Wake-up and Sleep with AUTOSAR

Technical Reference

Wakeup CAN, LIN, FlexRay via Communication Channel

Version 1.00.05

Authors	Mark A. Fingerle, Thomas Kuhl
Status	Released

Document Information

History

Author	Date	Version	Remarks
Mark A. Fingerle	2010-01-19	1.0	Initial version
Thomas Kuhl	2011-02-22	1.00.01	Change CAN wake up source validation handling refer to 2.3.5
Mark A. Fingerle, Klaus Emmert	2011-07-19	1.00.02	Remove Chapter “1.2.4 Wake-up by polling the wake-up source(s)” feature not supported by EcuM > Synchronous and asynchronous wake-up handling added (chapter 1.4.1). > New screen shots for ECUM and ICU
Klaus Emmert	2013-01-21	1.00.03	Chapter 1.3 added Chapter 2.3.5 example code updated
Thoms Kuhl	2013-05-22	1.00.04	correct typing errors inside source code samples (ESCAN00052636)
Thomas Kuhl	2013-08-01	1.00.05	correct typing errors (ESCAN00068119)

Reference Documents

No.	Source	Title	Version
[1]	ASR	Specification of ECU State Manager	1.2.0
[2]	Vector	AN-ISC-2-1098_Wakeup_by_GPT_ICU	1.0
[3]	Vector	TechnicalReference_Asr_ComM	3.7
[4]	Vector	TechnicalReference_Asr_Nm	2.7.1
[5]	NXP	TJA1041 AN00094_3.pdf	Rev 03
[6]	**Philips**	AN00093_2---TJA1020 LIN transceiver.pdf	Rev 02
[7]	**NXP**	071130 TJA1080A.pdf	Rev 02

Please note
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Introduction

This document describes the usage of MICROSAR (based on AUTOSAR 3.x) if the ECU shall wake up and start its own communication due to detected communication on the bus. Here is described how a communication channel (CAN, LIN or FlexRay) can be reactivated from sleep mode and if the system is in a power save mode, how the ECU can be reactivated via a wake-up event on a communication channel.

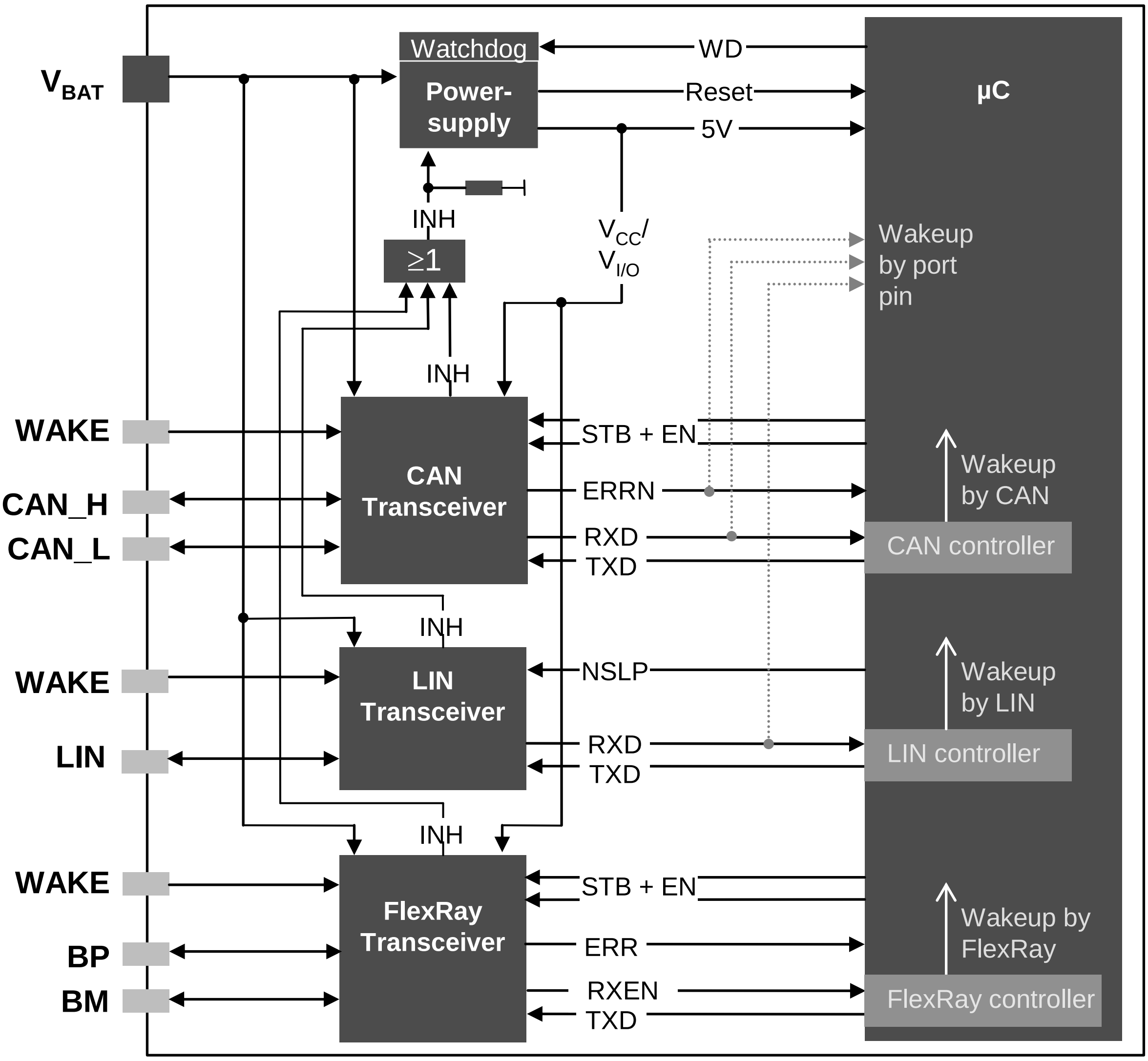


Figure 1-1 Example ECU schematic with CAN, LÍN and FlexRay transceiver

1.1 Procedure

- Configure the ECU wake-up sources (DaVinci Configurator Pro: ECUM)
- Configure the desired wake-up indication (DaVinci Configurator Pro: ICU interrupt, GENy: bus-specific interrupt, bus-specific polling mode)
- Implement the needed ECUM callback functions

1.2 Use Cases

For every wake-up interrupt use case the wake-up sources have to be configured.

> See 2.1 ECUM Configuration

1.2.1 Bus wake-up by internal run request

> The ECU is active and the application demands full communication mode for a network. The COMM starts the communication of the network (see [3]).

> If the communication of a channel is started the NM also starts the communication of the other synchronized channels (see [4]).

1.2.2 Wake-up by timer interrupt

> The GPT module triggers the wake-up interrupt and calls the function EcuM_CheckWakeup (see [2]).

1.2.3 Wake-up by µC I/O port

> Any desired wake-up source may be connected to an I/O port of the ECU. The event on the I/O port is detected by the ICU which starts the EcuM_CheckWakeup (not scope of this document).

1.2.4 Wake-up by CAN transceiver

> The used CAN controller does not support the feature "wakeup by CAN message", i.e. the CAN controller cannot detect incoming CAN messages and generate a wakeup interrupt. The ECU shall wake up and start its own communication due to detected communication on the CAN bus. In this case the CAN transceiver is used as interrupt source.

> See 2.2 ICU Interrupt (µC I/O port); 2.3.2 CAN Wake-up

1.2.5 Wake-up by CAN controller

> The used CAN controller supports the feature "wake-up by CAN message”.

> See 2.3.3 CAN Wake-up

1.2.6 Wake-up by LIN transceiver

> The used LIN controller does not support the feature "wake-up by message” and the transceiver should be used as wake-up source. In this case the LIN transceiver is used as interrupt source.

> See 2.2 ICU Interrupt (µC I/O port), 2.4.1 LIN Wake-up

1.2.7 Wake-up by LIN controller:

> The used LIN controller supports the feature "wake-up by message”.

> See 2.4.2 LIN Wake-up

1.2.8 Wake-up by FlexRay transceiver interrupt:

> The FlexRay transceiver should be used as wake-up source via interrupt.

> See 2.2 ICU Interrupt (µC I/O port), 2.5.1 FlexRay Wake-up by Transceiver Interrupt

1.2.9 Wake-up by FlexRay polling mode:

- The FlexRay transceiver should be used as wake-up source in polling mode.
- See 2.5.2 FlexRay Wake-up by Polling the Transceiver

1.3 Wake Up Detection at ECU Start-Up

The following listed conditions must be true for the ECU project:

- the used ECU supply voltage is switched by a CAN transceiver
- the ECU shall start up but the CAN wake-up event shall be validated before the ECU enters RUN state

Such a scenario is not fully supported by the AUTOSAR standard. The functional description below is an extension for AUTOSAR.

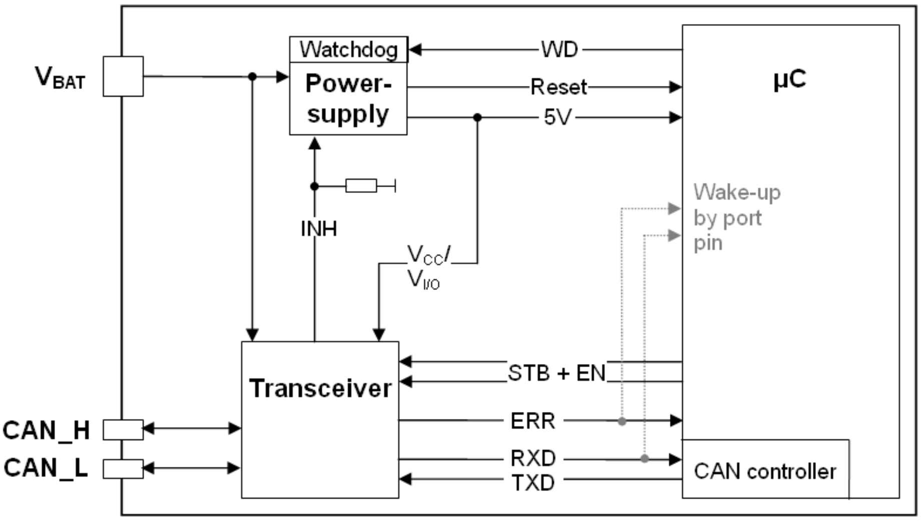


Figure 1-2 ECU sample

1.3.1 Functional Description

The bus wakeup detection is done inside the transceiver initialization, i.e. during `EcuM_AL_DriverInitTwo`. The transceiver monitors a detected bus wake-up to the EcuM via `EcuM_SetWakeupEvent(…)`.

An exception is partial networking as described in the following chapter.

1.3.1.1 Transceivers Supporting Partial Networking

The transceiver driver only supports 1 wake-up source and uses this source for the following wake-up reasons:

- POR – Power-On-Reset
- SYSERR
- WUP (wake-up pattern) / WUF (wake-up frame) – i.e. the wake-up event from the bus
- LWU (local wake-up/wakePin if supported by the hardware)

For Transceivers with partial networking the standard wake-up validation shall not be used. For that reason it is necessary to figure out the wake-up event. This can be done using the function `CanIf_GetTrcvWakeupReason()`:

Example

Here is an example how to figure out the wake-up reason.

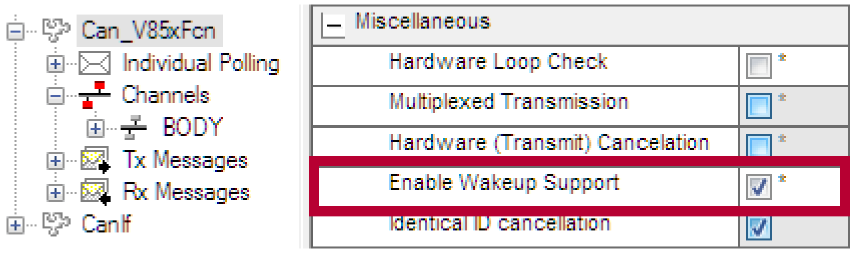
```
FUNC(void, ECUM_CODE) EcuM_AL_DriverInitTwo  (P2CONST(EcuM_ConfigType, AUTOMATIC,
ECUM_APPL_CONFIG) ConfigPtr)
{
  CanIf_TrcvWakeupModeType wakeupModeType;
  CanTrcv_Init() /* the Transceiver Treiber checks at initialization
                    on a corresponding wake-up event and informs the
                    ECUM via EcuM_SetWakeupEvent(<WK_Source_Handle>)!
                    The transceiver has to be initialized with the
                    state normal!*/

  /* Can driver initialization */
  CanDrv_Init()

  /* CAN interface initialization */
  CanIf_Init()

  /* The source code section below has to be performed per configured
  CAN channel */
  /* The source code section below must be performed before
  CanSM_Init () is called */

  if(EcuM_GetStatusOfWakeupSource(<WK_Source_Handle>) ==
  ECUM_WKSTATUS_PENDING)
  {
    (void)CanIf_GetTrcvWakeupReason(<TrcvId>, &wakeupModeType);

    if(wakeupModeType != CANIF_TRCV_WU_BY_BUS)
    {
      /* wakeup not triggered because of CAN bus event e.g. reception of
         wake up pattern (refer to the specific transceiver  driver
         technical reference about the supported value range of the
         CanIf_TrcvWakeupModeType
         Dependent on which wake-up event the bus should be started, the if
         condition has to be extended. */

      /* wake up event will be cleared to prevent invalid bus wake up */
      EcuM_ClearWakeupEvent(<WK_Source_Handle>);
    }
  }
}
```

If the wake-up reason is WUP or LWU, the bus can be woken up. Otherwise leave the bus sleeping.

1.3.2 Configuration

1.3.2.1 CAN Driver Configuration

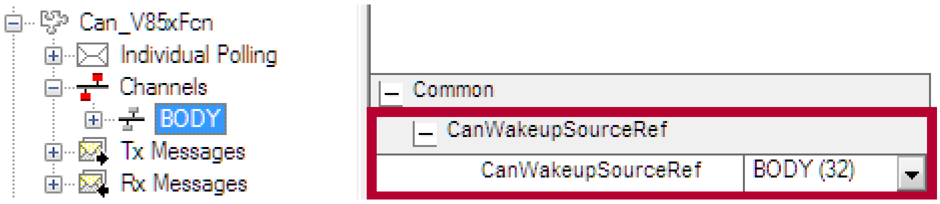
> Enable Wakeup Support



Miscellaneous	
Hardware Loop Check	☐ *
Multiplexed Transmission	☐ *
Hardware (Transmit) Cancelation	☐ *
Enable Wakeup Support	☑ *
Identical ID cancellation	☑

Figure 1-3 Enable Wakeup Source for CAN Driver

> Configure the **CanWakeupSourceRef** to the same value as you are going to set the values for the transceiver in the next chapter 1.3.2.2.

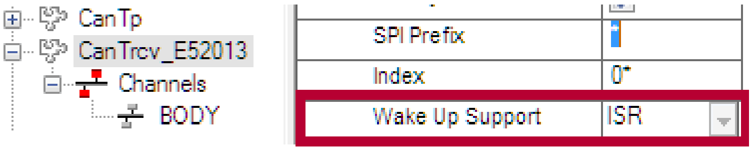


Common	
CanWakeupSourceRef	
CanWakeupSourceRef	BODY (32)

Figure 1-4 Wakeup Source on Driver Side

1.3.2.2 Transceiver Driver Configuration

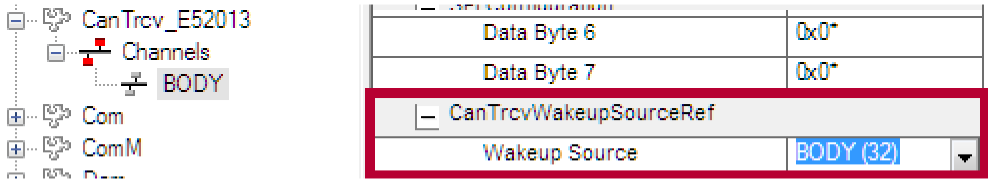
> Set **Wake Up Support**



SPI Prefix	
Index	0*
Wake Up Support	ISR

Figure 1-5 Wake Up support settings

> Set the **Wakeup Source** to appropriate value



Data Byte 6	0x0*
Data Byte 7	0x0*
CanTrcvWakeupSourceRef	
Wakeup Source	BODY (32)

Figure 1-6 Wakeup Source on Transceiver Side

1.4 Wake-up Process

This is a short explanation how a typical wake-up sequence is executed. For a detailed description of the execution flow when a wake-up event occurs please refer to the AUTOSAR ECUM [1] specification Chapter 9.

- **Enable Trigger:** Wake-up Enabled (init, shutdown, bus sleep). The wake-up sources will be enabled
  - if the according communication channel stops the communication or
  - if the ECU enters the sleep mode.
- **Trigger:** WakeupEvent (bus interrupt, ICU interrupt, SCHM(bus) polling)
  - A wake-up event is detected and the ECUM is informed about the wake-up source.
  - Alternatively the ECUM cyclically checks each wake-up source (which is configured as polling mode source) if a wake-up is pending.
- **Check:** EcuM_CheckWakeup triggers wake-up source-specific handling
  - The ECUM triggers the “check wake-up” function of the according <bus>-interface. If a wake-up is pending the ECUM is informed via the SetWakeupEvent function.
  - In case of shared interrupts multiple wake-up sources can be passed in one call.
  - ECUM starts wake-up validation timeout.
- **Set:** EcuM_SetWakeupEvent
  - The transceiver/controller driver of the wake-up source informs the ECUM if this wakeup source has triggered the wake-up event.
- **Disable:** Wake-up / Set MCU
  - After a valid wake-up the (interrupt notification of the wakeup) source becomes disabled. This might is done by the ECUM in case of an ECU wake-up or it has to be done by the wake-up source e.g. by the CanTrcv.
- **Validation**: (CAN)
  - The ECUM starts the bus which matches to the wake-up source (EcuM_StartWakeupSources).
  - The ECUM starts the validation of the wake-up event (if configured) and triggers the CanIf_CheckValidation function which matches to the wake-up source.
    - If a correct message is received on the CAN bus the CanIf_CheckValidation function has to call EcuM_ValidateWakeupEvent to indicate the ECUM about a valid wake-up event. The ECUM informs the COMM via ComM_EcuM_WakeUpIndication.
    - If the validation time expires the ECUM executes the callout EcuM_StopWakeupSource where activities are implemented to stop the bus again. The ECUM sets back the ECU into the preceding state again.

,

1.4.1 Synchronous and Asynchronous Wake-Up Concept

These concepts are working for all bus systems and any wake-up source. For this reason it is described only once. It can be transferred to your actual use case very easily.

Note
For the explanation we use CAN as example.

When a wake-up interrupt occurs, a `CanIf_CheckWakeup` has to be performed for figure out the reason for the wake-up interrupt. The result of this check can be

1. Wake-up reason detected, the reason is e.g. CAN message receipt.
2. No reason detected.

Synchronous Wake-Up Concept

In the synchronous case, the ECUM waits actively for the response of the `CanIf_CheckWakeup.`

In case of a longer process to figure out the wake-up reason, this would result in delay of the system.

Asynchronous Wake-Up Concept

The only difference to the synchronous wake-up is the timer you start, when you check for the wake-up reason via `CanIf_CheckWakeup`. There could be three results (Figure 1-8):

1. The successfully detected wake-up reason is returned before the timer expires
2. An `EcuM_EndCheckWakeup` is received before the timer expires. Then the timer is stopped and work can be continued.
3. The timer expires. In this case the result is, that there is no wake-up detected and work can be continued.

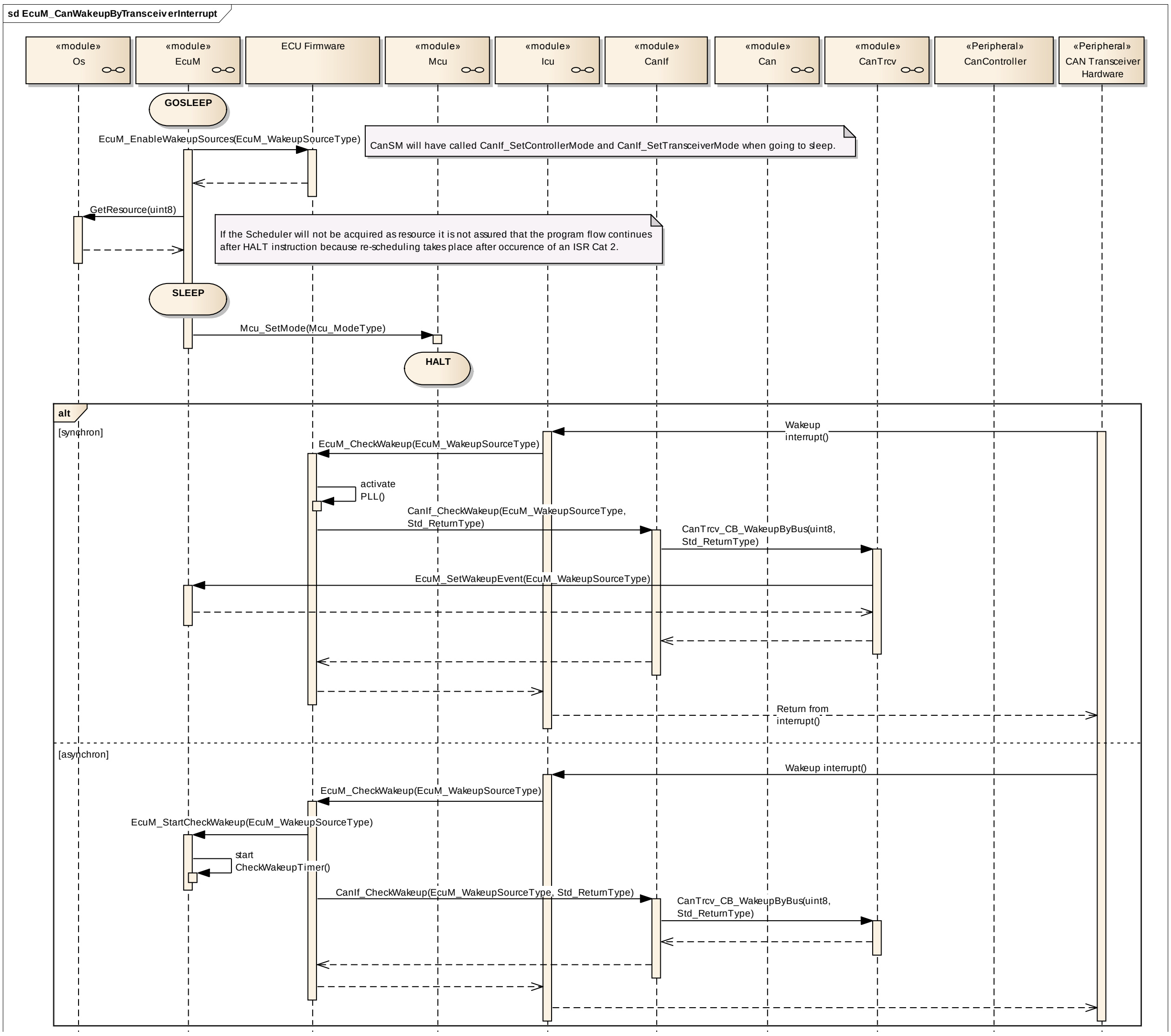

sd EcuM_CanWakeupByTransceiverInterrupt
«module» Os
«module» EcuM
ECU Firmware
«module» Mcu
«module» Icu
«module» CanIf
«module» Can
«module» CanTrcv
«Peripheral» CanController
«Peripheral» CAN Transceiver Hardware
GOSLEEP
EcuM_EnableWakeupSources(EcuM_WakeupSourceType)
CanSM will have called CanIf_SetControllerMode and CanIf_SetTransceiverMode when going to sleep.
GetResource(uint8)
If the Scheduler will not be acquired as resource it is not assured that the program flow continues after HALT instruction because re-scheduling takes place after occurence of an ISR Cat 2.
SLEEP
Mcu_SetMode(Mcu_ModeType)
HALT
alt
[synchron]
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
activate PLL()
CanIf_CheckWakeup(EcuM_WakeupSourceType, Std_ReturnType)
CanTrcv_CB_WakeupByBus(uint8, Std_ReturnType)
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
Return from interrupt()
[asynchron]
Wakeup interrupt()
EcuM_CheckWakeup(EcuM_WakeupSourceType)
EcuM_StartCheckWakeup(EcuM_WakeupSourceType)
start CheckWakeupTimer()
CanIf_CheckWakeup(EcuM_WakeupSourceType, Std_ReturnType)
CanTrcv_CB_WakeupByBus(uint8, Std_ReturnType)

Figure 1-7 Extract from the AUTSOAR document **AUTOSAR_SWS_ECU_StateManager.pdf**. It shows both use cases.

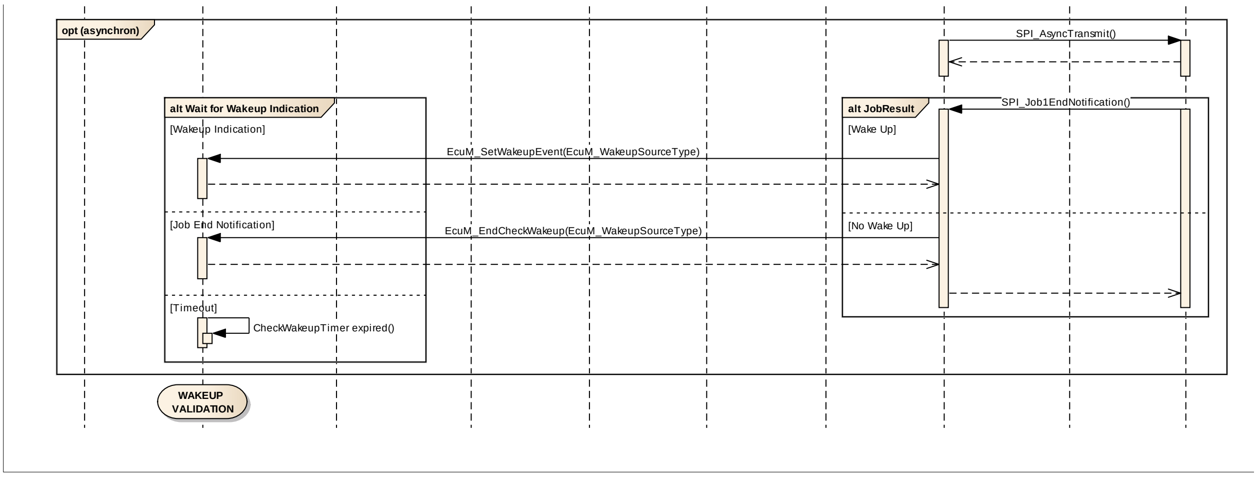
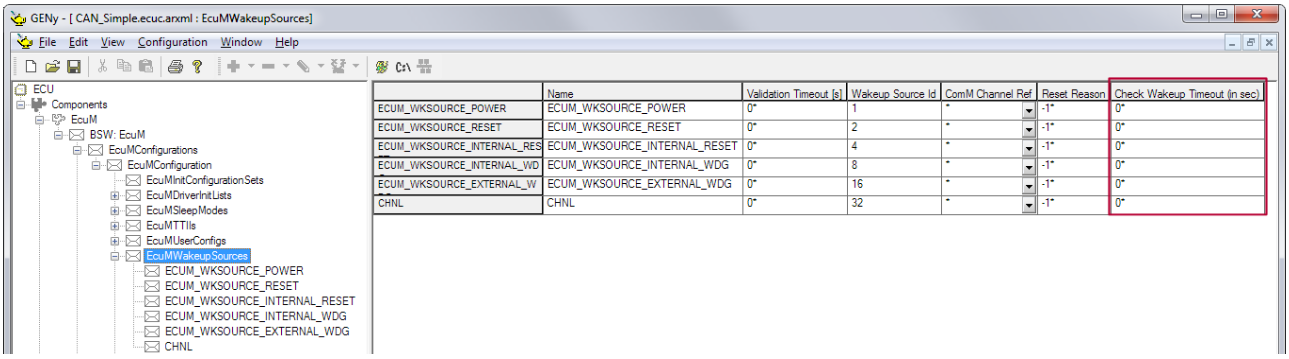

opt (asynchron)
SPI_AsyncTransmit()
alt Wait for Wakeup Indication
[Wakeup Indication]
EcuM_SetWakeupEvent(EcuM_WakeupSourceType)
[Job End Notification]
EcuM_EndCheckWakeup(EcuM_WakeupSourceType)
[Timeout]
CheckWakeupTimer expired()
alt JobResult
SPI_Job1EndNotification()
[Wake Up]
[No Wake Up]
WAKEUP VALIDATION

Figure 1-8 Possible results for asynchronous wake-up concept

1.4.2 Configuring Synchronous and Asynchronous Wake-Up

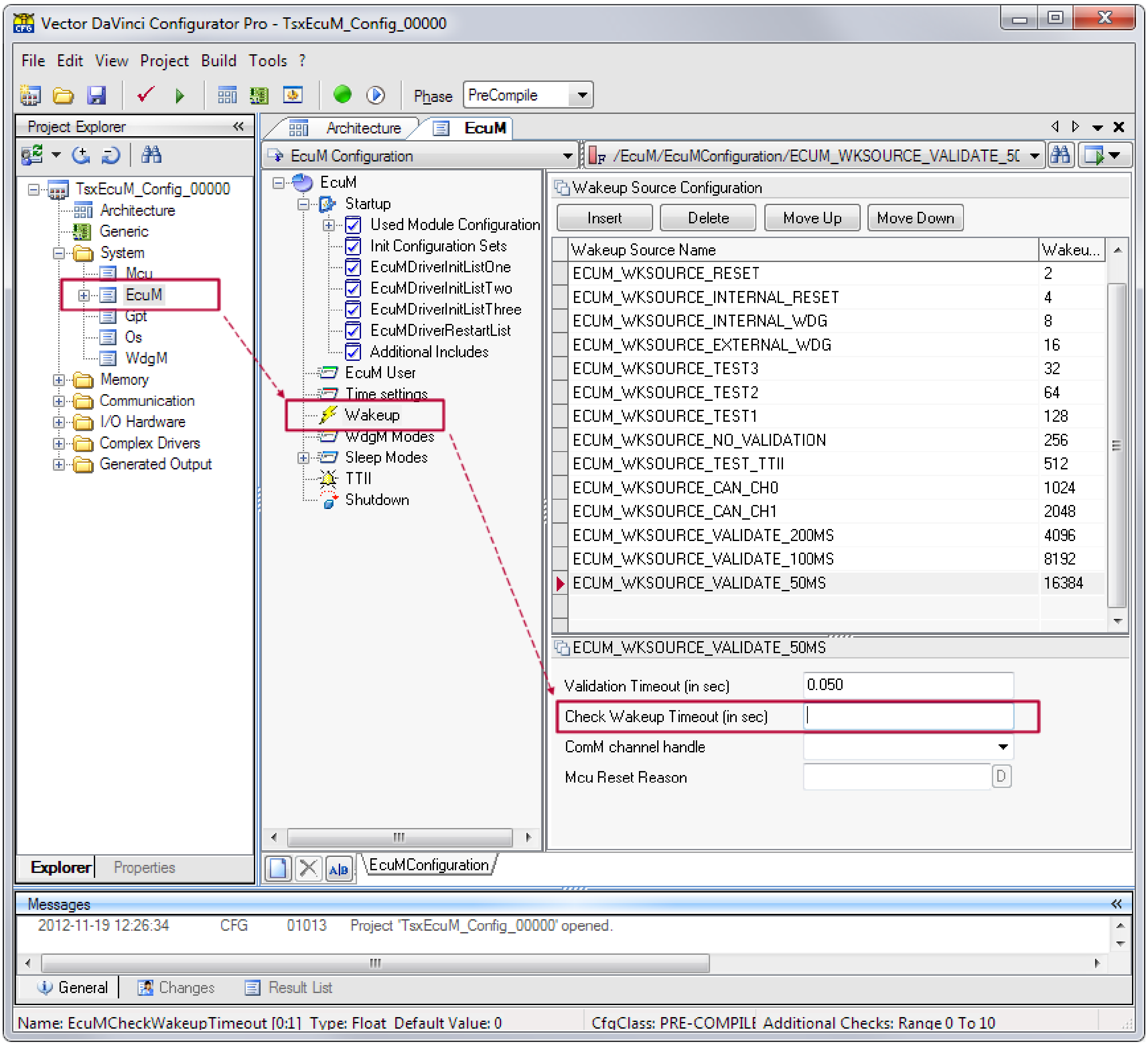
To get the asynchronous wake-up handling just enter a value in the field **Check Wakeup timeout (in sec)**.

Note
It depends on your project whether you do the configuration of ECUM in GENy or in DaVinci Configurator Pro. Both ways are shown in the following screen shots.



	Name	Validation Timeout [s]	Wakeup Source Id	ComM Channel Ref	Reset Reason	Check Wakeup Timeout (in sec)
ECUM_WKSOURCE_POWER	ECUM_WKSOURCE_POWER	0*	1	*	-1*	0*
ECUM_WKSOURCE_RESET	ECUM_WKSOURCE_RESET	0*	2	*	-1*	0*
ECUM_WKSOURCE_INTERNAL_RES	ECUM_WKSOURCE_INTERNAL_RESET	0*	4	*	-1*	0*
ECUM_WKSOURCE_INTERNAL_WD	ECUM_WKSOURCE_INTERNAL_WDG	0*	8	*	-1*	0*
ECUM_WKSOURCE_EXTERNAL_W	ECUM_WKSOURCE_EXTERNAL_WDG	0*	16	*	-1*	0*
CHNL	CHNL	0*	32	*	-1*	0*

Figure 1-9 Configuration using GENy.



Wakeup Source Name	Wakeu...
ECUM_WKSOURCE_RESET	2
ECUM_WKSOURCE_INTERNAL_RESET	4
ECUM_WKSOURCE_INTERNAL_WDG	8
ECUM_WKSOURCE_EXTERNAL_WDG	16
ECUM_WKSOURCE_TEST3	32
ECUM_WKSOURCE_TEST2	64
ECUM_WKSOURCE_TEST1	128
ECUM_WKSOURCE_NO_VALIDATION	256
ECUM_WKSOURCE_TEST_TTII	512
ECUM_WKSOURCE_CAN_CH0	1024
ECUM_WKSOURCE_CAN_CH1	2048
ECUM_WKSOURCE_VALIDATE_200MS	4096
ECUM_WKSOURCE_VALIDATE_100MS	8192
ECUM_WKSOURCE_VALIDATE_50MS	16384

ECUM_WKSOURCE_VALIDATE_50MS

Validation Timeout (in sec) 0.050

Check Wakeup Timeout (in sec)

ComM channel handle

Mcu Reset Reason

```
2012-11-19 12:26:34    CFG    01013    Project 'TsxEcuM_Config_00000' opened.
```

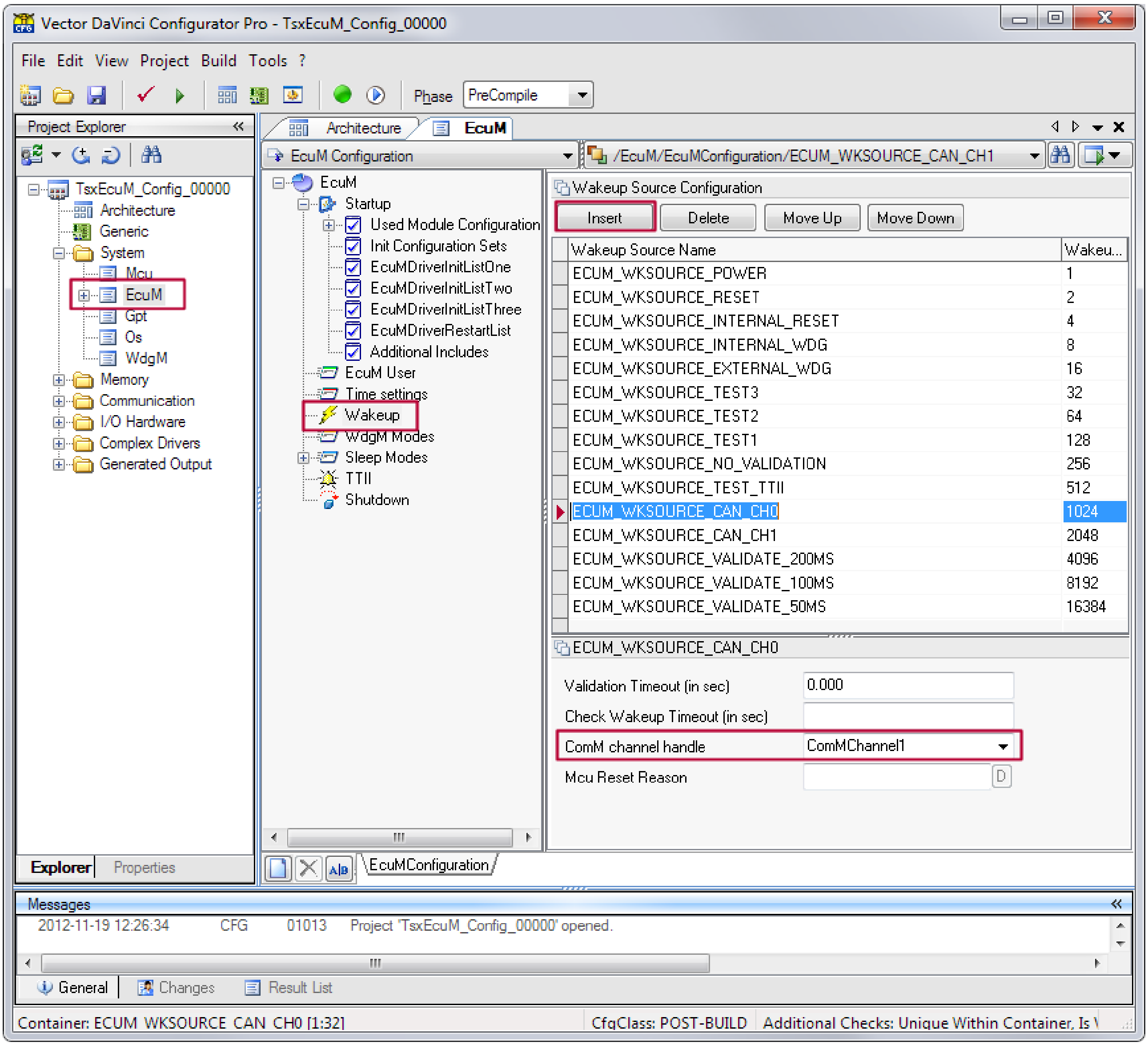
Name: EcuMCheckWakeupTimeout [0:1] Type: Float Default Value: 0 | CfgClass: PRE-COMPILE Additional Checks: Range 0 To 10

Figure 1-10 Configuration using DaVinci Developer

2 Wake-up Event

2.1 ECUM Configuration

This example shows the configuration of the ECUM with the MICROSAR Configurator Pro. It is also possible to configure the ECUM with GENy.



Wakeup Source Name	Wakeu...
ECUM_WKSOURCE_POWER	1
ECUM_WKSOURCE_RESET	2
ECUM_WKSOURCE_INTERNAL_RESET	4
ECUM_WKSOURCE_INTERNAL_WDG	8
ECUM_WKSOURCE_EXTERNAL_WDG	16
ECUM_WKSOURCE_TEST3	32
ECUM_WKSOURCE_TEST2	64
ECUM_WKSOURCE_TEST1	128
ECUM_WKSOURCE_NO_VALIDATION	256
ECUM_WKSOURCE_TEST_TTII	512
ECUM_WKSOURCE_CAN_CH0	1024
ECUM_WKSOURCE_CAN_CH1	2048
ECUM_WKSOURCE_VALIDATE_200MS	4096
ECUM_WKSOURCE_VALIDATE_100MS	8192
ECUM_WKSOURCE_VALIDATE_50MS	16384

ECUM_WKSOURCE_CAN_CH0

Validation Timeout (in sec) 0.000

Check Wakeup Timeout (in sec)

ComM channel handle ComMChannel1

Mcu Reset Reason

```
2012-11-19 12:26:34    CFG    01013    Project 'TsxEcuM_Config_00000' opened.
```

Container: ECUM_WKSOURCE_CAN_CH0 [1:32] | CfgClass: POST-BUILD | Additional Checks: Unique Within Container, Is \

Figure 2-1 Configuration of the wake-up source IDs

If the wake-up source needs to be validated enter a value greater than zero in the field **Validation Timeout**.

Create an identifier for each of your wake-up sources. The **Wake-up Source Name** has to be a valid C name.

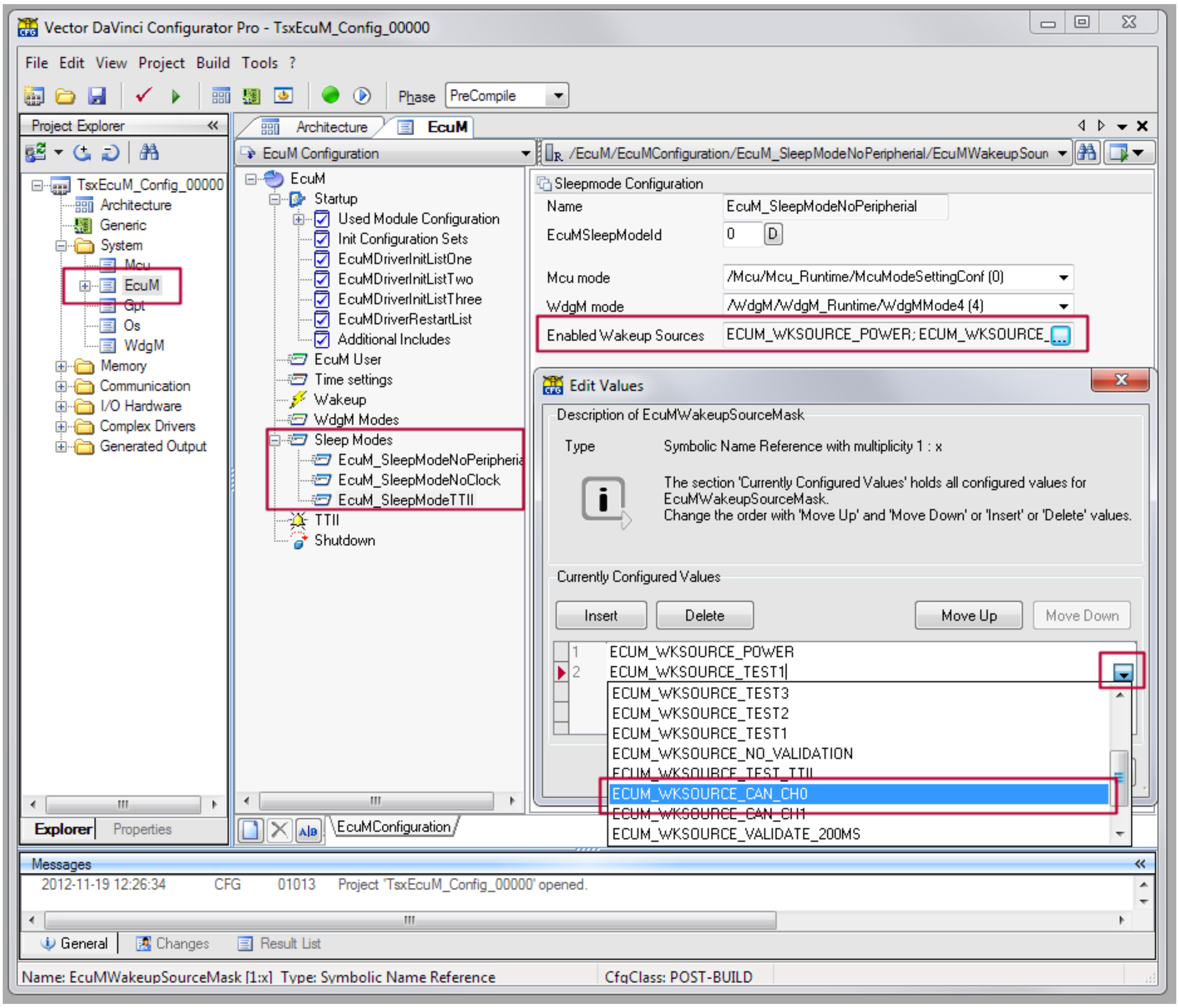
Select the according user handle of the COMM (**ComM channel handle**).

Info

The **Wakeup Source ID** has to be a multiple of two (2^n) and each ID has to be unique.

Info

The wake-up source belongs to the network (not to the transceiver or controller). In the example shown in Figure 2-1 the **Wake-up Source ID 64** is used for the first network **CAN0** independent if the wake-up event is triggered by the CAN transceiver or the CAN controller.



Vector DaVinci Configurator Pro - TsxEcuM_Config_00000

Phase: PreCompile

/EcuM/EcuMConfiguration/EcuM_SleepModeNoPeripherial/EcuMWakeupSoun

Sleepmode Configuration

Name: EcuM_SleepModeNoPeripherial

EcuMSleepModeId: 0

Mcu mode: /Mcu/Mcu_Runtime/McuModeSettingConf (0)

WdgM mode: /WdgM/WdgM_Runtime/WdgMMode4 (4)

Enabled Wakeup Sources: ECUM_WKSOURCE_POWER; ECUM_WKSOURCE_

Edit Values

Description of EcuMWakeupSourceMask

Type: Symbolic Name Reference with multiplicity 1 : x

The section 'Currently Configured Values' holds all configured values for EcuMWakeupSourceMask.
Change the order with 'Move Up' and 'Move Down' or 'Insert' or 'Delete' values.

Currently Configured Values

1 ECUM_WKSOURCE_POWER
2 ECUM_WKSOURCE_TEST1

```
ECUM_WKSOURCE_TEST3
ECUM_WKSOURCE_TEST2
ECUM_WKSOURCE_TEST1
ECUM_WKSOURCE_NO_VALIDATION
ECUM_WKSOURCE_TEST_TTII
ECUM_WKSOURCE_CAN_CH0
ECUM_WKSOURCE_CAN_CH1
ECUM_WKSOURCE_VALIDATE_200MS
```

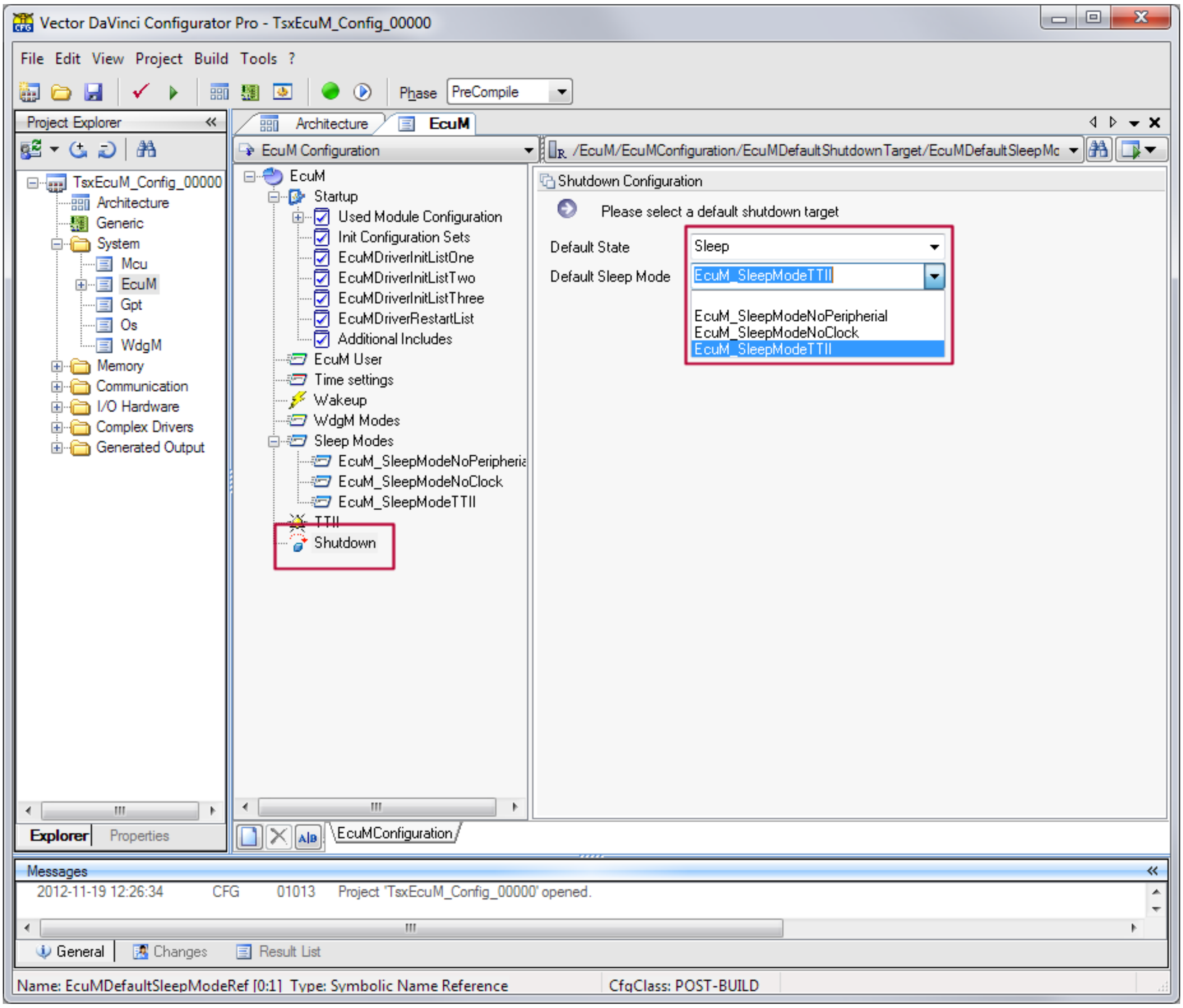
Messages

```
2012-11-19 12:26:34    CFG    01013    Project 'TsxEcuM_Config_00000' opened.
```

Name: EcuMWakeupSourceMask [1:x] Type: Symbolic Name Reference   CfgClass: POST-BUILD

Figure 2-2 Configuration of the ECUM Sleep mode

The ECUM performs the enabling of the wake-up source if the according channel enters the sleep mode. Add all needed wake-up sources to the list of the **Enabled Wakeup Sources**.



Please select a default shutdown target

Default State: Sleep

Default Sleep Mode: EcuM_SleepModeTTII

```
2012-11-19 12:26:34    CFG    01013    Project 'TsxEcuM_Config_00000' opened.
```

Name: EcuMDefaultSleepModeRef [0:1] Type: Symbolic Name Reference   CfgClass: POST-BUILD

Figure 2-3 Configuration of the ECUM shut down

This parameter specifies the sleep mode which should be selected when the default shutdown target state is **Sleep** see Figure 2-3. Select the name of the sleep mode which contains the list of your wake-up sources which has been configured in Figure 2-2.

The following callouts are used to enable and to disable the wake-up sources selected by the current sleep mode and to provide a specific reaction handling when a wake-up event has occurred.

- `EcuM_EnableWakeupSources()`
- `EcuM_DisableWakeupSources()`
- `EcuM_CheckWakeup()`
- `EcuM_StartWakeupSources()`

- `EcuM_StopWakeupSources()`
- `EcuM_CheckValidation()`

Info
In AUTOSAR R 2.1 the modules are specified to call EcuM_SetWakeupEvent API for correct wake-up detection.
In AUTOSAR 3.0 the modules should call EcuM_CheckWakeup. The ICU specification is inconsistent at this point. Please refer to the specific driver module documentation

The following parts are only necessary if the wake-up source shall also be able to wake up the ECU (see “Wake-up Capability” **Figure 2-4**).

Example

```
void EcuM_EnableWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* Enable CAN transceiver wakeup */
    CanIf_SetTransceiverWakeupMode(0, CANIF_TRCV_WU_ENABLE);
    /* Enable ECU wakeup which is activated in the configuration via
the “Wakeup Capability”. Parameter is the "ChannelName" of the
"IcuChannel"  */
    Icu_EnableWakeup(Icu_CAN0_TrcvWakeUp);
    /* Reduced power operation. Only those notifications are available
which are configured as wakeup capable */
    Icu_SetMode(ICU_MODE_SLEEP);
  }
  if ((wakeupSource & ECUM_WKSOURCE_LIN0) != 0)
  {
    /* Enable ECU wakeup which is activated in the configuration via
the “Wakeup Capability”. Parameter is the "ChannelName" of the
"IcuChannel"  */
    Icu_EnableWakeup(Icu_LIN0_TrcvWakeUp);
    /* Reduced power operation. Only those notifications are available
which are configured as wakeup capable */
    Icu_SetMode(ICU_MODE_SLEEP);
  }
  if ((wakeupSource & ECUM_WKSOURCE_FR) != 0)
  {
    /* Enable ECU wakeup which is activated in the configuration via
the “Wakeup Capability”. Parameter is the "ChannelName" of the
"IcuChannel"  */
    Icu_EnableWakeup(Icu_FlexRay_WakeUp);
    /* Reduced power operation. Only those notifications are available
which are configured as wakeup capable */
    Icu_SetMode(ICU_MODE_SLEEP);
  }
}
```

If the ECU enters the sleep mode, the wake-up sources which shall be able so wake the ECU has to be enabled. Enable the CAN transceiver wake-up as wake-up source if it should be the wake-up source. Enable the **Wake-up Capability** of the ICU channel via the function Icu_EnableWakeup with the ICU **ChannelName** of the **IcuChannel** as parameter (see Figure 2-4). Set the ICU in reduced power operation via Icu_SetMode.

Example

```
void EcuM_DisableWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* Disable ECU wakeup which is activated in the configuration via
the “Wakeup Capability”. Parameter is the "ChannelName" of the
"IcuChannel"  */
    Icu_DisableWakeup(Icu_CAN0_TrcvWakeUp);
    /* Normal operation, all used interrupts are enabled according to
the notification requests */
    Icu_SetMode(ICU_MODE_NORMAL);
    /* Disable CAN transceiver wakeup */
    CanIf_SetTransceiverWakeupMode(0, CANIF_TRCV_WU_DISABLE);
  }
  if ((wakeupSource & ECUM_WKSOURCE_LIN0) != 0)
  {
    /* Disable ECU wakeup which is activated in the configuration via
the “Wakeup Capability”. Parameter is the "ChannelName" of the
"IcuChannel"  */
    Icu_DisableWakeup(Icu_LIN0_TrcvWakeUp);
    /* Normal operation, all used interrupts are enabled according to
the notification requests */
    Icu_SetMode(ICU_MODE_NORMAL);
  }
  if ((wakeupSource & ECUM_WKSOURCE_FR) != 0)
  {
    /* Disable ECU wakeup which is activated in the configuration via
the “Wakeup Capability”. Parameter is the "ChannelName" of the
"IcuChannel"  */
    Icu_DisableWakeup(Icu_FlexRay_WakeUp);
    /* Normal operation, all used interrupts are enabled according to
the notification requests */
    Icu_SetMode(ICU_MODE_NORMAL);
  }
}
```

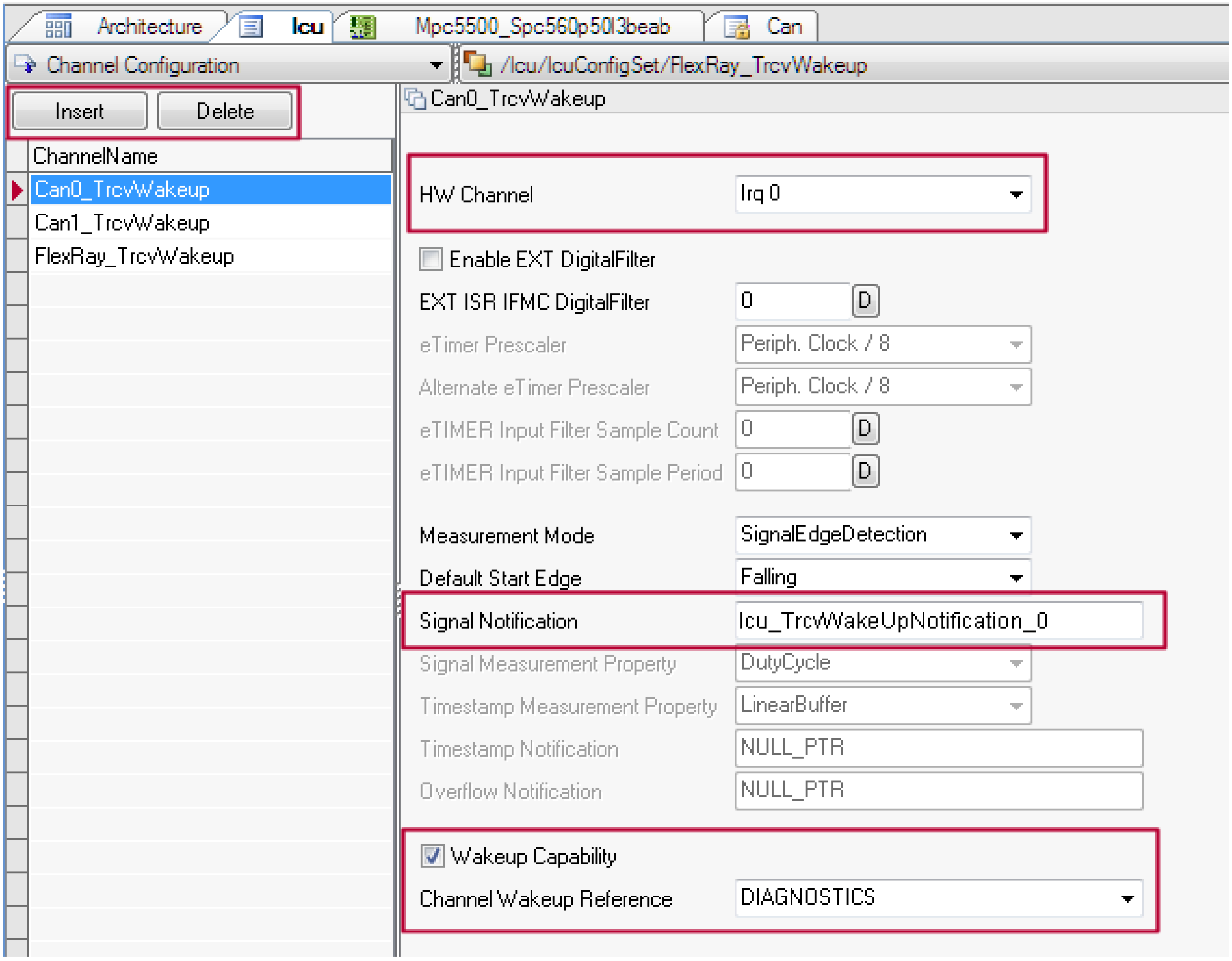
If the ECU wakes up the wakeup source has to be disabled. Disable the CAN transceiver wakeup for the wake-up source if it is used as wake-up source. Disable the **Wakeup Capability** of the ICU channel via the function Icu_DisableWakeup with the ICU **ChannelName** of the **IcuChannel** as parameter (see Figure 2-4). Set the ICU to `NORMAL` operation via Icu_SetMode.

2.2 ICU Interrupt (µC I/O port)

The proposal described in this chapter is using an additional µC I/O port to generate the wake-up information via the ICU driver. This I/O port is connected, e.g. to a Rx port of the wake-up source. The shown example pictures may differ, dependent on the hardware or the used configuration tool.

Add an IcuChannel (via **[Insert]**) for each of your wake-up sources which

- are not in polling mode
- do not have an interrupt function of its own and
- use the service of the ICU.



ChannelName
Can0_TrcvWakeup
Can1_TrcvWakeup
FlexRay_TrcvWakeup

HW Channel: Irq 0

□ Enable EXT DigitalFilter

EXT ISR IFMC DigitalFilter: 0

eTimer Prescaler: Periph. Clock / 8

Alternate eTimer Prescaler: Periph. Clock / 8

eTIMER Input Filter Sample Count: 0

eTIMER Input Filter Sample Period: 0

Measurement Mode: SignalEdgeDetection

Default Start Edge: Falling

Signal Notification: Icu_TrcvWakeUpNotification_0

Signal Measurement Property: DutyCycle

Timestamp Measurement Property: LinearBuffer

Timestamp Notification: NULL_PTR

Overflow Notification: NULL_PTR

☑ Wakeup Capability

Channel Wakeup Reference: DIAGNOSTICS

Figure 2-4 Configuration of the wake-up settings of a ICU channel

- Choose the **Hw Channel** which matches to the wake-up source.
- If the wake-up event shall be able to wake up the ECU (and not only the communication channel) it is necessary to activate the **Wake-up Capability** for this ICU channel, otherwise the wake-up event will be ignored when the ICU is set to ICU_MODE_SLEEP mode. If the **Signal Notification** function is NOT used then also activate **Report Wakeup Source** in the tab **General settings** (that the ICU triggers EcuM_CheckWakeup(“**Channel Wakeup Info**”) if a wakeup event occurs).
- In the **Channel Wake-up Info** drop down list choose the identifier which matches to the **Wakeup Source Name** set in the ECUM.
- Create a name for the **Signal Notification** function.

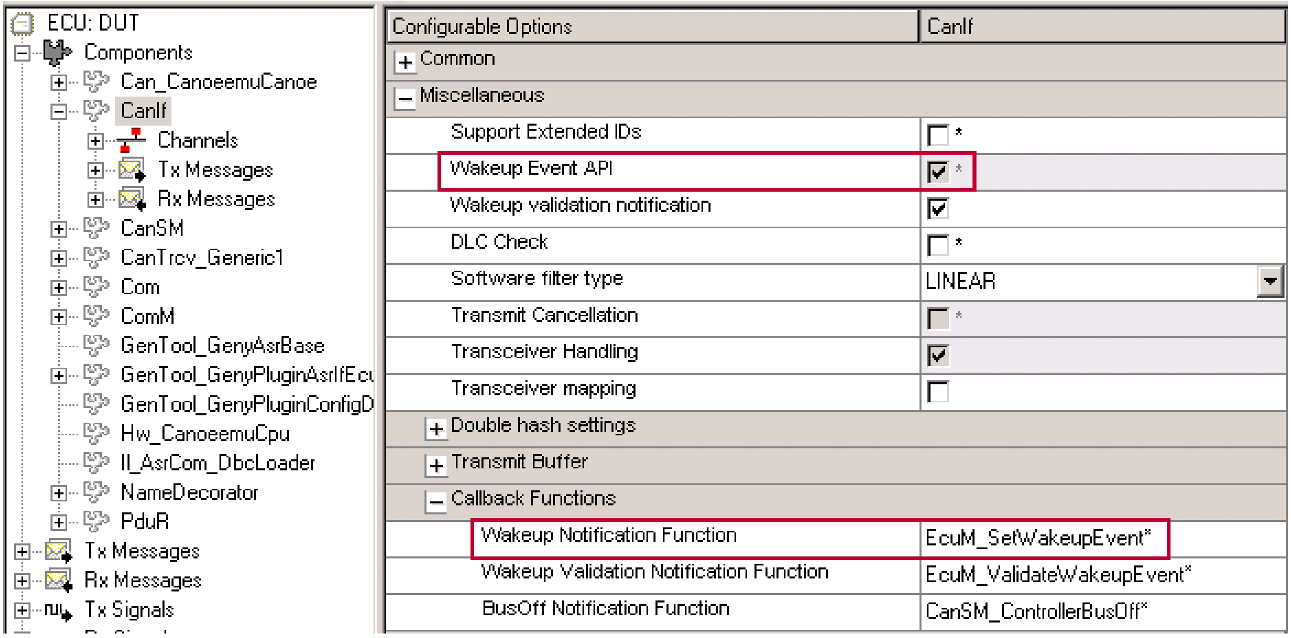
The user has to implement the signal notification function.

Example

```
void "Signal Notification" (void)
{
  EcuM_CheckWakeup("Wakeup Source Name");
}
```

2.3 CAN

2.3.1 General



- ECU: DUT
  - Components
    - Can_CanoeemuCanoe
    - CanIf
      - Channels
      - Tx Messages
      - Rx Messages
    - CanSM
    - CanTrcv_Generic1
    - Com
    - ComM
    - GenTool_GenyAsrBase
    - GenTool_GenyPluginAsrIfEcı
    - GenTool_GenyPluginConfigD
    - Hw_CanoeemuCpu
    - Il_AsrCom_DbcLoader
    - NameDecorator
    - PduR
  - Tx Messages
  - Rx Messages
  - Tx Signals

Configurable Options	CanIf
+ Common	
− Miscellaneous	
Support Extended IDs	☐ *
Wakeup Event API	☑ *
Wakeup validation notification	☑
DLC Check	☐ *
Software filter type	LINEAR
Transmit Cancellation	☐ *
Transceiver Handling	☑
Transceiver mapping	☐
+ Double hash settings	
+ Transmit Buffer	
− Callback Functions	
Wakeup Notification Function	EcuM_SetWakeupEvent*
Wakeup Validation Notification Function	EcuM_ValidateWakeupEvent*
BusOff Notification Function	CanSM_ControllerBusOff*

Figure 2-5 General CAN configuration in GENy

Enable the **Wake-up Event API** and enter the name of the **Wake-up Notification Function**. Here the notification EcuM_SetWakeupEvent of the ASR ECUM is used.

2.3.2 CAN Wake-up by Transceiver

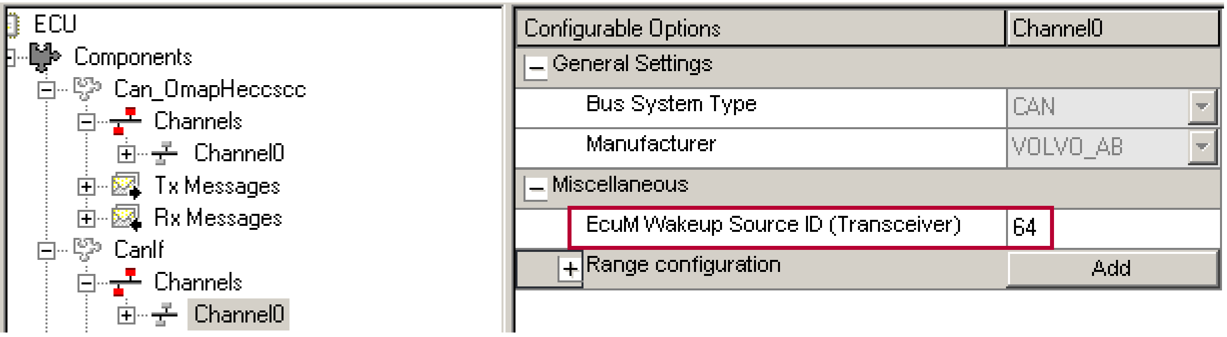
The wake-up information is generated via the ICU driver (see chapter 2.2) by using an additional µC I/O port. This I/O port is connected to the CAN transceiver pin which indicates the wake-up.

Example

If a TJA1041(A) CAN transceivers is used, connect the ERRN port with the µC I/O port. The RXD pin of the transceiver also indicates a wake-up and could be also used. But the advantage of the ERRN is that it distinguishes between a local wake-up (via WAKE pin) a remote wake-up (via CAN bus) (see [5]). A HIGH signal indicates a remote wake-up.

Info

In dependency of the provided ICU configuration options it is possible to configure the **IcuDefaultStartEdge**. This option should be configured to ICU_FALLING_EDGE, because the wake-up event is provided by the Rx pin and/or the ERRN pin via a level change from recessive to dominant.

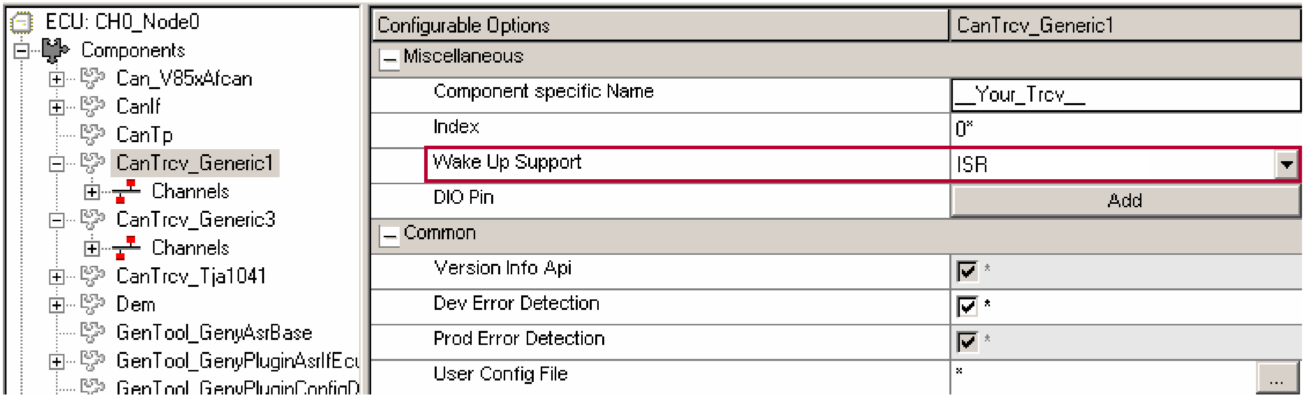


ECU
- Components
  - Can_OmapHeccscc
    - Channels
      - Channel0
    - Tx Messages
    - Rx Messages
  - CanIf
    - Channels
      - Channel0

Configurable Options	Channel0
General Settings	
Bus System Type	CAN
Manufacturer	VOLVO_AB
Miscellaneous	
EcuM Wakeup Source ID (Transceiver)	64
Range configuration	Add

Figure 2-6 Configuration of the CAN transceiver wake-up source

Enter the **Transceiver WakeUpSource** in the CAN interface. The value has to match to the **Wakeup Source ID** in the ECUM (cp. Chapter 2.1). Enter 0 (zero) if the channel should not be woken up by the transceiver. Set the wake-up source ID of the controller to zero as shown in Figure 2-10, .

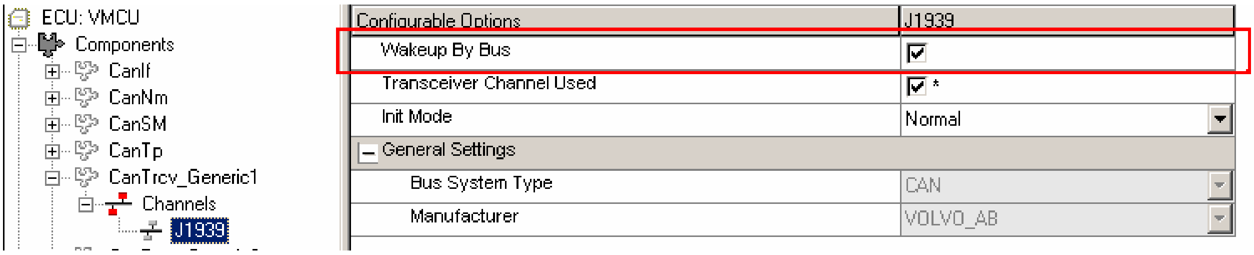


ECU: CH0_Node0
- Components
  - Can_V85xAfcan
  - CanIf
  - CanTp
  - CanTrcv_Generic1
    - Channels
  - CanTrcv_Generic3
    - Channels
  - CanTrcv_Tja1041
  - Dem
  - GenTool_GenyAsrBase
  - GenTool_GenyPluginAsrIfEcu
  - GenTool_GenyPluginConfigD

Configurable Options	CanTrcv_Generic1
Miscellaneous	
Component specific Name	__Your_Trcv__
Index	0*
Wake Up Support	ISR
DIO Pin	Add
Common	
Version Info Api	☑ *
Dev Error Detection	☑ *
Prod Error Detection	☑ *
User Config File	* ...

Figure 2-7 Selection of the desired interrupt service for the CAN transceiver wake-up

Select ISR for the **Wake Up Support**.



ECU: VMCU
- Components
  - CanIf
  - CanNm
  - CanSM
  - CanTp
  - CanTrcv_Generic1
    - Channels
      - J1939

Configurable Options	J1939
Wakeup By Bus	☑
Transceiver Channel Used	☑ *
Init Mode	Normal
General Settings	
Bus System Type	CAN
Manufacturer	VOLVO_AB

Figure 2-8 Activation of the CAN wake-up reason type in the configuration tool

Enable the feature **Wakeup By Bus** in the channel view of the CanTrcv.

The ICU triggers the notification function. This informs the ECUM about the wake-up interrupt by calling EcuM_CheckWakeup with all according wake-up sources as parameter.

Example

```
void Icu_TrcvWakeUpNotification_0 (void)
{
  /* inform the EcuM about the wakeup event, the parameter is the
configured transceiver wakeup source */
  EcuM_CheckWakeup(ECUM_WKSOURCE_CAN0);
}
```

The EcuM does not know if the function EcuM_CheckWakeup has been called by the wakeup source itself. Therefore, the EcuM_CheckWakeup has to ask the driver of the wakeup source if it was responsible for that wakeup. Add the CanIf_CheckWakeup function of the CAN bus with the according wake-up source as parameter.

Example

```
void EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* ask the driver of the wakeup source if it was responsible for
the wakeup */
   CanIf_CheckWakeup(ECUM_WKSOURCE_CAN0);
  }
}
```

To avoid wake-up events while the communication channel is in normal mode, it is necessary to disable the notification if the transceiver enters normal mode and to re-enable the notification if the transceiver enters standby mode. Adapt the CanTrcv_<TrcvName>_SetOpMode function according the example.

Example

```
Std_ReturnType CanTrcv_30_<TrcvName>_SetOpMode(CanIf_TransceiverModeType
OpMode, uint8 CanTrcvIndex)
{
  switch(OpMode)
    case CANTRCV_30_<TrcvName>_OP_MODE_NORMAL:
      /* Disable wakeup interrupt on normal operation mode */
      Icu_DisableNotification(Icu_CAN0_TrcvWakeUp);
    case CANTRCV_30_<TrcvName>_OP_MODE_STANDBY:
      Icu_EnableNotification(Icu_CAN0_TrcvWakeUp);
      case CANTRCV_30_<TrcvName>_OP_MODE_SLEEP
        Icu_EnableNotification(Icu_CAN0_TrcvWakeUp);
        /* example for TJA1041
Dio_WriteChannel(CanTrcv_30_xChannel[CanTrcvIndex].CanTrcvPinEN,STD_LOW);

Dio_WriteChannel(CanTrcv_30_xChannel[CanTrcvIndex].CanTrcvPinSTB,STD_LOW);
*/
}
```

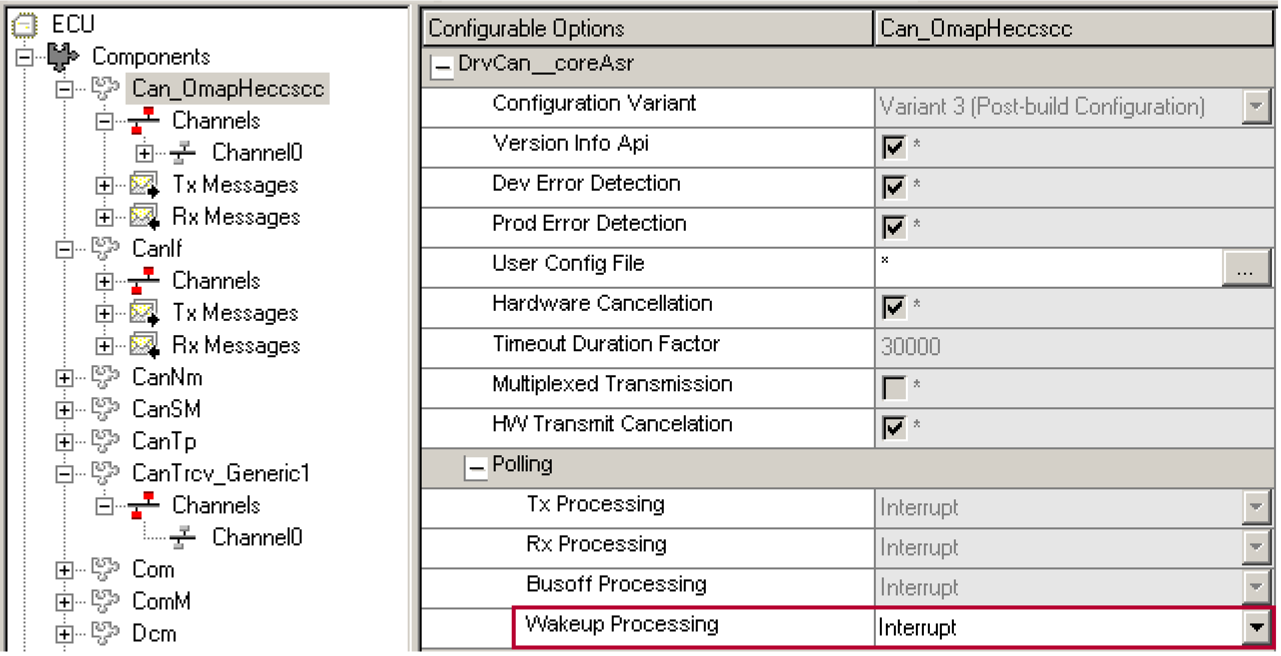
Caution
The ICU notification function activation, via `Icu_EnableNotification(...)`, has to be called before the transceiver HW is set into the standby mode (e.g. via `Dio_WriteChannel(..)`). Refer to the example above.

Caution
The function call Icu_DisableNotification does not disable the interrupts! The ICU notification functionality is one level above the interrupt handling. This is important if the RX line of the transceiver is connected to the ICU interrupt pin. With such a configuration, CAN communication would lead to a high µC load because of many interrupts on the ICU channel.

A possible solution is to also disable/enable the interrupt.

The CAN bus may be woken up by an EMC disturbance. In this case it is not necessary to wake up the bus or switch the ECU in RUN mode. To eliminate an unwanted wake-up the ECUM waits for a valid CAN message. The next step in the wake-up sequence is the CAN Wake-up Validation, see chapter 2.3.5, if **Validation Timeout** in the ECUM configuration in Figure 2-1 is greater than zero.

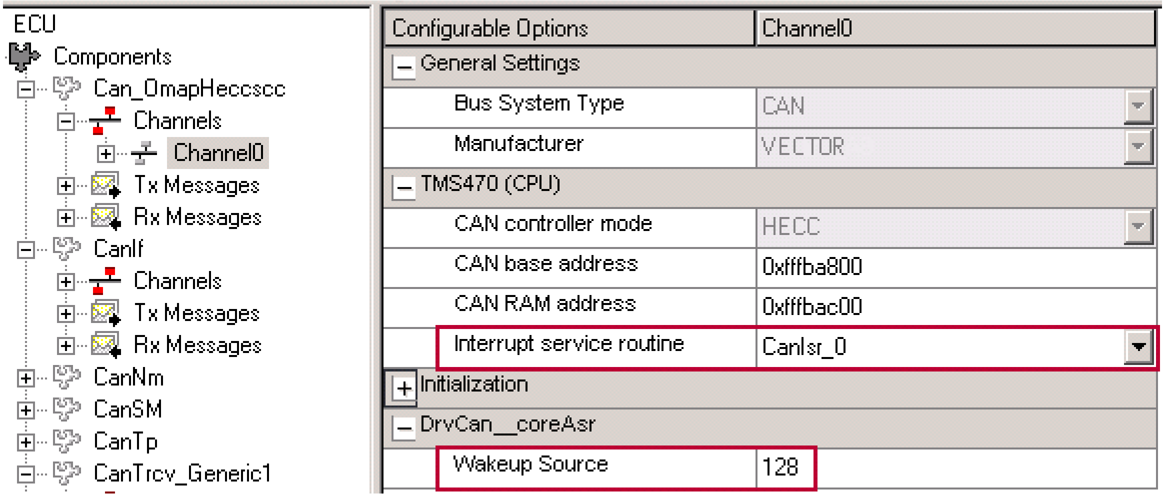
2.3.3 CAN Wake-up by Controller



Configurable Options	Can_OmapHeccscc
DrvCan__coreAsr	
Configuration Variant	Variant 3 (Post-build Configuration)
Version Info Api	☑ *
Dev Error Detection	☑ *
Prod Error Detection	☑ *
User Config File	*
Hardware Cancellation	☑ *
Timeout Duration Factor	30000
Multiplexed Transmission	□ *
HW Transmit Cancelation	☑ *
Polling	
Tx Processing	Interrupt
Rx Processing	Interrupt
Busoff Processing	Interrupt
Wakeup Processing	Interrupt

Figure 2-9 Selection of the CAN wake-up processing type interrupt

Select the Interrupt mode for the **Wakeup Processing**.



Configurable Options	Channel0
General Settings	
Bus System Type	CAN
Manufacturer	VECTOR
TMS470 (CPU)	
CAN controller mode	HECC
CAN base address	0xfffba800
CAN RAM address	0xfffbac00
Interrupt service routine	CanIsr_0
Initialization	
DrvCan__coreAsr	
Wakeup Source	128

Figure 2-10 Selection of the desired interrupt service and configuration of the CAN controller wake-up source

Enter the **Wakeup Source** in the channel view of the CAN driver. The value has to match to the **Wakeup Source ID** in the ECUM (cp. Chapter 2.1). Enter 0 (zero) if the channel should not be woken up by the controller. Set the wake-up source ID of the transceiver to zero as shown in Figure 2-6.

Choose the related interrupt service function for the CAN controller. This function has to be called by the interrupt dispatcher after identifying the interrupt source which belongs to the CAN controller.

```
void CanIsr_0( void )
{
  CanInterrupt(kCanPhysToLogChannelIndex_0);
}
```

The CAN Driver will detect an interrupt in the CanInterrupt function and triggers the EcuM_CheckWakeup with the according wake-up source ID of the controller as parameter.

The EcuM does not know if the function EcuM_CheckWakeup has been called by the wakeup source itself. Therefore, the EcuM_CheckWakeup has to ask the driver of the wakeup source if it was responsible for that wakeup. Add the CanIf_CheckWakeup function of the CAN bus with the according wake-up source as parameter.

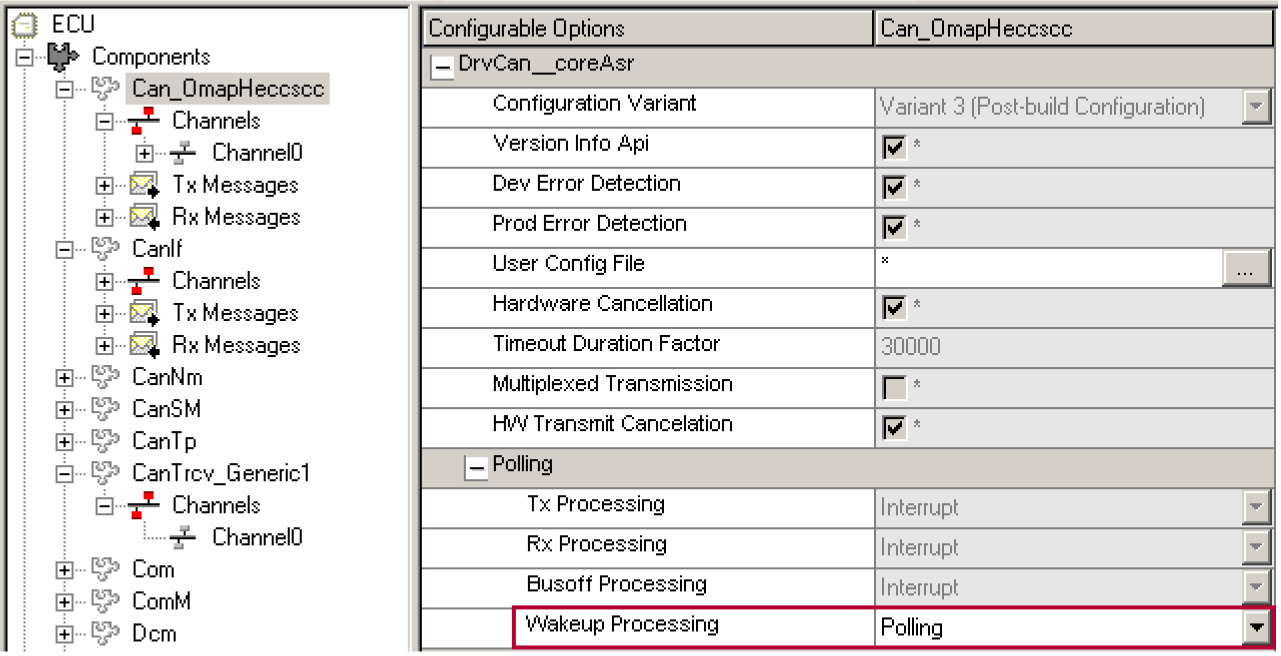
Example

```
void EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
   if ((wakeupSource & ECUM_WKSOURCE_CAN1) != 0)
   {
      /* ask the driver of the wakeup source if it was responsible for
the wakeup */
      CanIf_CheckWakeup(ECUM_WKSOURCE_CAN1);
   }
}
```

The CAN bus may be woken up by an EMC disturbance. In this case it is not necessary to wake up the bus or switch the ECU in RUN mode. To eliminate an unwanted wake-up the

ECUM waits for a valid CAN message. The next step in the wake-up sequence is the CAN Wake-up Validation, see chapter 2.3.5, if **Validation Timeout** in the ECUM configuration in Figure 2-1 is greater than zero.

2.3.4 CAN Wakeup by Polling the CAN Driver



- ECU
  - Components
    - Can_OmapHeccscc
      - Channels
        - Channel0
      - Tx Messages
      - Rx Messages
    - CanIf
      - Channels
      - Tx Messages
      - Rx Messages
    - CanNm
    - CanSM
    - CanTp
    - CanTrcv_Generic1
      - Channels
        - Channel0
    - Com
    - ComM
    - Dcm

Configurable Options	Can_OmapHeccscc
DrvCan__coreAsr	
Configuration Variant	Variant 3 (Post-build Configuration)
Version Info Api	☑ *
Dev Error Detection	☑ *
Prod Error Detection	☑ *
User Config File	*
Hardware Cancellation	☑ *
Timeout Duration Factor	30000
Multiplexed Transmission	☐ *
HW Transmit Cancelation	☑ *
Polling	
Tx Processing	Interrupt
Rx Processing	Interrupt
Busoff Processing	Interrupt
Wakeup Processing	Polling

Figure 2-11 Selection of the CAN wake-up processing type polling

Select the polling mode for the **Wakeup Processing**. The SCHM will call the Can_MainFunction_Wakeup cyclically. In this function the CAN driver detects if a wake-up is pending at any CAN controller or CAN transceiver and trigger the EcuM_CheckWakeup with the according wake-up source ID of the controller as parameter.

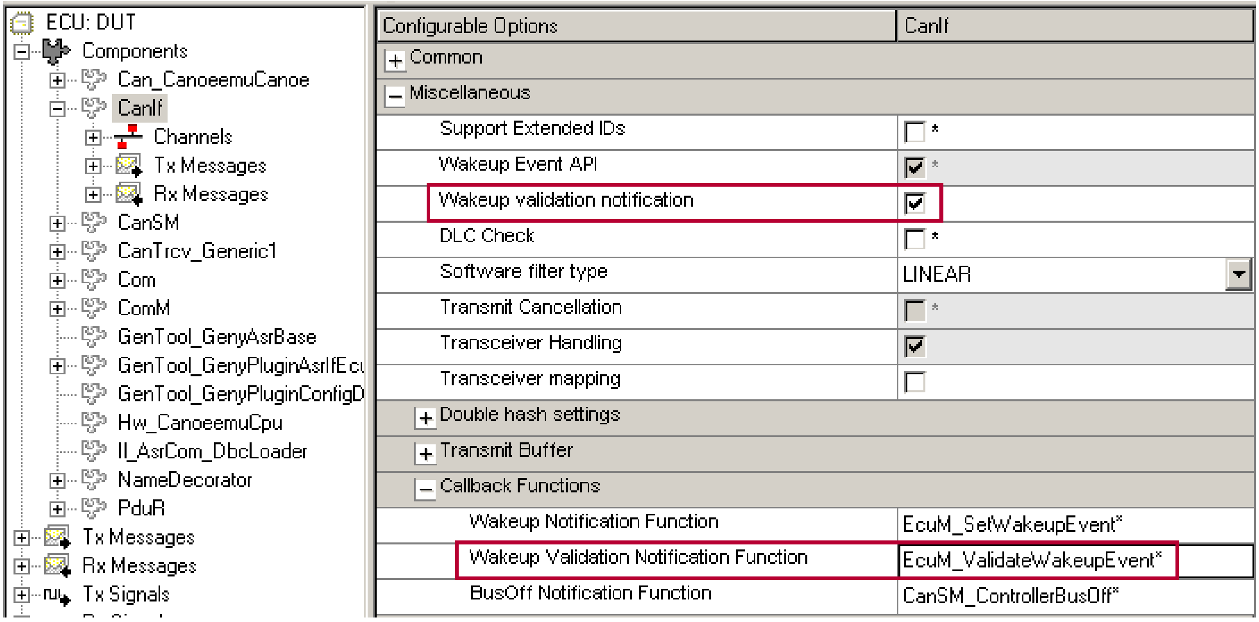
The EcuM does not know if the function EcuM_CheckWakeup has been called by the wakeup source itself. Therefore, the EcuM_CheckWakeup has to ask the driver of the wakeup source if it was responsible for that wakeup. Add the CanIf_CheckWakeup function of the CAN bus with the according wake-up source as parameter.

Example

```
void EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* ask the driver of the wakeup source if it was responsible for
the wakeup */
    CanIf_CheckWakeup(ECUM_WKSOURCE_CAN0);
  }
  if ((wakeupSource & ECUM_WKSOURCE_CAN1) != 0)
  {
    /* ask the driver of the wakeup source if it was responsible for
the wakeup */
    CanIf_CheckWakeup(ECUM_WKSOURCE_CAN1);
  }
}
```

The CAN bus may be woken up by an EMC disturbance. In this case it is not necessary to wake up the bus or switch the ECU in RUN mode. To eliminate an unwanted wake-up the ECUM waits for a valid CAN message. The next step in the wake-up sequence is the CAN Wake-up Validation, see chapter 2.3.5, if **Validation Timeout** in the ECUM configuration in Figure 2-1 is greater than zero.

2.3.5 CAN Wake-up Validation



ECU: DUT
- Components
  - Can_CanoeemuCanoe
  - CanIf
    - Channels
    - Tx Messages
    - Rx Messages
  - CanSM
  - CanTrcv_Generic1
  - Com
  - ComM
  - GenTool_GenyAsrBase
  - GenTool_GenyPluginAsrIfEcu
  - GenTool_GenyPluginConfigD
  - Hw_CanoeemuCpu
  - Il_AsrCom_DbcLoader
  - NameDecorator
  - PduR
- Tx Messages
- Rx Messages
- Tx Signals

Configurable Options	CanIf
+ Common	
– Miscellaneous	
Support Extended IDs	□ *
Wakeup Event API	☑ *
Wakeup validation notification	☑
DLC Check	□ *
Software filter type	LINEAR
Transmit Cancellation	□ *
Transceiver Handling	☑
Transceiver mapping	□
+ Double hash settings	
+ Transmit Buffer	
– Callback Functions	
Wakeup Notification Function	EcuM_SetWakeupEvent*
Wakeup Validation Notification Function	EcuM_ValidateWakeupEvent*
BusOff Notification Function	CanSM_ControllerBusOff*

Figure 2-12 Activation of the CAN wake-up validation and configuration of the validation function

Enable the **Wakeup validation notification** and enter the name of the **Wakeup Validation Notification Function**. Here the notification EcuM_ValidateWakeupEvent of the ECUM is used.

After the detection of the wake-up source in Ecu_CheckWakeup the wake-up validation will be performed. The ECUM triggers the callback which has to start the transceiver and controller of the wake-up source. Add the according function calls in the callback function EcuM_StartWakeupSources.

Example

```
void EcuM_StartWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
 ComM_ModeType CanMode = COMM_NO_COMMUNICATION;

 if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
 {
  /* determine in which is the current Can network state */
  (void)CanSM_GetCurrentComMode(0, &CanMode);
  if(COMM_NO_COMMUNICATION == CanMode)
  {
    /* set the controller and trcv mode into normal operation mode */
    CanIf_SetTransceiverMode(0 /* TrcvIdx */, CANIF_TRCV_MODE_NORMAL);
    CanIf_SetControllerMode (0 /* CtrlIdx */, CANIF_CS_STOPPED);
    CanIf_SetControllerMode (0 /* CtrlIdx */, CANIF_CS_STARTED);
  }
  else
  {
    /* stack already up and running */
  }
 }/* IF ECUM_WKSOURCE_CAN0*/


 if ((wakeupSource & ECUM_WKSOURCE_CAN1) != 0)
 {
  /* determine in which is the current Can Network state */
  (void)CanSM_GetCurrentComMode(1, &CanMode);
  if(COMM_NO_COMMUNICATION == CanMode)
  {
    /* set the controller and trcv mode into normal operation mode */
    CanIf_SetTransceiverMode(1 /* TrcvIdx */, CANIF_TRCV_MODE_NORMAL);
    CanIf_SetControllerMode (1 /* CtrlIdx */, CANIF_CS_STOPPED);
    CanIf_SetControllerMode (1 /* CtrlIdx */, CANIF_CS_STARTED);
  }
  else
  {
    /* stack already up and running */
  }
 }/* IF ECUM_WKSOURCE_CAN1*/
}
```

Alternatively with the latest CANSM the code could look like shown below. The CANIF calls can be replaced by `CanSM_StartWakeupSources(<Channel ID>)`.

Example for latest CANSM version

```
void EcuM_StartWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
 ComM_ModeType CanMode = COMM_NO_COMMUNICATION;

 if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
 {
  /* determine in which is the current Can network state */
  CanSM_StartWakeupSources(0);
 }/* IF ECUM_WKSOURCE_CAN0*/


 if ((wakeupSource & ECUM_WKSOURCE_CAN1) != 0)
 {
  /* determine in which is the current Can Network state */
  CanSM_StartWakeupSources(1);
 }/* IF ECUM_WKSOURCE_CAN1*/
}
```

The ECUM calls cyclically the validation function which has to trigger the channel specific validation of the wake-up source. CANIF indicates the passed validation by calling the function EcuM_ValidateWakeupEvent if it recognizes the successful reception of at least one message and the ECUM triggers the ComM_EcuM_WakeUpIndication to start the network.

Example

```
void EcuM_CheckValidation(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
    /* Query the driver if the wakeup event was valid */
    CanIf_CheckValidation(ECUM_WKSOURCE_CAN0);
  }
  if ((wakeupSource & ECUM_WKSOURCE_CAN1) != 0)
  {
    /* Query the driver if the wakeup event was valid */
    CanIf_CheckValidation(ECUM_WKSOURCE_CAN1);
  }
}
```

The validation fails if the CANIF could not recognize the successful reception of a message during the validation time. Due to the failed validation the ECUM initiates the shutdown of the network and sets the ECU back in the preceding state again. So the ECUM executes the GOSLEEP sequence (see [1]) or stays in RUN. Add the according function calls in the callback function EcuM_StopWakeupSources.

Example

```
void EcuM_StopWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  ComM_ModeType CanMode = COMM_NO_COMMUNICATION;

  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
   /* Validation was not successful*/
   (void)CanSM_GetCurrentComMode(0, &CanMode);
   if(COMM_NO_COMMUNICATION == CanMode)
   {
     /* Can channel is not started by the ECU internally, set the CAN
        controller and Transceiver back to sleep */
     CanIf_SetControllerMode(0 /* CtrlIdx */, CANIF_CS_STOPPED);
     CanIf_SetControllerMode(0 /* CtrlIdx */, CANIF_CS_SLEEP);
     CanIf_SetTransceiverMode(0 /*TrcvIdx*/, CANIF_TRCV_MODE_STANDBY);
   }
  }
  if ((wakeupSource & ECUM_WKSOURCE_CAN1) != 0)
  {
   /* Validation was not successful*/
   (void)CanSM_GetCurrentComMode(1, &CanMode);
   if(COMM_NO_COMMUNICATION == CanMode)
   {
     /* Can channel is not started by the ECU internally, set the CAN
        controller and Transceiver back to sleep */
     CanIf_SetControllerMode(1 /* CtrlIdx */, CANIF_CS_STOPPED);
     CanIf_SetControllerMode(1 /* CtrlIdx */, CANIF_CS_SLEEP);
     CanIf_SetTransceiverMode(1 /*TrcvIdx*/, CANIF_TRCV_MODE_STANDBY);
   }
  }
}
```

Alternatively with the latest CANSM, the code could look like shown below. The CANIF calls can be replaced by `CanSM_StopWakeupSources(<Channel ID>)`.

Example for latest CANSM version

```
void EcuM_StopWakeupSources(EcuM_WakeupSourceType wakeupSource)
{
  ComM_ModeType CanMode = COMM_NO_COMMUNICATION;

  if ((wakeupSource & ECUM_WKSOURCE_CAN0) != 0)
  {
   /* Validation was not successful*/
   CanSM_StopWakeupSources(0);
  }

  if ((wakeupSource & ECUM_WKSOURCE_CAN1) != 0)
  {
   /* Validation was not successful*/
  CanSM_StopWakeupSources(1);
  }
}
```

Caution

The EcuM functions EcuM_StartWakeupSources() and EcuM_StopWakeupSources() must not be interrupted by the ComM, CanSM and CanNm main function. This can be implemented by adding additional exclusive areas inside the two EcuM functions.

Additionally the EcuM_MainFunction must not interrupt the main functions of ComM, CanSM and CanNm, e.g. same task priority or same task context.

2.4 LIN

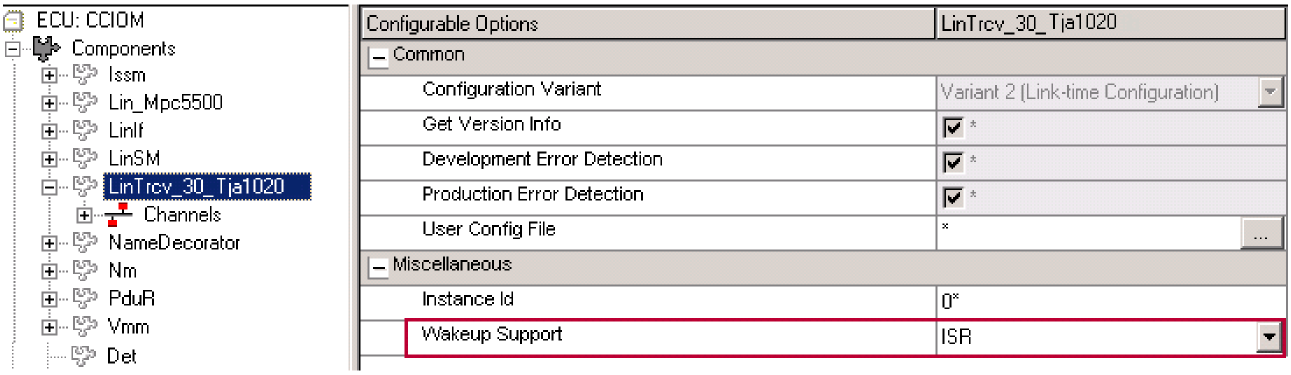
2.4.1 LIN Wake-up by Transceiver

The wake-up information is generated via the ICU driver (see chapter 2.2) by using an additional µC I/O port. This I/O port is connected to the pin of the LIN transceiver which indicates the wake-up.

Example

If TJA1020 transceivers is used connect the pin RXD with the µC I/O port. The RXD pin switches to low if the transceiver detects bus activity. To get a reliable negative edge it has to be ensured that the RXD pin has a high level if the transceiver is in SLEEP mode (see [6], Fig 7) e.g. the pin is connected to a pull-up reference. The wake-up source has to be deactivated to avoid interrupts during normal communication.

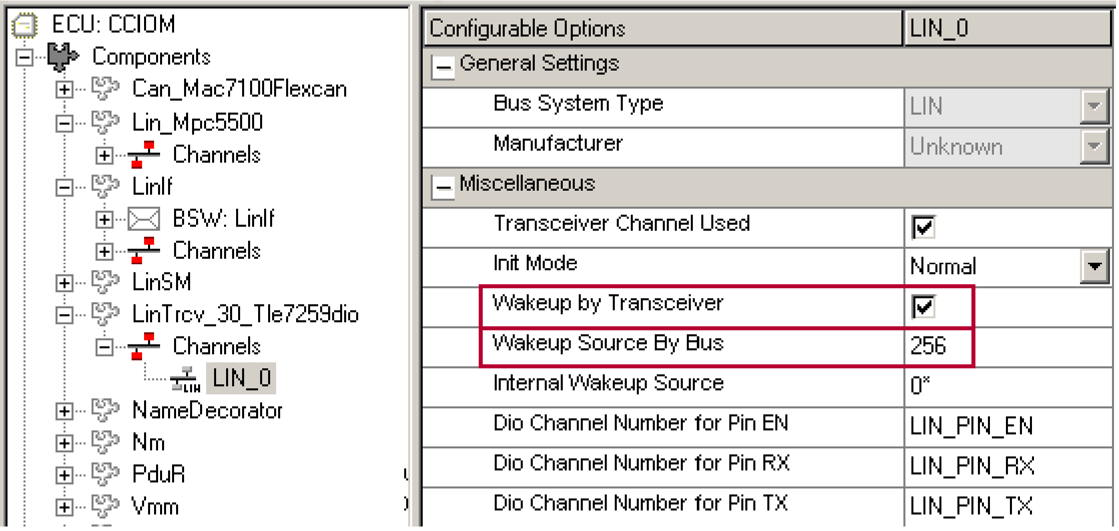
Info
The LIN transceiver TJA1020 in sleep mode triggers the inhibit pin only once in case of a valid wakeup event (see [6], Table 1). So it is not necessary to enable/disable the ICU notification function. But the voltage is also floating if the transceiver is in SLEEP mode (see [6], Fig 7).



Configurable Options	LinTrcv_30_Tja1020
Common	
Configuration Variant	Variant 2 (Link-time Configuration)
Get Version Info	☑ *
Development Error Detection	☑ *
Production Error Detection	☑ *
User Config File	*
Miscellaneous	
Instance Id	0*
Wakeup Support	ISR

Figure 2-13 Selection of the desired interrupt service for the LIN transceiver wake-up

Select ISR for the **Wakeup Support**.



Configurable Options	LIN_0
General Settings	
Bus System Type	LIN
Manufacturer	Unknown
Miscellaneous	
Transceiver Channel Used	☑
Init Mode	Normal
Wakeup by Transceiver	☑
Wakeup Source By Bus	256
Internal Wakeup Source	0*
Dio Channel Number for Pin EN	LIN_PIN_EN
Dio Channel Number for Pin RX	LIN_PIN_RX
Dio Channel Number for Pin TX	LIN_PIN_TX

Figure 2-14 Activation of the LIN transceiver wakeup and configuration of the wake-up source

Enable the feature **Wakeup by Transceiver** in the channel view of the LINTRCV.

Enter the **Wakeup Source By Bus** in the channel view of the LINTRCV. The value has to match to the **Wakeup Source ID** in the ECUM (cp. Chapter 2.1). Enter 0 (zero) if the channel should not be woken up by the transceiver.

The ICU triggers the notification function. This informs the ECUM about the wake-up interrupt by calling EcuM_CheckWakeup with all according wake-up sources as parameter.

Example

```
void Icu_LinTrcvWakeUpNotification_0(void)
{
  /* inform the EcuM about the wakeup event, the parameter is the
configured transceiver wakeup source */
  EcuM_CheckWakeup(ECUM_WKSOURCE_LIN0);
}
```

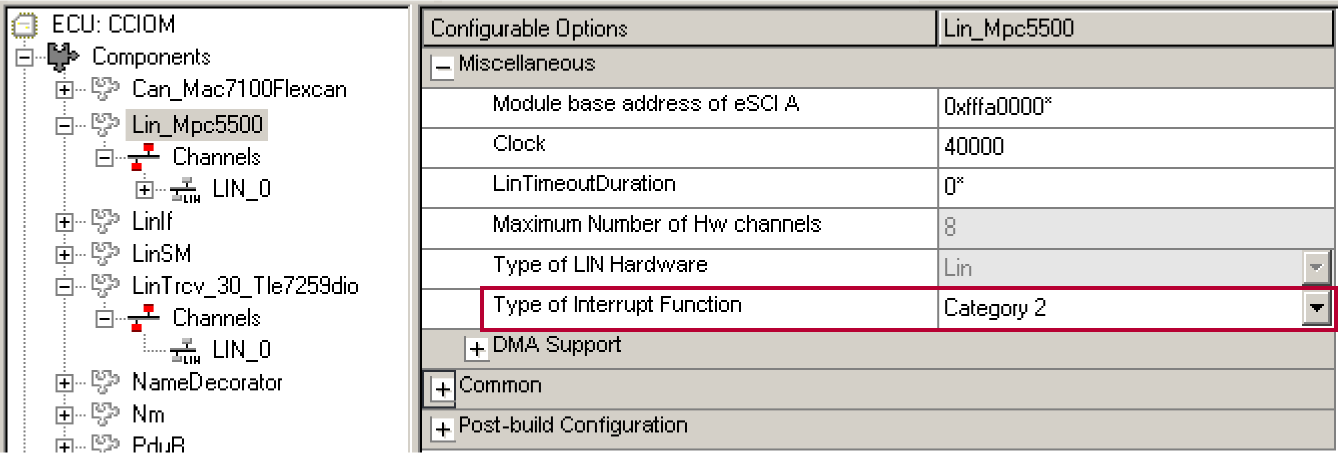
The EcuM does not know if the function EcuM_CheckWakeup has been called by the wakeup source itself. Therefore, the EcuM_CheckWakeup has to ask the driver of the wakeup source if it was responsible for that wakeup. Add the LinIf_Cbk_CheckWakeup callback function of the LIN bus with the according wake-up source as parameter.

Example

```
void EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_LIN0) != 0)
  {
    /* ask the driver of the wakeup source if it was responsible for
the wakeup */
    LinIf_Cbk_CheckWakeup(ECUM_WKSOURCE_LIN0);
  }
}
```

The bus will be woken up by a special signal pattern so it is improbable that the bus has been woken up by an EMC disturbance. From there no wakeup validation is necessary and the **Validation Timeout** in the ECUM configuration in Figure 2-1 has to be zero.

2.4.2 LIN Wake-up by Controller



Configurable Options	Lin_Mpc5500
Miscellaneous	
Module base address of eSCI A	0xfffa0000*
Clock	40000
LinTimeoutDuration	0*
Maximum Number of Hw channels	8
Type of LIN Hardware	Lin
Type of Interrupt Function	Category 2
DMA Support	
Common	
Post-build Configuration	

Figure 2-15 Selection of the desired interrupt service of the LIN controller

> Choose the needed **Type of Interrupt Function**.

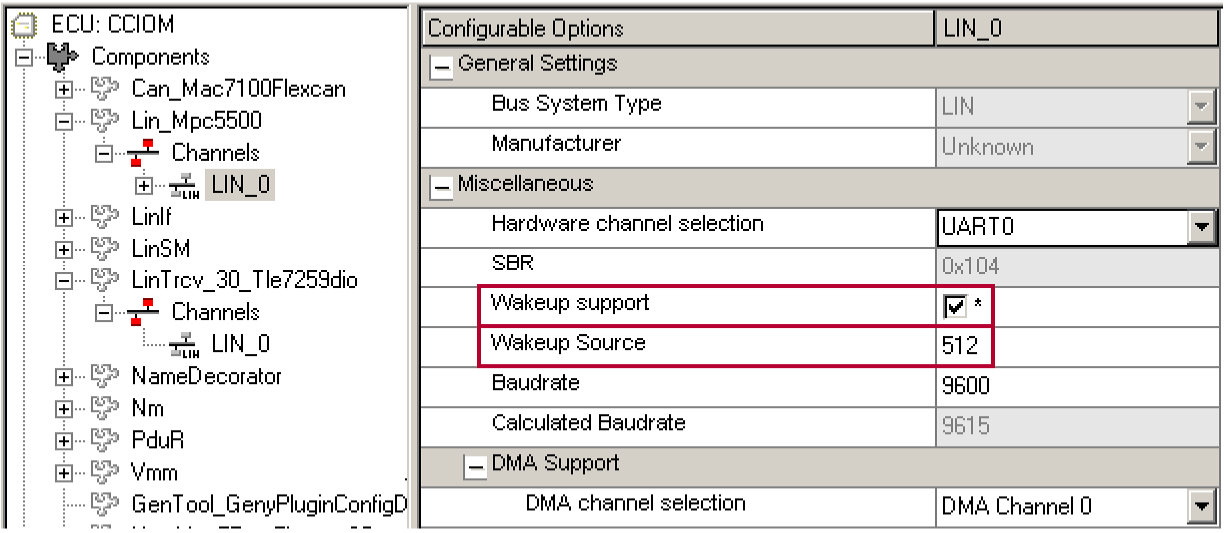
> Category 1: Interrupt function has to be added to the interrupt vector table.

- Category 2: Interrupt function is defined with ISR() define.
- void Func(void): Interrupt function is defined as void Func(void) function.

Choose the related interrupt service function for the LIN controller. This function has to be called by the interrupt dispatcher after identifying the interrupt source which belongs to the LIN controller.

```
ISR( LinIsr_0 ){
  Lin_Interrupt(0);
  }
```

The LIN driver will detect an interrupt in the Lin_Interrupt function and triggers the EcuM_CheckWakeup with the according wake-up source ID of the controller as parameter.



Configurable Options	LIN_0
General Settings	
Bus System Type	LIN
Manufacturer	Unknown
Miscellaneous	
Hardware channel selection	UART0
SBR	0x104
Wakeup support	☑ *
Wakeup Source	512
Baudrate	9600
Calculated Baudrate	9615
DMA Support	
DMA channel selection	DMA Channel 0

Figure 2-16 Activation of the LIN controller wake-up support and configuration wake-up source

Enable the feature **Wakeup support** in the channel view of the LIN driver.

Enter the **Wakeup Source** in the channel view of the LIN driver. The value has to match to the **Wakeup Source ID** in the ECUM (cp. Chapter 2.1). Enter 0 (zero) if the channel should not be woken up by the transceiver.

The bus will be woken up by a special signal pattern so it is improbable that the bus has been woken up by an EMC disturbance. From there no wake-up validation is necessary and the **Validation Timeout** in the ECUM configuration in Figure 2-1 has to be zero.

2.5 FlexRay

2.5.1 FlexRay Wake-up by Transceiver Interrupt

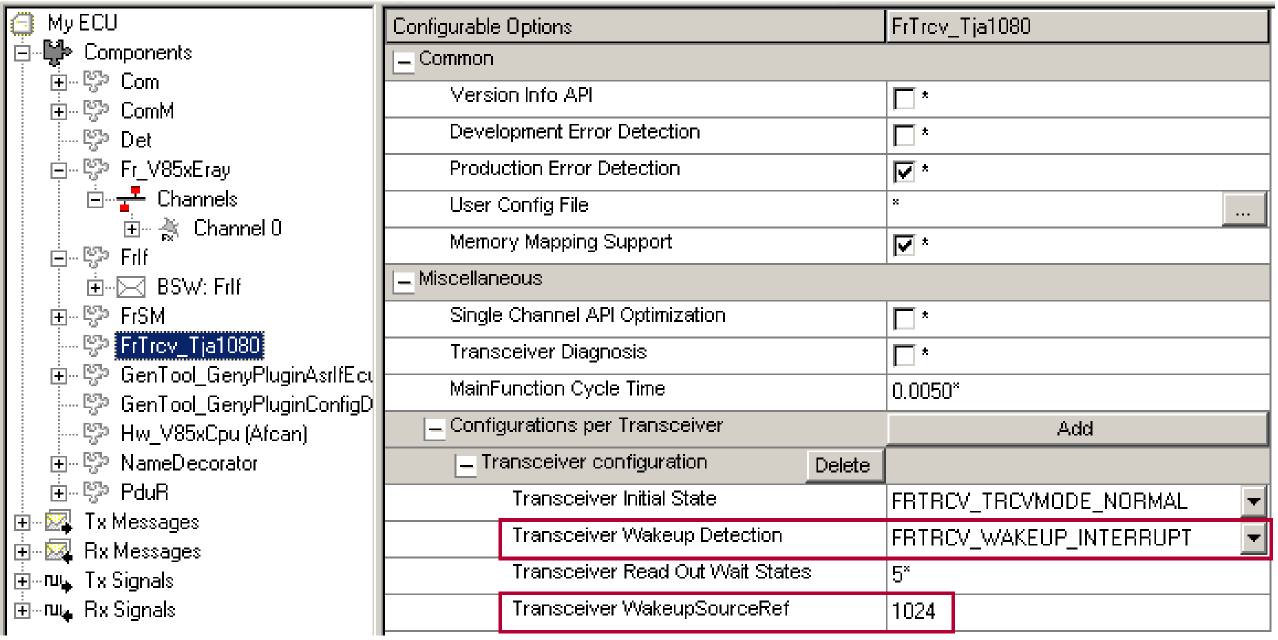
For FlexRay a wake-up is only possible via the FlexRay transceivers. There are two transceivers for the two different channels in a FlexRay cluster. They are treated as belonging to one network and thus, there should be only one wake-up source identifier configured for both channels. The wake-up information is generated via the ICU driver (see chapter 2.2) by using an additional µC I/O port which is connected to the pin of the transceiver which indicates the wake-up.

Example

If TJA1080A FlexRay transceiver is used connect the RXEN port of the transceiver with the µC I/O port. If the bus is idle the pin RXEN is switched to HIGH and if activity is detected on the bus lines the pin RXEN is switched to LOW (see [7], Table 9).

Info

In dependency of the provided ICU configuration options it is possible to configure the **IcuDefaultStartEdge**. This option should be configured to ICU_FALLING_EDGE, because the wake-up event is provided by the RXEN pin via a level change from recessive to dominant.



Configurable Options	FrTrcv_Tja1080
− Common	
Version Info API	□ *
Development Error Detection	□ *
Production Error Detection	☑ *
User Config File	* ...
Memory Mapping Support	☑ *
− Miscellaneous	
Single Channel API Optimization	□ *
Transceiver Diagnosis	□ *
MainFunction Cycle Time	0.0050*
− Configurations per Transceiver	Add
− Transceiver configuration Delete	
Transceiver Initial State	FRTRCV_TRCVMODE_NORMAL
Transceiver Wakeup Detection	FRTRCV_WAKEUP_INTERRUPT
Transceiver Read Out Wait States	5*
Transceiver WakeupSourceRef	1024

Figure 2-17 Selection of the FlexRay wakeup processing type interrupt and configuration of the wake-up source

Select FRTRCV_WAKEUP_INTERRUPT for the **Transceiver Wakeup Detection**.

Enter the according wake-up source ID of the ECUM in **Transceiver WakeupSourceRef** (cp. Chapter 2.1).

The ICU triggers the notification function. This informs the ECUM about the wake-up interrupt by calling EcuM_CheckWakeup with all according wake-up sources as parameter.

Example

```
void Icu_FrTrcvWakeUpNotification_0(void)
{
  /* inform the EcuM about the wakeup event, the parameter is the
configured transceiver wakeup source */
  EcuM_CheckWakeup(ECUM_WKSOURCE_FR);
}
```

The EcuM does not know if the function EcuM_CheckWakeup has been called by the wakeup source itself. Therefore, the EcuM_CheckWakeup has to ask the driver of the wakeup source if it was responsible for that wakeup. Add the FrIf_Cbk_WakeupByTransceiver callback function of the FlexRay bus with the according controller index.

Example

```
void EcuM_CheckWakeup(EcuM_WakeupSourceType wakeupSource)
{
  if ((wakeupSource & ECUM_WKSOURCE_FR) != 0)
  {
    /* ask the driver of the wakeup source if it was responsible for
the wakeup */
    FrIf_Cbk_WakeupByTransceiver(0 /* FrCtrlIdx */, FR_CHANNEL_A );
    FrIf_Cbk_WakeupByTransceiver(0 /* FrCtrlIdx */, FR_CHANNEL_B );
  }
}
```

Caution

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

To avoid the setting of wake-up events during the communication channel is in normal mode, it is necessary to disable the notification if the transceiver enters NORMAL mode and to re-enable the notification if the transceiver enters STANDBY mode. The FRTRCV enable and disables the ICU notification function already. Check if code fit to your needs.

It is possible to enable/disable the transceiver interrupt depending which state has been entered. This functionality can be enabled by defining the following parameter:

```
#define FRTRCV_WUPINT_CBK STD_ON
```

This switch can be defined in a user config file. When this feature is enabled the Icu_DisableNotification and Icu_EnableNotification will be called. Whether the transceiver interrupt will be enabled or disabled can be configured via the trcvIcuFctPtr option in FrTrcvPhy.c.

Example

```
/* Modes are { Unknown, Normal, Standby, Sleep, Receiveonly } */
STATIC CONST(trcvIcuFctPtrType, FRTRCV_CONST) trcvIcuFctPtr[] =
{ Icu_DisableNotification,
  Icu_DisableNotification,
  Icu_EnableNotification,
  Icu_EnableNotification,
  Icu_DisableNotification };
```

Enter the ICU channel name (Icu_FlexRay_WakeUp) in the array FrTrcvChannel FrTrcvPhy.c(line 78) or use the default name of the FrTrcv FRTRCV_CHANNEL_INT_0 as name for the ICU channel (cp. **Figure 2-4**).

Example

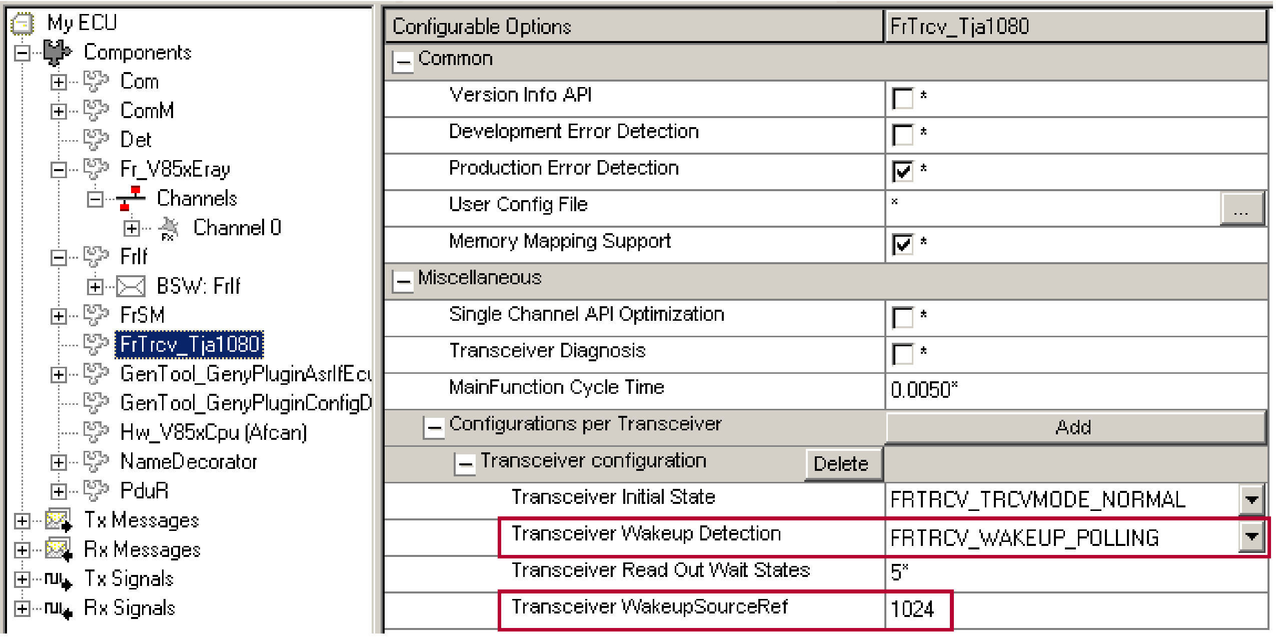
```
STATIC CONST( FrTrcvChannelType, FRTRCV_CONST ) FrTrcvChannel[] =
{
  { /* I/O used for Transceiver 1 */
    (Dio_ChannelType)FRTRCV_CHANNEL_EN_0,
    (Dio_ChannelType)FRTRCV_CHANNEL_STBN_0,
    (Dio_ChannelType)FRTRCV_CHANNEL_ERRN_0,
#if ( FRTRCV_WUPINT_CBK == STD_ON )
    (Icu_ChannelType) Icu_FlexRay_WakeUp
#endif
  },

}
```

Info

The (de)activation of the transceiver wake-up is done by the FlexRay state manager via FrIf_DisableTransceiverWakeup and FrIf_EnableTransceiverWakeup.

The bus will be wakened by a special signal pattern so it’s improbable that the bus has been woken up by an EMC disturbance. From there no wake-up validation is necessary and the **Validation Timeout** in the ECUM configuration in Figure 2-1 has to be zero.

2.5.2 FlexRay Wake-up by Polling the Transceiver



- My ECU
  - Components
    - Com
    - ComM
    - Det
    - Fr_V85xEray
      - Channels
        - Channel 0
    - FrIf
      - BSW: FrIf
    - FrSM
    - FrTrcv_Tja1080
    - GenTool_GenyPluginAsrIfEcu
    - GenTool_GenyPluginConfigD
    - Hw_V85xCpu (Afcan)
    - NameDecorator
    - PduR
  - Tx Messages
  - Rx Messages
  - Tx Signals
  - Rx Signals

Configurable Options	FrTrcv_Tja1080
Common	
Version Info API	□ *
Development Error Detection	□ *
Production Error Detection	☑ *
User Config File	* ...
Memory Mapping Support	☑ *
Miscellaneous	
Single Channel API Optimization	□ *
Transceiver Diagnosis	□ *
MainFunction Cycle Time	0.0050*
Configurations per Transceiver	Add
Transceiver configuration Delete	
Transceiver Initial State	FRTRCV_TRCVMODE_NORMAL
Transceiver Wakeup Detection	FRTRCV_WAKEUP_POLLING
Transceiver Read Out Wait States	5*
Transceiver WakeupSourceRef	1024

Figure 2-18 Selection of the FlexRay wake-up processing type polling and configuration of the wake-up source

Select FRTRCV_WAKEUP_POLLING for the **Transceiver Wakeup Detection**.

Enter the according wakeup source ID of the ECUM in **Transceiver WakeupSourceRef** (cp. Chapter 2.1).

Main function of the transceiver driver polls the respective transceiver for any wake-up events. In case a wake-up is detected and notifications are allowed the ECUM is notified via EcuM_SetWakeupEvent.

The bus will be woken up by a special signal pattern so it is improbable that the bus has been woken up by an EMC disturbance. From there no wake-up validation is necessary and the **Validation Timeout** in the ECUM configuration in Figure 2-1 has to be zero.

3 Special Use Cases

3.1 Multiple Wake-up Sources Share one µC I/O Port

For shared interrupts the ECU firmware may have to check multiple wake-up sources within EcuM_CheckWakeup. The ICU has to pass the identifiers of all wake-up sources that may have caused this interrupt to EcuM_CheckWakeup.

Configure the ICU as described in chapter 2.2. As **Signal Notification** function is Icu_Notification_0 used.

The ICU triggers the notification function if any of the wake-up sources causes an interrupt. Call the EcuM_CheckWakeup once with all the wake-up sources which are connected to the I/O port as parameter. EcuM_WakeupSourceType contains one bit for each wake-up source, so that multiple wake-up sources can be passed in one call. The EcuM will check each network which corresponds to a wake-up source if it was responsible for that wake-up. If the channel reports a positiv answer via EcuM_SetWakeupEvent the ECUM trigger the COMM to start the network.

Example

```
void Icu_Notification_0 (void)
{
  /* inform the EcuM about the wake up event, the parameter are the
configured wake up sources */
  EcuM_CheckWakeup(ECUM_WKSOURCE_CAN0 || ECUM_WKSOURCE_CAN1 ||
ECUM_WKSOURCE_FR);
}
```

3.2 CAN Wake-up Without Validation

Set the **Validation Timeout** to zero in the ECUM configuration (see Figure 2-1).

Disable the validation in the CANIF configuration (see Figure 2-12).

Skip the call EcuM_CheckWakeup and call EcuM_SetWakeupEvent direct and so the ECUM doesn't start the validation process.

Example

```
void Icu_TrcvWakeUpNotification_0(void)
{
  /* inform the EcuM about the wakeup event, the parameter is the
configured transceiver wakeup source */
  EcuM_SetWakeupEvent(ECUM_WKSOURCE_CAN0);
}
```

4 Integration hints

- The normal bus communication, started with COMM mode request at system initialization should work before you try to start the communication via wake-up.
- Check if the system reacts on an incoming (wake-up) signal and the configured interrupt function is called.
- Check if the interrupts are enabled (e.g. enable at start-up and skip disabling). There are several ways how the communication is disabled. All possible ways has to be considered E.g.
  - transceiver is set to `OP_MODE_STANDBY`,
  - transceiver is set to `OP_MODE_ SLEEP` or
  - ECUM switches to state SLEEP.
- Check if the wake-up source is the correct one.

5 Glossary and Abbreviations

5.1 Glossary

Term	Description
MICROSAR Configurator Pro	Generation tool for MICROSAR components
GENy	Generation tool for CANbedded and MICROSAR components
TJA1041	A CAN transceiver
TJA1080A	A FlexRay transceiver
TJA1020	A LIN transceiver

Table 5-1 Glossary

5.2 Abbreviations

Abbreviation	Description
µC	micro-controller
ASR	AUTOSAR
AUTOSAR	Automotive Open System Architecture
CAN	Controller Area Network
ComM	Communication Manager
Ctrl	Controller
ECU	Electronic Control Unit
EcuM	ECU State Manager
ERRN	Error
FR	FlexRay
GPT	General Purpose Timer
ICU	Input Capture Unit
I/O	input / output
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
Nm	MICROSAR Network Management Interface
Rx	Receive
SchM	BSW Scheduler Module
Trcv	Transceiver
Tx	Transmission

Table 5-2 Abbreviations

6 Contact

Visit our website for more information on

> News
> Products
> Demo software
> Support
> Training data
> Addresses

www.vector-informatik.com