来自CAN状态管理器模块的唤醒源所需的函数调用。

CanIf识别成功接收至少一条消息，并将其记录为成功验证。在验证期间，ECU状态管理器模块定期检查集成器代码EcuM\_CheckValidation中的CanIf。ECU状态管理器模块在验证成功后，将通过通信管理器模块继续正常启动CAN网络。

image-20230222153729030

#### LIN WAKEUP SEQUENCES

图9.8显示了LIN收发器通过中断唤醒。中断通常由ICU驱动程序处理，如9.2.2章所述。

image-20230222154253513

如图9.9所示，LIN控制器中断唤醒原理与CAN控制器中断唤醒原理类似。在这两种情况下，Driver模块封装了中断处理程序。

image-20230222154709800

LIN收发器和控制器可以轮询唤醒。ECU状态管理器模块会定期检查LIN接口模块，而LIN接口模块会询问LIN驱动模块或LIN收发器驱动模块，如图9.10所示。

image-20230222154959295

#### FLEXRAY WAKEUP SEQUENCES

对于FlexRay，只有通过FlexRay收发器才能实现唤醒。FlexRay集群中有两个不同通道的收发器。它们被视为属于一个网络，因此，两个通道应该只配置一个唤醒源标识符。图9.11显示了FlexRay收发器通过中断唤醒。

image-20230222155604356

注意，在EcuMM\_CheckWakeupHook中需要对FrIf\_WakeupByTransceiver进行两个单独的调用，每个FlexRay通道一个。

image-20230222155909126

#### ETHERNET WAKEUP SEQUENCE

在具有OA TC10兼容以太网硬件的以太网交换网络上，使用的以太网硬件(PHY)可以检测到唤醒。对于维护以太网交换机(主机ECU)的以太网ECU，建议使用按需轮询来检查以太网硬件通知的唤醒。因为检查所有受影响的EthSwtPort可能会花费时间，并且在中断中检查是不可接受的。因此，中断信号表明至少有一个以太网交换机端口检测到唤醒。在中断的上下文中，受影响的EthTrcv在EthTrcv\_MainFunction中被发出异步检查的信号。

每个EthTrcv应该有自己的唤醒源，以区分唤醒到达哪个EthSwtPort。如果EthSwtPort分配给相同的pnc，则可以共享唤醒源。

下面的以太网唤醒序列部分是可选的，因为没有“集成代码”的规范。因此，它是特定于实现的，例如在EcuM\_CheckWakeupHook期间调用EthIf来检查唤醒源。

image-20230222160547107

image-20230222160651224

单个以太网ECU(不维护以太网交换机的ECU)可以选择如何通过中断或轮询来检测唤醒。与主机ECU的不同之处在于，它不需要检查大量的以太网交换机端口。

image-20230222160909157

image-20230222161047220

### 3.1.5 Configuration

---

#### **Configuration specification**

## 4. 关于

---

### 4.1 发行说明

---

#### 4.1.1 升级

要将MkDocs升级到最新版本，请使用pip:

```
1 pip install -U mkdocs
```

您可以使用 `mkdocs --version` 确定当前安装的版本:

```
1 $ mkdocs --version
2 mkdocs, version 1.0 from /path/to/mkdocs (Python 3.6)
```

## 4.2 为MkDocs做贡献

为MkDocs项目做出贡献的介绍。

MkDocs项目欢迎并依赖于开源社区中开发人员和用户的贡献。 贡献可以通过多种方式进行，例如：

- 提交代码补丁
- 改进文档
- 错误报告和补丁评论

### 4.2.1 行为准则

在MkDocs项目的代码库，问题跟踪器，聊天室和邮件列表中进行交互的每个人都应遵循[PyPA行为准则]。

### 4.2.2 报告问题

请尽可能详细地提供详细信息。 让我们知道您的平台和MkDocs版本。 如果问题是可视的（例如主题或设计问题），请添加屏幕截图，如果出现错误，请包含完整错误和追溯。

### 4.2.3 测试开发版本

如果您只想安装并试用MkDocs的最新开发版本，可以使用以下命令执行此操作。 如果您想为新功能提供反馈或想要确认您遇到的错误是否已在git主服务器中修复，则此功能非常有用。 强烈建议您在virtualenv中执行此操作。

```
pip install https://github.com/mkdocs/mkdocs/archive/master.tar.gz
```

### 4.2.4 安装开发

首先，您需要fork并克隆存储库。 获得本地副本后，请运行以下命令。 强烈建议您在virtualenv中执行此操作。

```
pip install --editable .
```

这将在开发模式下安装MkDocs，它将`mkdocs`命令绑定到git存储库。

### 4.2.5 运行测试

要运行测试，建议您使用tox。

通过运行命令`pip install tox`使用pip安装Tox。 然后，可以通过在MkDocs存储库的根目录中运行命令`tox`来为MkDocs运行测试套件。

它将尝试针对我们支持的所有Python版本运行测试。 所以不要担心，如果你错过了一些，他们会失败。 当您提交pull request时，Travis将验证其余部分。

### 4.2.6 提交Pull Requests

一旦您对更改感到满意，或者您已准备好提供反馈，请将其推送到您的分支并发送拉取请求。 要接受更改，如果它是新功能，则很可能需要测试和文档。

## 4.3 许可

---

法律相关的东西。