Integration Manual

for MPC574XG CAN Driver

Document Number: IM35CANASR4.2 Rev0002 R1.0.0
Rev. 1.0

Contents

Section number **Title** **Page**

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	17/02/2017	Hieu Tran	First version for MPC574XG RTM 1.0.0 release

Chapter 2
Introduction

This integration manual describes the integration requirements for CAN Driver for MPC574XG microcontrollers.

2.1 Supported Derivatives

The software described in this document is intented to be used with the following microcontroller devices of NXP Semiconductor .

Table 2-1. MPC574XG Derivatives

NXP Semiconductor	MPC5748G_LQFP176, MPC5748G_MAPBGA256, MPC5748G_MAPBGA324, MPC5747G_LQFP176, MPC5747G_MAPBGA256, MPC5747G_MAPBGA324, MPC5746G_LQFP176, MPC5746G_MAPBGA256, MPC5746G_MAPBGA324, MPC5748C_LQFP176, MPC5748C_MAPBGA256, MPC5748C_MAPBGA324, MPC5747C_LQFP176, MPC5747C_MAPBGA256, MPC5747C_MAPBGA324, MPC5746C_LQFP176, MPC5746C_MAPBGA256, MPC5746C_MAPBGA324, MPC5746C_MAPBGA100, MPC5745C_LQFP176, MPC5745C_MAPBGA256, MPC5745C_MAPBGA100, MPC5744C_LQFP176, MPC5744C_MAPBGA256, MPC5744C_MAPBGA100, MPC5746B_LQFP176, MPC5746B_MAPBGA256, MPC5746B_MAPBGA100, MPC5744B_LQFP176, MPC5744B_MAPBGA256,

Table 2-1. MPC574XG Derivatives

	MPC5744B_MAPBGA100, MPC5745B_LQFP176, MPC5745B_MAPBGA256, MPC5745B_MAPBGA100

All of the above microcontroller devices are collectively named as MPC574XG .

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 4.2 Rev0002CAN Driver Software Specification Document.	V4.0.0
2	MPC5748G Reference Manual	Rev. 5, 12/2016
3	MPC5746C Reference Manual	Rev. 4, 12/2016
4	MPC5748G_1N81M_Rev.2 (official document) (1N81M)	Jun-16
5	MPC5748G_1N81M_0N78S_Comparison_Summary_v2_0 (internal document) (1N81M, 0N78S)	31.10.2016
6	MPC5746C_1N06M_Rev.4 (official document) (1N06M)	Jul-16
7	MPC5746C_cut1.1_cut2.0_cut2.1_comparison_v0 (internal document) (1N06M, 0N84S, 1N84S)	14-Sep-16
8	C3M_cut2.1_new_errata_20170113 (internal document) (1N84S)	13-Jan-17

Chapter 3
Building the Driver

This section describes the source files and various compilers, linker options used for building the Autosar CAN driver for NXP SemiconductorMPC574XG . It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

The CAN driver files are compiled using

- Windriver DIAB DIAB_5_9_6_2
- Green Hills Multi 7.1.4 / Compiler 2015.1.6

The compiler, linker flags used for building the driver are explained below:

Note

The TS_T2D35M10I0R0 plugin name is composed as follow:

TS_T = Target_Id

D = Derivative_Id

M = SW_Version_Major

I = SW_Version_Minor

R = Revision

(i.e. Target_Id = 2 identifies PA architecture and Derivative_Id = 35 identifies the MPC574XG)

3.1.1 DIAB Compiler/Linker/Assembler Options

Table 3-1. Compiler Options

Option	Description
-tPPCE200Z4204N3VEN:simple	Sets target processor to PPCE200Z4204N3VEN, generates ELF using EABI conventions, No floating point support (minimizes the required runtime), selects simple environment settings for Startup Module and Libraries
-tPPCE200Z210N3VEN:simple	Sets target processor to PPCE200Z210N3VEN, generates ELF using EABI conventions, No floating point support (minimizes the required runtime), selects simple environment settings for Startup Module and Libraries
-Xdialect-ansi	Follow the ANSI C standard with some additions
-XO	Enables extra optimizations to produce highly optimized code
-g3	Generate symbolic debugger information and do all optimizations.
-Xsize-opt	Optimize for size rather than speed when there is a choice
-Xsmall-data=0	Set Size Limit for 'small data' Variables to zero.
-Xsmall-const=0	Set Size Limit for “small const” Variables to zero.
-Xaddr-sconst=0x11	Specify addressing for constant static and global variables with size less than or equal to -Xsmall-const to far-absolute.
-Xaddr-sdata=0x11	Specify addressing for non-constant static and global variables with size less than or equal to -Xsmall-data in size to far-absolute.
-Xno-common	Disable use of the 'COMMON' feature so that the compiler or assembler will allocate each uninitialized public variable in the .bss section for the module defining it, and the linker will require exactly one definition of each public variable
-Xnested-interrupts	Allow nested interrupts
-Xdebug-dwarf2	Generate symbolic debug information in dwarf2 format
-Xdebug-local-all	Force generation of type information for all local variables
-Xdebug-local-cie	Create common information entry per module
-Xdebug-struct-all	Force generation of type information for all typedefs, struct, union and class types
-Xforce-declarations	Generates warnings if a function is used without a previous declaration
-ee1481	Generate an error when the function was used before it has been declared
-Xmacro-undefined-warn	Generates a warning when an undefined macro name occurs in a #if preprocessor directive
-Xlink-time-lint	Enable the checking of object and function declarations across compilation units, as well as the consistency of compiler options used to compile source files
-W:as:,-l	Pass the option '-l' (lower case letter L) to the assembler to get an assembler listing file
-Wa,-Xisa-vle	Instruct the assembler to expect and assemble VLE (Variable Length Encoding) instructions rather than BookE instructions.
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DUSE_SW_VECTOR_MODE	-D defines a preprocessor symbol and optionally can set it to a value. USE_SW_VECTOR_MODE: By default in the package, drivers are compiled to be used with interrupt controller configured to be in hardware vector mode. In case of AUTOSAR_OS_NOT_USED, the compiler option "-DUSE_SW_VECTOR_MODE" must be added to the list of compiler options to be used with interrupt controller configured to be in software vector mode.

Table continues on the next page...

Table 3-1. Compiler Options (continued)

Option	Description
-DDIAB	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the DIAB preprocessor symbol.
-DDISABLE_MCAL_INTERMODULE_ASR_CHECK	-D defines a preprocessor symbol to disable the inter-module version check for AR_RELEASE versions. DISABLE_MCAL_INTERMODULE_ASR_CHECK: By default in the package, drivers are compiled to perform the inter-module version check as per Autosar BSW004. When the inter-module version check needs to be disabled then the DISABLE_MCAL_INTERMODULE_ASR_CHECK global define must be added to the list of compiler options.
-c	Stop after assembly, produce object file.

Table 3-2. Assembler Options

Option	Description
-tPPCE200Z4204N3VEN:simple	Sets target processor to PPCE200Z4204N3VEN, generates ELF using EABI conventions, No floating point support (minimizes the required runtime), selects simple environment settings for Startup Module and Libraries
-tPPCE200Z210N3VEN:simple	Sets target processor to PPCE200Z210N3VEN, generates ELF using EABI conventions, No floating point support (minimizes the required runtime), selects simple environment settings for Startup Module and Libraries
-g	Dump the symbols in the global symbol table in each archive file.
-Xisa-vle	Expect and assemble VLE (Variable Length Encoding) instructions rather than Book E instructions. The default code section is named .text_vle instead of .text, and the default code section fill "character" is set to 0x44444444 instead of 0. The .text_vle code section will have ELF section header flags marking it as VLE code, not Book E code.
-Xasm-debug-on	Generate debug line and file information
-Xdebug-dwarf2	Generate symbolic debug information in dwarf2 format
-Xsemi-is-newline	Treat the semicolon (;) as a statement separator instead of a comment character.

Table 3-3. Linker Options

Option	Description
-tPPCE200Z4204N3VEN:simple	Sets target processor to tPPCE200Z4204N3VEN, generates ELF using EABI conventions, No floating point support (minimizes the required runtime), selects simple environment settings for Startup Module and Libraries
-tPPCE200Z210N3VEN:simple	Sets target processor to tPPCE200Z210N3VEN, generates ELF using EABI conventions, No floating point support (minimizes the required runtime), selects simple environment settings for Startup Module and Libraries
-Xelf	Generates ELF object format for output file
-m6	Generates a detailed link map and cross reference table
-Xlink-time-lint	Enable the checking of object and function declarations across compilation units, as well as the consistency of compiler options used to compile source files

3.1.2 GHS Compiler/Linker/Assembler Options

Table 3-4. Compiler Options

Option	Description
-cpu=ppc5748gz4204	Selects target processor: ppc5748gz4204
-cpu=ppc5748gz210	Selects target processor: ppc5748gz210
-ansi	Specifies ANSI C with extensions. This mode extends the ANSI X3.159-1989 standard with certain useful and compatible constructs.
-noSPE	Disables the use of SPE and vector floating point instructions by the compiler.
-Ospace	Optimize for size.
-sda=0	Enables the Small Data Area optimization with a threshold of 0.
-vle	Enables VLE code generation
-dual_debug	Enables the generation of DWARF, COFF, or BSD debugging information in the object file
-G	Generates source level debugging information and allows procedure call from debugger's command line.
--no_exceptions	Disables support for exception handling
-Wundef	Generates warnings for undefined symbols in preprocessor expressions
-Wimplicit-int	Issues a warning if the return type of a function is not declared before it is called
-Wshadow	Issues a warning if the declaration of a local variable shadows the declaration of a variable of the same name declared at the global scope, or at an outer scope
-Wtrigraphs	Issues a warning for any use of trigraphs
--prototype_errors	Generates errors when functions referenced or called have no prototype
--incorrect_pragma_warnings	Valid #pragma directives with wrong syntax are treated as warnings
-noslashcomment	C++ like comments will generate a compilation error
-preprocess_assembly_files	Preprocesses assembly files
-nostartfile	Do not use Start files
--short_enum	Store enumerations in the smallest possible type
--diag_error 223	Sets the specified compiler diagnostic messages to the level of error
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DUSE_SW_VECTOR_MODE	-D defines a preprocessor symbol and optionally can set it to a value. USE_SW_VECTOR_MODE: By default in the package, drivers are compiled to be used with interrupt controller configured to be in hardware vector mode. In case of AUTOSAR_OS_NOT_USED, the compiler option "-DUSE_SW_VECTOR_MODE" must be added to the list of compiler options to be used with interrupt controller configured to be in software vector mode.
-DDISABLE_MCAL_INTERMODULE_ASR_CHECK	-D defines a preprocessor symbol to disable the inter-module version check for AR_RELEASE versions. DISABLE_MCAL_INTERMODULE_ASR_CHECK: By default in the package, drivers are compiled to perform the inter-module version check as per Autosar BSW004. When the inter-module version check needs to be disabled then the DISABLE_MCAL_INTERMODULE_ASR_CHECK global define must be added to the list of compiler options.
-DGHS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GHS preprocessor symbol.
-c	Produces an object file (called input-file.o) for each source file.

Table 3-5. Assembler Options

Option	Description
-cpu=ppc5748gz4204	Selects target processor: ppc5748gz4204
-cpu=ppc5748gz210	Selects target processor: ppc5748gz210
-G	Generates source level debugging information and allows procedure call from debugger's command line.
-list	Creates a listing by using the name of the object file with the .lst extension

Table 3-6. Linker Options

Option	Description
-cpu=ppc5748gz4204	Selects target processor: ppc5748gz4204
-cpu=ppc5748gz210	Selects target processor: ppc5748gz210
-nostartfiles	Do not use Start files.
-vle	Enables VLE code generation
--nocpp	Do not Generate Constructors/Destructors
-Mn	sort numerically the MAP file
-delete	The -delete option instructs the linker to remove functions that are not referenced in the final executable.
-ignore_debug_references	Ignores relocations from DWARF debug sections when using -delete. DWARF debug information will contain references to deleted functions that may break some third-party debuggers.
-keepmap	keeps the MAP file in case of link error

3.2 Files Required for the Compilation

This section describes the include files required to compile, assemble (if assembler code) and link the CAN driver for MPC574XG microcontrollers.

To avoid integration of incompatible files, all the include files from other modules shall have the same AR_RELEASE_MAJOR_VERSION and AR_RELEASE_MINOR_VERSION, i.e. only files with the same AUTOSAR major and minor versions can be compiled.

CAN Files

- ..\Can_TS_T2D35M10I0R0\include\Can.h
- ..\Can_TS_T2D35M10I0R0\include\Can_IPW.h
- ..\Can_TS_T2D35M10I0R0\include\Can_FlexCan.h
- ..\Can_TS_T2D35M10I0R0\include\Reg_eSys_FlexCan.h
- ..\Can_TS_T2D35M10I0R0\src\Can.c

- ..\Can_TS_T2D35M10I0R0\src\Can_Irq.c
- ..\Can_TS_T2D35M10I0R0\src\Can_FlexCan.c

CAN Generated Files

- Can_PBcfg.c - This file should be generated by the user using a configuration tool for compilation.
- Can_Cfg.c - This file should be generated by the user using a configuration tool for compilation.
- Can_Cfg.h - This file should be generated by the user using a configuration tool for compilation.

Files from Base common folder

- ..\Base_TS_T2D35M10I0R0\include\Compiler.h
- ..\Base_TS_T2D35M10I0R0\include\Compiler_Cfg.h
- ..\Base_TS_T2D35M10I0R0\include\ComStack_Types.h
- ..\Base_TS_T2D35M10I0R0\include\MemMap.h
- ..\Base_TS_T2D35M10I0R0\include\Mcal.h
- ..\Base_TS_T2D35M10I0R0\include\Platform_Types.h
- ..\Base_TS_T2D35M10I0R0\include\Std_Types.h
- ..\Base_TS_T2D35M10I0R0\include\Reg_eSys.h
- ..\Base_TS_T2D35M10I0R0\include\Soc_Ips.h
- ..\Base_TS_T2D35M10I0R0\include\Reg_Macros.h
- ..\Base_TS_T2D35M10I0R0\include\Can_GeneralTypes.h

Files from CanIf folder:

- ..\CanIf_TS_T2D35M10I0R0\include\CanIf_Cbk.h
- ..\CanIf_TS_T2D35M10I0R0\include\CanIf_Types.h.h
- ..\CanIf_TS_T2D35M10I0R0\include\CanIf.h

Files from Det folder:

- ..\Det_TS_T2D35M10I0R0\src\Det.c
- ..\Det_TS_T2D35M10I0R0\include\Det.h

Files from Rte folder:

- ..\Rte_TS_T2D35M10I0R0\src\Schm_Can.c
- ..\Rte_TS_T2D35M10I0R0\include\Schm_Can.h

Files from EcuM folder:

- ..\EcuM_TS_T2D35M10I0R0\src\EcuM.c
- ..\EcuM_TS_T2D35M10I0R0\include\EcuM.h
- ..\EcuM_TS_T2D35M10I0R0\include\EcuM_Cbk.h

3.3 Setting up the Plugins

The CAN driver was designed to be configured by using the EB Tresos Studio (version EB tresos Studio 21.0.0 b160607-0933 or later.)

Location of various files inside the CAN module folder:

- VSMD (Vendor Specific Module Definition) file in EB tresos Studio XDM format:
  - ..\Can_TS_T2D35M10I0R0\config\Can.xdm
  - ..\CanIf_TS_T2D35M10I0R0\config\CanIf.xdm
  - ..\EcuM_TS_T2D35M10I0R0\config\EcuM.xdm
  - ..\Base_TS_T2D35M10I0R0\config\Base.xdm
  - ..\Resource_TS_T2D35M10I0R0\config\Resource.xdm
  - ..\Mcu_TS_T2D35M10I0R0\config\Mcu.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - ..\Can_TS_T2D35M10I0R0\autosar\Can.epd
  - ..\CanIf_TS_T2D35M10I0R0\autosar\CanIf.epd
  - ..\EcuM_TS_T2D35M10I0R0\autosar\EcuM.epd
  - ..\Mcu_TS_T2D35M10I0R0\autosar\Mcu.epd
- Code Generation Templates for Pre-Compile time configuration parameters:
  - ..\Can_TS_T2D35M10I0R0\generate_PC\include\Can_Cfg.h
  - ..\Can_TS_T2D35M10I0R0\generate_PC\include\Can_Cfg.c
  - ..\Can_TS_T2D35M10I0R0\generate_PC\Can_SourceClock.m
  - ..\Can_TS_T2D35M10I0R0\generate_PC\Can_VersionCheck_Inc.m
  - ..\Can_TS_T2D35M10I0R0\generate_PC\Can_VersionCheck_Src.m
  - ..\Can_TS_T2D35M10I0R0\generate_PC\Can_NotifyCheck_Src.m
- Code Generation Templates for Post-Build time configuration parameters:
  - ..\Can_TS_T2D35M10I0R0\generate_PB\include\Can_Cfg.h
  - ..\Can_TS_T2D35M10I0R0\generate_PB\src\ Can_PBcfg.c
  - ..\Can_TS_T2D35M10I0R0\generate_PB\Can_SourceClock.m
  - ..\Can_TS_T2D35M10I0R0\generate_PB\Can_VersionCheck_Src_PB.m
  - ..\Can_TS_T2D35M10I0R0\generate_PB\Can_NotifyCheck_Src_PB.m

Steps to generate the configuration:

1. Copy the module folders Can_TS_T2D35M10I0R0, CanIf_TS_T2D35M10I0R0, Base_TS_T2D35M10I0R0, Resource_TS_T2D35M10I0R0, EcuM_TS_T2D35M10I0R0, Mcu_TS_T2D35M10I0R0 into the Tresos plugins folder.
2. Set the desired Tresos Output location folder for the generated sources and header files.
3. Use the EB tresos Studio GUI to modify ECU configuration parameters values.

4. Generate the configuration files.

Dependencies

- **MCU** is required to use System Clock when clock source is used as Peripheral clock source to generate CAN Segment values.
- **RESOURCE** is required to select processor derivative. Current Can driver has support for the following derivatives, everyone having attached a Resource file: MPC5748G_LQFP176, MPC5748G_MAPBGA256, MPC5748G_MAPBGA324, MPC5747G_LQFP176, MPC5747G_MAPBGA256, MPC5747G_MAPBGA324, MPC5746G_LQFP176, MPC5746G_MAPBGA256, MPC5746G_MAPBGA324, MPC5748C_LQFP176, MPC5748C_MAPBGA256, MPC5748C_MAPBGA324, MPC5747C_LQFP176, MPC5747C_MAPBGA256, MPC5747C_MAPBGA324, MPC5746C_LQFP176, MPC5746C_MAPBGA256, MPC5746C_MAPBGA324, MPC5746C_MAPBGA100, MPC5745C_LQFP176, MPC5745C_MAPBGA256, MPC5745C_MAPBGA100, MPC5744C_LQFP176, MPC5744C_MAPBGA256, MPC5744C_MAPBGA100, MPC5746B_LQFP176, MPC5746B_MAPBGA256, MPC5746B_MAPBGA100, MPC5744B_LQFP176, MPC5744B_MAPBGA256, MPC5744B_MAPBGA100, MPC5745B_LQFP176, MPC5745B_MAPBGA256, MPC5745B_MAPBGA100.
- **CANIF** is required for reporting status of some events.
- **ECUM** is required for selecting the reference to the wakeup source for every Can controller.
- **DET** is required for signaling the development error detection (parameters out of range, null pointers, etc).

Chapter 4
Function calls to module

4.1 Function Calls during Start-up

The CAN module shall be initialized by Can_Init() service call during the start-up. API service Can_SetController_Mode(Can_Controller, CAN_T_START) shall be used for setting the CAN controller to running mode.

Note

Pin settings are not related to Can driver or plugin configuration. GPIO pins used for connection of CAN physical layer have to be properly assigned to the IPV_FlexCAN module prior the CAN initialization.

4.2 Function Calls during Shutdown

The IPV_FlexCAN IP has many Low Power Modes, with programmable wake up on bus activity.

- **Freeze Mode**

  This low power mode is entered when the HALT and FRZ bits in the MCR Register are asserted.

  Module ignores the Rx input pin and drives the Tx pin as recessive, stops the prescaler, thus halting all CAN protocol activities and grants write access to the Error Counters Register (ECR), which is read-only in other modes.

  Exit from this mode is done by negating the FRZ and HALT bits in the MCR Register or when the MCU is removed from Debug Mode

Note

It is not possible to exit from this mode by receiving a message on the Can bus.

- **Module Disable Mode**

  This low power mode is entered when the MDIS bit in the MCR Register is asserted.

  Module shuts down the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules.

  Exit from this mode is done by negating the MDIS bit in the MCR Register.

Note

It is not possible to exit from this mode by receiving a message on the Can bus.

- **Stop Mode**

  This low power mode is entered when Stop Mode is requested at MCU level.

  When in Stop Mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally.

  Exit from this mode happens when the Stop Mode request is removed or when activity is detected on the CAN bus and the Self Wake Up mechanism is enabled.

Note

Note that wake-up from Stop Mode only works when both bits, SLF_WAK and WAK_MSK, are asserted. If interrupt for Wakeup is implemented in INTC, the interrupt handler can change the state of the controller to RUN mode if it is needed. When exit from this mode controller is usually in Freeze mode.

Note

Refer the Reference Manual if wakuep is supported by hardware (bits register are implemented and INTC has connected the CAN wakeup interrupt signal).

4.3 Function Calls during Wake-up

The controller can be wakeup by a message when it is in Stop mode only if interrupt for wakeup is enabled or self wakeup mechanism is enabled.

SLF_WAK	WAK_MSK	MCU Clocks Enabled	Wake-up Interrupt Generated
0	0	No	No
0	1	No	No
1	0	No	No
1	1	Yes	Yes

Figure 4-1. Wake-up from Stop Mode

CAN stack can be changed from SLEEP mode to STOP mode by calling the Can_SetControllerMode(CAN_T_WAKEUP) service call.

Note

Refer the Reference Manual if wakuep is supported by hardware (bits register are implemented and INTC has connected the CAN wakeup interrupt signal).

Chapter 5
Module requirements

5.1 Exclusive Areas

CAN_EXCLUSIVE_AREA_00 - Used in “Can_DisableControllerInterrupts” function, to protect the variable for nesting level of enabling/disabling interrupts. Refer to CAN202 requirement.

CAN_EXCLUSIVE_AREA_01 - Used in “Can_EnableControllerInterrupts” function, to protect the variable for nesting level of enabling/disabling interrupts. Refer to CAN202 requirement.

CAN_EXCLUSIVE_AREA_02 - Used in “Can_Write” function to protect the operation for checking the status of MB and for reserving it as a free to use for transmission. If hardware transmit object is free the mutex for that HTH is set to “signaled”. Between this verification and signal operation the protection must be applied. Refer to CAN212 requirement.

CAN_EXCLUSIVE_AREA_04 - Used in "Can_IPW_ResetController" function to protect the operation for enabling the CAN module and requesting a Soft Reset.

CAN_EXCLUSIVE_AREA_05 - Used in "Can_ChangeBaudrate" function to protect the operation for the requests including: Enabling/Disabling the Can module, Puting/ Exiting Freeze mode, Setting value the bit fields for the baudrate configuration, Enabling/ Disabling Self Reception. In addition, this is used to protect for writting value to the variable which records the status of TX MBs.

CAN_EXCLUSIVE_AREA_06 - Used in a sub-function of "Can_SetControllerMode" function to protect the operation for putting the CAN module into Freeze mode, requesting a Soft Reset and setting up Software BusOff Recovery

CAN_EXCLUSIVE_AREA_07 - Used in a sub-function of “Can_DisableControllerInterrupts” function, to protect the operation for disabling interrupts of the Can module.

CAN_EXCLUSIVE_AREA_08 - Used in a sub-function of “Can_EnableControllerInterrupts” function, to protect the operation for enabling interrupts of the Can module.

CAN_EXCLUSIVE_AREA_11 - Used in a sub-function of "Can_ChangeBaudrate" function to protect the operation for configurating ID Acceptance Mode of the Rx FIFO ID Filter Table elements.

CAN_EXCLUSIVE_AREA_13 - Used in a sub-function of "Can_SetControllerMode" function to protect the operation for exiting Freeze mode.

Critical Region Exclusive Matrix

Below is the table depicting the exclusivity between different critical region IDs from the Can driver. If there is an “X” in a table, it means that those 2 critical regions cannot interrupt each other.

The critical regions from interrupts are grouped in “Interrupt Service Routines Critical Regions (composed diagram)”. If an exclusive area is “exclusive” with the composed “Interrupt Service Routines Critical Regions (composed diagram)” group, it means that it is exclusive with each one of the ISR critical regions.

Table 5-1. Exclusive Areas

	CAN_EXCLUSIVE_AREA_00	CAN_EXCLUSIVE_AREA_01	CAN_EXCLUSIVE_AREA_02	CAN_EXCLUSIVE_AREA_04	CAN_EXCLUSIVE_AREA_05	CAN_EXCLUSIVE_AREA_06	CAN_EXCLUSIVE_AREA_07	CAN_EXCLUSIVE_AREA_08	CAN_EXCLUSIVE_AREA_11	CAN_EXCLUSIVE_AREA_13	Interrupt Service Routines Critical Regions
CAN_EXCLUSIVE_AREA_00		X									
CAN_EXCLUSIVE_AREA_01	X										
CAN_EXCLUSIVE_AREA_02					X						X
CAN_EXCLUSIVE_AREA_04					X	X			X	X	

Table continues on the next page...

Table 5-1. Exclusive Areas (continued)

	CAN_EXCLUSIVE_AREA_00	CAN_EXCLUSIVE_AREA_01	CAN_EXCLUSIVE_AREA_02	CAN_EXCLUSIVE_AREA_04	CAN_EXCLUSIVE_AREA_05	CAN_EXCLUSIVE_AREA_06	CAN_EXCLUSIVE_AREA_07	CAN_EXCLUSIVE_AREA_08	CAN_EXCLUSIVE_AREA_11	CAN_EXCLUSIVE_AREA_13	Interrupt Service Routines Critical Regions
CAN_EXCLUSIVE_AREA_05			X	X		X			X	X	
CAN_EXCLUSIVE_AREA_06				X	X		X	X	X	X	
CAN_EXCLUSIVE_AREA_07						X		X			
CAN_EXCLUSIVE_AREA_08						X		X			
CAN_EXCLUSIVE_AREA_11				X	X	X				X	
CAN_EXCLUSIVE_AREA_13				X	X	X			X		
Interrupt Service Routines Critical Regions			X								

5.2 Peripheral Hardware Requirements

The CAN physical interface should be connected to the CAN module pins in order to get the CAN bus voltage levels.

There have to be another one CAN node present on the CAN bus in order to get the CAN bus synchronized.

5.3 ISR to Configure within OS - Dependencies

5.3.1 The following ISR's are used by the CAN driver

Table 5-2. CAN ISRs

ISR Name	Hardware Interrupt Vector
For CanCodeSizeOptimalization = STD_OFF	
Can_IsrFCA_PN	565
Can_IsrFCA_ERR	566
Can_IsrFCA_BO	567
Can_IsrFCA_MB_00_03	568
Can_IsrFCA_MB_04_07	569
Can_IsrFCA_MB_08_11	570
Can_IsrFCA_MB_12_15	571
Can_IsrFCA_MB_16_31	572
Can_IsrFCA_MB_32_63	573
Can_IsrFCA_MB_64_95	574
Can_IsrFCB_ERR	578
Can_IsrFCB_BO	579
Can_IsrFCB_MB_00_03	580
Can_IsrFCB_MB_04_07	581
Can_IsrFCB_MB_08_11	582
Can_IsrFCB_MB_12_15	583
Can_IsrFCB_MB_16_31	584
Can_IsrFCB_MB_32_63	585
Can_IsrFCB_MB_63_95	586
Can_IsrFCC_ERR	590
Can_IsrFCC_BO	591
Can_IsrFCC_MB_00_03	592
Can_IsrFCC_MB_04_07	593
Can_IsrFCC_MB_08_11	594
Can_IsrFCC_MB_12_15	595
Can_IsrFCC_MB_16_31	596
Can_IsrFCC_MB_32_63	597
Can_IsrFCC_MB_64_95	598
Can_IsrFCD_ERR	602
Can_IsrFCD_BO	603
Can_IsrFCD_MB_00_03	604
Can_IsrFCD_MB_04_07	605
Can_IsrFCD_MB_08_11	606
Can_IsrFCD_MB_12_15	607

Table continues on the next page...

Table 5-2. CAN ISRs (continued)

ISR Name	Hardware Interrupt Vector
Can_IsrFCD_MB_16_31	608
Can_IsrFCD_MB_32_63	609
Can_IsrFCD_MB_64_95	610
Can_IsrFCE_ERR	614
Can_IsrFCE_BO	615
Can_IsrFCE_MB_00_03	616
Can_IsrFCE_MB_04_07	617
Can_IsrFCE_MB_08_11	618
Can_IsrFCE_MB_12_15	619
Can_IsrFCE_MB_16_31	620
Can_IsrFCE_MB_32_63	621
Can_IsrFCE_MB_64_95	622
Can_IsrFCF_ERR	626
Can_IsrFCF_BO	627
Can_IsrFCF_MB_00_03	628
Can_IsrFCF_MB_04_07	629
Can_IsrFCF_MB_08_11	630
Can_IsrFCF_MB_12_15	631
Can_IsrFCF_MB_16_31	632
Can_IsrFCF_MB_32_63	633
Can_IsrFCF_MB_64_95	634
Can_IsrFCG_ERR	638
Can_IsrFCG_BO	639
Can_IsrFCG_MB_00_03	640
Can_IsrFCG_MB_04_07	641
Can_IsrFCG_MB_08_11	642
Can_IsrFCG_MB_12_15	643
Can_IsrFCG_MB_16_31	644
Can_IsrFCG_MB_32_63	645
Can_IsrFCG_MB_64_95	646
Can_IsrFCH_ERR	650
Can_IsrFCH_BO	651
Can_IsrFCH_MB_00_03	652
Can_IsrFCH_MB_04_07	653
Can_IsrFCH_MB_08_11	654
Can_IsrFCH_MB_12_15	655
Can_IsrFCH_MB_16_31	656
Can_IsrFCH_MB_32_63	657
Can_IsrFCH_MB_64_95	658

5.3.2 Macros for Interrupts

General Interrupts for every controller

CAN_BOISR(FC) expands ISR(Can_IsrFC##FC##_BO) for BusOff event.

CAN_WKPISR(FC) expands ISR(Can_IsrFC##FC##_WKP) for Wakeup event.

CAN_ERROR_NOTIFICATION_ENABLE == STD_ON

CAN_ERRISR(FC) expands ISR(Can_IsrFC##FC##_ERR) for error event.

CanCodeSizeOptimalization = STD_ON: All related ISRs are routed to one ISR function.

CAN_MB_UNIISRS(FC) expands ISR(Can_IsrFC##FC##_UNI) for Rx and Tx MBs.

CAN_MB_UNITXISRS(FC) expands ISR(Can_IsrFC##FC##_UNI) for Tx MBs.

CAN_MB_UNIRXISRS expands ISR(Can_IsrFC##FC##_UNI) for Rx MBs.

CanCodeSizeOptimalization = STD_OFF: All related ISRs have separate ISR functions taking care about the ISR processing.

CAN_RXFIFO_EVENTS(FC) expands ISR(Can_IsrFC##FC##_RxFifoEvents) for all Fifo events.

CAN_MB_ISRS(FC, Name, IdMin, IdMax) expands ISR(Can_IsrFC##FC##_##Name) for a group of Rx or Tx MBs.

CAN_MB_RXISRS(FC, Name, IdMin, IdMax) expands ISR(Can_IsrFC##FC##_##Name) for a group of Rx MBs.

CAN_MB_TXISRS(FC, Name, IdMin, IdMax) expands ISR(Can_IsrFC##FC##_##Name) for a group of Tx MBs.

Note

MPC574XG has the IFLAG1[7:4] bits assigned to the same interrupt, then the solution is to used CAN_RXFIFO_EVENTS interrupts macro generation. The CAN_RXFIFO_EVENT_UNIFIED define is generated by Tresos in Can_Cfg.h file and depends by the attribute “Can.CanConfigSet.RxFifoEventsUnified” from Resource properties file.

Interrupt Handlers Example for Can controller A:

ISR(**Can_IsrFCA_BO**)

ISR(**Can_IsrFCA_WKP**)

ISR(**Can_IsrFCA_UNI**)

ISR(**Can_IsrFCA_PN**)

ISR(**Can_IsrFCA_RxFifoEvents**) or ISR(**Can_IsrFCA_FrameAv**)
ISR(**Can_IsrFCA_Overf**) ISR(**Can_IsrFCA_Warn**)

ISR(**Can_IsrFCA_MB_00**)

ISR(**Can_IsrFCA_MB_00_03**)

ISR(**Can_IsrFCA_MB_04_07**)

ISR(**Can_IsrFCA_MB_08_11**)

ISR(**Can_IsrFCA_MB_12_15**)

ISR(**Can_IsrFCA_MB_16_31**)

ISR(**Can_IsrFCA_MB_32_63**)

ISR(**Can_IsrFCA_MB_64_95**)

Interrupt Handlers Example for Can controller B:

ISR(**Can_IsrFCB_BO**)

ISR(**Can_IsrFCB_WKP**)

ISR(**Can_IsrFCB_UNI**)

ISR(**Can_IsrFCB_RxFifoEvents**) or ISR(**Can_IsrFCB_FrameAv**)
ISR(**Can_IsrFCB_Overf**) ISR(**Can_IsrFCB_Warn**)

ISR(**Can_IsrFCB_MB_00_03**)

ISR(**Can_IsrFCB_MB_04_07**)

ISR(**Can_IsrFCB_MB_08_11**)

ISR(**Can_IsrFCB_MB_12_15**)

ISR(**Can_IsrFCB_MB_16_31**)

ISR(**Can_IsrFCB_MB_32_63**)

ISR(**Can_IsrFCB_MB_64_95**)

Interrupt Handlers Example for Can controller C:

ISR(**Can_IsrFCC_BO**)

ISR(**Can_IsrFCC_WKP**)

ISR(**Can_IsrFCC_UNI**)

ISR(**Can_IsrFCC_RxFifoEvents**) or ISR(**Can_IsrFCC_FrameAv**)
ISR(**Can_IsrFCC_Overf**) ISR(**Can_IsrFCC_Warn**)

ISR(**Can_IsrFCC_MB_00_03**)

ISR(**Can_IsrFCC_MB_04_07**)

ISR(**Can_IsrFCC_MB_08_11**)

ISR(**Can_IsrFCC_MB_12_15**)

ISR(**Can_IsrFCC_MB_16_31**)

ISR(**Can_IsrFCC_MB_32_63**)

ISR(**Can_IsrFCC_MB_64_95**)

Interrupt Handlers Example for Can controller D:

ISR(**Can_IsrFCD_BO**)

ISR(**Can_IsrFCD_WKP**)

ISR(**Can_IsrFCD_UNI**)

ISR(**Can_IsrFCD_RxFifoEvents**) or ISR(**Can_IsrFCD_FrameAv**)
ISR(**Can_IsrFCD_Overf**) ISR(**Can_IsrFCD_Warn**)

ISR(**Can_IsrFCD_MB_00_03**)

ISR(**Can_IsrFCD_MB_04_07**)

ISR(**Can_IsrFCD_MB_08_11**)

ISR(**Can_IsrFCD_MB_12_15**)

ISR(**Can_IsrFCD_MB_16_31**)

ISR(**Can_IsrFCD_MB_32_63**)

ISR(**Can_IsrFCD_MB_64_95**)

Interrupt Handlers Example for Can controller E:

ISR(**Can_IsrFCE_BO**)

ISR(**Can_IsrFCE_WKP**)

ISR(**Can_IsrFCE_UNI**)

ISR(**Can_IsrFCE_RxFifoEvents**) or ISR(**Can_IsrFCE_FrameAv**)
ISR(**Can_IsrFCE_Overf**) ISR(**Can_IsrFCE_Warn**)

ISR(**Can_IsrFCE_MB_00_03**)

ISR(**Can_IsrFCE_MB_04_07**)

ISR(**Can_IsrFCE_MB_08_11**)

ISR(**Can_IsrFCE_MB_12_15**)

ISR(**Can_IsrFCE_MB_16_31**)

ISR(**Can_IsrFCE_MB_32_63**)

ISR(**Can_IsrFCE_MB_64_95**)

Interrupt Handlers Example for Can controller F:

ISR(**Can_IsrFCF_BO**)

ISR(**Can_IsrFCF_WKP**)

ISR(**Can_IsrFCF_UNI**)

ISR(**Can_IsrFCF_RxFifoEvents**) or ISR(**Can_IsrFCF_FrameAv**)
ISR(**Can_IsrFCF_Overf**) ISR(**Can_IsrFCF_Warn**)

ISR(**Can_IsrFCF_MB_00_03**)

ISR(**Can_IsrFCF_MB_04_07**)

ISR(**Can_IsrFCF_MB_08_11**)

ISR(**Can_IsrFCF_MB_12_15**)

ISR(**Can_IsrFCF_MB_16_31**)

ISR(**Can_IsrFCF_MB_32_63**)

ISR(**Can_IsrFCF_MB_64_95**)

Interrupt Handlers Example for Can controller G:

ISR(**Can_IsrFCG_BO**)

ISR(**Can_IsrFCG_WKP**)

ISR(**Can_IsrFCG_UNI**)

ISR(**Can_IsrFCG_RxFifoEvents**) or ISR(**Can_IsrFCG_FrameAv**)
ISR(**Can_IsrFCG_Overf**) ISR(**Can_IsrFCG_Warn**)

ISR(**Can_IsrFCG_MB_00_03**)

ISR(**Can_IsrFCG_MB_04_07**)

ISR(**Can_IsrFCG_MB_08_11**)

ISR(**Can_IsrFCG_MB_12_15**)

ISR(**Can_IsrFCG_MB_16_31**)

ISR(**Can_IsrFCG_MB_32_63**)

ISR(**Can_IsrFCG_MB_64_95**)

Interrupt Handlers Example for Can controller H:

ISR(**Can_IsrFCH_BO**)

ISR(**Can_IsrFCH_WKP**)

ISR(**Can_IsrFCH_UNI**)

ISR(**Can_IsrFCH_RxFifoEvents**) or ISR(**Can_IsrFCH_FrameAv**)
ISR(**Can_IsrFCH_Overf**) ISR(**Can_IsrFCH_Warn**)

ISR(**Can_IsrFCH_MB_00_03**)

ISR(**Can_IsrFCH_MB_04_07**)

ISR(**Can_IsrFCH_MB_08_11**)

ISR(**Can_IsrFCH_MB_12_15**)

ISR(**Can_IsrFCH_MB_16_31**)

ISR(**Can_IsrFCH_MB_32_63**)

ISR(**Can_IsrFCH_MB_64_95**)

5.3.3 ISR Macro

MCAL drivers use the ISR macro to define the functions that will process hardware interrupts. Depending on whether the OS is used or not, this macro can have different definitions:

a. OS is not used - AUTOSAR_OS_NOT_USED is defined:

i. If USE_SW_VECTOR_MODE is defined:

```
#define ISR(IsrName) void IsrName(void)
```

In this case, drivers' interrupt handlers are normal C functions and the prolog/epilog handle the context save and restore.

ii. If USE_SW_VECTOR_MODE is not defined:

```
#define ISR(IsrName) INTERRUPT_FUNC void IsrName(void)
```

In this case, drivers' interrupt handlers must save and restore the execution context.

NXP Semiconductor OS is used - AUTOSAR_OS_NOT_USED is not defined

```
#define ISR(IsrName) void OS_isr_##IsrName()
```

In this case, OS is handling the execution context when an interrupt occurs. Drivers' interrupt handlers are normal C functions.

Other vendor's OS is used - AUTOSAR_OS_NOT_USED is not defined. Please refer to the OS documentation for description of the ISR macro.

5.4 Other AUTOSAR Modules - Dependencies

- **Mcu:** This module shall be initialized before using CAN. This module is required for setting the system clock frequency (clock for CAN).
- **Det** (only if CanDevErrorDetect=true): This module is necessary for enabling Development error detection. The API function used is Det_ReportError(). The activation/deactivation of Development error detection is configurable using 'CanDevErrorDetect' configuration parameter.
- **EcuM:** This module is necessary for a reference to the Wakeup source for this controller as defined in the ECU State Manager.
- **EcuC:** This module is necessary for handle Postbuild Variant.
- **Resource:** Sub-Derivative model is selected from Resource configuration.

5.5 Data Cache Restriction

In the DMA transfer mode, DMA transfers may issue cache coherency problems. To avoid possible coherency issues when D-CACHE is enabled, the user shall ensure that the buffers used as TCD source and destination are allocated in the NON-CACHEABLE area (by means of Memmap).

5.6 XBAR Configuration

5.7 User Mode Support Requirement

The CAN driver is supported to be able to run from User Mode. This mode is activated by a vendor specific pre-compile boolean configuration parameter CanEnabelUserModeSupport {CAN_ENABLE_USER_MODE_SUPPORT}.

If the parameter CanEnabelUserModeSupport is set to 'true' {CAN_ENABLE_USER_MODE_SUPPORT = STD_ON}, the CAN driver will perform the following measures:

(a) Using 'call trusted function' stubs for all internal function calls that access the CAN_MCR register which is requiring supervisor mode. The internal functions need 'call trusted function' as following:

Can_SetControllerMode()

Can_Init()

Can_ChangeBaudrate ()

Can_Write()(If the Pretended Networking mode is used)

Can_MainFunctionMode()

Can_SetIcomConfig()

Can_DeactivateIcom()

Can_ResetController()

NOTE

It is not required to initialize REG_PROT (Register Protection) for MPC574XG.

Chapter 6
Main API Requirements

6.1 Main Functions Calls within BSW Scheduler

CAN Driver support 4 main functions that can be configured to be scheduled by BSW scheduler:

- `FUNC (void, CAN_CODE) Can_MainFunction_Write( void )`
- `FUNC (void, CAN_CODE) Can_MainFucntion_Read( void )`
- `FUNC (void, CAN_CODE) Can_MainFunction_BusOff( void )`
- `FUNC (void, CAN_CODE) Can_MainFunction_Mode( void )`

These Autosar APIs are scheduled if these 3 events are configured to be in “Polling” mode by the following parameters:

- CanTxProcessing

```
#define CAN_TXPOLL_SUPPORTED (STD_ON)
```

- CanRxProcessing

```
#define CAN_RXPOLL_SUPPORTED (STD_ON)
```

- CanBusoffProcessing

```
#define CAN_BUSOFFPOLL_SUPPORTED (STD_ON)
```

The period for polling is configured by the following 4 parameters:

- CanMainFunctionWritePeriod

```
#define CAN_MAINFUNCTION_PERIOD_WRITE (uint32)0.0010U
```

- CanMainFunctionReadPeriod

```
#define CAN_MAINFUNCTION_PERIOD_READ (uint32)0.0010U
```

- CanMainFunctionBusoffPeriod

```
#define CAN_MAINFUNCTION_PERIOD_BUSOFF (uint32)0.0010U
```

- CanMainFunctionModePeriod

```
#define CAN_MAINFUNCTION_MODE_PERIOD (uint32)0.0010U
```

Note

A configuration for an hardware unit can be possible in such a way that one controller will handle events by interrupts and another by polling method.

6.2 API Requirements

CAN360,CAN361,CAN362,CAN363,CAN228,CAN112,CAN185,CAN186,CAN235,

6.3 Calls to Notification Functions, Callbacks, Callouts

Call-back Notifications

The CAN stack provides the following call-back notifications:

- `CanIf_TxConfirmation`: This CAN Interface call-back function is called when a CAN message has been transmitted.

```
FUNC (void, CAN_CODE) CanIf_TxConfirmation(PduIdType CanTxPduId)
```

- `CanIf_RxIndication`: This CAN Interface call-back function is called when valid CAN message is received.

```
FUNC (void, CAN_CODE) CanIf_RxIndication(uint8 Hrh, Can_IdType CanId, uint8 CanDlc,
uint8Ptr CanSduPtr)
```

- `CanIf_CancelTxConfirmation`: This CAN Interface call-back function is called when the CAN message has been canceled during the transmission.

```
FUNC (void, CAN_CODE) CanIf_CancelTxConfirmation(const Can_PduType * PduInfoPtr)
```

- `CanIf_ControllerBusOff`: This CAN Interface call-back function is called when the CAN controller reached the bus-off state (see CAN specification for further details).

```
FUNC (void, CAN_CODE) CanIf_ControllerBusOff(uint8 Controller)
```

User Notification

- None

Chapter 7
Memory Allocation

7.1 Sections to be defined in Can_MemMap.h

Table 7-1. Sections to be defined in Can_MemMap.h

Section name	Type of section	Description
CAN_START_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data
CAN_STOP_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	End of Memory Section for Config Data
CAN_START_SEC_CODE	Code	Start of memory Section for Code
CAN_STOP_SEC_CODE	Code	End of memory Section for Code
CAN_START_SEC_VAR_INIT_UNSPECIFIED	Variables	Used for variables, structures, arrays, when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. These variables are initialized with values after every reset.
CAN_STOP_SEC_VAR_INIT_UNSPECIFIED	Variables	End of above section.
CAN_START_SEC_VAR_NO_INIT_UNSPECIFIED	Variables	These variables are never cleared and never initialized by start-up code.
CAN_STOP_SEC_VAR_NO_INIT_UNSPECIFIED	Variables	End of above section.
CAN_START_SEC_CONST_UNSPECIFIED	Constant Data	Used for constants
CAN_STOP_SEC_CONST_UNSPECIFIED	Constant Data	End of above section.

7.2 Linker command file

Memory shall be allocated for every section defined in CAN_MemMap.h

Chapter 8
Configuration parameters considerations

Configuration parameter class for Autosar CAN driver fall into the following variants as defined below:

8.1 Configuration Parameters

Specifies whether the configuration parameter shall be of configuration class Post Build

Table 8-1. Configuration Parameters

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
Can	IMPLEMENTATION_CONFIG_VARIANT	Pre Compile parameter for all Variants of Configuration	Pre compile
CanGeneral			
	CanDevErrorDetection	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanVersionInfoApi	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanIndex	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanMainFunctionBusoffPeriod	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanMultiplexedTransmission	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanHardwareCancellation	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanIdenticalIdCancellation	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanTimeoutDuration	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanLPduReceiveCalloutFunction	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanCounterRef	Pre Compile parameter for all Variants of Configuration	Pre compile

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	CanMainFunctionReadPeriodRef	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanMainFunctionWritePeriodRef	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanSupportTTCANRef	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanCodeSizeOptimization	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanExtendedIDSupport	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanMBCountExtensionSupport	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanApiEnableMbAbort	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanEnableDualClockMode	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanChangeBaudrateApi	Pre Compile parameter for all Variants of Configuration	Pre compile
CanGeneral \CanMainFunctionRWPeriods			
	CanMainFunctionReadPeriod	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanMainFunctionWritePeriod	Pre Compile parameter for all Variants of Configuration	Pre compile
CanController			
	CanHwChannel	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanControllerActivation	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanControllerBaseAddress	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanControllerId	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanRxProcessing	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanTxProcessing	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanBusoffProcessing	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanBccSupport	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanListenOnlyMode	VariantPC or VariantPB	VariantPC or VariantPB
	CanLoopBackMode	VariantPC or VariantPB	VariantPC or VariantPB
	CanSoftwareBusOffRecovery	VariantPC or VariantPB	VariantPC or VariantPB
	CanAutoBusOffRecovery	VariantPC or VariantPB	VariantPC or VariantPB

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	CanTrippleSamplingEnable	VariantPC or VariantPB	VariantPC or VariantPB
	CanLowestBuffTransmitFirst	VariantPC or VariantPB	VariantPC or VariantPB
	CanLocalPriorityEn	VariantPC or VariantPB	VariantPC or VariantPB
	CanWarningEnable	VariantPC or VariantPB	VariantPC or VariantPB
	CanClockFromBus	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanCpuClockRef	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanCpuClockRef_Alternate	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanControllerRxFifoEnable	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanRxFifoWarningNotification	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanRxFifoOverflowNotification	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanErrorControllerNotifEn	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanErrorControllerNotification	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanControllerDefaultBaudrate	Pre Compile parameter for all Variants of Configuration	Pre compile
CanController \CanControllerBaudRate			
	CanControllerBaudRate	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerCheckCanStandard	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerPropSeg	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerSeg1	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerSeg2	VariantPC or VariantPB	VariantPC or VariantPB
	CanSyncJumpWidth	VariantPC or VariantPB	VariantPC or VariantPB
	CanAdvancedSetting	VariantPC or VariantPB	VariantPC or VariantPB
	CanBusLength	VariantPC or VariantPB	VariantPC or VariantPB
	CanPropDelayOfTranceiver	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerPrescaller	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerPrescaller_Alt	VariantPC or VariantPB	VariantPC or VariantPB
CanController\CanRxFifo			
	CanControllerIDAcceptanceMode	VariantPC or VariantPB	VariantPC or VariantPB
	CanIDValue0	VariantPC or VariantPB	VariantPC or VariantPB
	CanIDValue1	VariantPC or VariantPB	VariantPC or VariantPB
	CanIDValue2	VariantPC or VariantPB	VariantPC or VariantPB
	CanIDValue3	VariantPC or VariantPB	VariantPC or VariantPB

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	CanTableIDType	VariantPC or VariantPB	VariantPC or VariantPB
	CanMBFilterMaskValue	VariantPC or VariantPB	VariantPC or VariantPB
	CanRxFifoFiltersNumber	VariantPC or VariantPB	VariantPC or VariantPB
	CanRxFifoGlobalMaskValue	VariantPC or VariantPB	VariantPC or VariantPB
CanController\CanFilterMask			
	CanFilterMaskValue	VariantPC or VariantPB	VariantPC or VariantPB
CanController \CanHardwareObject			
	CanHandleType	VariantPC or VariantPB	VariantPC or VariantPB
	CanIdType	VariantPC or VariantPB	VariantPC or VariantPB
	CanIdValue	VariantPC or VariantPB	VariantPC or VariantPB
	CanMBPrio	VariantPC or VariantPB	VariantPC or VariantPB
	CanObjectId	VariantPC or VariantPB	VariantPC or VariantPB
	CanObjectType	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerRef	VariantPC or VariantPB	VariantPC or VariantPB
	CanFilterMaskRef	VariantPC or VariantPB	VariantPC or VariantPB
	CanMainFunctionRWPeriodRef	VariantPC or VariantPB	VariantPC or VariantPB
CanController \CanTTController			
	CanTTControllerApplWatchdogLimit	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerCycleCountMax	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerExpectedTxTrigger	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerExternalClockSynchronisation	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerGlobalTimeFiltering	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerInitialRefOffset	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerInterruptEnable	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerLevel2	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerNTUConfig	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerOperationMode	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerSyncDeviation	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerTURRestore	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerTimeMaster	VariantPC or VariantPB	VariantPC or VariantPB

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	CanTTControllerTimeMasterPriority	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerTxEnableWindowLength	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerWatchTriggerGapTimeMark	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTControllerWatchTriggerTimeMark	VariantPC or VariantPB	VariantPC or VariantPB
	CanTTIRQProcessing	VariantPC or VariantPB	VariantPC or VariantPB

Chapter 9
Integration Steps

This section gives a brief overview of the steps needed for integrating Controller Area Network :

- Generate the required CAN configurations. For more details refer to section Files Required for the Compilation
- Allocate proper memory sections in CAN_MemMap.h and linker command file. For more details refer to section Sections to be defined in Can_MemMap.h
- Compile & build the CAN with all the dependent modules. For more details refer to section Building the Driver

Chapter 10
External Assumptions for CAN driver

The section presents requirements that must be complied with when integrating CAN driver into the application.

[SMCAL_CPR_EXT4]

<< The external application shall call Can_Init function only when the driver state is CAN_UNINIT and the state of all controllers is UNINIT. >>

[SMCAL_CPR_EXT5]

<< The external application shall call Can_MainFunction_Write only after driver initialization. >>

[SMCAL_CPR_EXT6]

<< The external application shall call Can_MainFunction_Read only after driver initialization. >>

[SMCAL_CPR_EXT7]

<< The external application shall call Can_MainFunction_BusOff only after driver initialization. >>

[SMCAL_CPR_EXT8]

<< The external application shall call Can_MainFunction_Wakeup only after driver initialization. >>

[SMCAL_CPR_EXT9]

<< The external application shall call Can_SetControllerMode only after driver initialization >>

[SMCAL_CPR_EXT10]

<< The external application shall call Can_DisableControllerInterrupts function only after driver initialization. >>

[SMCAL_CPR_EXT11]

<< The external application shall call Can_EnableControllerInterrupts function only after driver initialization. >>

[SMCAL_CPR_EXT12]

<< The external application shall call Can_Write function only after driver initialization. >>

[SMCAL_CPR_EXT13]

<< The external application shall assure that Can_Init does not preempt and is not preempted by any other CAN driver API excepting Can_GetVersionInfo.

The external application shall assure that Can_Init does not preempt itself. >>

[SMCAL_CPR_EXT14]

<< The external application shall assure that Can_MainFunction_Write does not preempt and is not preempted by any other CAN driver API exceptin Can_GetVersionInfo.The external application shall assure that Can_MainFunction_Write does not preempt itself. >>

[SMCAL_CPR_EXT15]

<< The external application shall assure that Can_MainFunction_Read does not preempt and is not preempted by any other CAN driver API exceptin Can_GetVersionInfo.The external application shall assure that Can_MainFunction_Read does not preempt itself. >>

[SMCAL_CPR_EXT16]

<< The external application shall assure that Can_MainFunction_BusOff does not preempt and is not preempted by any other CAN driver API exceptin Can_GetVersionInfo.The external application shall assure that Can_MainFunction_BusOff does not preempt itself. >>

[SMCAL_CPR_EXT17]

<< The external application shall assure that Can_SetControllerMode does not preempt and is not preempted by any other CAN driver API using the same controller parameter.

The external application shall assure that Can_SetControllerMode does not preempt itself. >>

[SMCAL_CPR_EXT18]

<< The external application shall assure that Can_DisableControllerInterrupts does not preempt and is not preempted by any other CAN driver API using the same controller parameter. >>

[SMCAL_CPR_EXT19]

<< The external application shall assure that Can_EnableControllerInterrupts does not preempt and is not preempted by any other CAN driver API using the same controller parameter. >>

[SMCAL_CPR_EXT20]

<< The external application shall assure that Can_Write does not preempt and is not preempted by any other CAN driver API using the same controller as the hardware handle parameter.The external application shall assure that Can_Write does not preempt itself for the same hardware handle parameter. >>

[SMCAL_CPR_EXT21]

<< The external application shall call Can_ChangeBaudrate only when the CAN controller is in state STOPPED. >>

[SMCAL_CPR_EXT22]

<< The external application shall call Can_Init function only when the driver state is CAN_UNINIT and the state of all controllers is UNINIT. >>

[SMCAL_CPR_EXT23]

<< The external application shall assure that Can_CheckWakeup does not preempt and is not preempted by any other CAN driver API using the same controller parameter. The external application shall assure that Can_CheckWakeup does not preempt itself. >>

[SMCAL_CPR_EXT24]

<< The external application shall call Can_CheckWakeup function only after driver initialization. >>

[SMCAL_CPR_EXT25]

<< The external application shall call Can_MainFunction_Mode function only after driver initialization. >>

[SMCAL_CPR_EXT26]

<< The external application shall call Can_Init function only when the driver state is CAN_UNINIT and the state of all controllers is UNINIT. >>

[SMCAL_CPR_EXT27]

<< The external application shall call Can_SetControllerMode(CAN_T_START) only when the CAN controller is in state STOPPED. >>

[SMCAL_CPR_EXT28]

<< The external application shall call Can_SetControllerMode(CAN_T_STOP) only when the CAN controller is in state STARTED or STOPPED. >>

[SMCAL_CPR_EXT29]

<< The external application shall call Can_SetControllerMode(CAN_T_SLEEP) only when the CAN controller is in state SLEEP or STOPPED. >>

[SMCAL_CPR_EXT30]

<< The external application shall call Can_SetControllerMode(CAN_T_WAKEUP) only when the CAN controller is in state SLEEP or STOPPED. >>

[SMCAL_CPR_EXT31]

<< The external application shall assure that Can_ChangeBaudrate does not preempt and is not preempted by any other CAN driver API using the same controller parameter.The external application shall assure that Can_ChangeBaudrate does not preempt itself.The external application shal call Can_ChangeBaudrate only when the controller parameter is STOPPED. >>

[SMCAL_CPR_EXT32]

<< The external application shall call Can_ChangeBaudrate only after driver initialization and when the configured controller is in the STOPPED state. >>

[SMCAL_CPR_EXT33]

<< The external application shall assure that Can_CheckBaudrate does not preempt and is not preempted by any other CAN driver API using the same controller parameter. >>

[SMCAL_CPR_EXT34]

<< The external application shall call Can_CheckBaudrate only after driver initialization. >>

[SMCAL_CPR_EXT163]

<< If interrupts are locked a centralized function pair to lock and unlock interrupts shall be used. >>

[CAN023]

<< The consistency of the configuration must be checked by the configuration tool(s). >>

[CAN024]

<< The valid values that can be configured are hardware dependent. Therefore the rules and constraints can't be given in the standard. The configuration tool is responsible to do a static configuration checking, also regarding dependencies between modules (i.e. Port driver, MCU driver etc.) >>

[CAN039]

<< Can_ReturnType

Return values of CAN driver API

CAN_OK

CAN_NOT_OK

CAN_BUSY >>

NOTE

defined into Can_GeneralTypes.h included into Base module

[CAN240]

<< The Mcu module (SPAL see [8]) shall configure register settings that are 'shared' with other modules. >>

NOTE

not a requirement for CAN module

[CAN285]

<< Transmit cancellation may only be used when transmit buffers are enabled inside the CanIf module. >>

NOTE

This is a specification for CanIf module.Safety manual

[CAN288]

<< The TX request for the new L-PDU shall be repeated by the CanIf module, inside the notification function CanIf_CancelTxConfirmation.

Implementation note:

For sequence relevant streams the sender must assure that the next transmit request for the same CAN ID is only initiated after the last request was confirmed. >>

NOTE

This is a specification for CanIf module.Safety manual.

[CAN415]

<< Can_PduType

This type is used to provide ID, DLC and SDU from CAN interface to CAN driver. >>

NOTE

defined into Can_GeneralTypes.h included into Base module

[CAN416]

<< Can_IdType

Represents the Identifier of an L-PDU. For extended IDs the most significant bit is set. >>

NOTE

defined into Can_GeneralTypes.h included into Base module

[CAN417]

<< Can_StateTransitionType

CAN_T_START: CAN controller transition value to request state STARTED.

CAN_T_STOP: CAN controller transition value to request state STOPPED.

CAN_T_SLEEP: CAN controller transition value to request state SLEEP.

CAN_T_WAKEUP: CAN controller transition value to request state STOPPED from state SLEEP.

State transitions that are used by the function CAN_SetControllerMode >>

NOTE

defined into Can_GeneralTypes.h included into Base module

[CAN429]

<< Can_HwHandleType

Represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects use extended range. >>

NOTE

defined into Can_GeneralTypes.h included into Base module

[CAN436]

<< Can_GeneralTypes.h shall contain all types and constants that are shared among the AUTOSAR CAN modules Can, CanIf and CanTrcv. >>

NOTE

Implemented in base

[CAN437]

<< The integrator of the Can modules shall provide the file Can_GeneralTypes.h. >>

[CAN438]

<< The content of Can_GeneralTypes.h consists of types specified within [5] and the following type specifications within this document except Can_ConfigType. >>

NOTE

Implemented in base

[CAN455]

<< The service Can_CheckBaudrate(Controller, Baudrate) shall be called by CanIf_CheckBaudrate() for the requested CAN controller. >>

NOTE

This is a requirement for CanIf

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number IM35CANASR4.2 Rev0002 R1.0.0
Revision 1.0

AUTOSAR