User Manual

for MPC5634M CAN Driver

Document Number: UM14CANASR3.0R2.0.0
Rev. 1.1

freescale™

Contents

Section Number | Title | Page

Section Number	Title	Page

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	14/02/2011	Hari Sulgekar	BETA 1.9.0 Release
1.1	23/11/2011	Jhagadu Yadav	MPC5634M RTM 2.0.0 Release

Chapter 2 Introduction

This User Manual describes Freescale Semiconductor AUTOSAR Controller Area Network (CAN) for MPC5634M.

AUTOSAR CAN driver configuration parameters and deviations from the specification are described in Can Driver chapter of this document. AUTOSAR CAN driver requirements and APIs are described in the AUTOSAR CAN driver software specification document.

2.1 Supported Derivatives

The software described in this document is intended to be used with the following microcontroller devices of Freescale Semiconductor:

Table 2-1. MPC5634M Derivatives

Freescale Semiconductor	mpc5634m_bga208, mpc5634m_qfp144, mpc5634m_qfp176

All of the above microcontroller devices are collectively named as MPC5634M.

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.

- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About this Manual

This Technical Reference employs the following typographical conventions:

Boldface type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
ASM	Assembler
BSMI	Basic Software Make file Interface
CAN	Controller Area Network
DEM	Diagnostic Event Manager
DET	Development Error Tracer
C/CPP	C and C++ Source Code
VLE	Variable Length Encoding
N/A	Not Applicable
MCU	Micro Controller Unit

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	AUTOSAR 3.0 CAN Driver Software Specification Document.	V2.2.0 R3.0 Rev 0001
2	MPC5634M Reference Manual	Rev. 6, 4 October 2011

Chapter 3
Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 3.0 CAN Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

The MPC5634M contains up to two Controller Area Network (FlexCAN) blocks.

Each FlexCAN module is a communication controller implementing the CAN protocol. The CAN protocol interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The message buffer management (MBM) sub-module handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The bus interface unit (BIU) sub-module controls the access to and from the internal interface bus, to establish connection to the CPU and other blocks.

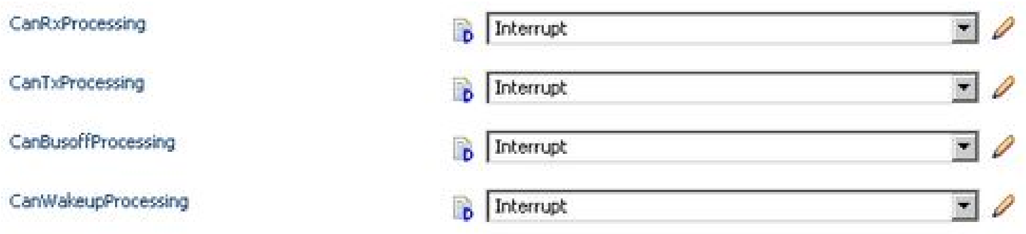
The FlexCAN has these major features:

- 64(for FlexCAN_A)/32(for FlexCAN_C) flexible message buffers (MBs) of zero to eight bytes data length.
- Individual Rx mask registers per message buffer.
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 Extended, 16 Standard, or 32 Partial (8 bits) IDs, with individual masking capability.
- ListenOnly capability.
- Programmable loop-back mode supporting self-test operation.
- Maskable interrupts.

- Low power modes.
- Unused MB and Rx Mask Register space can be used as general purpose RAM space.

General Aspects for Interrupt Implementation

Autosar specifications permit to configure Can controller in interrupt mode for the following events: Rx Processing, Tx Processing, BusOff Processing, Wakeup Processing.



CanRxProcessing	Interrupt
CanTxProcessing	Interrupt
CanBusoffProcessing	Interrupt
CanWakeupProcessing	Interrupt

Figure 3-1. Can Interrupts Selection

Interrupts implementation and mapping is done according to MPC5634M Reference Manual.

General Aspects for RxFifo Implementation

The receive-only FIFO is enabled by asserting the FEN bit in the MCR register. The RxFifo configuration in the Tresos plugin is implemented by “CanControllerRxFifoEnable” parameter under “CanController” container.

When the Fifo is enabled, the memory region normally occupied by the first 8 MBs ($80 - $FF) is normally reserved for use of the Fifo engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing the MB0 structure.

The interrupts corresponding to MB0 to7 have a different behavior when RxFifo in enabled. Bit 7 of the IFLAG1 becomes the “Fifo Overflow” flag, bit 6 becomes the “Fifo Warning” flag, bit 5 becomes the “Frame Available in Fifo” flag and bits 4 to 0 are unused.

Note

The hardware objects configured in the “Can_MBConfigObjectType” structure are used for the initialization of MBs. IF RxFifo is enabled then the MB initialization will start from the MB8 because first 8 MBs (0 to 7) are reserved for the RxFifo. Also the total number of hardware objects per controller is reduced to 56 from 64(for FlexCAN_A)/32(for FlexCAN_C) when Rx Fifo is enabled.

If RxFifo is enabled the user can define proper handlers for overflow and warnings notification events. If the precompile parameter “CanCodeSizeOptimization” is set true or Rx processing is configured in polling mode, the 3 events of Fifo (overflow, warning and frame available) are processed by the same low level driver routine (see Can_LLD_Process_Rx).

Usage of Symbolic Names for Controllers and Message Buffers.

CAN controllers are declared by the user in the "CanController" list. For every controller a name is set by the user. This name will be translated into a define which is generated in Can_Cfg.h file of type (uint8).

CAN message buffers are declared by the user in the "CanHardwareObject" list. For every message buffer a name is set by the user. This name will be translated into a define which is generated in Can_Cfg.h file of type (Can_HWObjectCountType).

IRQ Usage

For every controller from "CanController" list a define is generated in Can_Cfg.h file. According to event handling (interrupts or polling) extra defines are generated for including interrupt routines or polling APIs.

Example of generated defines for FlexCan_A:

```
  /* When FlexCan_A is configured this define will be generated in Can_Cfg.h file. */
  #define CAN_FCA_INDEX       (uint8)0U
  /* If transmission is configured in interrupt mode for FlexCan_A, this define will be set
to STD_ON, else STD_OFF. */
  #define CAN_A_TXINT_SUPPORTED  (STD_OFF)
  /* If at least one controller has the transmission configured in polling mode this define
will be STD_ON. */
  #define CAN_TXPOLL_SUPPORTED   (STD_ON)
  /* If reception is configured in interrupt mode for FlexCan_A, this define will be set to
STD_ON, else STD_OFF. */
  #define CAN_A_RXINT_SUPPORTED  (STD_OFF)
  /* If at least one controller has the reception configured in polling mode this define
will be STD_ON. */
  #define CAN_RXPOLL_SUPPORTED   (STD_ON)
  /* If busoff is configured in interrupt mode for FlexCan_A, this define will be set to
STD_ON, else STD_OFF. */
  #define CAN_A_BUSOFFINT_SUPPORTED  (STD_OFF)
  /* If at least one controller has the busoff configured in polling mode this define will
be STD_ON. */
  #define CAN_BUSOFFPOLL_SUPPORTED   (STD_ON)
  /* If wakeup is configured in interrupt mode for FlexCan_A, this define will be set to
STD_ON, else STD_OFF. */
  #define CAN_A_WAKEUPINT_SUPPORTED  (STD_OFF)
  /* If at least one controller has the wakeup configured in polling mode this define will
be STD_ON. */
  #define CAN_WAKEUPPOLL_SUPPORTED   (STD_ON)
  /* If at least one controller has the RxFifo enabled, this define will be STD_ON. */
  #define CAN_RXFIFO_ENABLE (STD_OFF)
  /* If FlexCan_A has the RxFifo enabled, this define will be STD_ON. */
  #define CAN_A_FIFO_EN       (STD_OFF)
  /* If global precompile wakeup is enabled, this define will be STD_ON. */
```

```
#define CAN_WAKEUP_SUPPORT (STD_OFF)
```

Note

Refer to Can_Irq.c file for checking how are interrupts included/excluded at precompile based above defines.

3.3 Calling Driver APIs

1. void Can_GetVersionInfo (Std_VersionInfoType* versioninfo)

This function returns the version information of this module.

versioninfo - Pointer where to store the version information of this module.

```
typedef struct {
    uint16 vendorID;
    uint16 moduleID;
    uint8 instanceID;
    uint8 sw_major_version;
    uint8 sw_minor_version;
    uint8 sw_patch_version;
} Std_VersionInfoType;
```

Sample code:

```
/* main_app.c */
....................
Std_VersionInfoType version;
uint16  vendor_id;

Can_Init( Can_Cfg1);
#if (CAN_VERSION_INFO_API == STD_ON)
    Can_GetVersionInfo( &version);
    vendor_id = version.vendorID;
#endif /* (CAN_VERSION_INFO_API == STD_ON) */
....................
```

Note

This API can be used only if CAN_VERSION_INFO_API is set to STD_ON.

2. void Can_Init (const Can_ConfigType* Config)

This function initializes the module.

Config - Pointer to driver configuration. This is the type of the external data structure containing the overall initialization data for the CAN driver and SFR settings affecting all controllers. Furthermore it contains pointers to controller configuration structures. The contents of the initialization data structure are CAN hardware specific.

For Pre-Compile configuration the Can_cfg.h file will contain the extern declaration of "Can_ConfigType" structure.

```
....................
#ifdef CAN_PRECOMPILE_SUPPORT
    #define CAN_INIT_CONFIG_PC_DEFINES \
        extern CONST(Can_ConfigType, CAN_CONST) Can_ConfigSet_PC;
#endif
....................
```

Can_Cfg.c file will contains the structure type:

```
....................
CONST(Can_ConfigType, CAN_CONST) Can_ConfigSet_PC = {
.....
};
....................
```

For Post-Build configuration the Can_Cfg.h file will contain the extern declaration of "Can_ConfigType" structure.

```
....................
#ifndef CAN_PRECOMPILE_SUPPORT
    #define CAN_INIT_CONFIG_PB_DEFINES \
        extern CONST(Can_ConfigType, CAN_CONST) CanConfigSet_0_PB;
    #endif
....................
```

Can_PBcfg.c file will contains the structure type:

```
....................
CONST(Can_ConfigType, CAN_CONST) CanConfigSet_0_PB = {
    .....
};
....................
```

Can.h export driver configuration:

```
#ifdef CAN_PRECOMPILE_SUPPORT
    /* Export Driver configuration */
    CAN_INIT_CONFIG_PC_DEFINES
#else
    /* Export Driver configuration */
    CAN_INIT_CONFIG_PB_DEFINES
#endif
```

Sample code:

```
            /* main_app.c */
            ....................
            #ifdef CAN_PRECOMPILE_SUPPORT
                CONST(Can_ConfigType, CAN_CONST) *Can_Cfg1 = &Can_ConfigSet_PC;
            #else
                CONST(Can_ConfigType, CAN_CONST) *Can_Cfg1 = &CanConfigSet_0_PB;
            #endif
            Can_Init( Can_Cfg1);
            ....................
```

3. void Can_InitController (uint8 Controller, const Can_ControllerConfigType* Config)

This function initializes only CAN controller specific settings.

Controller - CAN controller to be initialized

Config - Pointer to controller configuration.

Can_Cfg.h contains the controller indexes:

```
            ....................
            #define CanA0 0U  /* Default configuration for FlexCAN_A */
            ....................
```

For Pre-Compile configuration the Can_cfg.h file will contain the extern declaration of "Can_ControllerConfigType" structure.

```
            ....................
            #ifdef CAN_PRECOMPILE_SUPPORT
                #define CAN_INIT_CONFIG_CONTROLLERS_PC \
                    extern CONST(Can_ControllerConfigType, CAN_CONST)
ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT_0];
                #endif
            ....................
```

Can_Cfg.c file will contains the structure type:

```
            ....................
            CONST(Can_ControllerConfigType, CAN_CONST)
ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT_0] = {
                .....
            };
            ....................
```

For Post-Build configuration the Can_Cfg.h file will contain the extern declaration of "Can_ControllerConfigType" structure.

```
            ....................
            #ifndef CAN_PRECOMPILE_SUPPORT
                #define CAN_INIT_CONFIG_CONTROLLERS_PB \
                    extern CONST(Can_ControllerConfigType, CAN_CONST)
ControllerConfigs0_PB[CAN_MAXCONTROLLERCOUNT_0];
```

```
            #endif
        ....................
```

Can_PBcfg.c file will contains the structure type:

```
            ....................
            CONST(Can_ControllerConfigType, CAN_CONST)
ControllerConfigs0_PB[CAN_MAXCONTROLLERCOUNT_0] = {
                .....
            };
            ....................
```

Can.h export Can controller configurations:

```
            #ifdef CAN_PRECOMPILE_SUPPORT
                /* Export Controllers configuration */
                CAN_INIT_CONFIG_CONTROLLERS_PC
            #else
                /* Export Controllers configuration */
                CAN_INIT_CONFIG_CONTROLLERS_PB
            #endif
```

Sample code:

```
            /* main_app.c */
            ....................
            #ifdef CAN_PRECOMPILE_SUPPORT
                CONST(Can_ControllerConfigType, CAN_CONST) *Can_ControllerCfg =
&ControllerConfigs_PC;
            #else
                CONST(Can_ControllerConfigType, CAN_CONST) *Can_ControllerCfg =
&ControllerConfigs0_PB;
            #endif
            Can_Init( Can_Cfg1);
            Can_InitController( CanA0, Can_ControllerCfg);
            ....................
```

4. Can_ReturnType Can_SetControllerMode (uint8 Controller, Can_StateTransitionType Transition)

This function performs software triggered state transitions of the CAN controller State machine.

Controller - CAN controller for which the status shall be changed

Transition - CAN state transition: CAN_STOP / CAN_T_START / CAN_T_SLEEP / CAN_T_WAKEUP

Sample code:

```
            /* main_app.c */
            ....................
            Can_ReturnType ret_val = CAN_NOT_OK;
```

```
Can_Init( Can_Cfg1);
ret_val = Can_SetControllerMode( CanA0, CAN_T_START);
....................
```

5. void Can_DisableControllerInterrupts (uint8 Controller)

This function disables all interrupts for this CAN controller.

Controller - CAN controller for which interrupts shall be disabled.

Sample code:

```
/* main_app.c */
....................
Can_Init( Can_Cfg1);
Can_DisableControllerInterrupts( CanA0);
....................
```

6. void Can_EnableControllerInterrupts (uint8 Controller)

This function enables all allowed interrupts.

Controller - CAN controller for which interrupts shall be re-enabled.

```
/* main_app.c */
....................
Can_Init( Can_Cfg1);
Can_EnableControllerInterrupts( CanA0);
....................
```

7. Can_ReturnType Can_Write (uint8 Hth, const Can_PduType* PduInfo)

This function transmit a message on the bus.

Hth - information which HW-transmit handle shall be used for transmit. Implicitly this is also the information about the controller to use because the Hth numbers are unique inside one hardware unit.

PduInfo - Pointer to SDU user memory, DLC and Identifier.

Can_Cfg.h will contains the MB indexes:

```
#define CTRL0_MB0 0U /* RECEIVE object of Can Controller ID = 0 */
#define CTRL0_MB1 1U /* RECEIVE object of Can Controller ID = 0 */
#define CTRL0_MB2 2U /* RECEIVE object of Can Controller ID = 0 */
#define CTRL0_MB3 3U /* TRANSMIT object of Can Controller ID = 0 */
#define CTRL0_MB4 4U /* TRANSMIT object of Can Controller ID = 0 */
#define CTRL0_MB5 5U /* TRANSMIT object of Can Controller ID = 0 */
#define CTRL1_MB0 6U /* RECEIVE object of Can Controller ID = 1 */
#define CTRL1_MB1 7U /* TRANSMIT object of Can Controller ID = 1 */
```

Sample code:

```
/* main_app.c */
....................
Can_ReturnType ret_val = CAN_NOT_OK;
Can_PduType    CanMessage;
VAR(uint8, CAN_VAR) data[8U] = {0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77, 0x77};

Can_Init( Can_Cfg1);
CanMessage.length = 8U;
CanMessage.sdu = data;
CanMessage.id = 0x1U;

ret_val = Can_Write( CTRL0_MB3, &CanMessage);
....................
```

8. void Can_MainFunction_Write ()

This function performs the polling of TX confirmation and TX cancellation confirmation when CAN_TX_PROCESSING is set to POLLING.

Notice that:

```
/* Can.h */
#if (CAN_TXPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_Write( void );
#else
    #define Can_MainFunction_Write()
#endif
```

Sample code:

```
/* main_app.c */
....................
ret_val = Can_Write( CTRL0_MB3, &CanMessage);
Can_MainFunction_Write();
....................
```

Note

This API can be used only if CAN_TXPOLL_SUPPORTED is set to STD_ON.

9. void Can_MainFunction_Read ()

This function performs the polling of RX indications when CAN_RX_PROCESSING is set to POLLING.

Notice that:

```
/* Can.h */
#if (CAN_RXPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_Read( void );
#else
```

```
    #define Can_MainFunction_Read()
#endif
```

Note

This API can be used only if CAN_RXPOLL_SUPPORTED is set to STD_ON.

10. void Can_MainFunction_BusOff ()

This function performs the polling of bus-off events that are configured statically as 'to be polled'.

Notice that:

```
/* Can.h */
#if (CAN_BUSOFFPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_BusOff( void );
#else
    #define Can_MainFunction_BusOff()
#endif
```

Note

This API can be used only if CAN_BUSOFFPOLL_SUPPORTED is set to STD_ON.

11. void Can_MainFunction_Wakeup ()

This function performs the polling of wake-up events that are configured statically as 'to be polled'.

Notice that:

```
/* Can.h */
#if (CAN_WAKEUPPOLL_SUPPORTED == STD_ON)
    extern FUNC (void, CAN_CODE) Can_MainFunction_Wakeup( void );
#else
    #define Can_MainFunction_Wakeup()
#endif
```

Note

This API can be used only if CAN_WAKEUPPOLL_SUPPORTED and CAN_WAKEUP_SUPPORT are set to STD_ON.

12. void Can_Cbk_CheckWakeup (uint8 controller)

This function checks if a wakeup has occurred for the given controller.

Sample code:

```
/* main_app.c */
....................
Std_ReturnType std_ret_val = E_NOT_OK;

Can_Init( Can_Cfg1);
.....
std_ret_val = Can_Cbk_CheckWakeup( CanA0);
....................
```

Note

This API can be used only if CAN_WAKEUP_SUPPORT is set to STD_ON.

13. void Can_AbortMb (uint8 can_controller, uint8 can_mb)

This function write a abort code (b'1001) to MBCS[CODE] field of the MB.

Sample code:

```
/* main_app.c */
....................
Can_PduType    CanMessage;
Can_ReturnType ret_val = CAN_NOT_OK;

Can_Init( Can_Cfg1);
ret_val = Can_SetControllerMode( CanA0, CAN_T_START);
ret_val = Can_Write( CTRL0_MB3, &CanMessage);
Can_AbortMb ( CanA0,CTRL0_MB3);
....................
```

Note

This API can be used only if CAN_API_ENABLE_ABORT_MB is set to STD_ON.

14. Std_ReturnType Can_SetClockMode(uint8 can_controller, Can_ClockMode can_clk_mode)

This function is configuring Can controller to run on the same baudrate, but having a different MCU source clock..

Sample code:

```
    /* main_app.c */
Std_ReturnType   can_return = E_NOT_OK;
Can_ReturnType  ret_val = CAN_NOT_OK;
    ....................
    Can_Init( Can_Cfg1);
can_return = Can_SetClockMode( controller, CAN_NORMAL);
.............
can_return = Can_SetClockMode( controller, CAN_ALTERNATE);
ret_val = Can_SetControllerMode( controller, CAN_T_START);
    ....................
```

Note

This API can be used only if CAN_DUAL_CLOCK_MODE is set to STD_ON.

3.4 Deviation from Requirements

The driver deviates from the AUTOSAR CAN Driver software specification in some places.

There are also some additional requirements (on top of requirements detailed in AUTOSAR CAN Driver software specification) which need to be satisfied for correct operation.

1. The driver does not distinguish between “Extended” and “Mixed” MB types for receiving way: All Rx MBs configured as MIXED type will be converted to EXTENDED type. For transmission the CanIf will prepare the message ID with MSB bit set and based on this fact the Can module will send the message as STANDARD or EXTENDED type.See CANIF188 and CANIF281 requirements.
2. The maximum number of nested calls to Can_DisableControllerInterrupts is limited to 127.
3. The interrupt service routines are not coded as re-entrant and may only be preempted by code which does not call any of the driver functions.
4. Functions Can_DisableControllerInterrupts and Can_EnableControllerInterrupts may not be preempted by code which calls function Can_InitController.
5. Function Can_MainFunction_Read may not be preempted by code which calls any of the driver functions.
6. Priority inversion may occur even if Cancellation and Transmission multiplexing is enabled. When all message buffers in the transmission pool are full and scheduled for transmission a new call to Can_Write will cause a search which identifies the lowest priority message. If the lowest priority message has lower priority than the new message submitted to Can_Write, then cancellation of the message currently stored in the message buffer will be attempted. There is, however, a possibility that this lowest priority message might be successfully transmitted after the Can_Write has read the message buffer during its search. If a new high priority message is immediately scheduled for transmission (via preemptive call to Can_Write) the identification of the message buffer holding the lowest priority message will no longer be correct (in the underlying Can_Write which has been preempted). This may lead to the message not being cancelled (as it now may have a higher priority). In this case Can_Write will not repeat the search for the lowest priority message and priority inversion may occur (if there is another message with lower priority

scheduled for transmission in a different message buffer). Whether this scenario can or cannot occur in a particular application depends on implementation of the CanIf.

Table Table 3-1 provides Status column description.

Table 3-1. Deviations Status Column Description

Term	Definition
N/A	Not available
N/T	Not testable
N/S	Out of scope
N/I	Not implemented
N/F	Not fully implemented

Below table identifies the AUTOSAR requirements that are not fully implemented, implemented differently, or out of scope for the CAN driver.

Table 3-2. CAN Deviations Table

Requirement	Status	Description	Notes
CAN011	N/S	The CAN driver directly uses the buffer of the upper layer. It is the responsibility of the upper layer to keep the buffer consistent. The function CanIf_Transmit copies the L-SDU either directly in the CAN Hardware or buffers it if CAN Hardware transmit resources are presently not available. The L-SDU source buffer (provided by source layer, e.g. COM) can be written with the next value as soon as CanIf_Transmit returns. The source layer (i.e. COM) is responsible that the L-SDU buffer is not overwritten until the function CanIf_Transmit returned. SDU must be protected by the layer that calls CanIf_Transmit.	Can module is not responsible for keeping the buffers consistent.Upper layer should provide it.
CAN324	N/I	CAN_HANDLE_TYPE Specifies the type (Full-CAN or Basic-CAN) of a hardware object.	Controller doesn’t provide any bit field to differentiate BASIC-CAN and FULL-CAN.
CAN242	N/S	If an off-chip CAN controller is used, the driver uses services of other MCAL drivers (i.e. SPI). These drivers need to be up and running before the CAN controller can be initialized. The sequence of initialization of different drivers is partly specified in [7]. Only Synchronous APIs may be used because the CAN driver does not provide callback functions that can be called by the MCAL driver. Thus the type of connection between µC and CAN Hardware Unit has only impact on implementation and not on the API.	No off-chip controller is used.
CAN110	N/S	There is no requirement regarding the execution order of the CAN main processing functions.	Application Code Requirement.

Table continues on the next page...

Table 3-2. CAN Deviations Table (continued)

Requirement	Status	Description	Notes
CAN240	N/S	The Can module's environment shall make sure that the Mcu module is initialized before initializing the Can module.	Can driver cannot access any variable of Mcu module for checking the state.
CAN243	N/S	If the Can module uses services of other MCAL drivers (e.g. SPI), the Can module's environment shall make sure that these drivers are up and running before initializing the Can module.	No services of other drivers are used by Can driver.
CAN077	N/S	For CAN Hardware Units of different type, different Can modules shall be implemented.	Current platforms have only one type of hardware unit.

3.5 Function Definitions

APIs of all functions supported by the driver are as per AUTOSAR CAN Driver software specification Version 3.0.

3.5.1 Function Can_Init

Initialize the CAN driver. SID = 0x00.

Prototype:`void Can_Init(const Can_ConfigType *Config);`

Table 3-3. Can_Init Arguments

Type	Name	Direction	Description
const `Can_ConfigType` *	Config	**input**	Pointer to driver configuration

Initialize all the controllers.

The CAN module shall be initialized by Can_Init(<&Can_Configuration>) service call during the start-up.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Can_Init shall be called at most once during runtime.

post: Can_Init shall initialize all the controllers and set the driver in READY state.

3.5.2 Function Can_GetVersionInfo

Returns the version information of this module. SID = 0x07.

Prototype:`void Can_GetVersionInfo(Std_VersionInfoType *versioninfo);`

Table 3-4. Can_GetVersionInfo Arguments

Type	Name	Direction	Description
Std_VersionInfoType *	versioninfo	**input**	A pointer to a variable to store version info.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: The CAN_VERSION_INFO_API define must be configured on.

post: The version information is return if the parameter versionInfo is not a null pointer.

3.5.3 Function Can_InitController

Initialize the CAN controllers. SID = 0x02.

Prototype:`void Can_InitController(uint8 Controller, const Can_ControllerConfigType *Config);`

Table 3-5. Can_InitController Arguments

Type	Name	Direction	Description
uint8	Controller	**input**	Controller Can controller to be initialized - based on configuration order list (CanControllerId).
const `Can_ControllerConfigType` *	Config	**input**	Pointer to controller configuration.

Initialize the controller based on ID input parameter.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Before controller re-initalization the driver must be initialized and the controllers must be in Stop state.

post: Interrupts and MBs must be configured to be able to communicate over CAN bus.

3.5.4 Function Can_SetControllerMode

Put the controller into a required state. SID = 0x03.

Prototype:

```
Can_ReturnType Can_SetControllerMode(uint8 Controller, Can_StateTransitionType Transition);
```

Table 3-6. Can_SetControllerMode Arguments

Type	Name	Direction	Description
`uint8`	Controller	**input**	CAN controller for which the status shall be changed
`Can_StateTransitionType`	Transition	**input**	Possible transitions (CAN_T_STOP / CAN_T_START / CAN_T_SLEEP / CAN_T_WAKEUP)

Table 3-7. Can_SetControllerMode Returns

Value	Description
CAN_OK	Transition initiated.
CAN_NOT_OK	Development or production error.

Switch the controller from one state to another.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

Return: Can_ReturnType Result of the transition.

pre: Before changing the controller state the driver must be initialized.

post: After the transition to the new state the interrupts required for that state must be enabled.

3.5.5 Function Can_DisableControllerInterrupts

Disable INTs. SID = 0x04.

Prototype:`void Can_DisableControllerInterrupts(uint8 Controller);`

Table 3-8. Can_DisableControllerInterrupts Arguments

Type	Name	Direction	Description
uint8	Controller	**input**	CAN controller for which interrupts shall be disabled.

Switch OFF the controller's interrupts.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized before changing the interrupts state (en/dis).

post: Controller must not respond to any interrupt assertion.

3.5.6 Function Can_EnableControllerInterrupts

Enable INTs. SID = 0x05.

Prototype:`void Can_EnableControllerInterrupts(uint8 Controller);`

Table 3-9. Can_EnableControllerInterrupts Arguments

Type	Name	Direction	Description
uint8	Controller	**input**	CAN controller for which interrupts shall be disabled

Switch ON the controller's interrupts.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized before changing the interrupts state (en/dis).

post: Controller must respond to interrupt assertion.

3.5.7 Function Can_Write

Transmit information on CAN bus. SID = 0x06.

Prototype`Can_ReturnType Can_Write(Can_HWObjectCountType Hth, const Can_PduType *PduInfo);`

Table 3-10. Can_Write Arguments

Type	Name	Direction	Description
`Can_HWObjectCountType`	Hth	**input**	Information which HW-transmit handle shall be used for transmit. Implicitly this is also the information about the controller to use because the Hth numbers are unique inside one hardware unit.
const `Can_PduType` *	PduInfo	**input**	Pointer to SDU user memory, DLC and Identifier

Can_Write checks if hardware transmit object that is identified by the HTH is free.

Can_Write checks if another Can_Write is ongoing for the same HTH.

a) hardware transmit object is free: The mutex for that HTH is set to 'signaled' the ID, DLC and SDU are put in a format appropriate for the hardware (if necessary) and copied in the appropriate hardware registers/buffers. All necessary control operations to initiate the transmit are done. The mutex for that HTH is released. The function returns with CAN_OK.

b) hardware transmit object is busy with another transmit request. The function returns with CAN_BUSY.

c) A preemptive call of Can_Write has been issued, that could not be handled reentrant (i.e. a call with the same HTH). The function returns with CAN_BUSY the function is non blocking

d) The hardware transmit object is busy with another transmit request for an L-PDU that has lower priority than that for the current request. The transmission of the previous L-PDU is cancelled (asynchronously). The function returns with CAN_BUSY.

This routine is called by:

- CanIf or an upper layer according to Autosar requierements.

Return: `Can_ReturnType`

- CAN_OK - write command has been accepted
- CAN_NOT_OK - development error occurred
- CAN_BUSY - no TX hardware buffer available or preemptive call of `Can_Write()` that can't be implemented reentrant

pre: Driver must be initialized and MB must be configured for Tx.

post: The data can be transmitted or rejected because of another data with a higher priority.

3.5.8 Function Can_Cbk_CheckWakeup

Process check of WakeUp condition. SID = 0x0B.

Prototype`Std_ReturnType Can_Cbk_CheckWakeup(const uint8 controller);`

Table 3-11. Can_Cbk_CheckWakeup Arguments

Type	Name	Direction	Description
const uint8	controller	**input**	controller ID

This service shall evaluate the WakeupSource parameter to get the information, which dedicate wakeup source needs to be checked, either a CAN transceiver or controller device.

This routine is called by:

- CanIf or an upper layer according to Autosar requierements.

Return: Std_ReturnType

- E_OK - a wakeup was detected for the given controller.
- E_NOT_OK - no wakeup was detected for the given controller.

pre: Driver must be initialized.

post: Return the wakeup event occurrence.

3.5.9 Function Can_MainFunction_Write

Function called at fixed cyclic time. SID 0x01.

Prototype:`void Can_MainFunction_Write(void);`

Service for performs the polling of TX confirmation and TX cancellation confirmation when CAN_TX_PROCESSING is set to POLLING.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized.

post: Send the data from that MB that is configured for Tx.

3.5.10 Function Can_MainFunction_Read

Function called at fixed cyclic time. SID = 0x08.

Prototype:`void Can_MainFunction_Read(void);`

Service for performs the polling of RX indications when CAN_RX_PROCESSING is set to POLLING.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized.

post: Receive the data from that MB that is configured for Rx.

3.5.11 Function Can_MainFunction_BusOff

Function called at fixed cyclic time. SID - 0x09.

Prototype:`void Can_MainFunction_BusOff(void);`

Service for performs the polling of bus-off events that are configured statically as 'to be polled'.

This routine is called by:

- CanIf or an upper layer according to Autosar requierements.

pre: Driver must be initialized.

post: Handle the BusOff event.

3.5.12 Function Can_MainFunction_Wakeup

Function called at fixed cyclic time. SID = 0x0A.

Prototype`void Can_MainFunction_Wakeup(void);`

Service for performs performs the polling of wake-up events that are configured statically as 'to be polled'.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized.

post: Handle the wakeup event.

3.5.13 Function Can_AbortMb

Process a message buffer abort.

Prototype:`void Can_AbortMb(Can_HWObjectCountType Hth);`

Table 3-12. Can_AbortMb Arguments

Type	Name	Direction	Description
`Can_HWObjectCountType`	Hth	**input**	HW-transmit handler

This function write a abort code (b'1001) to MBCS[CODE] field of the MB.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized and the current MB transmission should be ready for transmit.

3.5.14 Function Can_SetClockMode

Process a transition from one clock source to another.

Prototype:`void Can_SetClockMode(uint8 can_controller, Can_ClockMode can_clk_mode);`

Table 3-13. Can_SetClockMode Arguments

Type	Name	Direction	Description
`uint8`	can_controller	**input**	controller ID
`Can_ClockMode`	can_clk_mode	**input**	clock mode selection

This function is configuring Can controllers to run on the same baudrate, but having a different MCU source clock.

This routine is called by:

- CanIf or an upper layer according to Autosar requirements.

pre: Driver must be initialized and the current controller should be in Stop state.

3.6 Structure Definitions

Data structures supported by the driver are as per AUTOSAR CAN Driver software specification Version 3.0.

3.6.1 Structure Can_ConfigType

Top Level structure containing all Driver configuration. A pointer to this structure is transmitted to `Can_Init()` to initialize the driver at startup. The application selects one of the configurations by using a pointer to one of the elements of this array as a parameter of the Can_Init function.

Declaration

```
typedef struct
        {
            uint8 Can_ControllersConfigured,
            const Can_RxFiFoTableIdConfigType *const Can_RxFiFoTableIdConfigPtr,
            const Can_ControlerDescriptorType * ControlerDescriptors,
            const Can_ControllerConfigType *const ControllerConfigsPtr,
            Can_IdPtrType FilterMasksPtr,
            const Can_MBConfigContainerType MBConfigContainer,
            Can_HWObjectCountType Can_max_object_Id,
            uint8 Can_ControllerID_Mapping,
            Can_ObjType Can_ObjectType_Mapping
        } Can_ConfigType;
```

Table 3-14. Structure Can_ConfigType member description

Member	Description
Can_ControllersConfigured	Number of Can Controllers configured in Tresos plugin.
Can_RxFiFoTableIdConfigPtr	Pointer to the Table IDs for the RX Fifo.
ControlerDescriptors	Pointer to the first FlexCAN Controller description.
ControllerConfigsPtr	Pointer to the Configuration of FlexCAN controller (CTRL value register & Options).
FilterMasksPtr	Pointer to the first FilterMask value - any controller can have many filter masks for Can messages.
MBConfigContainer	Pointer to the first MB configuration of this Controller.
Can_max_object_Id	Maximum Object IDs configured.
Can_ControllerID_Mapping	Controller ID mapping.
Can_ObjectType_Mapping	Can Object Type mapping.

3.6.2 Structure Can_ControlerDescriptorType

Structure for describing individual FlexCAN controllers on the chip. HRH = Hardware Receive Handle (HRH) is defined and provided by the CAN driver. Each HRH represents exactly one hardware object. The HRH can be used to optimize software filtering. HTH = The Hardware Transmit Handle (HTH) is defined and provided by the CAN driver. Each HTH represents one or several hardware objects, that are configured as hardware transmit pool.

Declaration

```
        typedef struct
        {
            const Can_PCallBackType Can_ErrorNotification,
            const Can_PCallBackType Can_RxFifoOverflowNotification,
            const Can_PCallBackType Can_RxFifoWarningNotification,
            const uint32 CanWakeupSourceID,
            constCan_HWObjectCountType HthOffset,
            const uint8 ControllerOffset,
            const Can_HWObjectCountType FirstIndex,
            const uint8 MaxMBCount,
            const uint8 RxFiFoTableIdConfigIndex
        } Can_ControlerDescriptorType;
```

Table 3-15. Structure Can_ControlerDescriptorType member description

Member	Description
Can_ErrorNotification	Pointer to Error interrupt notification function (ESR[ERR_INT]).
Can_RxFifoOverflowNotification	Pointer to RX FIFO Overflow notification function.
Can_RxFifoWarningNotification	Pointer to RX FIFO Warning notification function.
CanWakeupSourceID	WakeUp source ID (typedef uint32 EcuM_WakeupSourceType - in EcuM.h (uint32)).
HthOffset	HthOffset is calculated (FirstHTH - No.of Rx MBs).
ControllerOffset	Hardware Offset for Can controller: FLEXCAN_A = Offset[0], FLEXCAN_B = Offset[1], ...
FirstIndex	Index of the first MB for this controller (it is in the range of 0 and MaxMBCount).
MaxMBCount	Max Message Buffer number.
RxFiFoTableIdConfigIndex	RxFifo TableID Controller index.

3.6.3 Structure Can_ControllerConfigType

Configuration of FlexCAN controller. This structure is initialized by Tresos considering user settings.

- used by `Can_ConfigType`.

Declaration

```
typedef struct
{
    const uint32 ControlRegister,
    const uint32 ControlRegister_Alternate,
    const uint16 Options,
    const uint8 Can_NumberOfMB
} Can_ControllerConfigType;
```

Table 3-16. Structure Can_ControllerConfigType member description

Member	Description
ControlRegister	Content of the Control Register (CTRL) fields: PRESDIV, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.
ControlRegister_Alternate	Content of the Control Register (CTRL) fields: PRESDIV_Alternate, RJW, PSEG1, PSEG2, CLKSRC, LPB, SMP, BOFF_REC, LOM, PROPSEG.

Table continues on the next page...

Table 3-16. Structure Can_ControllerConfigType member description (continued)

Member	Description
Options	BusOff Sw Recovery, RXFifo En, IDAM Type, Event Trigger Mode TxProcessing/RxProcessing/BusoffProcessing/WakeuProcessing: Polling vs Interrupt mode.
Can_NumberOfMB	Number of message Buffers available for FlexCan unit. Parameter available only if CAN_MIX_MB_SUPPORT is STD_ON.

3.6.4 Structure Can_ControllerStatusType

Records the status of a CAN Controller during runtime.

Declaration

```
typedef struct
{
    Can_HWObjectCountType CancelMBIndex,
    Can_StateTransitionType ControllerState,
    uint8 ControllerSWFlag,
    uint8 FirstTxMBIndex,
    sint8 IntDisableLevel,
    uint8 InterruptMode,
    uint32 MBInterruptMask[2],
    uint32 TxGuard[2],
    PduIdType TxPduId[64],

    uint32 Can_MBMapping[CAN_MAXMB_SUPPORTED]
} Can_ControllerStatusType;
```

Table 3-17. Structure Can_ControllerStatusType member description

Member	Description
CancelMBIndex	Index of MB buffer being cancelled.
ControllerState	FlexCAN controller power state.
ControllerSWFlag	Software flags for tracking interrupt changes state.
FirstTxMBIndex	Index of the first MB used for Tx for a sepcific controller. This value is relative to 0 (which is first MB).
IntDisableLevel	Storage space for Can_DisableControllerInterrupts nesting level.
InterruptMode	Global interrupt autorization state.
MBInterruptMask	Pre-calculated MB INT masks (used in Can_EnableControllerInterrupts).
TxGuard	Guard bits for EXCLUSIVE ACCESS to Tx MBs.
TxPduId	Storage space for PDU_ID (supplied in call to Can_Write and needed after Tx in CanIf_TxConfirmation).
Can_MBMapping	Map for every MB the HOH assigned according to configuration.

Remark: TxGuard and MBInterruptMask vectors can have a size of 1 if only 32 MBs are available on hardware.

Remark: TxPduId vector can have a size of 32 if only 32 MBs are available on hardware.

3.6.5 Structure Can_MBConfigContainerType

Type for storing Message Buffer configurations. The MessageBufferConfigs array is sorted according to:

- HRHs first, HTHs next (AutoSAR requirement)
- Controller ID (HRHs and HTHs belonging to all controllers must be grouped together)
- Message ID (to ensure top priority IDs are first which means they will be serviced first).

Declaration

```
typedef struct
{
    const Can_HWObjectCountType MessageBufferConfigCount,
    const Can_MBConfigObjectType *const MessageBufferConfigsPtr
} Can_MBConfigContainerType;
```

Table 3-18. Structure Can_MBConfigContainerType member description

Member	Description
MessageBufferConfigCount	Number of elements in the array -(having 6 controllers with 64MBs each uint8 is not enough to store this value -> the type is extended to uint16).
MessageBufferConfigsPtr	Pointer to the MB array.

3.6.6 Structure Can_MBConfigObjectType

Type for storing information about Message Buffers (CAN hardware objs).

- used by `Can_MBConfigContainerType`.

Declaration

```
typedef struct
{
    const uint8 ControllerId,
    Can_HWObjectCountType IdMaskIndex,
    CanIdType IdType,
    const uint8 LocalPriority,
    Can_ObjType MBType,
    Can_IdType MessageId,

    uint32 HWObjID
} Can_MBConfigObjectType;
```

Table 3-19. Structure Can_MBConfigObjectType member description

Member	Description
ControllerId	Controller ID (index into controller address array containing Can_ControllerPtrType).
IdMaskIndex	Index into array of Can_FilterMaskType values (uint8/uint16), Current MB and the coresponding filter mask.
IdType	ID type: EXTENDED, STANDARD, MIXED.
LocalPriority	Local priority bits used for arbitration.
MBType	Receive/Transmit.
MessageId	Message ID (extended identifier) (uint16/uint32). configurable by CanHardwareObject/CanIdValue.
HWObjID	Hardware Obiect ID.

3.6.7 Structure Can_PduType

Type used to provide ID, DLC, SDU from CAN interface to CAN driver. HTH/HRH = ID+DLC+SDU.

Declaration

```
typedef struct
{
  Can_IdType id,
  uint8 length,
  uint8 * sdu,
  PduIdType swPduHandle
} Can_PduType;
```

Table 3-20. Structure Can_PduType member description

Member	Description
id	CAN L-PDU = Data Link Layer Protocol Data Unit. Consists of Identifier, DLC and Data(SDU) It is uint32 for CAN_EXTENDEDID=STD_ON, else is uint16.
length	DLC = Data Length Code (part of L-PDU that describes the SDU length).
sdu	CAN L-SDU = Link Layer Service Data Unit. Data that is transported inside the L-PDU.
swPduHandle	The L-PDU Handle = defined and placed inside the CanIf module layer. Each handle represents an L-PDU, which is a constant structure with information for Tx/Rx processing.

3.6.8 Structure Can_RxFiFoTableIdConfigType

Rx Fifo Table ID.

Declaration

```
typedef struct
{
    const uint32 TableFilterMask,
    const uint32 TableId
} Can_RxFiFoTableIdConfigType;
```

Table 3-21. Structure Can_RxFiFoTableIdConfigType member description

Member	Description
TableFilterMask	Table with the Filter masks.
TableId	Table with the IDs specific for Rx Fifo.

3.7 Define Definitions

Data defines supported by the driver are as per AUTOSAR CAN Driver software specification Version 3.0.

3.7.1 Define CAN_ISROPTCODESIZE

Optimization of interrupt service code for size.

Definition:`#define CAN_ISROPTCODESIZE (STD_ON)`

3.7.2 Define CAN_MAINFUNCTION_PERIOD_BUSOFF

Periods for cyclic call of Main function.

Definition:`#define CAN_MAINFUNCTION_PERIOD_BUSOFF 0.0010U`

3.7.3 Define CAN_MAINFUNCTION_PERIOD_READ

Periods for cyclic call of Main function.

Definition:`#define CAN_MAINFUNCTION_PERIOD_READ 0.0010U`

3.7.4 Define CAN_MAINFUNCTION_PERIOD_WAKEUP

Periods for cyclic call of Main function.

Definition: `#define CAN_MAINFUNCTION_PERIOD_WAKEUP 0.0010U`

3.7.5 Define CAN_MAINFUNCTION_PERIOD_WRITE

Periods for cyclic call of Main function.

Definition: `#define CAN_MAINFUNCTION_PERIOD_WRITE 0.0010U`

3.7.6 Define CAN_RXFIFO_ENABLE

Support for Rx Fifo - If at leats one controller supports RxFifo, then this define is set to STD_ON. This define is global for the entire hardware unit (all controllers). Every controller has particular configuration for Rx Fifo (see `Can_ControllerConfigType` structure).

Definition: `#define CAN_RXFIFO_ENABLE (STD_ON)`

3.7.7 Define CAN_RXFIFO_EVENT_UNIFIED

Set if RxFifo events (Warning/Overflow/FrameAvailable) are configured on the same int on INTC vector table. Based on this define separate interrupts handlers or not will be used for the 3 events of the RxFifo.

Definition: `#define CAN_RXFIFO_EVENT_UNIFIED (STD_ON)`

3.7.8 Define CAN_RXPOLL_SUPPORTED

This macro enables `Can_MainFunction_Read()` if at least one controller is set to process Rx in Polling Mode.

Definition: `#define CAN_RXPOLL_SUPPORTED (STD_OFF)`

3.7.9 Define CAN_TIMEOUT_DURATION

Maximum number of tight SW loops to execute while waiting for a state change.

Definition: `#define CAN_TIMEOUT_DURATION (uint32)1000U`

3.7.10 Define CAN_TXCANCEL_SUPPORTED

Support for Transmision Cancellation.

Definition:`#define CAN_TXCANCEL_SUPPORTED (STD_ON)`

3.7.11 Define CAN_TXMUX_SUPPORTED

Support for Multiplexed Transmision - MB set for Tx is put on bus from any TxMB that is available (free).

Definition:`#define CAN_TXMUX_SUPPORTED (STD_ON)`

3.7.12 Define CAN_TXPOLL_SUPPORTED

This macro enables `Can_MainFunction_Write()` if at least one controller is set to process Tx in Polling Mode.

Definition:`#define CAN_TXPOLL_SUPPORTED (STD_OFF)`

3.7.13 Define CAN_VERSION_INFO_API

Support for version info API.

Definition:`#define CAN_VERSION_INFO_API (STD_ON)`

3.7.14 Define CAN_WAKEUP_SUPPORT

CAN driver support for wakeup over CAN Bus.

Definition:`#define CAN_WAKEUP_SUPPORT (STD_ON)`

3.7.15 Define CAN_WAKEUPPOLL_SUPPORTED

This macro enables `Can_MainFunction_Wakeup()` if at least one controller is set to process WakeUp in Polling Mode.

Definition:`#define CAN_WAKEUPPOLL_SUPPORTED (STD_OFF)`

3.7.16 Define CAN_WKP_EXT_SUPPORT

WakeUp support settings. Internal Wakeup is triggered by WakeUp bits Interrupt from IntVectorTable. External Wakeup is supported by Rx pin through WAKEUP_UNIT module.

Definition:`#define CAN_WKP_EXT_SUPPORT (STD_OFF)`

3.7.17 Define CAN_WKP_INT_SUPPORT

WakeUp support settings. Internal Wakeup is triggered by WakeUp bits Interrupt from IntVectorTable. External Wakeup is supported by Rx pin through WAKEUP_UNIT module.

Definition:`#define CAN_WKP_INT_SUPPORT (STD_ON)`

3.8 Symbolic Names DISCLAIMER

All containers having the symbolic name tag set as true in the Autosar schema will generate defines like:

#define <Container_Short_Name> <Container_ID>

For this reason it is forbidden to duplicate the name of such containers across the MCAL configuration, or to use names that may trigger other compile issues (e.g. match existing #ifdefs arguments).

3.9 Configuration Parameters

As per the AUTOSAR specification the driver has two types of configurations parameters:**Pre-Compile** parameters and **Post-Build** parameters. Pre-Compile parameters are stored in the file Can_Cfg.h and Can_Cfg.c. Post-Build parameters are stored in the file Can_PBcfg.c.

The files to be used for different configuration types are listed below:

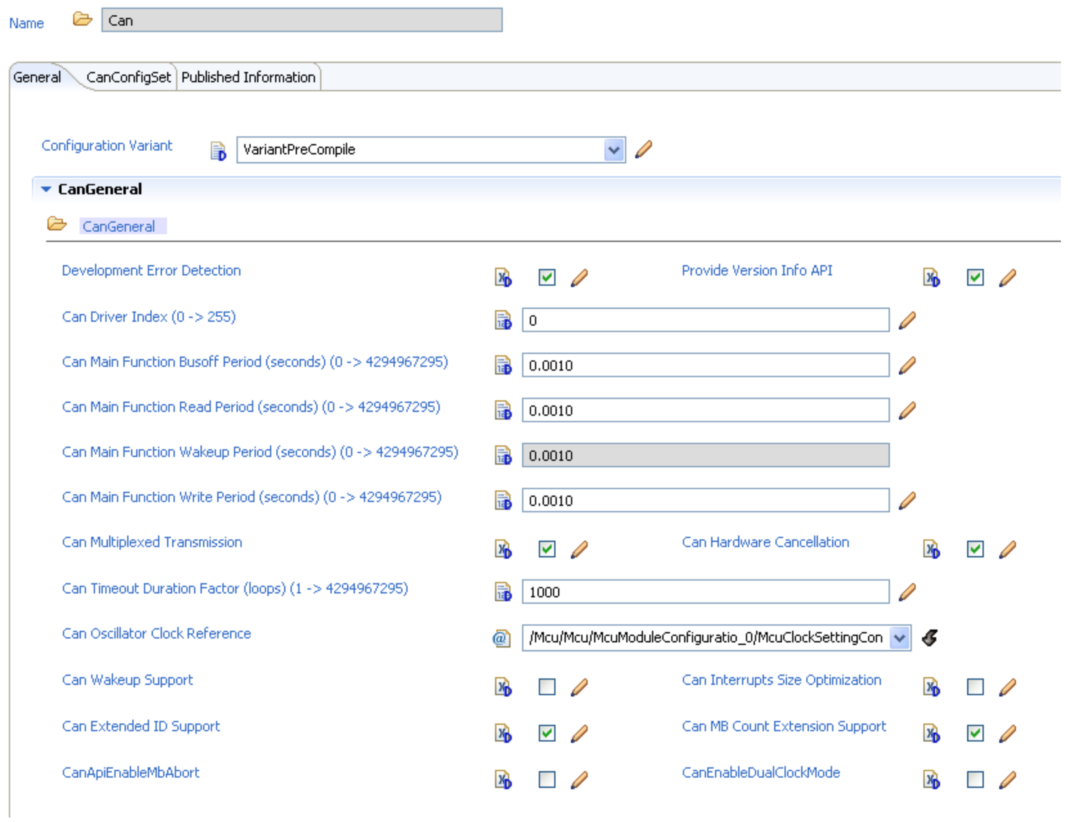
1. **Variant PC**:Can_Cfg.h, Can_Cfg.c
2. **Variant PB**: Can_Cfg.h, Can_PBcfg.c
3. **Variant LT**: Not Applicable

A section for **Can_PBcfg.c** file is needed in linker file to place the post build configuration in desired location.

Please refer to section-8 of **AUTOSAR_SWS_C_ImplementationRules** file for complete details on configuration types.

3.9.1 Can-General Parameters

CanGeneral parameters, their possible values and meaning are described in the following text. CanGeneral parameters are implemented as preprocessor defines.



Name: Can

Configuration Variant: VariantPreCompile

CanGeneral

Development Error Detection	✓	Provide Version Info API	✓
Can Driver Index (0 -> 255)	0		
Can Main Function Busoff Period (seconds) (0 -> 4294967295)	0.0010		
Can Main Function Read Period (seconds) (0 -> 4294967295)	0.0010		
Can Main Function Wakeup Period (seconds) (0 -> 4294967295)	0.0010		
Can Main Function Write Period (seconds) (0 -> 4294967295)	0.0010		
Can Multiplexed Transmission	✓	Can Hardware Cancellation	✓
Can Timeout Duration Factor (loops) (1 -> 4294967295)	1000		
Can Oscillator Clock Reference	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingCon		
Can Wakeup Support	☐	Can Interrupts Size Optimization	☐
Can Extended ID Support	✓	Can MB Count Extension Support	✓
CanApiEnableMbAbort	☐	CanEnableDualClockMode	☐

Figure 3-2. Can General Parameters

3.9.1.1 IMPLEMENTATION_CONFIG_VARIANT

Table 3-22. IMPLEMENTATION_CONFIG_VARIANT

Description	Defines whether precompile version is used. Using this option with VariantPostBuild value, Tresos can generate many CanConfigSet variants.
Class	Implementation Specific Parameter
Range	VariantPreCompile, VariantPostBuild

Table continues on the next page...

Table 3-22. IMPLEMENTATION_CONFIG_VARIANT (continued)

Default	VariantPreCompile
Source File	Can_Cfg.h
Source Representation	#define CAN_PRECOMPILE_SUPPORT
Autosar 3.0 Requirement	NA

Note

This parameter permit to generate many configurations if VariantPostBuild is selected.

Note

This parameter has the label "Configuration Variant" in Tresos plugin interface.

3.9.1.2 CanDevErrorDetection

Table 3-23. CanDevErrorDetection

Description	Switches the Development Error Detection and Notification ON or OFF.
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_DEV_ERROR_DETECT (STD_ON)
Autosar 3.0 Requirement	CAN064

Note

Setting this parameter off code size is reduced, but some Autosar requirements are not tested (no Det errors are reported in this way).

Note

This parameter has the label "Development Error Detection" in Tresos plugin interface.

3.9.1.3 CanVersionInfoApi

Table 3-24. CanVersionInfoApi

Description	Defines whether version information reporting should be included at compile time (STD_ON) or excluded (STD_OFF).
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_VERSION_INFO_API (STD_ON)
Autosar 3.0 Requirement	CAN106

Note

Setting this parameter off code size is reduced, but no support for "Can_GetVersionInfo" API will be available.

Note

This parameter has the label "Provide Version Info API " in Tresos plugin interface.

3.9.1.4 CanIndex

Table 3-25. CanIndex

Description	Specifies the Instance ID of this module instance. If only one instance is present it shall have the ID 0.
Class	Autosar Parameter
Range	Integer
Default	0
Source File	Can_Cfg.h
Source Representation	#define CAN_INSTANCE (uint8)0U
Autosar 3.0 Requirement	CAN320
NOTE	*This parameter is transmitted as the second parameter to the Det_Reporterror function.*

Note

This parameter is transmitted as the second parameter to the Det_ReportError() function.

Note

This parameter has the label "Can Driver Index" in Tresos plugin interface.

3.9.1.5 CanMainFunctionBusOffPeriod

Table 3-26. CanMainFunctionBusOffPeriod

Description	Describes the period for cyclic call to Can_MainFunction_Busoff (in seconds).
Class	Autosar Parameter
Range	Float
Default	0.001
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_PERIOD_BUSOFF (uint32)0.001U
Autosar 3.0 Requirement	CAN355

Note

This period value is not used in the Can driver. It should be exported to SchM for using it when polling mode is selected.

Note

This parameter has the label "Can Main Function Busoff Period (seconds)" in Tresos plugin interface.

3.9.1.6 CanMainFunctionReadPeriod

Table 3-27. CanMainFunctionReadPeriod

Description	Describes the period for cyclic call to Can_MainFunction_Read (in seconds).
Class	Autosar Parameter
Range	Float
Default	0.001
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_PERIOD_READ (uint32)0.001U
Autosar 3.0 Requirement	CAN356

Note

This period value is not used in the Can driver. It should be exported to SchM for using it when polling mode is selected.

Note

This parameter has the label "Can Main Function Read Period (seconds)" in Tresos plugin interface.

3.9.1.7 CanMainFunctionWakeupPeriod

Table 3-28. CanMainFunctionWakeupPeriod

Description	Describes the period for cyclic call to Can_MainFunction_Wakeup (in seconds).
Class	Autosar Parameter
Range	Float
Default	0.001
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_PERIOD_WAKEUP (uint32)0.001U
Autosar 3.0 Requirement	CAN357

Note

This period value is not used in the Can driver. It should be exported to SchM for using it when polling mode is selected.

Note

This parameter has the label "Can Main Function Wakeup Period (seconds)" in Tresos plugin interface.

3.9.1.8 CanMainFunctionWritePeriod

Table 3-29. CanMainFunctionWritePeriod

Description	Describes the period for cyclic call to Can_MainFunction_Write (in seconds).
Class	Autosar Parameter
Range	Float
Default	0.001
Source File	Can_Cfg.h
Source Representation	#define CAN_MAINFUNCTION_PERIOD_WRITE (uint32)0.001U
Autosar 3.0 Requirement	CAN358

Note

This period value is not used in the Can driver. It should be exported to SchM for using it when polling mode is selected.

Note

This parameter has the label "Can Main Function Write Period (seconds)" in Tresos plugin interface.

3.9.1.9 CanMultiplexedTransmission

Table 3-30. CanMultiplexedTransmission

Description	Defines whether support for multiplex transmission should be included at compile time (STD_ON) or excluded (STD_OFF).
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_TXMUX_SUPPORTED (STD_ON)
Autosar 3.0 Requirement	CAN095

Note

When Multiplex Transmission is enabled (ON), several TRANSMIT HardwareObjects can have the same CanObjectID, but they must belong to the same controller. Multiplex transmission means the driver will look for a free MessageBuffer in the range of the Hth value used in Can_Write().

Note

This parameter has the label "Can Multiplexed Transmission" in Tresos plugin interface.

3.9.1.10 CanHardwareCancellation

Table 3-31. CanHardwareCancellation

Description	Specifies if hardware cancellation shall be supported ON or OFF.
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_TXCANCEL_SUPPORTED (STD_ON)
Autosar 3.0 Requirement	CAN069

Note

Setting this control to false the code size is reduced, but no support for message buffer cancellation is available.

Note

This parameter has the label "Can Hardware Cancellation" in Tresos plugin interface.

3.9.1.11 CanTimeoutDurationFactor

Table 3-32. CanTimeoutDurationFactor

Description	Specifies the maximum number of loops for blocking function until a timeout is raised in short term wait loops.
Class	Autosar Parameter
Range	Integer
Default	10000
Source File	Can_Cfg.h
Source Representation	#define CAN_TIMEOUT_DURATION (uint32)10000U
Autosar 3.0 Requirement	CAN113

Note

This value represents a number of finite "while-loops" that the Driver can wait until a hardware set is configured. There is no correspondence between this number of loops and the number of uC cycles.

Note

This parameter has the label "Can Timeout Duration Factor (loops)" in Tresos plugin interface.

3.9.1.12 CanOscillatorClockRef

Table 3-33. CanOscillatorClockRef

Description	Defines the crystal frequency used.
Class	Implementation Specific Parameter
Range	NA
Default	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingConfig
Source File	NA
Source Representation	NA
Autosar 3.0 Requirement	NA

Note

This parameter should have the same value according to external oscillator board.

Note

This parameter has the label "Can Oscillator Clock Reference" in Tresos plugin interface.

3.9.1.13 CanWakeupSupport

Table 3-34. CanWakeupSupport

Description	Defines whether Can driver supports for Wakeup over Can bus.
Class	Autosar Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_WAKEUP_SUPPORT (STD_ON)
Autosar 3.0 Requirement	CAN330

Note

This parameter enables the wakeup functionality in case the platform supports this feature. It is related to internal and external wakeup.

Note

This parameter has the label "Can Wakeup Support" in Tresos plugin interface.

3.9.1.14 CanCodeSizeOptimization

Table 3-35. CanCodeSizeOptimization

Description	Enables optimization of interrupt service routines for code size.
Class	Implementation Specific Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_ISROPTCODESIZE (STD_ON)
Autosar 3.0 Requirement	NA

Note

This parameter permit to implement all interrupts routines for all MBs with the same routine. Code size is reduced. Ex: Can_IsrFCA_MB_00_03, Can_IsrFCA_MB_04_07, etc, are replaced by Can_IsrFCA_UNI.

Note

This parameter has the label "Can Interrupts Size Optimization" in Tresos plugin interface.

3.9.1.15 CanExtendedIdSupport

Table 3-36. CanExtendedIdSupport

Description	Enables support of Extended/Mixed mode.
Class	Implementation Specific Parameter
Range	True, False
Default	True
Source File	Can_Cfg.h
Source Representation	#define CAN_EXTENDEDID (STD_ON)
Autosar 3.0 Requirement	NA

Note

This parameter permit to enable the Message ID representation on (uint32) size variable, else (uint16) is used.

Note

This parameter has the label "Can Extended ID Support" in Tresos plugin interface.

3.9.1.16 CanMBCountExtensionSupport

Table 3-37. CanMBCountExtensionSupport

Description	Enables support of more than 255 Can Hardware Objects.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_MBCOUNTEXTENSION (STD_ON)
Autosar 3.0 Requirement	NA

Note

This parameter permit to declare more than 255 MBs in the configuration. If total MBs for all controllers exceed 255 then this parameter must be ON.

Note

This parameter has the label "Can MB Count Extension Support" in Tresos plugin interface.

3.9.1.17 CanApiEnableMbAbort

Table 3-38. CanApiEnableMbAbort

Description	Enables an additional API, to write an ABORT code (b1001) to the message buffer CODE filed to abort a message transmission.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_API_ENABLE_ABORT_MB (STD_ON)
Autosar 3.0 Requirement	NA

Note

This parameter value is considered only if "CanHardwareCancellation" is true.

Note

This parameter has the label "CanApiEnableMbAbort" in Tresos plugin interface.

3.9.1.18 CanEnableDualClockMode

Table 3-39. CanEnableDualClockMode

Description	Enables support for dual clock API - Can_SetClockMode(). Can controller is able to run on the same bit rate over CAN bus using two different clock sources that can be changed while the controller is not running.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_DUAL_CLOCK_MODE (STD_ON)

Table continues on the next page...

Table 3-39. CanEnableDualClockMode (continued)

Autosar 3.0 Requirement	NA

Note

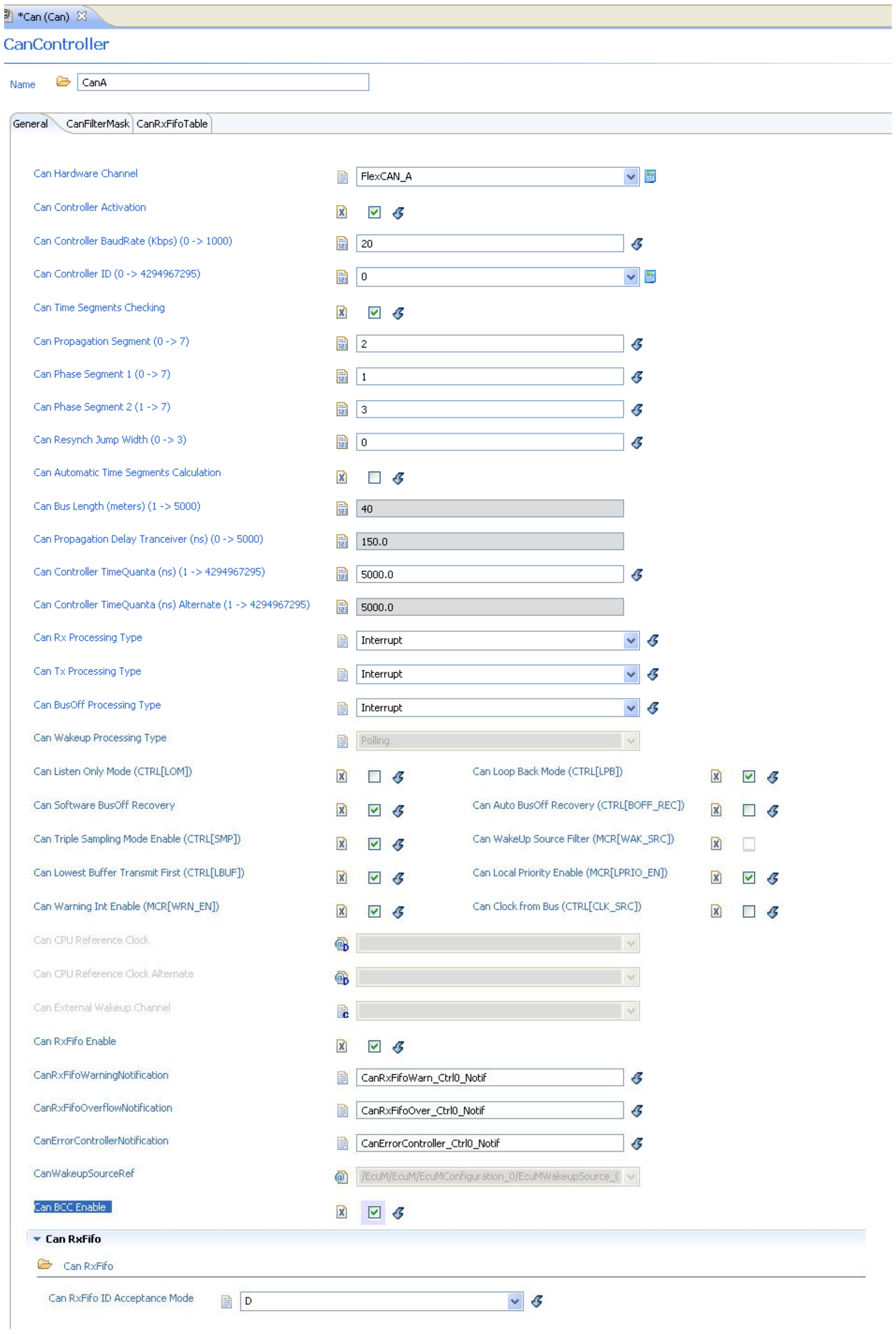
This parameter is visible and configurable only if Can.CanConfig.DualClockMode=STD_ON from Resource files.

Note

This parameter has the label "CanEnableDualClockMode" in Tresos plugin interface.

3.9.2 Can-Controller Parameters

Can-Controller parameters, their possible values and meaning are described in the following text. The Can-Controller parameters are implemented as constant structures and arrays stored in flash memory of the MCU.



*Can (Can)

CanController

Name: CanA

General | CanFilterMask | CanRxFifoTable

Can Hardware Channel	FlexCAN_A
Can Controller Activation	✓
Can Controller BaudRate (Kbps) (0 -> 1000)	20
Can Controller ID (0 -> 4294967295)	0
Can Time Segments Checking	✓
Can Propagation Segment (0 -> 7)	2
Can Phase Segment 1 (0 -> 7)	1
Can Phase Segment 2 (1 -> 7)	3
Can Resynch Jump Width (0 -> 3)	0
Can Automatic Time Segments Calculation	☐
Can Bus Length (meters) (1 -> 5000)	40
Can Propagation Delay Tranceiver (ns) (0 -> 5000)	150.0
Can Controller TimeQuanta (ns) (1 -> 4294967295)	5000.0
Can Controller TimeQuanta (ns) Alternate (1 -> 4294967295)	5000.0
Can Rx Processing Type	Interrupt
Can Tx Processing Type	Interrupt
Can BusOff Processing Type	Interrupt
Can Wakeup Processing Type	Polling

Can Listen Only Mode (CTRL[LOM])	☐	Can Loop Back Mode (CTRL[LPB])	✓
Can Software BusOff Recovery	✓	Can Auto BusOff Recovery (CTRL[BOFF_REC])	☐
Can Triple Sampling Mode Enable (CTRL[SMP])	✓	Can WakeUp Source Filter (MCR[WAK_SRC])	☐
Can Lowest Buffer Transmit First (CTRL[LBUF])	✓	Can Local Priority Enable (MCR[LPRIO_EN])	✓
Can Warning Int Enable (MCR[WRN_EN])	✓	Can Clock from Bus (CTRL[CLK_SRC])	☐

Can CPU Reference Clock	
Can CPU Reference Clock Alternate	
Can External Wakeup Channel	
Can RxFifo Enable	✓
CanRxFifoWarningNotification	CanRxFifoWarn_Ctrl0_Notif
CanRxFifoOverflowNotification	CanRxFifoOver_Ctrl0_Notif
CanErrorControllerNotification	CanErrorController_Ctrl0_Notif
CanWakeupSourceRef	/EcuM/EcuM/EcuMConfiguration_0/EcuMWakeupSource_(
Can BCC Enable	✓

Can RxFifo

Can RxFifo

Can RxFifo ID Acceptance Mode	D

Figure 3-3. Can Controller Parameters

3.9.2.1 CanHwChannel

Table 3-40. CanHwChannel

Description	Specifies which one of the on-chip FlexCan interface is associated with the controller ID.
Class	Implementation Specific Parameter
Range	FlexCAN_A, FlexCAN_B, ...
Default	FlexCAN_A
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* Can Controller Offset on chip */
	FLEXCAN_A_OFFSET,
	....
	}

Autosar 3.0 Requirement	NA

Note

This parameter set the Can hardware channel for which the settings are implemented. The order from hardware (Can_A first and Can_B second) is not mandatory to be respected in the CanController list from Tresos plugin.

Note

This parameter has the label "Can Hardware Channel" in Tresos plugin interface.

3.9.2.2 CanControllerActivation

Table 3-41. CanControllerActivation

Description	Defines if a CAN controller is used in the configuration.
Class	Autosar Parameter
Range	True , False
Default	True
Source File	Can_Cfg.c, Can_PBcfg.c

Table continues on the next page...

Table 3-41. CanControllerActivation (continued)

Source Representation	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* Can Controller Offset on chip */
	FLEXCAN_A_OFFSET,
	....
	}

Autosar 3.0 Requirement	CAN315

Note

This parameter permit to use the current settings for one of Can controllers. If this control is set to ‘false’ no Can controller initialization is made (NULL_PTR). Can controller offset is set to FLEXCAN_NULL_OFFSET.

Note

This parameter has the label "Can Controller Activation" in Tresos plugin interface.

3.9.2.3 CanControllerBaudRate

Table 3-42. CanControllerBaudRate

Description	Specifies the bit drate of the CAN controller in kbps. Can bus maximum speed is 1Mbps.
Class	Autosar Parameter
Range	0 - 1000
Default	20
Source File	NA
Source Representation	NA
Autosar 3.0 Requirement	CAN005

Note

This parameter set the bit rate for serial Can message transmission. The value must respect the Can standards and also considering this value and clock value will set the time segments according to Can protocol.

Note

This parameter has the label "Can Controller BaudRate (Kbps)" in Tresos plugin interface.

3.9.2.4 CanControllerId

Table 3-43. CanControllerId

Description	This parameter provides the controller ID which is unique in a given CAN Driver. The value for this parameter starts with 0 and continues without any gaps.
Class	Autosar Parameter
Range	Integer
Default	0
Source File	Can_Cfg.h
Source Representation	#define CanController_0 (uint8)0U
Autosar 3.0 Requirement	CAN316

Note

This parameter set an ID for the current Can controller configuration. A define is generated for each Can controller.

Note

This parameter has the label "Can Controller ID" in Tresos plugin interface.

3.9.2.5 CanControllerCheckCanStandard

Table 3-44. CanControllerCheckCanStandard

Description	If enabled, Can plugin checks that CanControllerPropSeg, CanControllerSeg1, CanControllerSeg2 and CanSyncJumpWidth settings match the CAN Standard Compliant Bit Time Segment Settings.
Class	Implementation Specific Parameter.
Range	True, False
Default	True
Source File	NA
Source Representation	NA
Autosar 3.0 Requirement	NA

Note

This parameter is used in the Can_Cfg.c and Can_PBcfg.c files validating time segments values if it is enabled.

Note

This parameter has the label "Can Time Segments Checking" in Tresos plugin interface.

3.9.2.6 CanControllerPropSeg

Table 3-45. CanControllerPropSeg

Description	Propogation delay in time quanta.
Class	Autosar Parameter
Range	0 - 7
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[PROPSEG] - Propagation segment */
	0U ,
	....
	}

Autosar 3.0 Requirement	CAN073

Note

It is used to compensate the physical delay within the CAN network (CTRL[PROPSEG] - 0..7).

Note

This parameter has the label "Can Propagation Segment" in Tresos plugin interface.

3.9.2.7 CanControllerSeg1

Table 3-46. CanControllerSeg1

Description	Specifies Phase Segment 1
Class	Autosar Parameter
Range	0 - 7
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-46. CanControllerSeg1 (continued)

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[PSEG1] - Segment 1 */
	....
	}

Autosar 3.0 Requirement	CAN074

Note

Specifies the Phase Segment 1 in time quantas (CTRL[PSEG1] = 0..7). PHASE_BUF_SEG1 = (PSEG1 + 1) * Tq . The PHASE_BUF_SEG1 valid values are 1-8 Tq.

Note

This parameter has the label "Can Phase Segment 1" in Tresos plugin interface.

3.9.2.8 CanControllerSeg2

Table 3-47. CanControllerSeg2

Description	Specifies Phase Segment 2
Class	Autosar Parameter
Range	1 - 7
Default	2
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[PSEG2] - Segment 2 */
	....
	}

Autosar 3.0 Requirement	CAN075

Note

Specifies Phase Segment 2 in time quantum (CTRL[PSEG2] = 1..7). PHASE_BUF_SEG2 = (PSEG2 + 1) * Tq . The PHASE_BUF_SEG2 valid values are 2-8 Tq.

Note

This parameter has the label "Can Phase Segment 2" in Tresos plugin interface.

3.9.2.9 CanSyncJumpWidth

Table 3-48. CanSyncJumpWidth

Description	Specifies Synchronization Jump Width.
Class	Implementation Specific Parameter
Range	0 - 3
Default	2
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[RJW] - Resynchronization Jump Width */
	....
	}

Autosar 3.0 Requirement	NA

Note

Specifies Synchronization Jump Width: CTRL[RJW] = 0..3.

Note

This parameter has the label "Can Resynch Jump Width" in Tresos plugin interface.

3.9.2.10 CanAdvancedSetting

Table 3-49. CanAdvancedSetting

Description	If True initiates the derivation of the Can bit timing values from the CanControllerBaudRate parameter and source clock value.
Class	Implementation Specific Parameter

Table continues on the next page...

Table 3-49. CanAdvancedSetting (continued)

Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	NA
Autosar 3.0 Requirement	NA

Note

If this parameter is set to “True” then “CanControllerPropSeg”, “CanControllerSeg1”, “CanControllerSeg2” and “CanSyncJumpWidth” are disabled and these values are calculated indirectly. In the same time two another parameters are enabled: "BusLength" and "PropagationDelayOfTranceiver".

Note

This parameter has the label "Can Automatic Time Segments Calculation" in Tresos plugin interface.

3.9.2.11 CanBusLength

Table 3-50. CanBusLength

Description	Specifies the Can bus length in meters. This parameter is used for PropSeg calculation when “CanAdvancedSetting” is set to true.
Class	Implementation Specific Parameter
Range	1 – 5000
Default	40
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 3.0 Requirement	NA

Note

This parameter is available only if “AdvancedSetting” is set to true.

Note

This parameter has the label "Can Bus Length (meters)" in Tresos plugin interface.

3.9.2.12 CanPropDelayTranceiver

Table 3-51. PropagationDelayOfTranceiver

Description	Specifies the propagation delay in ns for the Can transceiver. This parameter is used for PropSeg calculation when “CanAdvancedSetting” is set to true.
Class	Implementation Specific Parameter
Range	1 – 5000
Default	150
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 3.0 Requirement	NA

Note

This parameter is available only if “CanAdvancedSetting” is set to true.

Note

This parameter has the label "Can Propagation Delay Tranceiver (ns)" in Tresos plugin interface.

3.9.2.13 CanControllerTimeQuanta

Table 3-52. CanControllerTimeQuanta

Description	Specifies the time quanta for the controller (it's time unit expressed in nanoseconds). The calculation of the resulting prescaler value depending on module clocking and time quanta shall be done offline. FreqTq = 1 / CanControllerTimeQuanta; Prescaler = FreqCanClk / FreqTq.
Class	Autosar Parameter
Range	1 - 4294967295
Default	500
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 3.0 Requirement	CAN063

Note

This parameter has the label "Can Controller TimeQuanta (ns)" in Tresos plugin interface.

3.9.2.14 CanControllerTimeQuanta_Alt

Table 3-53. CanControllerTimeQuanta_Alt

Description	Specifies the alternate time quanta for the controller (it's time unit expressed in nanoseconds). The calculation of the resulting prescaler value depending on module clocking and time quanta shall be done offline. FreqTq = 1 / CanControllerTimeQuanta; Prescaler = FreqCanClk / FreqTq. This value should have the same value as "CanControllerTimeQuanta" parameter and is recommended to be changed only if prescaller is out of range with the new source clock.
Class	Implementation Specific Parameter
Range	1 - 4294967295
Default	500
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 3.0 Requirement	NA

Note

This parameter is available/editable only if "CanEnableDualClockMode" is set to true.

Note

This parameter has the label "Can Controller TimeQuanta (ns) Alternate" in Tresos plugin interface.

3.9.2.15 CanRxProcessing

Table 3-54. CanRxProcessing

Description	Specifies if Rx events are polled inside Can_MainFunction_Read() or cause an interrupt.
Class	Autosar Parameter
Range	Polling , Interrupt
Default	Polling
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Controller Options ===== */
	/* RxPoll Enabled */*/
	CAN_CONTROLLERCONFIG_RXPOL_EN |
	....
	}

Autosar 3.0 Requirement	CAN317

Note

This parameter set how it is implemented the handling of Rx confirmation events: by polling or by interrupt.

Note

This parameter has the label "Can Rx Processing Type" in Tresos plugin interface.

3.9.2.16 CanTxProcessing

Table 3-55. CanTxProcessing

Description	Specifies if TX events are polled inside Can_MainFunction_Write() or cause an interrupt.
Class	Autosar Parameter
Range	Polling , Interrupt
Default	Polling
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Controller Options ===== */
	/* TxPoll Enabled */*/
	CAN_CONTROLLERCONFIG_TXPOL_EN |
	....
	}

Autosar 3.0 Requirement	CAN318

Note

This parameter set how it is implemented the handling of Tx confirmation events: by polling or by interrupt.

Note

This parameter has the label "Can Tx Processing Type" in Tresos plugin interface.

3.9.2.17 CanBusOffProcessing

Table 3-56. CanBusOffProcessing

Description	Specifies if bus-off events are polled inside Can_Main_Function BusOff() or cause an interrupt.
Class	Autosar Parameter

Table continues on the next page...

Table 3-56. CanBusOffProcessing (continued)

Range	Polling , Interrupt
Default	Polling
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Controller Options ===== */
	/* BusOffPoll Enabled */*/
	CAN_CONTROLLERCONFIG_BOPOL_EN |
	....
	}

Autosar 3.0 Requirement	CAN314

Note

This parameter set how it is implemented the handling of BusOff confirmation events: by polling or by interrupt.

Note

This parameter has the label "Can BusOff Processing Type" in Tresos plugin interface.

3.9.2.18 CanWakeupProcessing

Table 3-57. CanWakeupProcessing

Description	Specifies if wakeup events are polled inside Can_Main_FunctionWakeup() or cause an interrupt.
Class	Autosar Parameter
Range	Polling , Interrupt
Default	Polling
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Controller Options ===== */
	/* WakeupPoll Enabled */*/
	CAN_CONTROLLERCONFIG_WKPOL_EN |
	....
	}

Table continues on the next page...

Table 3-57. CanWakeupProcessing (continued)

Autosar 3.0 Requirement	CAN319

Note

This parameter set how it is implemented the handling of Wakeup confirmation events: by polling or by interrupt.

Note

This parameter has the label "Can Wakeup Processing Type" in Tresos plugin interface.

3.9.2.19 CanListenOnlyMode

Table 3-58. CanListenOnlyMode

Description	Enables the Listen only mode.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[LOM] - Listen only mode */
	....
	}

Autosar 3.0 Requirement	NA

Note

In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode.

Note

This parameter has the label "Can Listen Only Mode (CTRL[LOM])" in Tresos plugin interface.

3.9.2.20 CanLoopBackMode

Table 3-59. CanLoopBackMode

Description	Enables the Loop Back Mode.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	 /* ===== Control Register - CTRL ===== */
	 ....
	 /* CTRL[LPB] - Loop-back mode */
	 ....
	 }

Autosar 3.0 Requirement	NA

Note

The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic "1").

Note

This parameter has the label "Can Loop Back Mode (CTRL[LPB])" in Tresos plugin interface.

3.9.2.21 CanSoftwareBusOffRecovery

Table 3-60. CanSoftwareBusOffRecovery

Description	Enables Automatic Bus Recovery Off.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Table continues on the next page...

Table 3-60. CanSoftwareBusOffRecovery (continued)

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Controller Options ===== */
	/* Software BusOff Recovery */
	CAN_CONTROLLERCONFIG_BUSOFFSWREC |
	....
	}

Autosar 3.0 Requirement	NA

Note

Enables Software Bus Off recovery when automatic recovering from BusOff state is disabled for CAN controller.

Note

This parameter has the label "Can Software BusOff Recovery" in Tresos plugin interface.

3.9.2.22 CanAutoBusOffRecovery

Table 3-61. CanAutoBusOffRecovery

Description	Enables Automatic Bus Recovery Off.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[BOFF_REC] - Bus off recovery */
	....
	}

Autosar 3.0 Requirement	NA

Note

Enables automatic BusOff recovery (CTRL[BOFF_REC] bit). If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user.

Note

This parameter has the label "Can Auto BusOff Recovery (CTRL[BOFF_REC])" in Tresos plugin interface.

3.9.2.23 CanTripleSamplingEnable

Table 3-62. CanTripleSamplingEnable

Description	Enables acquisition of 3 samples and majority voting for the value of received bit.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[SMP] - Sampling mode */
	....
	}

Autosar 3.0 Requirement	NA

Note

This bit defines the sampling mode of CAN bits at the Rx input. 1 - Enables acquisition of 3 samples and majority voting for the value of received bit. 0 - Just one sample is used to determine the bit value.

Note

This parameter has the label "Can Triple Sampling Mode Enable (CTRL[SMP])" in Tresos plugin interface.

3.9.2.24 CanWakeUpSourceFilter

Table 3-63. CanWakeUpSourceFilter

Description	Defines whether the integrated low-pass filter is applied to protect the Rx Can input from spurious wake up.
Class	Implementation Specific Parameter

Table continues on the next page...

Table 3-63. CanWakeUpSourceFilter (continued)

Range	True, False
Default	True
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Controller Options ===== */
	/* This setting defines whether the integrated low-pass filter is applied to protect the Rx CAN input from spurious wake up. */
	CAN_CONTROLLERCONFIG_WAKEUP_SRC |
	....
	}

Autosar 3.0 Requirement	NA

Note

Defines whether the integrated low-pass filter is applied to protect the Rx CAN input from spurious wake up. 1 = FlexCAN uses the filtered Rx input to detect recessive to dominant edges on the CAN bus. 0 = FlexCAN uses the unfiltered Rx input to detect recessive to dominant edges on the CAN bus. This option will provide the setting for init the controller with MCR[WAK_SRC] bit set.

Note

This parameter has the label "Can WakeUp Source Filter (MCR[WAK_SRC])" in Tresos plugin interface.

3.9.2.25 CanLowestBuffTransmitFirst

Table 3-64. CanLowestBuffTransmitFirst

Description	This parameter defines the ordering mechanism for MB transmission.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Table continues on the next page...

Table 3-64. CanLowestBuffTransmitFirst (continued)

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[LBUF] - Lowest Buffer Transmitted First */
	....
	}

Autosar 3.0 Requirement	NA

Note

CTRL[LBUF]. This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the MCR[LPRIO_EN] bit doesn't affect the priority arbitration.

Note

This parameter has the label "Can Lowest Buffer Transmit First (CTRL[LBUF])" in Tresos plugin interface.

3.9.2.26 CanLocalPriorityEn

Table 3-65. CanLocalPriorityEn

Description	This field is used when MCR[LPRIO_EN] is set and makes sense only for Tx MBs.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Controller Options ===== */
	....
	/* Local Priority Feature */
	CAN_CONTROLLERCONFIG_LPRIO_EN |
	}

Autosar 3.0 Requirement	NA

Note

MCR[LPRIO_EN]. This bit controls whether the local priority feature is enabled or not.

Note

This parameter has the label "Can Local Priority Enable (MCR[LPRIO_EN])" in Tresos plugin interface.

3.9.2.27 CanWarningEnable

Table 3-66. CanWarningEnable

Description	This parameter defines if warning interrupt is enabled for Rx and Tx.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs0_PB[CAN_MAXCONTROLLERCOUNT] = { …….. / * Warning Interrupt Enable Feature */ CAN_CONTROLLERCONFIG_WRNINT_EN | ………..
Autosar 3.0 Requirement	NA

Note

This parameter has the label "Can Warning Int Enable (MCR[WRN_EN])" in Tresos plugin interface.

3.9.2.28 CanClockFromBus

Table 3-67. CanClockFromBus

Description	Switches the source clock for the module to the system bus (rather than crystal).
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.c, Can_PBcfg.c

Table continues on the next page...

Table 3-67. CanClockFromBus (continued)

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	/* ===== Control Register - CTRL ===== */
	....
	/* CTRL[CLK_SRC] - Clock source */
	....
	}

Autosar 3.0 Requirement	NA

Note

Switches the source clock for the module to the system bus (rather than crystal). 1 = The CAN engine clock source is the bus clock.(from MCU). 0 = The CAN engine clock source is the oscillator clock.

Note

This parameter has the label "Can Clock from Bus (CTRL[CLK_SRC])" in Tresos plugin interface.

3.9.2.29 CanCpuClockRef

Table 3-68. CanCpuClockRef

Description	Reference to the CPU clock configuration, which is set in the MCU driver configuration. This parameter is enabled only if “CanClockFromBus” is set to true.
Class	Autosar Parameter
Range	NA
Default	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingConfig/McuClockReferencePoin_0
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	NA
Autosar 3.0 Requirement	CAN313

Note

Reference to the CPU clock configuration, which is set in the MCU driver configuration. This control is used only if "CanClockFromBus" = "true". MCU plugin need to be added and then give the reference to it.

Note

This parameter has the label "Can CPU Reference Clock" in Tresos plugin interface.

3.9.2.30 CanCpuClockRef_Alternate

Table 3-69. CanCpuClockRef_Alternate

Description	Reference to the CPU clock configuration, which is set in the MCU driver configuration. This parameter is enabled only if “CanClockFromBus” is set to true.
Class	Implementation Specific Parameter
Range	NA
Default	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingConfig/McuClockReferencePoin_0
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	NA
Autosar 3.0 Requirement	NA

Note

Reference to the CPU clock configuration, which is set in the MCU driver configuration. This parameter is available/editable only if "Can.CanConfig.DualClockMode" is set to STD_ON from Resource files, "CanEnableDualClockMode" is set to true and "CanClockFromBus" = "true". MCU plugin need to be added and then give the reference to it.

Note

This parameter has the label "Can CPU Reference Clock Alternate" in Tresos plugin interface.

3.9.2.31 CanControllerExternalWakeupCh

Table 3-70. CanControllerExternalWakeupCh

Description	Defines the WKPU channel that is assigned to Rx pin of the current controller. This option is enabled only if External Wakeup is enabled from Resource files.
Class	Implementation Specific Parameter
Range	WKPU0 – WKPU23
Default	WKPU4
Source File	Can_Cfg.c, Can_PBcfg.c

Table continues on the next page...

Table 3-70. CanControllerExternalWakeupCh (continued)

Source Representation	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* WKPU source */
	0
	....
	}

Autosar 3.0 Requirement	NA

Note

Identify the External WakeUp channel ID associated to the selected CAN controller/network to be used to wake-up the core.

Note

This parameter has the label "Can External Wakeup Channel" in Tresos plugin interface.

3.9.2.32 CanControllerRXFifoEnable

Table 3-71. CanControllerRXFifoEnable

Description	Defines if RX FIFO feature of CAN controller is used in the configuration.
Class	Implementation Specific Parameter
Range	True, False
Default	False
Source File	Can_Cfg.h
Source Representation	#define CAN_A_FIFO_EN (STD_ON)
Autosar 3.0 Requirement	NA

Note

Defines if RxFifo feature of CAN controller is used in the configuration. If Fifo feature of CAN controller is enabled, First 8 Message Buffers will be used by Fifo engine.

Note

This parameter has the label "Can RxFifo Enable" in Tresos plugin interface.

3.9.2.33 CanRxFifoWarningNotification

Table 3-72. CanRxFifoWarningNotification

Description	Defines the handler for Rx Fifo warning.
Class	Implementation Specific Parameter
Range	String
Default	NULL_PTR
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	extern FUNC(void, CAN_CODE) handler_name (void);
Autosar 3.0 Requirement	NA

Note

This parameter has the label "CanRxFifoWarningNotification" in Tresos plugin interface.

3.9.2.34 CanRxFifoOverflowNotification

Table 3-73. CanRxFifoOverflowNotification

Description	Defines the handler for Rx Fifo overflow.
Class	Implementation Specific Parameter
Range	String
Default	NULL_PTR
Source File	Can_Cfg.c, Can_PBcfg.c
Source Representation	extern FUNC(void, CAN_CODE) handler_name (void);
Autosar 3.0 Requirement	NA

Note

This parameter has the label "CanRxFifoOverflowNotification" in Tresos plugin interface.

3.9.2.35 CanErrorControllerNotification

Table 3-74. CanErrorControllerNotification

Description	Defines the handler for error controller.
Class	Implementation Specific Parameter
Range	String
Default	NULL_PTR
Source File	Can_Cfg.c, Can_PBcfg.c

Table continues on the next page...

Table 3-74. CanErrorControllerNotification (continued)

Source Representation	extern FUNC(void, CAN_CODE) handler_name (void);
Autosar 3.0 Requirement	NA

Note

This parameter has the label "CanErrorControllerNotification" in Tresos plugin interface.

3.9.2.36 CanWakeupSourceRef

Table 3-75. CanWakeupSourceRef

Description	This parameter contains a reference to the Wakeup source for this controller as defined in the ECU State Manager.
Class	Autosar Parameter
Range	NA
Default	/EcuM/EcuM/EcuMConfiguration_0/EcuMWakeupSource_0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_ControlerDescriptorType, CAN_CONST) ControlerDescriptors_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* WakeUp source ID (ECU State Manager) */
	0U,
	....
	}

Autosar 3.0 Requirement	CAN359

Note

This parameter contains a reference to the Wakeup Source for this controller as defined in the ECU State Manager. Type: reference to "EcuM_WakeupSourceType". EcuM plugin need to be added and then give the reference to it.

Note

This parameter has the label "CanWakeupSourceRef" in Tresos plugin interface.

3.9.2.37 CanBccSupport

Table 3-76. CanBccSupport

Description	This parameter defines if Backwards Compatibility Configuration (BCC) feature of CAN controller is used in the configuration.
Class	Implementation Specific Parameter
Range	True,False
Default	False
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_ControllerConfigType, CAN_CONST)ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = { .... /*---------Controller Options----------*/ #if (CAN_BCC_SUPPORT_ENABLE == STD_ON) CAN_CONTROLLERCONFIG_BCC_EN | #endif /* (CAN_BCC_SUPPORT_ENABLE == STD_ON) */ .... }
Autosar 3.0 Requirement	NA

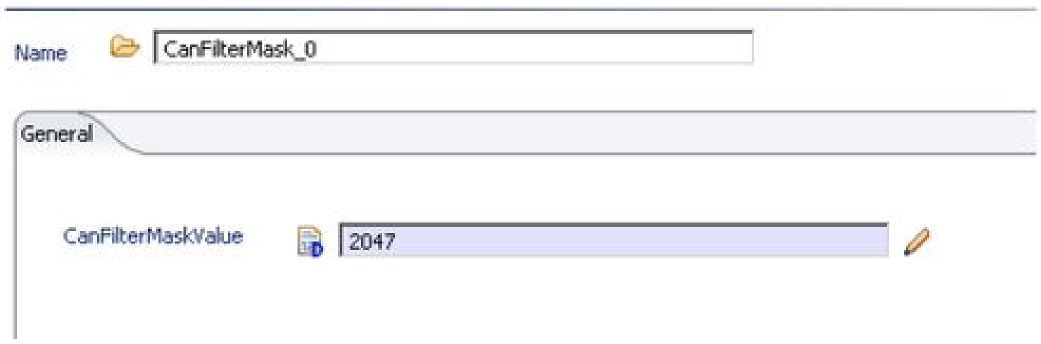
Note

If BCC feature of CAN controller is enabled, Individual Rx masking and queue feature are disabled.

Note

If BCC feature of CAN controller is disabled, Individual Rx masking and queue feature are enabled.

3.9.3 CanFilterMask



Name: CanFilterMask_0

CanFilterMaskValue: 2047

Figure 3-4. CanFilterMask

3.9.3.1 CanFilterMaskValue

Table 3-77. CanFilterMaskValue

Description	Reference to the filter mask that is used for hardware filtering together with the CAN_ID_VALUE
Class	Autosar Parameter
Range	0 - 4294967296
Default	2047
Source File	Can_Cfg.c, Can_PBCfg.c

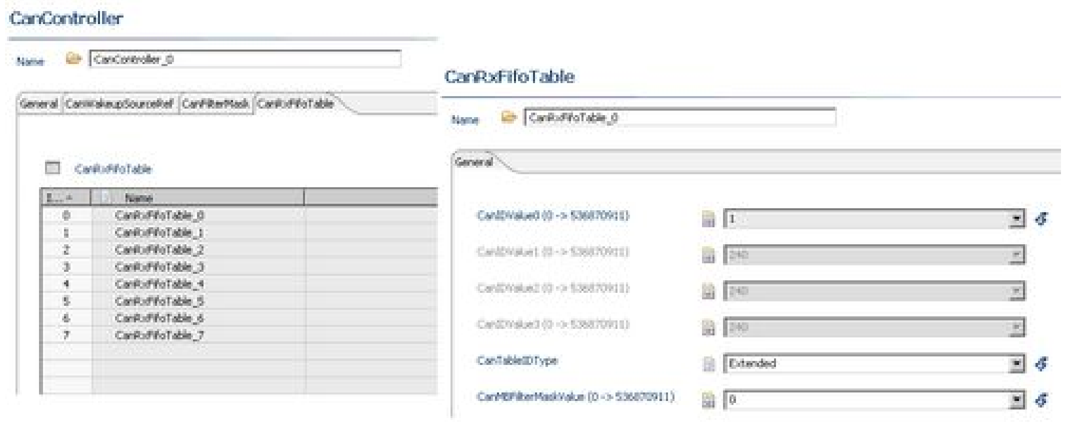
Source Representation	CONST(Can_IdType, CAN_CONST) Can_FilterMasks_PC[CAN_MAXFILTERCOUNT_0] = {
	....
	/* FilterMasks_PC[0], "CanFilterMask_0" */
	0x7ffU,
	....
	}

Autosar 3.0 Requirement	CAN066

Note

Describes a mask for hardware-based filtering of CAN identifiers.

3.9.4 Can RxFifo



CanController

Name: CanController_0

General | CanWakeupSourceRef | CanFilterMask | CanRxFifoTable

CanRxFifoTable

I...	Name
0	CanRxFifoTable_0
1	CanRxFifoTable_1
2	CanRxFifoTable_2
3	CanRxFifoTable_3
4	CanRxFifoTable_4
5	CanRxFifoTable_5
6	CanRxFifoTable_6
7	CanRxFifoTable_7

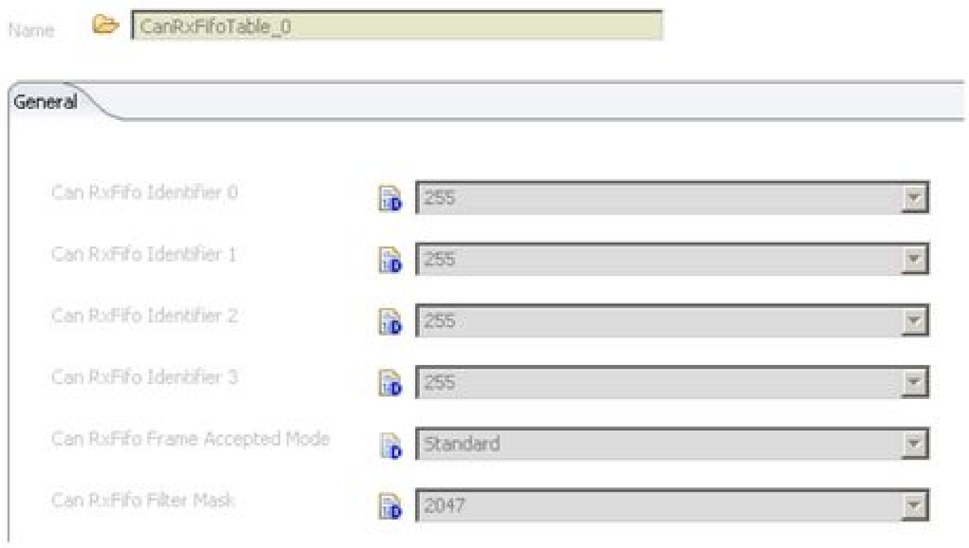
CanRxFifoTable

Name: CanRxFifoTable_0

General

CanIDValue0 (0 -> 536870911)	1
CanIDValue1 (0 -> 536870911)	240
CanIDValue2 (0 -> 536870911)	240
CanIDValue3 (0 -> 536870911)	240
CanTableIDType	Extended
CanMBFilterMaskValue (0 -> 536870911)	0

Figure 3-5. Can RxFifo



Name CanRxFifoTable_0

General

Can RxFifo Identifier 0: 255

Can RxFifo Identifier 1: 255

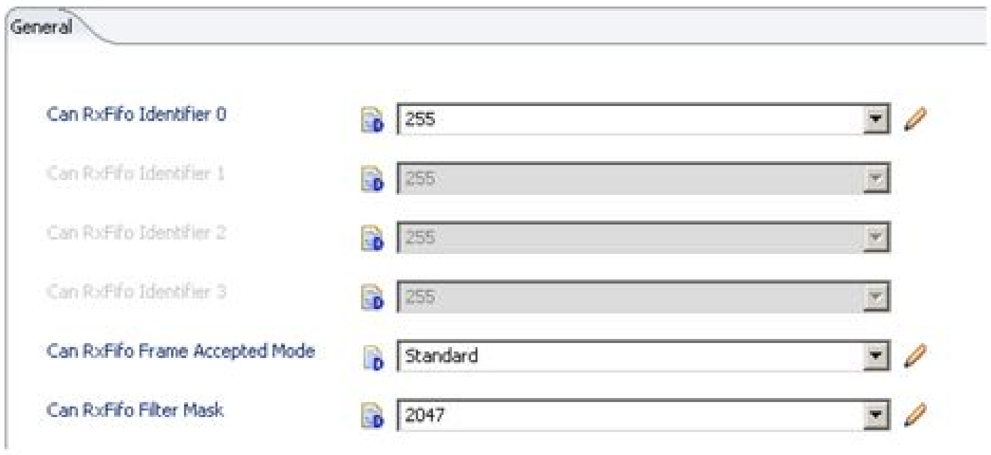
Can RxFifo Identifier 2: 255

Can RxFifo Identifier 3: 255

Can RxFifo Frame Accepted Mode: Standard

Can RxFifo Filter Mask: 2047

Figure 3-6. Can CanRxFifoTable - CanControllerRxFifoEnable = FALSE



General

Can RxFifo Identifier 0: 255

Can RxFifo Identifier 1: 255

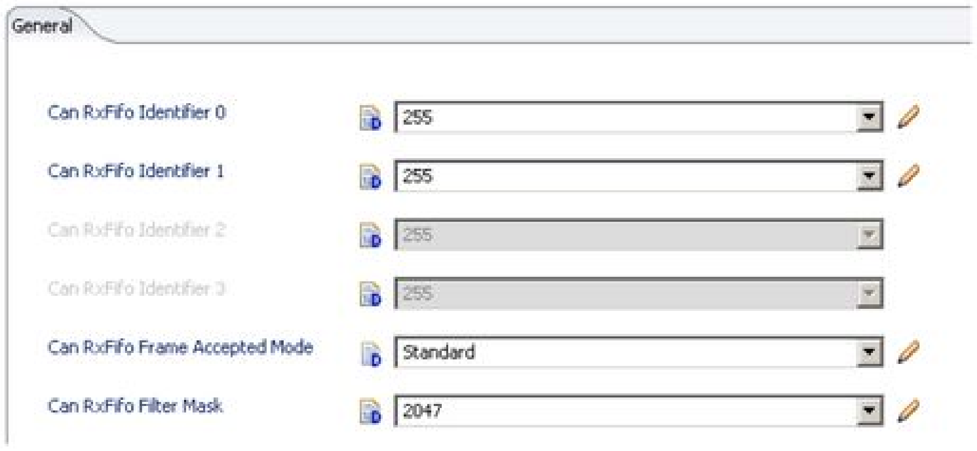
Can RxFifo Identifier 2: 255

Can RxFifo Identifier 3: 255

Can RxFifo Frame Accepted Mode: Standard

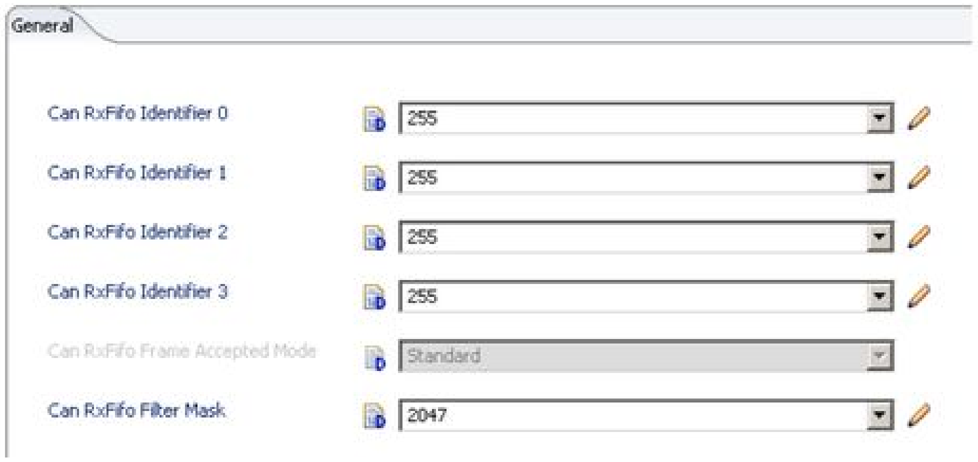
Can RxFifo Filter Mask: 2047

Figure 3-7. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = A


General

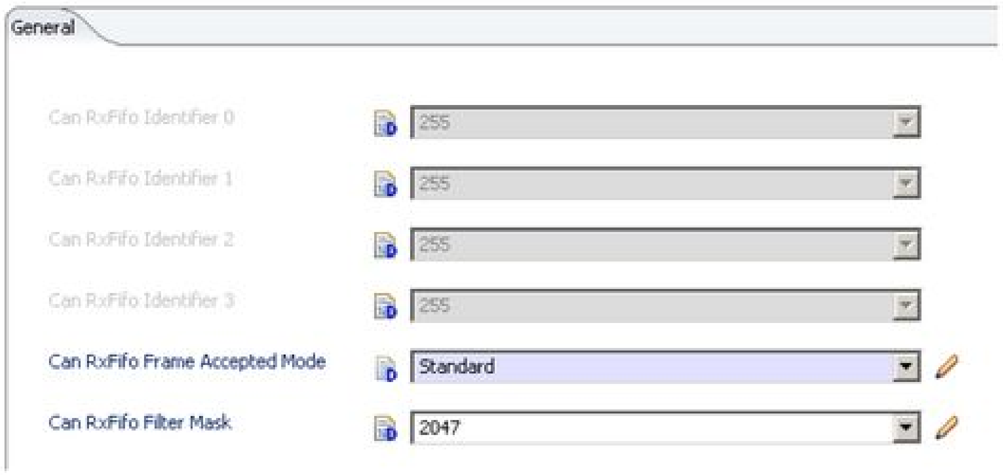
Can RxFifo Identifier 0	255
Can RxFifo Identifier 1	255
Can RxFifo Identifier 2	255
Can RxFifo Identifier 3	255
Can RxFifo Frame Accepted Mode	Standard
Can RxFifo Filter Mask	2047

Figure 3-8. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = B


General

Can RxFifo Identifier 0	255
Can RxFifo Identifier 1	255
Can RxFifo Identifier 2	255
Can RxFifo Identifier 3	255
Can RxFifo Frame Accepted Mode	Standard
Can RxFifo Filter Mask	2047

Figure 3-9. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = C


General

Can RxFifo Identifier 0	255
Can RxFifo Identifier 1	255
Can RxFifo Identifier 2	255
Can RxFifo Identifier 3	255
Can RxFifo Frame Accepted Mode	Standard
Can RxFifo Filter Mask	2047

Figure 3-10. Can CanRxFifoTable - CanControllerRxFifoEnable = TRUE and CanControllerIDAcceptanceMode = D

3.9.4.1 CanControllerIDAcceptanceMode

Table 3-78. CanControllerIDAcceptanceMode

Description	This 2-bit field identifies the format of the elements of the Rx FIFO filter table.
Class	Implementation Specific Parameter
Range	A - D
Default	A
Source File	Can_Cfg.c, Can_PBcfg.c

Source Representation	CONST(Can_ControllerConfigType, CAN_CONST) ControllerConfigs_PC[CAN_MAXCONTROLLERCOUNT] = {
	....
	/* ===== Controller Options ===== */
	/* ID Acceptance Mode A */
	CAN_CONTROLLERCONFIG_IDAM_A |
	}

Autosar 3.0 Requirement	NA

3.9.4.2 CanIDValue0

Table 3-79. CanIDValue0

Description	Specifies an ID to be used as acceptance criteria for the ID Table0. Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below: Format A - One full ID (standard or extended) per filter element Format B - One full standard ID if the CanTableIDType is Standard or one 14 most significant bit value of Extended ID if the CanTableIDType is Extended Format C - One 8 most significant bit value of Standard or Extended ID.
Class	Implementation Specific Parameter
Range	0 - 2047 for Format A and B– Standard CanTableIDType typ 0 – 536870911 for Format A – Extended CanTableIDType type 0 – 1683 for Format B – Extended CanTableIDType type 0 – 255 for Format C
Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-79. CanIDValue0 (continued)

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PCConfig[CAN_MAXTABLEID_0] = {
	{0x**11**121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff }}
	Format B
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{ {0x**1ff8**07f8, /* CanRxFifoTable_0 of type Standard and formatB for FlexCAN_A */
	0xfff8fff8 }}
	Format A
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{0x**4001fffe**, /* CanRxFifoTable_3 of type Extended and formatA for FlexCAN_A */
	0xfffffffe }}

Autosar 3.0 Requirement	NA

Note

Specifies an ID to be used as acceptance criteria for the ID Table 0.

3.9.4.3 CanIDValue1

Table 3-80. CanIDValue1

Description	Specifies an ID to be used as acceptance criteria for the ID Table1. Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below: Format B - One full standard ID if the CanTableIDType is Standard or one 14 most significant bit value of Extended ID if the CanTableIDType is Extended Format C - One 8 most significant bit value of Standard or Extended ID.
Class	Implementation Specific Parameter
Range	0 - 2047 for Format B– Standard CanTableIDType type 0 – 1683 for Format B – Extended CanTableIDType type 0 – 255 for Format C
Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-80. CanIDValue1 (continued)

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PCConfig[CAN_MAXTABLEID_0] = {
	{0x11**12**1314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff }}
	Format B
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{ {0x1ff8**07f8**, /* CanRxFifoTable_0 of type Standard and formatB for FlexCAN_A */
	0xfff8fff8 }}

Autosar 3.0 Requirement	NA

Note

Specifies an ID to be used as acceptance criteria for the ID Table 1.

3.9.4.4 CanIDValue2

Table 3-81. CanIDValue2

Description	Specifies an ID to be used as acceptance criteria for the ID Table2. Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below C- One 8 most significant bit value of Standard or Extended ID.
Class	Implementation Specific Parameter
Range	0 - 255
Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig [CAN_MAXTABLEID]= {
	{0x1112**13**14, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff }}

Autosar 3.0 Requirement	NA

Note

Specifies an ID to be used as acceptance criteria for the ID Table 2.

3.9.4.5 CanIDValue3

Table 3-82. CanIDValue3

Description	Specifies an ID to be used as acceptance criteria for the ID Table3. Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below C- One 8 most significant bit value of Standard or Extended ID.
Class	Implementation Specific Parameter
Range	0 - 255
Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	Format C
	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig[CAN_MAXTABLEID]= {
	{0x111213**14**, /* CanRxFifoTable_0 of formatC for FlexCAN_A */
	0xffffffff } }

Autosar 3.0 Requirement	NA

Note

Specifies an ID to be used as acceptance criteria for the ID Table 3.

3.9.4.6 CanMBFilterMaskValue

Table 3-83. CanMBFilterMaskValue

Description	Specifies filter mask value to be used as acceptance criteria for the ID Table. Note: Value for this parameter should be entered based on the CanControllerIDAcceptanceMode selected as explained below: A- Filtermask value for One full standard or Extended ID based on the CanTableIDType selected. B- Filtermask value for CanIDValue0(bit field : 0-11 for standard ID type and 0-14 for extended ID type) and CanIDValue1 (bit field : 12- 22 for standard ID type and 15-28 for extended ID type) C- Filtermask value for CanIDValue0(bit field : 0-8), CanIDValue1(bit field : 9-16), CanIDValue2(bit field : 17-24) and CanIDValue3(bit field : 25-32).
Class	Implementation Specific Parameter
Range	0 – 2047 for Format A Standard CanTableIDType 0 - 536870911 Format A Extended CanTableIDType 0 – 268435455 for Format B Extended CanTableIDType 0 - 4194303 for Format B Standard CanTableIDType 0 – 4294967295 for Format C

Table continues on the next page...

Table 3-83. CanMBFilterMaskValue (continued)

Default	0xF0
Source File	Can_Cfg.c, Can_PBCfg.c

Source Representation	CONST(Can_RxFiFoTableIdConfigType, CAN_CONST) RxFifoTableID_PBConfig[CAN_MAXTABLEID]= {
	 {0x11121314, /* CanRxFifoTable_0 of formatC for FlexCAN_A */,
	 0xFFFFFFFF } }

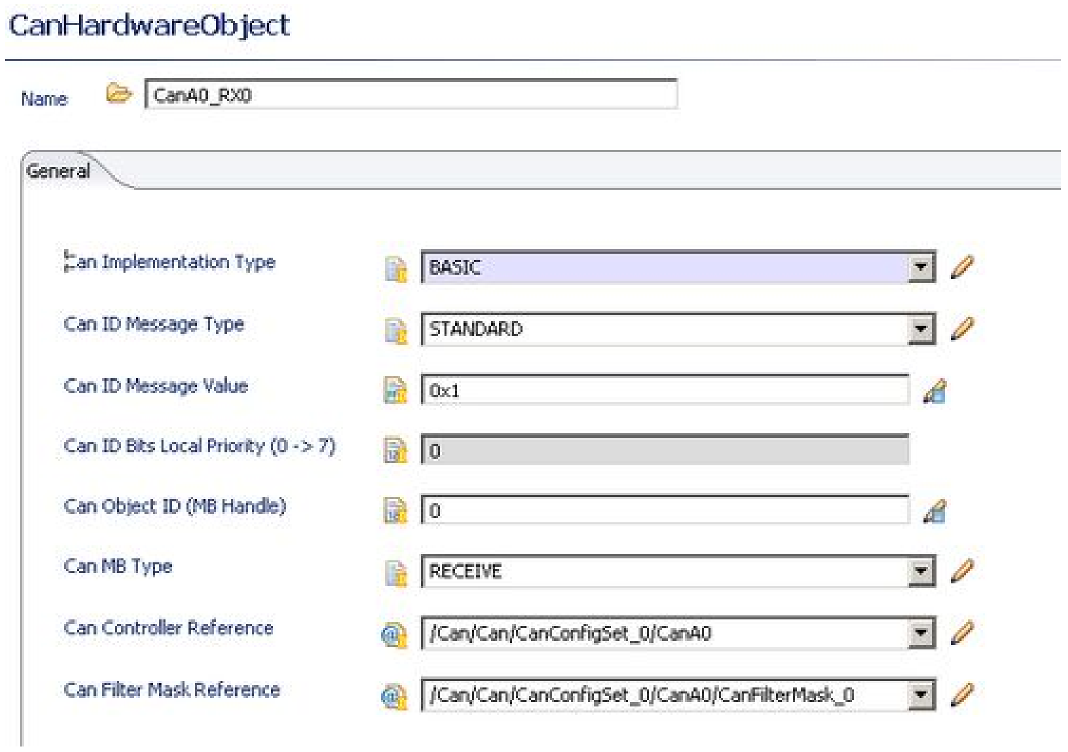
Autosar 3.0 Requirement	NA

3.9.4.7 CanTableIDType

Table 3-84. CanTableIDType

Description	Specifies whether extended or standard frames are accepted into the FIFO.
Class	Implementation Specific Parameter
Range	Standard, Extended
Default	Standard
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	NA
Autosar 3.0 Requirement	NA

3.9.5 Can HardwareObject


CanHardwareObject

Name: CanA0_RX0

Can Implementation Type	BASIC
Can ID Message Type	STANDARD
Can ID Message Value	0x1
Can ID Bits Local Priority (0 -> 7)	0
Can Object ID (MB Handle)	0
Can MB Type	RECEIVE
Can Controller Reference	/Can/Can/CanConfigSet_0/CanA0
Can Filter Mask Reference	/Can/Can/CanConfigSet_0/CanA0/CanFilterMask_0

Figure 3-11. Can HardwareObject

Note

When "CanControllerRxFifoEnable" parameter is true for a controller, then the maximum number of hardware objects to be configured for that controller is 56.

Note

When the FEN bit is set in the MCR register, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. For reading data received from Fifo it should be used as the reading from MB0.

3.9.5.1 CanHandleType

Table 3-85. CanHandleType

Description	Specifies the type (Full-Can or Basic-Can) of the hardware object.
Class	Autosar Parameter
Range	BASIC, FULL
Default	BASIC

Table continues on the next page...

Table 3-85. CanHandleType (continued)

Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_MessageBufferConfigObjectType, CAN_CONST) MessageBufferConfigs0[]= { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* IdMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
Autosar 3.0 Requirement	CAN324

3.9.5.2 CanIdType

Table 3-86. CanIdType

Description	Specifies whether the IdValue is of type: standard identifier, extended identifier, mixed mode.
Class	Autosar Parameter
Range	Standard, Mixed, Extended
Default	Standard
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-86. CanIdType (continued)

Source Representation	CONST(Can_MessageBufferConfigObjectType, CAN_CONST) MessageBufferConfigs0[]= { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* IdMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, **/* ID type: EXTENDED, STANDARD, MIXED */** **STANDARD,** /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
Autosar 3.0 Requirement	CAN065

Note

Specifies whether the IdValue is of type: - standard identifier (ID - 11 bits length), - extended identifier (ID - 29 bits length), - mixed mode (standard or extended).

3.9.5.3 CanIdValue

Table 3-87. CanIdValue

Description	Specifies (together with the filter mask) the identifiers that pass the hardware filter for of RX objects.
Class	Autosar Parameter
Range	Integer
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-87. CanIdValue (continued)

Source Representation	CONST(Can_MessageBufferConfigObjectType, CAN_CONST) MessageBufferConfigs0[]= { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* IdMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, **/* MessageId */** **0x1U,** /* Local priority bits used for arbitration */ 0U,
Autosar 3.0 Requirement	CAN325

Note

Specifies (together with the filter mask)- the identifiers range that passes the hardware filter for of RX objects. Parameter ranges from 0 to 0x7FF (11 bits) for Standard IDs and 0 to 0x1FFFFFFF (29 bits) for Extended IDs. User can assign any code to this parameter, but must to respect the above rule related to Standard/Extended IDs.

3.9.5.4 CanMBPrio

Table 3-88. CanMBPrio

Description	This field is used when MCR[LPRIO_EN] is set and makes sense only for Tx MBs.
Class	Implementation Specific Parameter
Range	0 - 7
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-88. CanMBPrio (continued)

Source Representation	CONST(Can_MessageBufferConfigObjectType, CAN_CONST) MessageBufferConfigs0[]= { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* IdMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, **/* Local priority bits used for arbitration */** **0U**,
Autosar 3.0 Requirement	NA

Note

MBCS[PRIO]: Local priority. This 3-bit field is used when MCR[LPRIO_EN] is set and makes sense only for TX buffers. If CTRL[LBUF] is set this field is not used. These bits are not transmitted. They are appended to the regular ID to define the transmission priority.

3.9.5.5 CanObjectId

Table 3-89. CanObjectId

Description	Holds the handle ID of Hrh or Hth. The value of this parameter is unique in a given CAN Driver and should start with 0 and continue without any gaps. The Hrh and Hth IDs are defined under two different name-spaces. Examples: Hrh0-0, Hrh1-1, Hth0-2, Hth1-3
Class	Autosar Parameter
Range	Integer
Default	0
Source File	Can_Cfg.h
Source Representation	#define CanA0_RX0 0U /* RECEIVE object */
Autosar 3.0 Requirement	CAN326

Note

Holds the handle ID of HRH or HTH.

3.9.5.6 CanObjectType

Table 3-90. CanObjectType

Description	Specifies if the HardwareObject is used as Transmit or as Receive object
Class	Autosar Parameter
Range	Transmit, Receive
Default	Receive
Source File	Can_Cfg.c, Can_PBCfg.c
Source Representation	CONST(Can_MessageBufferConfigObjectType, CAN_CONST) MessageBufferConfigs0[]= { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* IdMaskIndex */ 0U, /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, **/* Receive/Transmit MB configuration */** **RECEIVE,** /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
Autosar 3.0 Requirement	CAN327

Note

Specifies if the HardwareObject is used as Transmit or as Receive object.

3.9.5.7 CanControllerRef

Table 3-91. CanControllerRef

Description	This associates the hardware object to the CAN controller that uses this hardware object.
Class	Autosar Parameter
Range	Integer
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

Table 3-91. CanControllerRef (continued)

Source Representation	CONST(Can_MessageBufferConfigObjectType, CAN_CONST) MessageBufferConfigs0[]= { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * /* IdMaskIndex */ 0U, **/* ControllerId - based on the order from CanController list */ 0U,** /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
Autosar 3.0 Requirement	CAN322

Note

Reference to CAN Controller to which the HOH is associated to.

3.9.5.8 CanFilterMaskRef

Table 3-92. CanFilterMaskRef

Description	Describes a mask for hardware-based filtering of CAN identifiers. It shall be distinguished between: - Standard identifier mask, - Extended identifier mask
Class	Autosar Parameter
Range	Integer
Default	0
Source File	Can_Cfg.c, Can_PBCfg.c

Table continues on the next page...

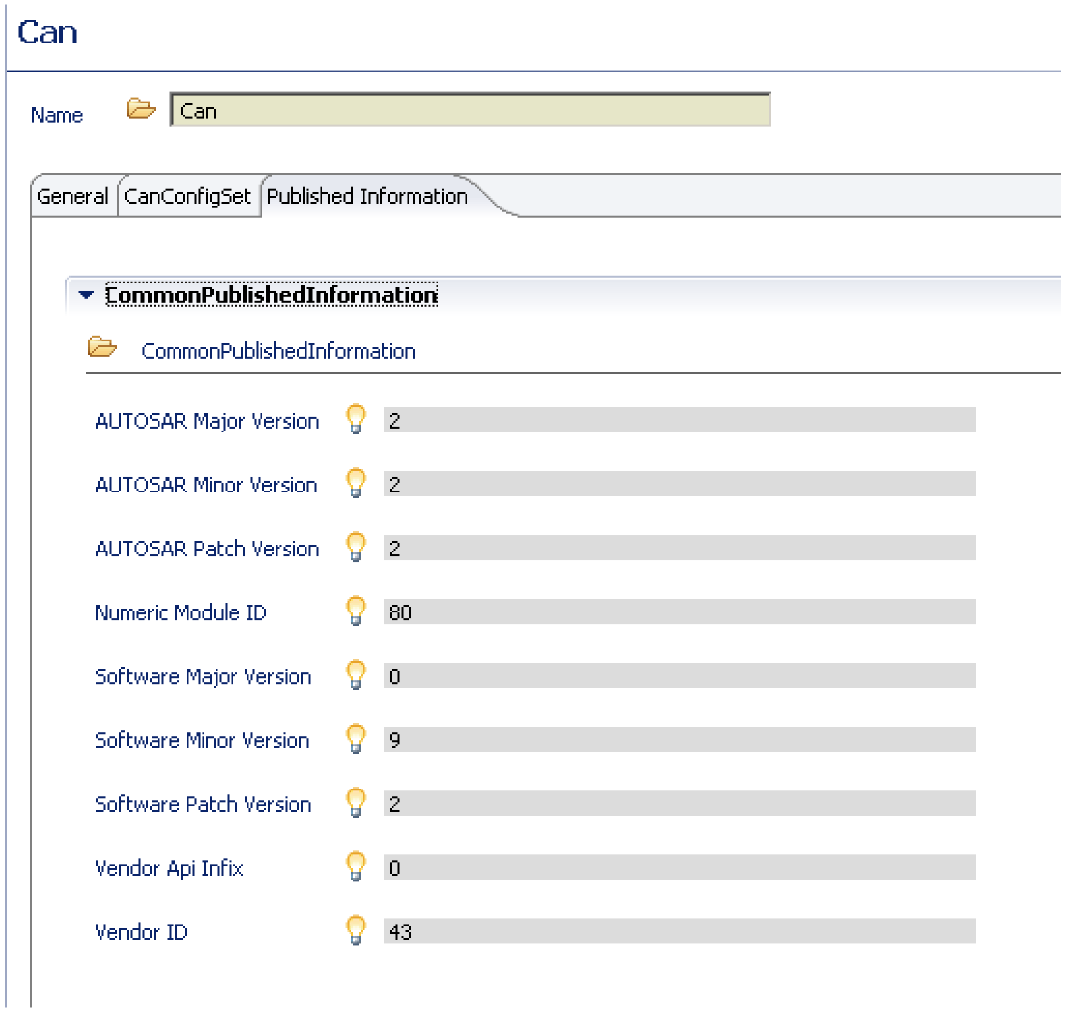
Table 3-92. CanFilterMaskRef (continued)

Source Representation	CONST(Can_MessageBufferConfigObjectType, CAN_CONST) MessageBufferConfigs0[]= { /* MessageBufferConfigs_PB[0], "CanA0_RX0" * **/* IdMaskIndex */** **0U,** /* ControllerId - based on the order from CanController list */ 0U, /* ID type: EXTENDED, STANDARD, MIXED */ STANDARD, /* Receive/Transmit MB configuration */ RECEIVE, /* MessageId */ 0x1U, /* Local priority bits used for arbitration */ 0U,
Autosar 3.0 Requirement	CAN321

Note

Reference to the filter mask that is used for hardware filtering togerther with the CAN_ID_VALUE. This value is used as acceptance masks for ID filtering in RX MBs and the FIFO.

3.9.6 Can Common Published Information


Can

Name: Can

AUTOSAR Major Version	2
AUTOSAR Minor Version	2
AUTOSAR Patch Version	2
Numeric Module ID	80
Software Major Version	0
Software Minor Version	9
Software Patch Version	2
Vendor Api Infix	0
Vendor ID	43

Figure 3-12. Can Common Published Information

3.9.6.1 ArMajorVersion

Table 3-93. ArMajorVersion

Description	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_SPEC_VERSION_MAJOR
Source File	All
Source Representation	ARVersion **M4_SRC_AR_SPEC_VERSION_MAJOR**.M4_SRC_AR_SPEC_VERSION_MINOR.M4_SRC_AR_SPEC_VERSION_PATCH
Autosar 3.0 Requirement	NA

3.9.6.2 ArMinorVersion

Table 3-94. ArMinorVersion

Description	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_SPEC_VERSION_MINOR
Source File	All
Source Representation	ARVersion M4_SRC_AR_SPEC_VERSION_MAJOR.**M4_SRC_AR_SPEC_VERSION_MINOR**.M4_SRC_AR_SPEC_VERSION_PATCH
Autosar 3.0 Requirement	NA

3.9.6.3 ArPatchVersion

Table 3-95. ArPatchVersion

Description	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_SPEC_VERSION_PATCH
Source File	All
Source Representation	ARVersion M4_SRC_AR_SPEC_VERSION_MAJOR.M4_SRC_AR_SPEC_VERSION_MINOR.**M4_SRC_AR_SPEC_VERSION_PATCH**
Autosar 3.0 Requirement	NA

3.9.6.4 ModuleId

Table 3-96. ModuleId

Description	Module ID of this module from Module List.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_XDM_AR_MODULE_ID
Source File	NA
Source Representation	NA
Autosar 3.0 Requirement	NA

3.9.6.5 SwMajorVersion

Table 3-97. SwMajorVersion

Description	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_SW_VERSION_MAJOR
Source File	All
Source Representation	@version **M4_SRC_SW_VERSION_MAJOR**.M4_SRC_SW_VERSION_MINOR.M4_SRC_SW_VERSION_PATCH
Autosar 3.0 Requirement	NA

3.9.6.6 SwMinorVersion

Table 3-98. SwMinorVersion

Description	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_SW_VERSION_MINOR
Source File	All
Source Representation	@version M4_SRC_SW_VERSION_MAJOR.**M4_SRC_SW_VERSION_MINOR**.M4_SRC_SW_VERSION_PATCH
Autosar 3.0 Requirement	NA

3.9.6.7 SwPatchVersion

Table 3-99. SwPatchVersion

Description	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_SW_VERSION_PATCH
Source File	All

Table continues on the next page...

Table 3-99. SwPatchVersion (continued)

Source Representation	@version M4_SRC_SW_VERSION_MAJOR.M4_SRC_SW_VERSION_MINOR.**M4_SRC_SW_VERSION_PATCH**
Autosar 3.0 Requirement	NA

3.9.6.8 VendorApiInfix

Table 3-100. VendorApiInfix

Description	Vendor API.
Class	Implementation Specific Parameter
Range	Integer
Default	NA
Source File	NA
Source Representation	NA
Autosar 3.0 Requirement	NA

3.9.6.9 VendorId

Table 3-101. VendorId

Description	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.
Class	Implementation Specific Parameter
Range	Integer
Default	M4_SRC_AR_MODULE_VENDOR_ID
Source File	All
Source Representation	#define CAN_VENDOR_ID_C **M4_SRC_AR_MODULE_VENDOR_ID**
Autosar 3.0 Requirement	NA

3.10 Wakeup Configuration

The current driver implementation is using two forms of wakeup events detection and handling.

a. Internal Wakeup

A macro is generated in Can_Cfg.h file that guard the code that handles internal wakeup functionality:

```
#define CAN_WKP_INT_SUPPORT (STD_ON)
```

Note

MPC5634M doesn't support internal wakeup functionality. INTC table has the CAN wakeup interrupt reserved.Also is not support available for specific wakeup bits in the MCR, ESR registers.

b. External Wakeup

A macro is generated in Can_Cfg.h file that guard the code that handles external wakeup functionality

```
#define CAN_WKP_EXT_SUPPORT (STD_ON)
```

Note

MPC5634M doesn't support external wakeup functionality. This platform doesn't have WKUP module for this.

Note

Values for internal and external wakeup support are set according to Reference Manual specifications.

Wakeup Implementation

General Wake-Up mechanism can be enabled or disabled by the “CanWakeupSupport” control. This is the implementation of the CAN330 requirement. See below the definition of the macro that enables/ disables this functionality (located in Can_Cfg.h file).

```
#define CAN_WAKEUP_SUPPORT (STD_ON)
```

The CAN Controller can handle wakeup events by interrupts or by polling mode. The selection is done by the precompile parameter “CanWakeupProcessing” that is specific to every controller.

a. Interrupt Mode

ISR(Can_IsrFCx_WKP) is the function that handles with the wakeup events. Global Wakeup parameter must be enabled and also the specific controller must have enabled the support of interrupts.

```
#if (CAN_WAKEUP_SUPPORT == STD_ON)
    #if (CAN_A_WAKEUPINT_SUPPORTED == STD_ON)
    #endif
#endif
```

b. Polling Mode

Can_MainFunction_Wakeup is the function that handles with the wakeup events in polling mode. Global Wakeup parameter must be enabled and also the specific controller must have enabled the support of wakeup polling mode.

```
#if (CAN_WAKEUP_SUPPORT == STD_ON)
    #if (CAN_WAKEUPPOLL_SUPPORTED == STD_ON)
    #endif
#endif
```

3.11 Driver Usage and Configuration Tips

1. A CAN Hardware Unit consists of one or multiple CAN controllers of the same type. Can_MainFunction_Write(), Can_MainFunction_Read(), Can_MainFunction_BusOff() and Can_MainFunction_Wakeup() APIs are defined if at least one Can controller from the Hardware Unit is configured to Polling mode for write, read, busoff and wakeup operation - else are empty functions. Refer to CAN178, CAN180, CAN183 and CAN185.

2. Can_Cbk_CheckWakeup() and Can_MainFunction_Wakeup() APIs are not defined if Wakeup support is disabled.

3. Can_AbortMb() API (Non Autosar) is defined if this feature is enabled from the Tresos plugin.

4. Can_SetClockMode() API (Non Autosar) is defined if this feature is enabled from the Tresos plugin.

5. The CAN Hardware Unit can be initialized using Can_Init() API.

6. A single CAN controller can be initialized using Can_InitController() API. The condition is that controller should be in STOP state (not participating to bus communication) while it is initialized.

7. Every CAN controller initialization is preceded by a software reset. See Can_LLD_ResetController() routine from low level driver.

8. Can_InitController() API should configure the MCR, CTRL, RXIMR registers, RxFifo block structure (if enabled), Message Buffers (for Rx MB is configured also every RXIMR register - used for message filtering).

9. Can_LLD_SetControllerMode() API can use transitions to CAN_T_SLEEP and CAN_T_WAKEUP only if Wakeup feature is enabled or supported by the platform.

10. Interrupts can be enabled calling Can_EnableControllerInterrupts() only if Can_DisableControllerInterrupts() was called prior. Refer to CAN208.

11. Multiplex transmission (supported by Can_Write() API) means to send a message from any Tx MB that is free to be used, not necessary from the one which is transmitted as "Hth" parameter. This feature can be used only if it's enabled from the Tresos Plugin. Refer to CAN277.

12. For Tx MBs the difference between Standard and Extended mode is done by the most significant bit of the Can ID. Refer to CANIF243 and CANIF188.

13. For Rx MBs the MIXED message buffer type is handled as EXTENDED type. Based on the MB type the RXIMR register is configured according: for STANDARD type the value is left shift with 18 bits.

Chapter 4
The Configuration of Can Bit Timing

4.1 Clock Source Description

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW.

The PRESDIV field (CTRL[PRESDIV]) controls a prescaler that generates the Serial Clock (Sclock), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

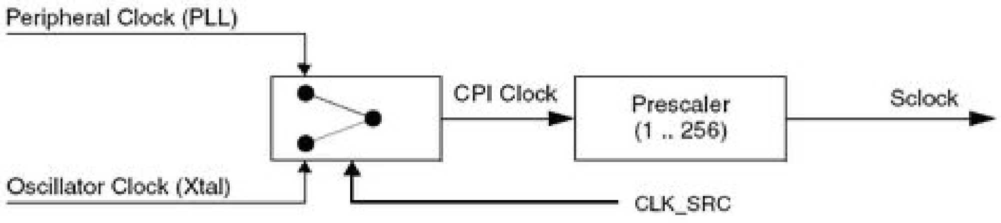


Figure 4-1. Can Engine Clock Scheme

A bit time is subdivided into three segments:

SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.

Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL Register so that their sum (plus 2) is in the range of 4 to 16 time quanta.

Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL Register (plus 1) to be 2 to 8 time quanta long.

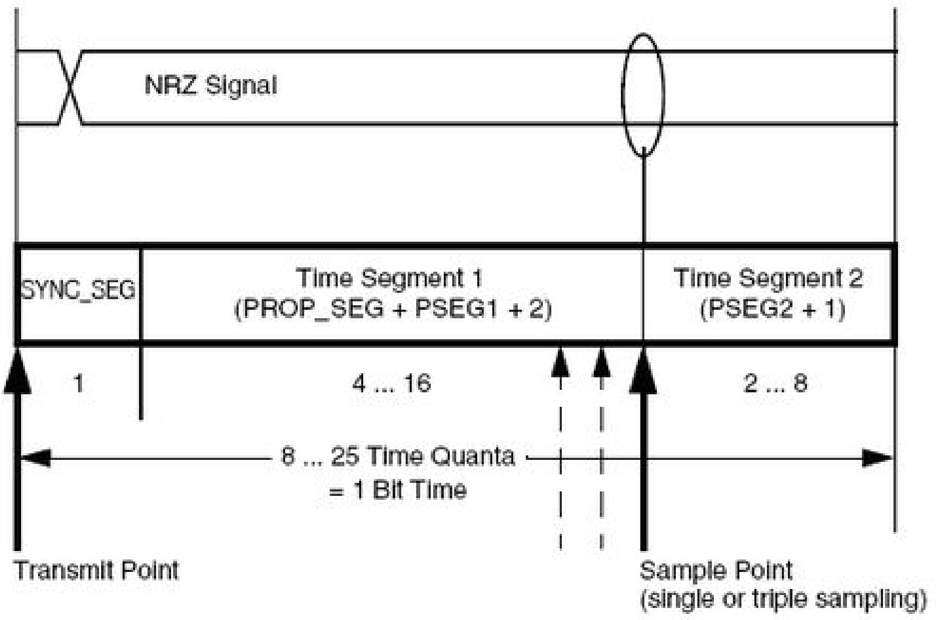
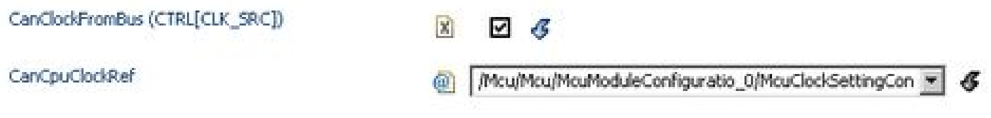


Figure 4-2. Can Time Segments

Understanding the Clock Selection for Can

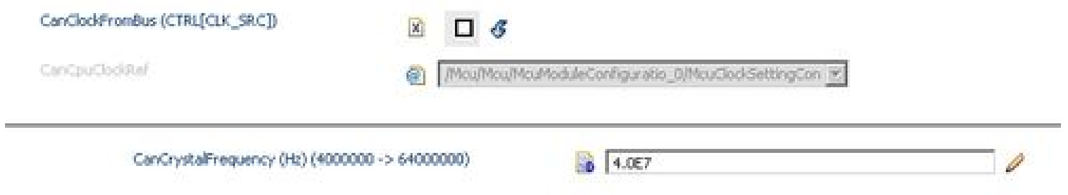
The clock source for CAN module can be obtained in 2 modes, depending by the setting of the "CanClockFromBus" control in Tressos.



CanClockFromBus (CTRL[CLK_SRC])	✓
CanCpuClockRef	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingCon

Figure 4-3. Can Clock Selection: bus

When “CanClockFromBus” is set to false the clock value is extracted from the “CanCrystalFrequencyHz” control of Can Tresos plugin.



CanClockFromBus (CTRL[CLK_SRC])	☐
CanCpuClockRef	/Mcu/Mcu/McuModuleConfiguratio_0/McuClockSettingCon
CanCrystalFrequency (Hz) (4000000 -> 64000000)	4.0E7

Figure 4-4. Can Clock Selection: osc_clock

4.2 Time Segments Calculation

Basic-Can vs. Full-Can

The terms **Basic CAN** and **Full CAN** must not be confused with the terms Standard CAN - also known as Base Frame Format (11 bit identifier, Version 2.0A data format) and Extended CAN - also known as Extended Frame Format (29 bit identifier, or Version 2.0B data format). Suitably configured, each implementation (Basic or Full CAN) can handle both Base and Extended data formats.

NOTE: Can Driver doesn’t make any difference between Full and Basic Hardware Object Type. This is implemented in the plugin configuration just because it’s a requirement of Autosar (CAN324)

Time Segments Calculation

Every CanController container of Tresos configuration tool has a parameter named “CanClockFromBus” which selects the source clock from MCU configuration (if “CanClockFromBus” is true) or from “CanCrystalFrequencyHz” parameter of “CanGeneral” container (if “CanClockFromBus” is false). In both cases the value is expressed in Hertz.

Step.1

First step is to calculate the Prescaler (CTRL[PRESDIV]). Autosar permit to configure the "CanControllerTimeQuanta" (it’s time unit expressed in nanoseconds). From this unit can calculate the frequency of the CanControllerQuanta (in hertz):

CanControllerQuanta = 1 / CanControllerTimeQuanta (hertz = 1 / seconds)

Prescaler value is calculated:

Prescaler = CanClockFreq / CanControllerQuanta (constant = hertz / hertz)

NOTE: Valid interval for Prescaler is between 1 and 256.

Step.2

Second step is to calculate the number of CanTimeQuantas per Can bit. Autosar permit to to configure the BaudRate (in Kbps) using parameter "CanControllerBaudRate" from "CanController" container of Tressos Can Plugin.

CanTimeQuanta = (1 / CancontrollerBaudRate) / CanControllerTimeQuanta (const = (1 / hertz) / seconds)

NOTE: Valid interval for No of CanTimeQuantas is between 8 and 25.

Step.3

Third step is to check the compatibility for parameters. The sum of Propagation Segment, Pseg1, Pseg2 and SyncSeg must be equal with no of CanTimeQuantas resulted as above.

No of Time Quantas per bit = 1 + PropSeg + 1 + Pseg1 + 1 + Pseg2 + 1

4.3 CAN Bit Timing

Selecting bit timing parameters that work well on the bench may not equate with the situation that the product moves into the real environment (maximum wiring length, worst case configuration, oscillator tolerance, etc) and we can find that bit timing parameters are inadequate.

For the CAN protocol (J1939, J2284) is is recommended to use a sample point in the range of 80% to 90%.

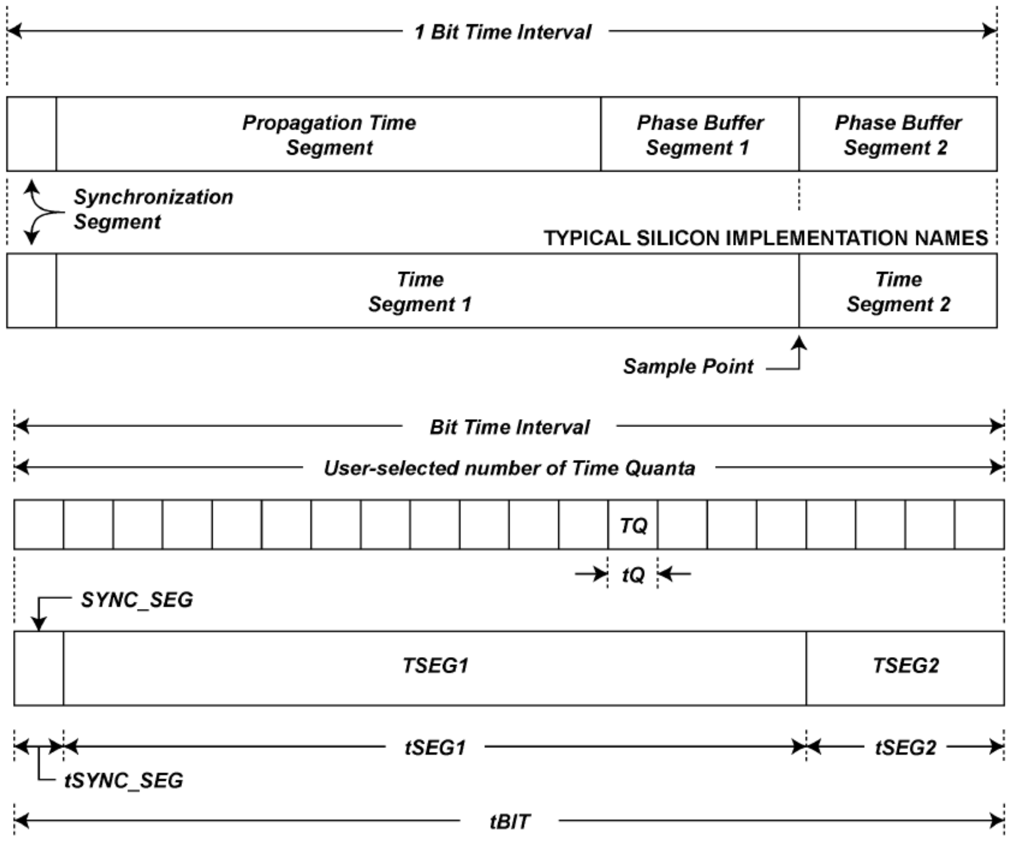


Figure 4-5. Can Bit Timing

The most important part of CAN bit timing is the Time Quantum (TQ). The time duration of the time quantum is derived from the CAN controller clock oscillator and the adjustable clock divider (prescaler). For Autosar CAN requirements we have to define Clock source and Time Quantum and calculate indirectly the prescaler.

The Synchronization Segment (SYNC_SEG) time interval is used to synchronize all the nodes across the network. SYNC_SEG time interval has a fixed period of one Time Quantum.

Time Segment 1 (TSEG1) is the time interval used to compensate for both positive phase errors in synchronization between nodes on the network and propagation delay between network nodes.

Time Segment 2 (TSEG2) is the time interval used to compensate for negative phase errors in synchronization between nodes.

Re-synchronization Jump Width (SJW) is not directly a segment of the bit time, but is used to dynamically adjust TSEG1 and TSEG2. SJW is the maximum amount of time by which TSEG1 may be lengthened or TSEG2 shortened to compensate for synchronization differences between nodes on the CAN network.

Bit Timing Example

```
Clock Source= 20MHz
Prescaler = 2
Bit Rate = 500K bits per second
From these input data will result:
    Time Bit = 2 microseconds ( 1/ 500Kbps)
    CAN clock source = 10 MHz (Clock Source / Prescaler)
    Selecting TQ = 100 nsec -> No. TQs per bit = 20 ( 2 usec / 100 nsec)
Bit Time Segments Calculation:

    SYNC_SEG = 1 (according to CAN ISO standard)

    TSEG1 = 15

    TSEG2 = 4

Sample Point Calculation:

    SP = (1 + TSEG1) / (1 + TSEG1 + TSEG2) = 0.8 -> 80%
```

Chapter 5
DEM events reported by CAN driver

5.1 DEM events

Production errors (DEM) reported by CAN driver are explained in below table.

Table 5-1. Production Errors (DEM) reported by CAN driver

Function in which DEM error is reported	DEM event ID	Conditions that generate this DEM event	fatal HW error, i.e. ECU is defective and needs to be replaced ?
Can_LLD_ResetController()	CAN_E_TIMEOUT	DEM error is reported when MCR[MCR_NOTRDY] flag is not set even after putting the controller in STOP/HALT mode.	YES
Can_LLD_ResetController()	CAN_E_TIMEOUT	DEM error is reported when MCR[SOFT_RST] flag is not set even after performing the controller reset.	YES
Can_LLD_SetControllerMode()	CAN_E_TIMEOUT	DEM error is reported when MCR[SLF_WAK] flag is not set even after setting it has been set to enable the self wakeup feature.	YES
Can_LLD_DisableControllerInterrupts()	CAN_E_TIMEOUT	DEM error is reported when IMASK1 and IMASK2 register bits are not reset even after the register bits are cleared to disable the interrupts.	YES
Can_LLD_EnableControllerInterrupts()	CAN_E_TIMEOUT	DEM error is reported when IMASK1 and IMASK2 register bits are not set even after the register bits are set to enable the interrupts.	YES

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: UM14CANASR3.0R2.0.0
Rev. 1.1

freescale™