Integration Manual

for MPC5634M FLS Driver

Document Number: IM14FLSASR3.0R2.0.0
Rev. 2.6

freescale™

Contents

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

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
2.5	03-Feb-2011	Gaetano Stabile	Update for Monaco automatic documentation
2.6	21-Dec-2011	Khanindra Jyoti Deka	Update for Monaco RTM 2.0.0

Chapter 2
Introduction

This integration manual describes the integration requirements for Fls Driver for MPC5634M microcontrollers.

2.1 Supported Derivatives

The software described in this document is intented 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
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
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEE	Flash EEPROM Emulation
FLS	Flash
XML	Extensible Markup Language

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	AUTOSAR 3.0Fls 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 Fls driver for Freescale SemiconductorMPC5634M . It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

The Fls driver files are compiled using

- GHS 5.2.4
- DIAB 5_8_0_02 wind00198363 20100511 123238
- CW Version 4.3 build 182

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 CW Compiler/Linker/Assembler Options

Table 3-1. 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
-ir	Include the debug information
-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
-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

Table continues on the next page...

Table 3-1. Compiler Options (continued)

Option	Description
-DMCAL_VERSION_CHECK	-D defines enable the cross check between the AutoSar component Version Numbers
-DMWERKS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the CWpreprocessor symbol.

Table 3-2. 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 3-3. 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.

3.1.2 DIAB Compiler/Linker/Assembler Options

Table 3-4. Compiler Options

Option	Description
-tPPCE200Z3VEG:simple	Sets target processor to PPCE200Z3, 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

Table continues on the next page...

Table 3-4. Compiler Options (continued)

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

Table 3-5. Assembler Options

Option	Description
-tPPCE200Z3VEN:simple	Selects target processor: PPCE200Z3, 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
-tPPCE200Z3VEN:simple	Selects target processor: PPCE200Z3, 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 GHS Compiler/Linker/Assembler Options

Table 3-7. Compiler Options

Option	Description
-cpu=ppc563xm	Selects target processor: ppc563xm
-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

Table continues on the next page...

Table 3-7. Compiler Options (continued)

Option	Description
-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
-DGHS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GHS preprocessor symbol.
-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 3-8. Assembler Options

Option	Description
-cpu=ppc563xm	Selects target processor: ppc563xm

Table 3-9. Linker Options

Option	Description
-cpu=ppc563xm	Selects target processor: ppc563xm
-nostartfiles	Do not use Start files.
-vle	Enables VLE code generation
-linker_warnings	Display linker warnings

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.
-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 Fls driver for MPC5634M 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.

Fls Files

- ..\Fls_TS_T2D14M20I0R0\src\Fls.c
- ..\Fls_TS_T2D14M20I0R0\src\Fls_Ac.c
- ..\Fls_TS_T2D14M20I0R0\include\Fls.h
- ..\Fls_TS_T2D14M20I0R0\include\Fls_Api.h
- ..\Fls_TS_T2D14M20I0R0\include\Fls_InternalTypes.h
- ..\Fls_TS_T2D14M20I0R0\include\Fls_Types.h
- ..\Fls_TS_T2D14M20I0R0\include\Fls_Version.h
- ..\Fls_TS_T2D14M20I0R0\include\Fls_FlashMem.h

- Fls_PBcfg.c - this file should be generated by the user using a configuration/ generation tool
- Fls_Cfg.h - this file should be generated by the user using a configuration/generation tool
- Note: Fls_Ac.c that implements Erase/Write access codes are implemented in VLE instruction code only.

Other includes files:

Files from MemIf folder:

- ..\MemIf_TS_T2D14M20I0R0\include\MemIf_Types.h

Files from Base common folder

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

Files from Dem folder:

- ..\Dem_TS_T2D14M20I0R0\include\Dem.h

Files from Det folder:

- ..\Det_TS_T2D14M20I0R0\include\Det.h

Files from SchM folder:

- ..\SchM_TS_T2D14M20I0R0\include\SchM_Fls.h

Files from MCI(Machine Check Interrupt):

- Exc_Types.h (only if FlsDsiHandlerApi=TRUE)

3.3 Setting up the Plug-ins

The Fls driver was designed to be configured by using the EB Tresos Studio (version Tresos 2010a.sr4 20100415-release2010a-sr4 or later.)

- VSMD (Vendor Specific Module Definition) file in EB tresos Studio XDM format: ..\Fls_TS_T2D14M20I0R0\config\Fls.xdm
- VSMD (Vendor Specific Module Definition) file in AUTOSAR compliant EPD format: ..\Fls_TS_T2D14M20I0R0\autosar\(one EPD file for each supported sub-derivative)
- Code Generation Templates for Post-Build time configuration parameters:
  - ..\Fls_TS_T2D14M20I0R0\generate\include\Fls_Cfg.h
  - ..\Fls_TS_T2D14M20I0R0\generate\src\Fls_PBcfg.c

Steps to generate the configuration:

1. Copy the module folders Fls_TS_T2D14M20I0R0 , Base_TS_T2D14M20I0R0 and Resource_ 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.

Chapter 4
Function calls to module

None.

4.1 Function Calls during Start-up

Fls shall be initialized during STARTUP phase of EcuM initialization. The API to be called for this is Fls_Init().

The MCU module should be initialized before the Fls is initialized.

If the FLS driver is used in user mode, be sure that the Flash memory controller registers are accessible and that accessed Flash memory partition is not protected. For more information please refer to the "Memory Protection Unit" and "Register Protection" chapters in the device reference manual.

The Flash memory physical sectors that are going to be modified by Fls driver (i.e. erase and write operations) have to be unlocked for a successful operation. It is also handled by Fls driver, but it should be configured using Fls_PhysicalSectorUnlock parameter for all configured blocks.

Note: if the unlock is not handled by Fls driver must be setup on an application level. Example of unlock code can be found in Fls driver UM document.

4.2 Function Calls during Shutdown

None.

4.3 Function Calls during Wake-up

None.

4.4 Implementing an Exception Handler in case of non correctable ECC error

When reading a FLASH location with a non correctable ECC error an IVOR exception is thrown.

On Z0 and Z6 core based platforms, a different IVOR is thrown according to the value of the bits ME and EE in the MSR (Machine Check Register) as show in the table below:

Table 4-1. IVOR selection on Z0 and Z6 core based platform

MSR[ME]	MSR[EE]	data/instruction	IVOR	Function to call
0	0	X	N/A	No IVOR thrown: Machine Checkstop state is entered (TO BE AVOIDED)
1	1	flash data read	IVOR2 (DSI)	Fls_DsiHandler
1	0	flash data read	IVOR1 (MCI)	Fls_MciHandler

On Z4 and Z7 core based platform, IVOR1 is thrown independently from MSR bits state as show in the table below.

Table 4-2. IVOR selection on Z4 and Z7 core based platform

MSR[ME]	MSR[EE]	data/instruction	IVOR	Function to call
0	0	X	N/A	Machine check stop not implemented (can cause infinite loop, TO BE AVOIDED)
1	0, 1	flash data read	IVOR1	Fls_DsiHandler

The Flash driver provide two API which can be called inside the user ECC error recovery handler (IVOR handler driver functions):

- **Fls_CompHandlerReturnType Fls_DsiHandler (Fls_ExceptionDetailsType *);**
- **Fls_CompHandlerReturnType Fls_MciHandler (Fls_ExceptionDetailsType *);**

For Z0 and Z6 both functions should be used as explained on above tables, while for Z4 and Z7 core only **Fls_DsiHandler()**must be used.

This API is available only if the configuration parameter **FlsDsiHandlerApi = true**;

The **Fls_ExceptionDetailsType** data structure contains some information about the details of Exception and in particular:

- a pointer to the statement that generated the ECC (**Fls_InstructionAddressType**);
- an address of the data that caused the error in ECC (**Fls_DataAddressType**);
- details on the type of exception (**uint32**).

IVOR Hander Driver functions examine the job executed by the driver and the data contained in the structure **Fls_ExceptionDetailsType** in particular:

- Check whether there is pending read or compare job;
- Check if exception syndrome Indicates DSI (or MCI) reason;
- Data address which cause the exception matches address currently accessed by pending flash read or flash compare job.

If these conditions are verified the IVOR handler driver functions return **FLS_HANDLED_SKIP** and set the job to failed value. This information can be retrieved using **Fls_GetJobResult()** which will return **MEMIF_JOB_FAILED** or, if the job error notification parameter is configured, the notification function will be called.

With these information the following recovery strategies may be implemented:

- skip the instruction that caused the error (**FLS_HANDLED_SKIP**);
- retry the execution (**FLS_HANDLED_RETRY**);
- perform a controlled shutdown of current activity (**FLS_HANDLES_STOP**);
- do nothing (infinite loop).

IVOR handler driver functions is not available for ISI (Instruction Storage Interrupt).

In addition to this information, a basic flow is depicted in the following figure:

XML CONTENT ERROR: The placement attribute value of the following image element is currently 'inline', but should be 'break' in order for the align attribute value of 'left' to take effect.

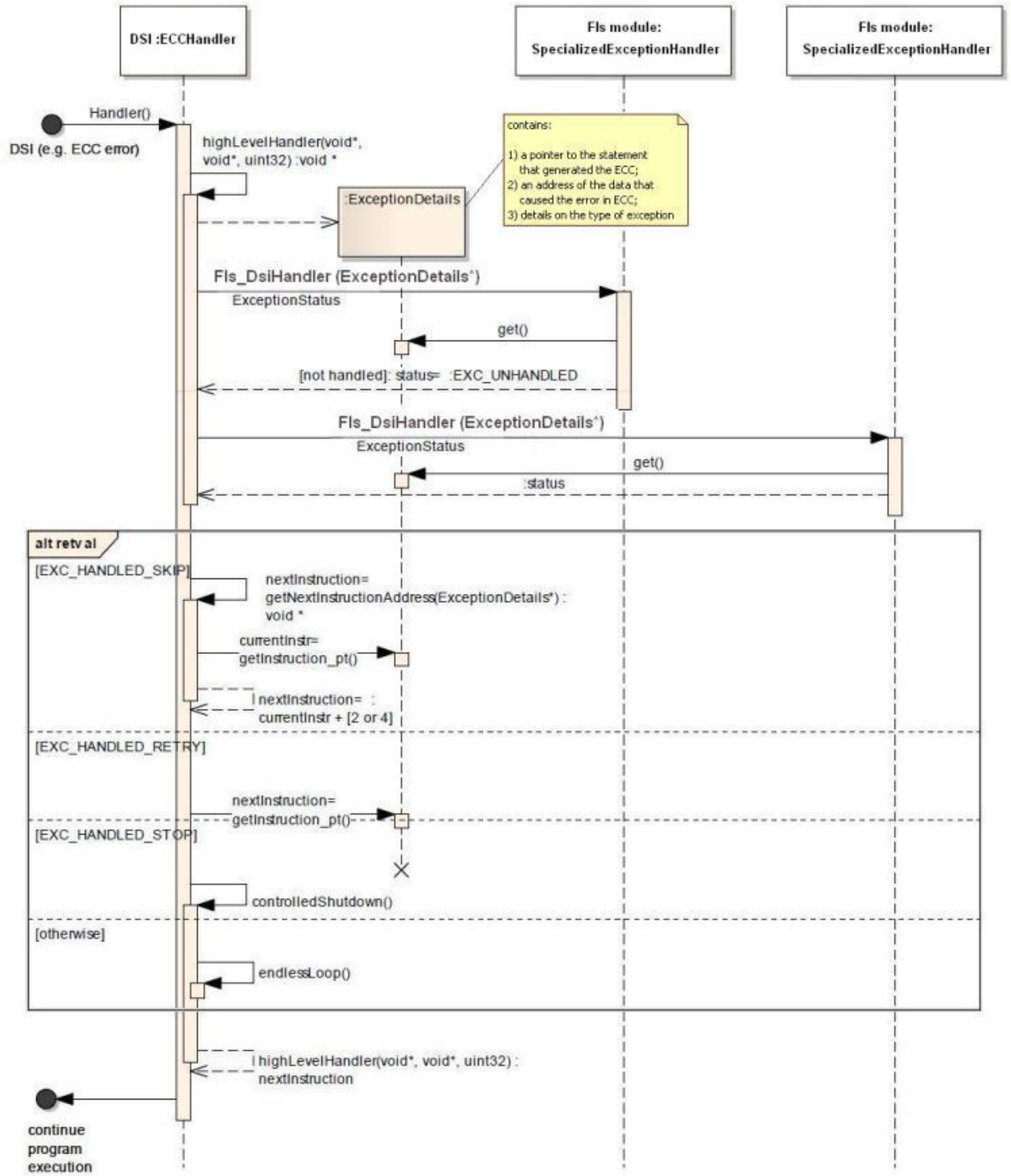

Figure 4-1. Example of ECC Handling

Chapter 5
Module requirements

5.1 Exclusive areas to be defined in BSW scheduler

Fls driver has two set of exclusive areas (EA):

FLS_EXCLUSIVE_AREA_00

FLS_EXCLUSIVE_AREA_01

FLS_EXCLUSIVE_AREA_02

FLS_EXCLUSIVE_AREA_03

This Exclusive areas (EA) are used to make Fls_MainFunction Thread Safe.

Note: These critical sections are the result of unclear Fls SWS document requirement number FLS215.

FLS215: “The FLS module’s flash access routines shall only disable interrupts and wait for the completion of the erase/write command if necessary (that is if it has to be ensured that no other code is executed in the meantime).”

On the contrary no BSW module is allowed directly control the global ECU interrupts, the Rte (and OS) module shall be used for this purposes. The actual implementation/ behavior of this critical section is left on the ECU integrator. It means in case no other executed code (task-s) access the ‘code’ or ‘constant data’ from affected Flash area (sector-s) which is being modified by current Fls job (erase or write operations) then the implementation could be ‘void’ (as there is no Flash read-while-write error possible). In all other cases you have to block the execution of the code (task-s) which would access this affected Flash area (sector-s).

Within these critical sections the Fls driver also access subset of the Flash PFC (Platform Flash Controller) registers (like PFCR0 or PFCR1) that can be potentially also used by Mcu driver, hence these are also protected against simultaneous access from both Fls and Mcu drivers.

FLS_EXCLUSIVE_AREA_10

FLS_EXCLUSIVE_AREA_11

FLS_EXCLUSIVE_AREA_12

FLS_EXCLUSIVE_AREA_13

Exclusive areas to make the Fls_Erase, Fls_Write, Fls_Read, and Fls_Compare functions thread safe (FLS_DEV_ERROR_DETECT==STD_ON). The exclusive areas above protect Fls internal job variables and thus all shall use (map to) same system resources in terms of OS objects.

5.2 Peripheral Hardware Requirements

The FLS driver uses/controls the "Flash Memory" MCU peripheral. For more details about peripheral and its structure refers to MCU reference manual.

5.3 ISR to configure within OS – dependencies

None.

5.4 ISR Macro

None.

5.5 Other AUTOSAR modules - dependencies

- **Base**
- **Dem:** This module is necessary for enabling reporting of production relevant error status. The API function used is Dem_ReportErrorStatus().

- **Det** 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.
- **SchM:** Exclusive areas implementations.
- **MemIf:** Memory Interface
- **Resource:** Sub-Derivative model is selected from Resource configuration.
- **DSI(Data Storage Interrupt) and MCI(Machine Check Interrupt):** (only if FlsDsiHandlerApi=true)

Chapter 6
Main API Requirements

6.1 Main functions calls within SchM module

Fls_MainFunction (call rate depends on target application, i.e. how fast the data needs to be read/written/compared in/to Flash memory).

6.2 API Requirements

None

6.3 Calls to Notification Functions, Callbacks, Callouts

The FLS driver provides notifications that are user configurable:

- FlsAcCallback (usually routed to Wdg module)
- FlsJobEndNotification (usually routed to Fee module)
- FlsJobErrorNotification (usually routed to Fee module)

6.4 Tips for FLS integration

Synchronous vs. Asynchronous write mode

Asynchronous write mode works in the way, that Fls_MainFunction() just schedules the HW write operation and does not wait for its completion. In the next Fls_MainFunction() it is checked if the write operation is finished. If yes (depends on how often the

Fls_MainFunction() is called), another write operation is scheduled. This process is repeated until all data is written. In this mode, FlsMaxWriteFastMode/ FlsMaxWriteNormalMode values are ignored, data is written just by FlsPageSize length.

When synchronous write mode is used, Fls_MainFunction() initializes write operation and also waits for its completion.

So the main differences between these two modes are in the time consumption and number of calls of the Fls_MainFunction(). The Fls_MainFunction() takes less time in asynchronous mode, but the whole write operation uses more Fls_MainFunction() executions.

Example1:

Table 6-1. Example: Synchronous vs. Asynchronous write mode

Fls_MainFunction()	Asynchronous mode	Synchronous mode
Time consumption (avrg/max)	15,7/ 96,8 µs	62,3/ 169,3 µs
Calls needed (avrg/max)	13/ 680	5/377

Example2:

FLS Max Write = 16 Byte, FLS Page Size = 8 Bytes and we are going to write 32 Bytes. In asynchronous write mode it will take at least 5 Fls_MainFunctions() (4 x 8 Bytes + 1 finish job check). In synchronous write mode it will take just 2 Fls_MainFunctions() to finish the write job (2x 16 bytes).

Possible values after interrupted HW write

Following value can be read from the flash when the HW write operation is interrupted:

1. Valid value (no ECC exception) and correct (the same as was intended to be written).

2. Valid value (no ECC exception) but incorrect (not the same as was intended to be written) – write operation was interrupted when ECC was not fully written. 1 bit error was improperly detected which lead to unwanted data correction. Example: we are going to write 00 C4 00 00, but the write operation is interrupted by reset. After reset we can see that the flash contains value 10 C4 00 00.

3. Always wrong value (stable ECC error).

4. Value with low margin – sometimes valid value is read (scenario 1 or 2) and sometimes ECC.

Note: One FLS HW write operation writes “FlsPageSize” Bytes from higher to lower addresses. It is possible to interrupt (e.g. by reset) this job after each byte.

Example: value 00 00 00 00 shall be written to the FLS. We can reset the write operation to obtain values: FF FF FF 00, FF FF 00 00 or FF 00 00 00 (of course not all of them can be seen because of ECC error. This is just an example to explain the write operation process).

Chapter 7
Memory Allocation

7.1 Sections to be defined in MemMap.h

For Post Build data:

```
#ifdef FLS_START_CONFIG_DATA_UNSPECIFIED
#undef FLS_START_CONFIG_DATA_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef FLS_STOP_CONFIG_DATA_UNSPECIFIED
#undef FLS_STOP_CONFIG_DATA_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
```

For Code:

```
#ifdef FLS_START_SEC_CODE
#undef FLS_START_SEC_CODE
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef FLS_STOP_SEC_CODE
#undef FLS_STOP_SEC_CODE
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
```

For Variables:

```
#ifdef FLS_START_SEC_VAR_UNSPECIFIED
#undef FLS_START_SEC_VAR_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef FLS_STOP_SEC_VAR_UNSPECIFIED
#undef FLS_STOP_SEC_VAR_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
```

```
#endif
```

For Constant data:

```
#ifdef FLS_START_SEC_CONST_UNSPECIFIED
#undef FLS_START_SEC_CONST_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef FLS_STOP_SEC_CONST_UNSPECIFIED
#undef FLS_STOP_SEC_CONST_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
```

For Ram Code:

```
#ifdef FLS_START_SEC_RAMCODE
#undef FLS_START_SEC_RAMCODE
#undef MEMMAP_ERROR
#pragma section CODE ".ramcode" far-absolute
#endif
#ifdef FLS_STOP_SEC_RAMCODE
#undef FLS_STOP_SEC_RAMCODE
#undef MEMMAP_ERROR
#pragma section CODE
#endif
```

7.2 Linker command file

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

Additionally if FlsAcLoadOnJobStart=true there have to be reserved space in RAM at locations defined by:

- FlsAcErase of space FlsAcSizeErase [in bytes]
- FlsAcWrite of space FlsAcSizeWrite [in bytes]

Using following configuration parameters

- FlsAcErasePointer
- FlsAcWritePointer

it is possible to use symbolic name instead of absolute addresses, but in this case the linker should define them.

Function Fls_InvalidRestore_Ram should be executed from RAM, FLS_START_SEC_RAMCODE and FLS_STOP_SEC_RAMCODE has been implemented for this porpouse in MemMap.h which define a new pragma section "**.ramcode**".

In the linker command file "**.ramcode**" section should be defined; then a separate routine should be implemented by the user to copy this section from flash to ram, for example into startup file.

Chapter 8
Configuration parameters considerations

Configuration parameter class for Autosar Fls 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 an PreCompile.

Table 8-1. Configuration Parameters

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
FlsGeneral			
	FlsAcLoadOnJobStart	VariantPostBuild	PreCompile
	FlsBaseAddress	VariantPostBuild	PreCompile
	FlsCancelApi	VariantPostBuild	PreCompile
	FlsCompareApi	VariantPostBuild	PreCompile
	FlsDevErrorDetect	VariantPostBuild	PreCompile
	FlsDriverIndex	VariantPostBuild	PreCompile
	FlsGetJobResultApi	VariantPostBuild	PreCompile
	FlsGetStatusApi	VariantPostBuild	PreCompile
	FlsTotalSize	VariantPostBuild	PreCompile
	FlsUseInterrupts	VariantPostBuild	PreCompile
	FlsVersionInfoApi	VariantPostBuild	PreCompile
	FlsDsiHandlerApi	VariantPostBuild	PreCompile
	FlsDsiHandlerInclude	VariantPostBuild	PreCompile
	FlsEraseBlankCheck	VariantPostBuild	PreCompile
	FlsWriteBlankCheck	VariantPostBuild	PreCompile
	FlsWriteVerifyCheck	VariantPostBuild	PreCompile

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	FlsMaxEraseBlankCheck	VariantPostBuild	PreCompile
FlsConfigSet			
	FlsAcErase	VariantPostBuild	PostBuild
	FlsAcWrite	VariantPostBuild	PostBuild
	FlsAcErasePointer	VariantPostBuild	PostBuild
	FlsAcWritePointer	VariantPostBuild	PostBuild
	FlsCallCycle	VariantPostBuild	PostBuild
	FlsAcCallback	VariantPostBuild	PostBuild
	FlsJobEndNotification	VariantPostBuild	PostBuild
	FlsJobErrorNotification	VariantPostBuild	PostBuild
	FlsMaxReadFastMode	VariantPostBuild	PostBuild
	FlsMaxReadNormalMode	VariantPostBuild	PostBuild
	FlsMaxWriteFastMode	VariantPostBuild	PostBuild
	FlsMaxWriteNormalMode	VariantPostBuild	PostBuild
	FlsProtection	VariantPostBuild	PostBuild
FlsSector			
	FlsPhysicalSectorUnlock	VariantPostBuild	PostBuild
	FlsPhysicalSector	VariantPostBuild	PostBuild
	FlsNumberOfSectors	VariantPostBuild	PostBuild
	FlsPageSize	VariantPostBuild	PostBuild
	FlsSectorSize	VariantPostBuild	PostBuild
	FlsSectorStartaddress	VariantPostBuild	PostBuild
	FlsSectorEraseAsynch	VariantPostBuild	PostBuild
	FlsPageWriteAsynch	VariantPostBuild	PostBuild

Chapter 9
Integration Steps

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

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

Chapter 10
ISR Reference

ISR functions exported by the Fls driver.

10.1 Software specification

The following sections contains driver software specifications.

10.1.1 Define Reference

Constants supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

10.1.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

10.1.3 Function Reference

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

10.1.4 Structs Reference

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

10.1.5 Types Reference

Types supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

10.1.6 Variables Reference

Variables supported by the driver are as per AUTOSAR Fls Driver software specification Version 3.0 .

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: IM14FLSASR3.0R2.0.0
Rev. 2.6