Integration Manual

for MPC5634M CAN Driver

Document Number: IM14CANASR3.0R2.0.0
Rev. 1.1

freescale™

Contents

Section Number **Title** **Page**

Chapter 6
Main API Requirements

Chapter 7
Memory Allocation

Chapter 8
Configuration Parameters

Chapter 9
Integration Steps

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 integration manual describes the integration requirements for CAN Driver for MPC5634M microcontrollers.

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
Building the Driver

This section describes the source files and various compilers, linker options used for building the Autosar CAN driver for Freescale SemiconductorMPC5634M. It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

The CAN driver files are compiled using

- GHS 5.1.7
- DIAB 5_8_0_02 wind00198363 20100511 123238
- CW Version 4.3 build 182
- COSMIC Software PPC C Cross Compiler V4.3.4 - 16 Nov 2011 - Win32-F

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

Note: The TS_T2D14M20I0R0 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 PowerPC architecture and Derivative_Id = 14 identifies the MPC5634M)

3.1.1 GHS Compiler/Linker/Assembler Options

Table 3-1. Compiler Options

Option	Description
-cpu= ppc5646	Selects target processor: ppc5646x
-ansi	Enforces strict ANSI mode (C89 standard)
-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.
--no_commons	Allocates uninitialized global variables to a section and initializes them to zero at program startup. This may improve optimizations by giving the compiler optimizer more information about the location of the variable.
-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
-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.
-DGHS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GHS preprocessor symbol.

Table 3-2. Assembler Options

Option	Description
-cpu= ppc5646	Selects target processor: ppc5646

Table 3-3. Linker Options

Option	Description
-cpu= ppc5646	Selects target processor: ppc5646
-nostartfiles	Do not use Start files.
-vle	Enables VLE code generation
-linker_warnings	Display linker warnings

3.1.2 DIAB Compiler/Linker/Assembler Options

Table 3-4. Compiler Options

Option	Description
-tPPCE200Z4VEG:simple	Sets target processor to PPCE200Z4, generates ELF using EABI conventions, All Single Hardware Floating Point (Single precision uses hardware, double precision is mapped to single precision), 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
-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.
-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
-Xalign-functions=4	Align each function on an address boundary divisible by 4
-g	Generate symbolic debugger information. Do most target-independent optimizations. Also, disable most target-dependent optimizations: option -g2 also disables basic reordering and all peephole optimizations.
-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
-Xforce-prototypes	Generate warnings if a function is used without a previous prototype declaration
-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
-Xlint	Generate warnings when suspicious and non-portable C code is encountered. Enables all warnings

Table continues on the next page...

Table 3-4. Compiler Options (continued)

Option	Description
-ei1604	Suppress the warning messages 1604.
-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.
-DDIAB	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the DIAB preprocessor symbol.

Table 3-5. Assembler Options

Option	Description
-tPPCE200Z4VEN:simple	Selects target processor: PPCE200Z4, generates ELF using EABI conventions, NO floating point support, 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

Table 3-6. Linker Options

Option	Description
-tPPCE200Z4VEN:simple	Selects target processor: PPCE200Z4, generates ELF using EABI conventions, NO floating point support, 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
-lc	Specifies to linker to search for libc.a
-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.
-Xlibc-old	Enables usage of legacy (pre-release 5.6) libraries

3.1.3 CW Compiler/Linker/Assembler Options

Table 3-7. Compiler Options

Option	Description
-proc Zen	Generates and links object code for Zen processor. The compiler uses unsigned as the default parameter for the -char switch
-lang c	Expects source code to conform to the language specified by the ISO/IEC 9899-1990 ("C90") standard
-opt all	This option is selected all optimization (the same as -opt speed,level=4,intrinsics,noframe)
-common off	Disables moving uninitialized data into a common section
-sdatathreshold 0	Specifies the threshold size (in bytes) for an item considered by the linker to be small data. (The linker stores small data items in the Small Data address space. The compiler can generate faster code to access this data.)
-sdata2threshold 0	Specifies the threshold size (in bytes) for an item considered by the linker to be small constant data. (The linker stores small constant data items in the Small Constant Data address space.)
-vle	Tells the compiler and linker to generate and lay out Variable Length Encoded (VLE) instructions, available on Zen variants of Power Architecture processors
-use_lmw_stmw on	Enables the use of multiple load and store instructions for function prologues and epilogues
-ppc_asm_to_vle	Converts regular Power Architecture assembler mnemonics to equivalent VLE (Variable Length Encoded) assembler mnemonics in the inline assembler
-cpp_exceptions off	When on, generates executable code for C++ exceptions. When off, generates smaller, faster executable code
-func_align 4	Specifies alignment of functions in executable code
-sym dwarf-2,full	Generate DWARF-2-conforming debugging information (Debug With Arbitrary Record Format)
-gdwarf-2	Generate DWARF-2-conforming debugging information (Debug With Arbitrary Record Format). The linker ignores debugging information that is not in the Dwarf 1, Dwarf 2 format
-w on	Turns on most warning messages
-r	Compiler should expect function prototypes
-w undefmacro	Issues warning messages on the use of undefined macros in #if and #elif conditionals
-char unsigned	Controls the default sign of the char data type: char data items are unsigned
-nosyspath	Performs a search of both the user and system paths, treating #include statements of the form #include xyz the same as the form #include "xyz"
-fp none	No floating point code generation
-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 3-7. Compiler Options (continued)

Option	Description
-DMWERKS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the CWpreprocessor symbol.

Table 3-8. Assembler Options

Option	Description
-proc Zen	Generates and links object code for Zen processor. The compiler uses unsigned as the default parameter for the -char switch
-vle	Tells the compiler and linker to generate and lay out Variable Length Encoded (VLE) instructions, available on Zen variants of Power Architecture processors
-sym dwarf-2,full	Generate DWARF-2-conforming debugging information (Debug With Arbitrary Record Format)
-gdwarf-2	Generate DWARF-2-conforming debugging information (Debug With Arbitrary Record Format). The linker ignores debugging information that is not in the Dwarf 1, Dwarf 2 format.

Table 3-9. Linker Options

Option	Description
-proc Zen	Generates and links object code for Zen processor. The compiler uses unsigned as the default parameter for the -char switch
-code_merging all	Removes duplicated functions to reduce object code size
-far_near_addressing	Simplifies address computations to reduce object code size and improve performance
-vle_enhance_merging	Removes duplicated functions that are called by functions that use VLE instructions to reduce object code size
-listdwarf	DWARF debugging information in the linker's map file
-sym dwarf-2,full	Generate DWARF-2-conforming debugging information (Debug With Arbitrary Record Format)
-char unsigned	Controls the default sign of the char data type: char data items are unsigned.

3.1.4 CSMC Compiler/Linker/Assembler Options

Table 3-10. Compiler Options

Option	Description
-l	Create listing file; this option directs the compiler to produce an assembly language file with C source line interspersed in it. Please note that the C source lines are commented in the assembly language file: they start with ';'.
+modvc	Memory model with "medium size" application, in detail: "data" less than 64kb, "constants" less than 64kb, no code size limit
+rev	Tells the compiler to reverse the order of bits in the bitfields. You need this option in order to use most non-Cosmic header files.

Table continues on the next page...

Table 3-10. Compiler Options (continued)

Option	Description
-pc99	authorize the repetition of the const and volatile modifiers in the declaration either directly or indirectly in the typedef.
-odB5	disable the optimization B5.
-pxf	prefix filenames in the debug information with absolute full path name.
+debug	produce debug information to be used by the debug utilities provided with the compiler and by any external debugger.
-DCSMC	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the CSMC preprocessor symbol.
-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
-DEU_DISABLE_ANSILIB_CALLS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the EU_DISABLE_ANSILIB_CALLS preprocessor symbol.
-DMCAL_CER_VALIDATION	-D defines a preprocessor symbol for CER Report
-DMCAL_VERSION_CHECK	-D defines enable the cross check between the AutoSar component Version Numbers

Table 3-11. Assembler Options

Option	Description
-l	create a listing file. The name of the listing file is derived from the input file name by replacing the suffix by the ".ls" extension

Table 3-12. Linker Options

Option	Description
-p	display symbols with physical address instead of logical address in the map file.

3.2 Files required for Compilation

This section describes the include files required to compile, assemble (if assembler code) and link the Autosar CAN driver for Freescale SemiconductorMPC5634M microcontrollers.

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

Can Files:

Table 3-13. Include Files

..\Can_TS_T2D14M20I0R0\include\	FlexCan_LLD.h
	Reg_eSys_FlexCan.h
	Can.h
	Can_LLD.h
	CanIf_Cbk.h
	CanIf.h

Table 3-14. Source Files

..\Can_TS_T2D14M20I0R0\src\	Can_Irq.c
	Can.c
	Can_LLD.c
	FlexCan_LLD.c

Can_Cfg.c (For PC Variant) - This file should be generated by the user using a configuration tool for compilation

Can_PBcfg.c (For PB Variant) - 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

Other include files:

Table 3-15. Files from Base folder:

..\Base_TS_T2D14M20I0R0\specific\include\	Compiler.h
	Compiler_Cfg.h
	MemMap.h
	Platform_Types.h
	Std_Types.h
	Reg_eSys.h
	Reg_Macros.h
	Cer.h
	ComStack_Types.h
	Soc_Ips.h
	Mcal.h

Table 3-16. Files from Dem folder:

..\Dem_TS_T2D14M20I0R0\generic\include\	Dem.h
	Dem_IntErrId.h
	Dem_Types.h

Table 3-17. Files from Det folder:

..\Det_TS_T2D14M20I0R0\generc\include\	Det.h

Table 3-18. Files from EcuM folder:

..\EcuM_TS_T2D14M20I0R0\include\	EcuM.c
	EcuM.h
	EcuM_Cbk.h
	EcuM_Cfg.h

Table 3-19. Files from SchM folder:

..\SchM_TS_T2D14M20I0R0\include\	SchM_Can.c
	SchM_Can.h

3.3 Setting up the Plugins

The CAN driver was designed to be configured by using the EB Tresos Studio (version Tresos 2010a.sr4 20100415-release2010a-sr4 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_T2D14M20I0R0\config\Can.xdm
  - ..\CanIf_TS_T2D14M20I0R0\config\CanIf.xdm
  - ..\EcuM_TS_T2D14M20I0R0\config\EcuM.xdm
  - ..\Base_TS_T2D14M20I0R0\config\Base.xdm
  - ..\Resource_TS_T2D14M20I0R0\config\Resource.xdm
  - ..\Mcu_TS_T2D14M20I0R0\config\Mcu.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - ..\Can_TS_T2D14M20I0R0\autosar\Can.epd
  - ..\CanIf_TS_T2D14M20I0R0\autosar\CanIf.epd

  - ..\EcuM_TS_T2D14M20I0R0\autosar\EcuM.epd
  - ..\Mcu_TS_T2D14M20I0R0\autosar\Mcu.epd

- Code Generation Templates for Pre-Compile time configuration parameters:
  - ..\Can_TS_T2D14M20I0R0\generate_PC\include\Can_Cfg.h
  - ..\Can_TS_T2D14M20I0R0\generate_PC\include\Can_Cfg.c
  - ..\Can_TS_T2D14M20I0R0\generate_PC\Can_SourceClock.m
  - ..\Can_TS_T2D14M20I0R0\generate_PC\Can_VersionCheck_Inc.m
  - ..\Can_TS_T2D14M20I0R0\generate_PC\Can_VersionCheck_Src.m
  - ..\Can_TS_T2D14M20I0R0\generate_PC\Can_NotifyCheck_Src.m

- Code Generation Templates for Post-Build time configuration parameters:
  - ..\Can_TS_T2D14M20I0R0\generate_PB\include\Can_Cfg.h
  - ..\Can_TS_T2D14M20I0R0\generate_PB\src\ Can_PBcfg.c
  - ..\Can_TS_T2D14M20I0R0\generate_PB\Can_SourceClock.m
  - ..\Can_TS_T2D14M20I0R0\generate_PB\Can_VersionCheck_Src_PB.m
  - ..\Can_TS_T2D14M20I0R0\generate_PB\Can_NotifyCheck_Src_PB.m

Steps to generate the configuration:

1. Copy the module folders Can_TS_T2D14M20I0R0, CanIf_TS_T2D14M20I0R0, Base_TS_T2D14M20I0R0, Resource_TS_T2D14M20I0R0, EcuM_TS_T2D14M20I0R0, Mcu_TS_T2D14M20I0R0 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: mpc5634m_bga208, mpc5634m_qfp144, mpc5634m_qfp176.
- **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).
- **DEM** is required for signaling the production error detection (hardware failure, etc).

Chapter 4
Function Calls to Module

The CAN module shall be initialized by Can_Init(Can_configuration) service call during the start-up.

Note

The settings for pins Rx/Tx required for CAN bus communication are not related to Can driver or plugin configuration.

Note

After Can driver initialization the messages will not be handled for specific controller. API service Can_SetController_Mode(can_controller, CAN_T_START) shall be used for setting the CAN controller to running mode.

Note

At least one another CAN node have to be connected to the CAN bus in order to synchronize the CAN communication.

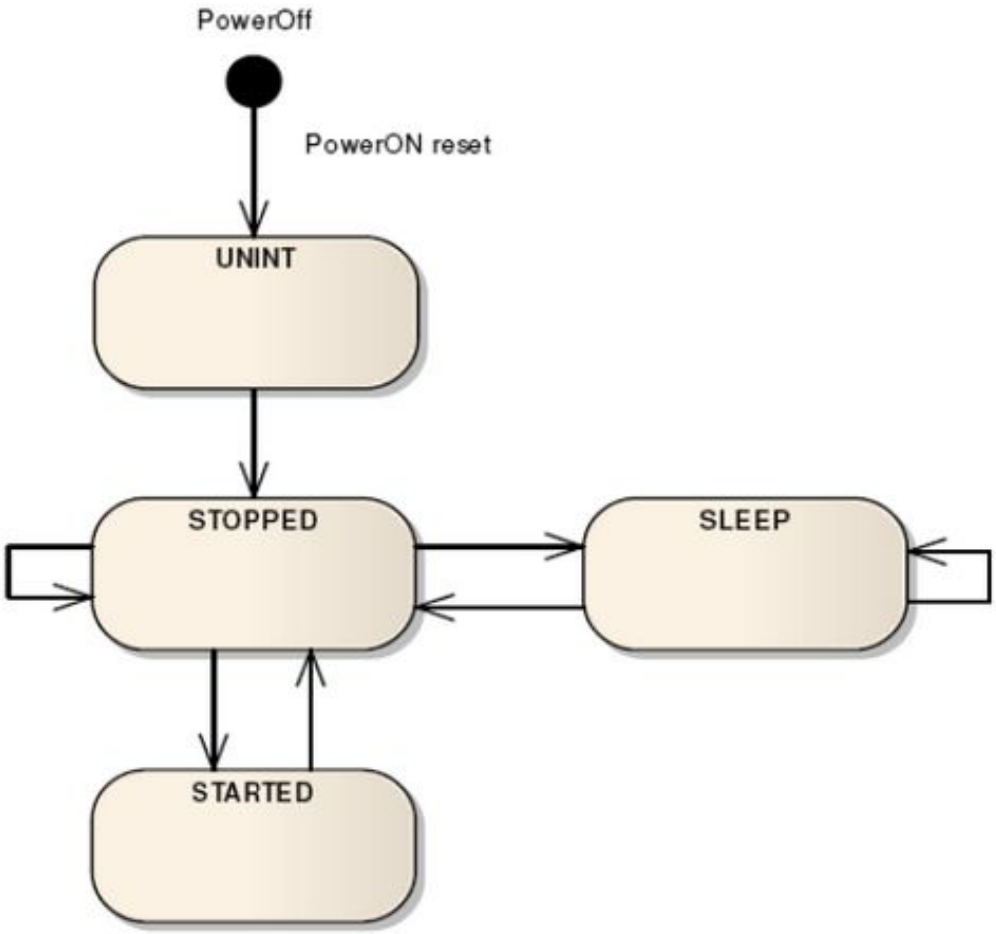

Figure 4-1. CAN Controller State Machine

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 FlexCAN module prior the CAN initialization.

4.2 Function Calls during Shutdown

The 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 4-2. 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

In the current implementation, CAN is using the services of Schedule Manager (SchM) for entering and exiting the critical regions. SchM implementation is done by the integrators of the MCAL using OS or non-OS services. For testing the CAN, stubs are used for SchM.

Some CAN driver global variables updates are performed by ISRs before calling the user notification functions. In order to avoid the scenario where an executing CAN driver function is preempted by a CAN ISR, which is modifying some of the global variables, some exclusive areas are defined.

The ISR critical regions must not block the other critical regions to avoid deadlocks. This is ensured by exiting the ISR critical region before calling the user notification functions.

The following critical regions are used in the CAN driver:

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.

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	Interrupt Service Routines Critical Regions(composed diagram)
CAN_EXCLUSIVE_AREA_00	X			X
CAN_EXCLUSIVE_AREA_01		X		X
CAN_EXCLUSIVE_AREA_02			X	X
Interrupt Service Routines Critical Regions (composed diagram)	X	X	X	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
Can_IsrFCA_ERR	153

Table continues on the next page...

Table 5-2. CAN ISRs (continued)

ISR Name	Hardware Interrupt Vector
Can_IsrFCA_BO	152
Reserved for Wakeup	154
Can_IsrFCA_MB_00	155
Can_IsrFCA_MB_01	156
Can_IsrFCA_MB_02	157
Can_IsrFCA_MB_03	158
Can_IsrFCA_MB_04	159
Can_IsrFCA_MB_05	160
Can_IsrFCA_MB_06	161
Can_IsrFCA_MB_07	162
Can_IsrFCA_MB_08	163
Can_IsrFCA_MB_09	164
Can_IsrFCA_MB_10	165
Can_IsrFCA_MB_11	166
Can_IsrFCA_MB_12	167
Can_IsrFCA_MB_13	168
Can_IsrFCA_MB_14	169
Can_IsrFCA_MB_15	170
Can_IsrFCA_MB_16-31	171
Can_IsrFCA_MB_32-63	172
Can_IsrFCC_ERR	174
Can_IsrFCC_BO	173
Reserved for Wakeup	175
Can_IsrFCC_MB_00	176
Can_IsrFCC_MB_01	177
Can_IsrFCC_MB_02	178
Can_IsrFCC_MB_03	179
Can_IsrFCC_MB_04	180
Can_IsrFCC_MB_05	181
Can_IsrFCC_MB_06	182
Can_IsrFCC_MB_07	183
Can_IsrFCC_MB_08	184
Can_IsrFCC_MB_09	185
Can_IsrFCC_MB_10	186
Can_IsrFCC_MB_11	187

Table continues on the next page...

Table 5-2. CAN ISRs (continued)

ISR Name	Hardware Interrupt Vector
Can_IsrFCC_MB_12	188
Can_IsrFCC_MB_13	189
Can_IsrFCC_MB_14	190
Can_IsrFCC_MB_15	191
Can_IsrFCC_MB_16-31	192

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_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(FC)` 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. This is used when IFLAG[4:7] bits are connected to the same INTC hardware vector.

`CAN_MB_RXOVER(FC)` expands `ISR(Can_IsrFC##FC##_Overf)` for Overflow RxFifo event. This is used when IFLAG[4:7] bits are connected to different INTC hardware vectors.

`CAN_MB_RXWARN(FC)` expands `ISR(Can_IsrFC##FC##_Warn)` for Warning RxFifo event. This is used when IFLAG[4:7] bits are connected to different INTC hardware vectors.

`CAN_MB_FRAV(FC)` expands `ISR(Can_IsrFC##FC##_FrameAv)` for Frame Available RxFifo event. This is used when IFLAG[4:7] bits are connected to different INTC hardware vectors.

`CAN_MB_ISRS(FC, Name, IdMin, IdMax)` expands `ISR(Can_IsrFC##FC##_##Name)` for a group Rx or Tx MB. Special for MBs from 0 to 63.

`CAN_MB_TXISRS(FC, Name, IdMin, IdMax)` expands `ISR(Can_IsrFC##FC##_##Name)` for a single Tx MB. Special for MBs from 0 to 63.

`CAN_MB_RXISRS(FC, Name, IdMin, IdMax)` expands `ISR(Can_IsrFC##FC##_##Name)` for a group of Rx MBs. Special for MBs from 0 to 63.

Note:MPC5634M has the IFLAG1[4:7] bits assigned to a single interrupt, then the solution is to used CAN_RXFIFO_EVENTS interrupt 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.

5.4 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.

Freescale 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.

5.5 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.
- **Dem:** This module is necessary for enabling reporting of production relevant error status. The API function used is Dem_ReportErrorStatus().
- **EcuM:** This module is necessary for a reference to the Wakeup source for this controller as defined in the ECU State Manager.
- **Resource:** Sub-Derivative model is selected from Resource configuration.

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_Wakeup( void )`
- `FUNC (void, CAN_CODE) Can_MainFunction_BusOff( void )`

These Autosar APIs are scheduled if these 4 events are configured to be in “Polling” mode by the following parameters:

- CanTxProcessing

```
#defineCAN_TXPOLL_SUPPORTED (STD_ON)
```

- CanRxProcessing

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

- CanWakeupProcessing

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

- CanBusoffProcessing

```
#define CAN_WAKEUPPOLL_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
```

- CanMainFunctionWakeupPeriod

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

- CanMainFunctionBusoffPeriod

```
#define CAN_MAINFUNCTION_PERIOD_BUSOFF (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

Not Applicable.

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 MemMap.h

For Post Build data:

#ifdef CAN_START_CONFIG_DATA_UNSPECIFIED

#undef CAN_START_CONFIG_DATA_UNSPECIFIED

#undef MEMMAP_ERROR

*/*Memory Section for Post Build Data to be defined here. Example given in the next line*/*

#pragma ghs section const=".pb CAN_cfg"

#endif

#ifdef CAN_STOP_CONFIG_DATA_UNSPECIFIED

#undef CAN_STOP_CONFIG_DATA_UNSPECIFIED

#undef MEMMAP_ERROR

*/*End of section to be mentioned here. Example given in the next line.*/*

#pragma ghs section

#endif

For Code:

#ifdef CAN_START_SEC_CODE

#undef CAN_START_SEC_CODE

#undef MEMMAP_ERROR

*/*Memory Section for Code to be defined here.*/*

#endif

#ifdef CAN_STOP_SEC_CODE

#undef CAN_STOP_SEC_CODE

#undef MEMMAP_ERROR

*/*End of section to be mentioned here*/*

#endif

For Variables:

#ifdef CAN_START_SEC_VAR_UNSPECIFIED

#undef CAN_START_SEC_VAR_UNSPECIFIED

#undef MEMMAP_ERROR

*/*Memory Section for Variables to be defined here.*/*

#endif

#ifdef CAN_STOP_SEC_VAR_UNSPECIFIED

#undef CAN_STOP_SEC_VAR_UNSPECIFIED

#undef MEMMAP_ERROR

*/*End of section to be mentioned here*/*

#endif

For Constant data:

#ifdef CAN_START_SEC_CONST_UNSPECIFIED

```
#undef CAN_START_SEC_CONST_UNSPECIFIED
#undef MEMMAP_ERROR
/*Memory Section for Constants to be defined here.*/
#endif

#ifdef CAN_STOP_SEC_CONST_UNSPECIFIED
#undef CAN_STOP_SEC_CONST_UNSPECIFIED
#undef MEMMAP_ERROR
/*End of section to be mentioned here*/
#endif
```

7.2 Linker Command File

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

Chapter 8
Configuration Parameters

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
	CanMainFunctionReadPeriod	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanMainFunctionWakeupPeriod	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanMainFunctionWritePeriod	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
	CanTimeoutDurationFactor	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
	CanOscillatorClockRef	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanWakeupSupport	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
CanController			
	CanHwChannel	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanControllerActivation	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanControllerBaudRate	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerId	Pre Compile parameter for all Variants of Configuration	Pre compile
	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
	CanControllerTimeQuanta	VariantPC or VariantPB	VariantPC or VariantPB
	CanControllerTimeQuanta_Alt	VariantPC or VariantPB	VariantPC or VariantPB
	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

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	CanWakeupProcessing	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
	CanTrippleSamplingEnable	VariantPC or VariantPB	VariantPC or VariantPB
	CanWakeUpSourceFilter	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
	CanErrorControllerNotification	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanWakeupSourceRef	Pre Compile parameter for all Variants of Configuration	Pre compile
	CanBccSupport	Pre Compile parameter for all Variants of Configuration	Pre compile
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
	CanTableIDType	VariantPC or VariantPB	VariantPC or VariantPB
	CanMBFilterMaskValue	VariantPC or VariantPB	VariantPC or VariantPB
CanController\CanFilterMask			

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	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

Chapter 9
Integration Steps

This section gives a brief overview of the steps needed for integrating CAN:

1. Generate the required CAN configuration.
2. Allocate proper memory sections in MemMap.h and linker command file.
3. Make sure all include files for compilation are available.
4. Map the ISRs to their vector locations.
5. Compile and build the CAN module with all the dependent modules.

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: IM14CANASR3.0R2.0.0
Rev. 1.1