User's Manual

AUTOSAR MCAL R4.0.3

User's Manual

FLS Driver Component Ver.1.0.5

Embedded User's Manual

Target Device:
RH850/P1x

Renesas Electronics
www.renesas.com

Rev.1.02 May 2017

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Abbreviations and Acronyms

Abbreviation / Acronym	Description
ANSI	American National Standards Institute
ADC	Analog to Digital Converter
API	Application Programming Interface
AUTOSAR	AUTomotive Open System ARchitecture
BSW	Basic SoftWare
BSWMDT	Basic Software Module Description Template
BUS	Bidirectional Universal Switch
CPU	Central Processing Unit
CAN	Controller Area Network
DET/Det	Development Error Tracer
DEM/Dem	Diagnostic Event Manager
DIO	Digital Input Output
DMA	Direct Memory Access
DED	Double bit Error Detection
EEPROM	Electrically Erasable Programmable Read Only Memory
ECU	Electronic Control Unit
ECC	Error Correction Code
FACI	Flash Application Command Interface
FCU	Flash Control Unit
FLS	FLaSh Driver
GPT	General Purpose Timer
GNU	GNU's Not Unix
HW	HardWare
ID/Id	Identifier
IO	InOut
ICU	Input Capture Unit
I/O	Input/Output
ISR	Interrupt Service Routine
KB	Kilo Byte
LIN	Local Interconnect Network
MB	Mega Byte
MHz	Mega Hertz
MCU	Micro Controller Unit
MCAL	Microcontroller Abstraction Layer
NA	Not Applicable
OS	Operating System
PDF	Parameter definition file
PWM	Pulse Width Modulation
RAM	Random Access Memory

Abbreviation / Acronym	Description
ROM	Read Only Memory
R/W/RW	Read/Write/Read and Write
RTE	Run Time Environment
SCHM/SchM	Scheduler Manager
SCI	Serial Communications Interface
SPI	Serial Peripheral Interface
SED	Single bit Error Detection
SW	SoftWare
WDT	Watch Dog Timer

Definitions

Term	Represented by
Sl. No.	Serial Number

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The purpose of this document is to describe the information related to FLS Driver Component for Renesas P1x microcontrollers.

This document shall be used as reference by the users of FLS Driver Component. The system overview of complete AUTOSAR architecture is shown in the below Figure:

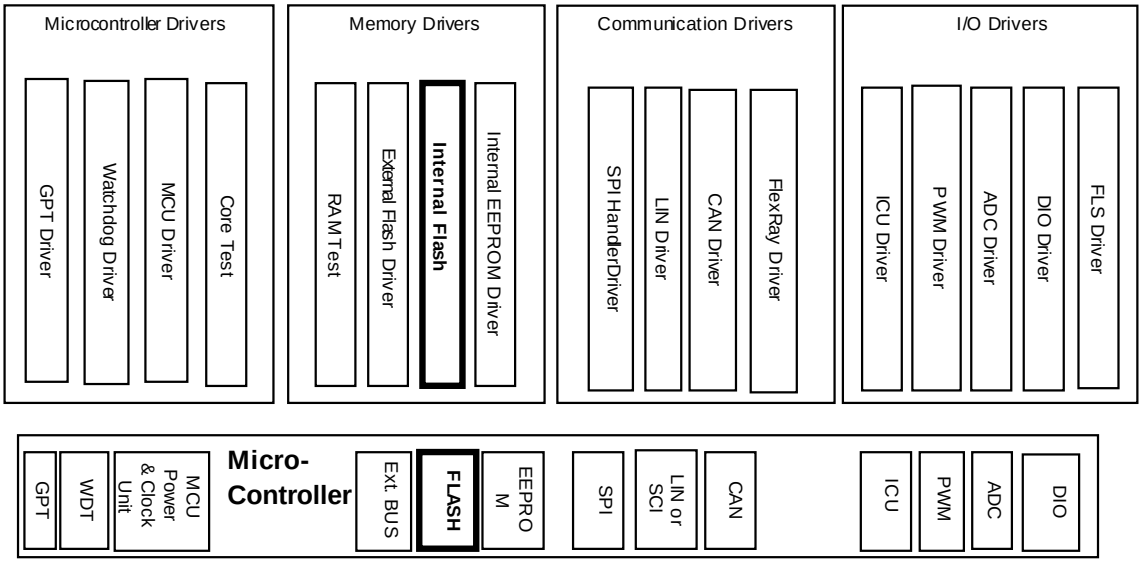

Figure 1-1 System Overview of FLS Driver Component in AUTOSAR MCAL Layer

The FLS Driver Component is part of BSW which is accessible by RTE. This RTE is a middle ware layer providing communication services for the application software and thereby it is possible to map the application software components between different ECUs.

The RTE provides the encapsulation of Hardware channels and basic services to the Application Software Components. So it is possible to map the Application Software-Components between different ECUs.

The Basic Software Modules are located below the RTE. The Basic Software itself is divided into the subgroups: System Services, Memory, Communication and IO Hardware-Abstraction. The Complex Drivers are also located below the RTE. Among others, the Operating System (OS), the Watchdog manager and the Diagnostic services are located in the System Services subgroup. The Memory subgroup contains modules to provide access to the non-volatile memories, namely Flash and EEPROM. Here the flash operation will be handled by flash driver.

On board Device Abstraction provides an interface to physical values for AUTOSAR software components. It abstracts the physical origin of signals (their paths to the hardware FLSs) and normalizes the signals with respect to their physical appearance. The microcontroller driver provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required from the upper layers.

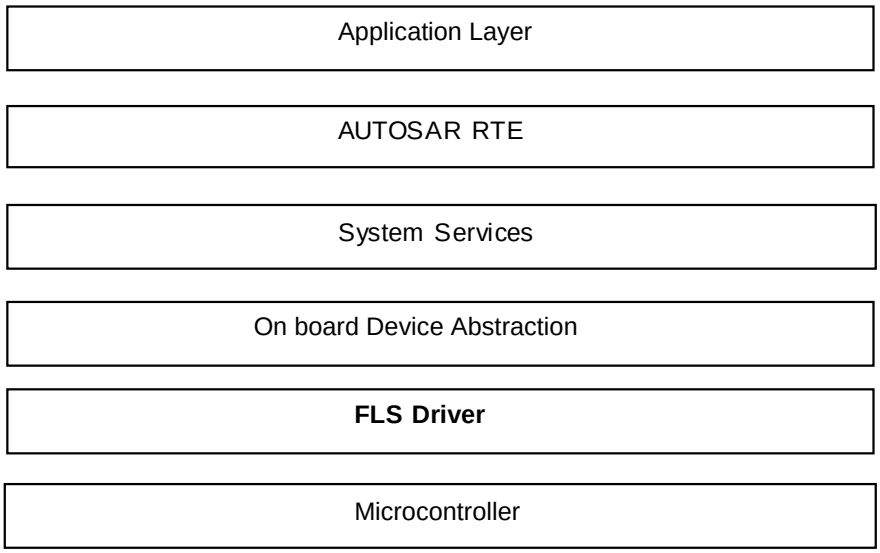

Figure 1-2 System Overview of AUTOSAR Architecture

The FLS application software components are located at the top and can gain access to the rest of the ECU and also to other ECUs only through the RTE. This RTE is a middleware layer providing communication services for the application software and thereby it is possible to map the application software components between different ECUs.

This FLS Software Module is located below the RTE. The FLS Component APIs are directly invoked by the application or RTE. The FLS Component is responsible for erase/write/read/compare data on the data flash memory.

The FLS component perform the activities like accessing and programming the on-chip data flash hardware.

The FLS Component layer comprises of API for erase/write data to on-chip data flash memory of the device. The FLS Component conforms to the AUTOSAR standard and is implemented mapping to the AUTOSAR FLS Software Specification.

The functional parameters of FLS software components are statically configurable to fit as far as possible to the real needs of each ECU.

1.1 Document Overview

The document has been segmented for easy reference. The table below provides user with an overview of the contents of each section:

Section	Contents
Section1 (Introduction)	This section provides an introduction and overview of FLS Driver Component.
Section 2 (Reference Documents)	This section lists the documents referred for developing this document.
Section 3 (Integration and Build Process)	This section explains the folder structure, Make file structure for FLS Driver Component. This section also explains about the Make file descriptions, Integration of FLS Driver Component with other components, building the FLS Driver Component along with a sample application.
Section 4 (Forethoughts)	This section provides brief information about the FLS Driver Component, the preconditions that should be known to the user before it is used, diagnostic channel, limit check feature, sample and hold feature, conversion time and stabilization time, DMA and ISR operations, data consistency details, deviation list and user mode and supervisor mode.
Section 5 (Architecture Details)	This section describes the layered architectural details of the FLS Driver Component.
Section 6 (Registers Details)	This section describes the register details of FLS Driver Component.
Section 7 (Interaction between The User And FLS Driver Component)	This section describes interaction of the FLS Driver Component with the upper layers.
Section 8 (FLS Driver Component Header And Source File Description)	This section provides information about the FLS Driver Component source files is mentioned. This section also contains the brief note on the tool generated output file.
Section 9 (Generation Tool Guide)	This section provides information on the FLS Driver Component Code Generation Tool.
Section 10 (Application Programming Interface)	This section explains all the APIs provided by the FLS Driver Component.
Section 11 (Development And Production Errors)	This section lists the DET and DEM errors.
Section 12 (Memory Organization)	This section provides the typical memory organization, which must be met for proper functioning of component.
Section 13 (P1M Specific Information)	This section provides the P1M Specific Information.
Section 14 (Release Details)	This section provides release details with version name and base version.

Chapter 2 Reference Documents

Sl. No.	Title	Version
1.	AUTOSAR_SWS_FlashDriver.pdf	3.2.0
2.	r01uh0436ej0130_rh850p1x.pdf	1.30
3.	AUTOSAR_SWS_CompilerAbstraction.pdf	3.2.0
4.	AUTOSAR_SWS_MemoryMapping.pdf	1.4.0
5.	AUTOSAR_SWS_PlatformTypes.pdf	2.5.0
6.	AUTOSAR BUGZILLA (http://www.autosar.org/bugzilla) Note: AUTOSAR BUGZILLA is a database, which contains concerns raised against information present in AUTOSAR Specifications.	-

Chapter 3 Integration and Build Process

In this section the folder structure of the FLS Driver Component is explained. Description of the Make files along with samples is provided in this section.

Remark The details about the C Source and Header files that are generated by the FLS Driver Generation Tool are mentioned in the "R20UT3711EJ0101-AUTOSAR.pdf".

3.1. FLS Driver Component Make file

The Make file provided with the FLS Driver Component consists of the GNU Make compatible script to build the FLS Driver Component in case of any change in the configuration. This can be used in the upper level Make file (of the application) to link and build the final application executable.

3.1.1. Folder Structure

The files are organized in the following folders:

Remark Trailing slash '\' at the end indicates a folder

X1X\common_platform\modules\fls\src
\Fls.c
\Fls_Internal.c
\Fls_Ram.c
\Fls_Version.c
\Fls_Private_Fcu.c
\Fls_Irq.c

X1X\common_platform\modules\fls\include
\Fls.h
\Fls_Debug.h
\Fls_Internal.h
\Fls_Private_Fcu.h
\Fls_PBTypes.h
\Fls_Ram.h
\Fls_Types.h
\Fls_Version.h
\Fls_Irq.h
\Fls_RegWrite.h

X1X\P1x\modules\fls\sample_application\<SubVariant>\make\ghs
\App_FLS_<SubVariant>_Sample.mak

X1X\P1x\modules\fls\sample_application\<SubVariant>\obj\<compiler>

(Note: For example, compiler can be ghs.)

X1X\common_platform\modules\fls\generator\Fls_X1x.dll

tools/RUCG/RUCG.exe

X1X\P1x\common_family\generator\Global_Application_P1x.trxml

\Sample_Application_P1x.trxml

\P1x_translation.h

X1X\P1x\modules\fls\generator
\R403_FLS_P1x_BSWMDT.arxml

X1X\P1x\modules\fls\user_manual
(User manuals will be available in this folder)

Notes:

1. <Compiler> can be ghs.
2. <Device_name> can be 701304, 701305, 701310, 701311, 701312, 701313, 701314, 701315, 701318, 701319, 701320, 701321, 701322 or 701323.
3. <SubVariant> can be P1M.
4. <AUTOSAR_version> can be 4.0.3.

Chapter 4 Forethoughts

4.1. General

Following information will aid the user to use the FLS Driver Component software efficiently:

FLS General

- The FLS Driver Component supports Data Flash access only. Code Flash access is out of scope. User application shall not program Code Flash in the application mode. Code Flash shall only be programmed in safe environment in the boot mode.
- The start-up code is ECU specific. FLS Driver Component does not implement the start-up code.
- Example code mentioned in this document shall be taken only as a reference for implementation.
- All development errors will be reported to DET by using the API Det_ReportError provided by DET.
- All production errors will be reported to DEM by using the API Dem_ReportErrorStatus provided by DEM.
- The FLS Driver Component developed supports only on-chip ROM and no external devices are considered. Hence the parameters related to external devices are ignored by the Generation Tool.
- The FLS Driver Component does not provide functionalities for setting of protection flags, boot cluster size, swapping of boot block and flashing of boot block and they are out of scope for FLS Driver Component implementations.
- Maximum value of 'FlsMaxReadNormalMode' parameter specifies the size of a temporary buffer in RAM which is used when Fls_ReadImmediate and Fls_Compare APIs are called. The resulting RAM consumption has to be considered.
- The length of the data that has to be programmed on to the flash should be in multiples of flash page. The FLS Driver Component does not pad bytes if the length is not in multiples of flash page. It is the responsibility of the application to pad bytes such that the length of the data is in multiples of flash page.
- Erase, Write, Read and Blank check jobs are initiated within the corresponding APIs itself. Fls_MainFunction API shall act as a checker function and it shall check whether the job is completed and initiate the next round of job cycle if the job is not completed.
- The normal write verification using the direct memory read access is performed when DET is enabled.
- The FLS Driver Component can invoke user configurable call-back notification functions. However, the implementation of the call back functions is the responsibility of the upper layer.
- The parameter 'FlsCallCycle' shall be used for timeout implementation. The Erase, Write and BlankCheck timeout count values shall be generated based on FlsCallCycle and hardware specific atomic operations' time ('FlsEraseTime', 'FlsWriteTime' and 'FlsBlankCheckTime'). To report timeout, 'FlsTimeOutMonitoring' parameter needs to be configured as TRUE'. In case if the parameter 'FlsDevErrorDetect' is also enabled, time out DET shall be reported. The 'FlsCallCycle' parameter shall be configured by the user correctly. Incorrect value may lead to reporting of timeout DET by Fls_MainFunction.
- There are two possible errors that can be detected by ECC are Single-bit errors (SED) and Double-bit errors (DED). The ECC error notification feature is incorporated in Read functionality only. So whenever the read is

initiated this feature will be enabled always and only notifying to the upper layer happens via configurable notification functions. The configuration of single bit and double bit error notification function parameters are user selectable. The error notification functions for both single bit and double bit ECC error report are configurable with parameters from configuration.
The parameters are:
FlsEccSedNotification: This parameter mapped to Single-bit error (SED) notification routine provided by some upper layer module.
FlsEccDedNotification: This parameter mapped to Double-bit error (DED) notification routine provided by some upper layer module. The Double bit error is reported to DEM in addition to notification functions.

- The file Interrupt_VectorTable.c provided is just a Demo and not all interrupts will be mapped in this file. So the user has to update the Interrupt_VectorTable.c as per his configuration.
- The accesses to HW registers is possible only in the low level driver layer. The user shall never write or read directly from any register, but shall use the AUTOSAR standard API provided by the MCAL.
- FlsTimeOutCountValue: This parameter specifies the time out count required for erase, write and blank check operations during interrupt mode.Incorrect configuration of this parameter shall lead to erroneous operations of FLS Driver.
- FlsLoopCount: This parameter is used to avoid the risk of endless loops in FLS driver. The loop count can be minimum 32 to maximum 255.
- The configurations provided for fast mode write operation is ignored by the Generation Tool and only configurations for normal mode operations and fast mode read operation are accepted.
- Fls_SetMode API sets the flash driver operation mode (FAST Mode/SLOW Mode) for read operation. This API allows the user to read more number of bytes during run time if in case the default mode is configured as 'MEMIF_MODE_FAST'. Fls_SetMode API is not applicable for Erase/Write/Blank Check operations, because underlying hardware does not support it.

FLS Initialization

- Fls_Init API shall enable the flash memory erase/write protection settings if it is supported by hardware. Before the flash operation protection shall be disabled and after the completion of job, protection shall be again enabled.
- During activation of flash environment (in Fls_Init), the access to Code flash is not possible. Hence the user should ensure that all the application and supporting components code that needs to be executed during flash operation need to locate in RAM.
- The device supports servicing of interrupts during self-programming. During activation of flash environment (in Fls_Init), the interrupt vector address in the flash will not be available. The interrupt vectors can be relocated to RAM during flash programming. For details please refer *Exception Handling Address Switching Function* in the according device CPU user manual.

FLS Schedule operation

- The FLS Driver Component's job processing function (Fls_MainFunction) is a polled function.
- In a single cycle of Fls_MainFunction API, the maximum number of bytes processed for the fast read command and normal read depends on the configuration of parameters 'FlsMaxReadNormalMode' (if default mode is MEMIF_MODE_SLOW) and 'FlsMaxReadFastMode' (if default mode is MEMIF_MODE_FAST).

- In a single cycle of Fls_MainFunction call, FLS driver performs write operation for 4 bytes, or blank check operation for 4 bytes, or erase operation for 64 bytes.

FLS Erase Operation

- The Fls_Erase API computes the sectors that need to be erased based on the provided target address and length. When DET is enabled the error will be reported if the length of the bytes to be erased is not in multiples of flash sector size.

FLS Read Operation

- Data Flash Memory Read Cycle Setting Register (EEPRDCYCL) is used to specify the number of wait cycles to be inserted when reading the data in the data flash. The initial value of the register is taken by default. If required user application shall set this register as per P1M device user manual.
- Blank Check operation is done implicitly when performing Read operation

FLS Blank Check Operation

- The processing of blank check operation is applicable for Data flash only.

FLS Fast Read (Read Immediate) Operation

- The functional behavior of FLS driver when calling Fls_ReadImmediate API will be the same as calling Fls_Read API except that blank check is excluded.
- Blank check is time consuming and is not mandatory for reading an already programmed flash area. Fls_ReadImmediate API can be used to perform fast read operation without blank check when the user is sure that the area to be read is already written with content.

Suspend and Resume operation

- Fls_Suspend API is used to suspend an ongoing flash operation in order to do other flash operations. Only erase and write operations are suspend able.
- Fls_Resume can only be used to resume a suspended flash operation. Only erase and write operations are resume able.
- It is not always possible to suspend.
  E.g.: Any operation ► suspend ► suspend – is not possible.
  Write or Erase ► suspend ► Erase operation – is not possible
  Write operation ► suspend ► other Write operation – is not possible
  Any operation ► suspend ► other operation ► suspend–isn't possible

FLS Timeout Monitoring

- The configuration parameter FlsTimeoutMonitoring in the FlsGeneral container can be used to enable/disable the timeout supervision for FLS driver independent of DET settings.
- Only when FlsTimeoutMonitoring is set to TRUE and DET is switched ON, a DET error FLS_E_TIMEOUT will be reported in case of detection of a timeout error.
- In order to perform timeout monitoring/supervision on flash operations, the following configuration parameters should be used properly according to use-cases.
  - In the polling mode of FLS, the parameter FlsCallCycle shall be configured to specify the cycle time of calls of the FLS main function (in

seconds). The timeout count values are calculated internally based on the CPU frequency for the respective flash operations, i.e., erase, write, blank check, etc.

- In the interrupt mode of FLS, the parameter FlsTimeOutCountValue shall be configured to directly specify the timeout count value required for erase, write and blank check operations.
- Fls_MainFunction is crucial for timeout supervision. The call frequency of Fls_MainFunction shall be handled properly in the upper layer software to be in line with the FLS module configuration.

Note: since read, read immediate, compare operations are not supported in FLS interrupt mode, only the parameter FlsCallCycle is used to calculate timeout count values for them irrespective of interrupt or polling mode. For write, erase, blank check operations, FlsCallCycle is used in the polling mode of FLS, while FlsTimeOutCountValue is used in the interrupt mode of FLS.

In FlsGeneral container the configuration parameter FlsLoopCount is used to avoid the risk of endless loops in the FLS driver. FlsLoopCount is always used in the implementation, hence it is not dependent on the parameter FlsTimeoutMonitoring

4.2. Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the FLS Driver Component:

FLS General

- The user should ensure that FLS Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Correct frequency configuration is essential for Flash programming quality and stability. Wrong configuration could lead to loss of data retention or Flash operation fail. The limits for CPU frequency are device dependent. Please refer to the respective device user manuals for correct range. If the CPU frequency is a fractional value, round up the value to the nearest integer. Do not change power mode (voltage or CPU clock) while FLS is performing a Data Flash operation. If power mode must change the user can wait until operations are no longer busy or cancel the ongoing operation and reinitialize the FLS module with proper CPU frequency value.
- In case of Flash modification operation (Erase/Write) interruption due to e.g. power failure, reset etc., the electrical conditions of the affected Flash range (Flash block on erase, Flash write unit on Write) get undefined. It is impossible to give a statement on the read value after the interruption. Thus, the resulting read value is not reliable; the electrical margin for the specified data retention may not be given. In such case, erase and re-write the affected Flash block(s) to ensure data integrity and retention.
- It is not possible to modify the Code Flash in parallel to a modification of the Data Flash or vice versa due to shared hardware resources.
- Data Flash blocks are aligned to 64 bytes and Data Flash words are aligned to 4 bytes. RH850 devices also add alignment restrictions for types larger than 8 bits. Please refer to device hardware manual for details.
- Validation of input parameters is done only when the static configuration parameter FLS_DEV_ERROR_DETECT is enabled. Application should ensure that the right parameters are passed while invoking the APIs when FLS_DEV_ERROR_DETECT is disabled.
- A mismatch in the version numbers will result in compilation error. Ensure that the correct versions of the header and the source files are used.
- The files Fls_Cfg.h, Fls_Cbk.h and Fls_PBcfg.c generated using FLS Generation Tool have to be linked along with FLS Driver Component

source files.

- Values for production code Event Ids should be assigned externally by the configuration of the DEM.
- Calling FLS functions, especially Cancel/Suspend/Resume/MainFunction APIs by a higher priority ISR must be prevented by upper layer to avoid possible re-entrancy issue.
- Interrupt mode supports Erase, Write, and Blank Check operations only.
- Cancel and suspend/resume operations are not allowed in case of two instances of FLS Driver Component as the effect is not evaluated.
- All functions are not re-entrant. So, re-entrant calls of any not re-entrant function must be avoided.
- User have the responsibility to enable or disable the critical protection using the parameter FlsCriticalSectionProtection. By enabling parameter FlsCriticalSectionProtection, Microcontroller HW registers which suffer from concurrent access by multiple tasks are protected.
- To use the FLS driver, the FCU firmware shall be stored and executed from in FCURAM. And the software that reads the FCU firmware area shall be placed in internal RAM.
- Due to these preconditions, the following four private functions used in Fls_Init() shall be placed in internal RAM at link time using linker derivative in order to perform FLS driver initialization properly.

  Fls_FcuCopytoRam() - size 118 Bytes,
  Fls_FcuSwitchBFlash() - size 46 Bytes,
  Fls_FcuClearCache() - size 76 Bytes,
  Fls_FcuGetFWParam() - size 46 Bytes.

  For the integration, the FLS driver code comes under the memory sections FLS_START_SEC_PRIVATERAM_CODE / FLS_STOP_SEC_PRIVATERAM_CODE shall be placed in internal RAM for the execution.

  Please refer to the FLS sample application linker script for more details.
- As the atomic time-out monitoring in POLLING mode depends FlsCallCycle parameter, which is the schedule time of Fls_MainFunction call, the user should take care the configuration of that parameter as per the intended cycle time of Fls_MainFunction. i.e: If there are any scenarios where Fls_MainFunction may be called with different cycle time in the same application, the user shall configure FlsCallCycle with the least cycle time, otherwise it may hit early time-out error.

FLS Initialization

- Fls_Init function temporarily disables Code Flash. During this time, since the Code Flash is not available, the FLS code shall be executed from internal RAM. Please ensure that: (1) User application code execution is done from other locations than Code Flash (e.g. internal RAM). (2) No access to Code Flash is allowed, e.g. by jump to interrupt/exception functions, direct Code Flash read/execution from the CPU, DMA accesses to Code Flash.
- The FLS Driver Component needs to be initialized by calling Fls_Init before calling any other Fls functions.
- Fls_Init shall do verification of ECC control registers, so as to ensure ECC 1-bit error detection and correction, ECC 2-bit error detection are enabled for data flash before initialization of FCU. If the user configurable ECC check for FACI is enabled and if the verification of FACI ECC register fails, DEM error FLS_E_ECC_FAILED shall be reported.
- Fls_Init function temporarily disables Code Flash in order to copy the FCU firmware and initialize FCU properly. Afterwards, Code Flash will be re-enabled. When failure occurs during re-enabling Code Flash, a call-back function is invoked to notify the upper layer software. This call-back function Fls_CallSwitchBFlashErrorNotification() is declared in "Fls.h" as

user interface, and shall be implemented in the upper layer software. Countermeasures (e.g. hardware reset) against such failure shall be considered in the call-back function implementation. And this function Fls_CallSwitchBFlashErrorNotification() must be properly mapped in RAM and executed out of RAM in case of failure.

FLS Schedule operation

- The Fls_MainFunction should be invoked regularly by the Basic Scheduler. Though not specified by AUTOSAR, calling Fls_MainFunction by polling mechanism is also possible. Ensure that the FLS Driver Component is initialized before enabling the invocation of this scheduled function to avoid reporting of a DET error when enabled.

FLS Write Operation

- Due to RV40 Flash technology, hardware will implicitly reject the write operation if the target Flash cells are not blank (a kind of "overwriting guard"). Writing to non-blank Flash cells will result in write error.
- Writing the same area more than once is prohibited. To write again the flash memory area where data has already been written to, user shall erase the corresponding area in advance.

FLS Read Operation

- Data Flash on RH850 devices is made with differential cells for storage. This means that reading erased but non-programmed Data Flash areas directly (bypassing FLS) will produce undefined data with a tendency to the previously written data, and it will most probably cause ECC error exceptions. To avoid this exceptions, use FLS read APIs.
- Fls_ReadImmediate API should not be used to read blank cells. User application shall handle the errors associated with blank cell read using Fls_ReadImmediate API.

FLS Blank Check Operation

- A blank check pass does not confirm that it is possible to write to this word (4 Bytes). Also partly written/erased words may have a blank check pass but write is not allowed under this condition. A blank check fail does not confirm a stable read value. Even though parts of a word are at least partly written, random read data are still possible, so are ECC error indications for single error corrections and double error detection.
- Due to the above shown limitations the information which can be given by Fls_BlankCheck, either passing or failing, is limited. It cannot be used to determine the current state of a flash cell in a meaning full way without additional information obtained by other means. The blank check should only be used to confirm or check some flow status but should not be used to determine if a flash cell can be read or written. FLS055 from AUTOSAR Specification of Flash Driver are not fulfilled here because blank check itself is not able to identify erasure state of flash cell which is ready for write operation. Please refer to application note document "RV40F DataFlash Usage" for more details about blank check and usage hints

FLS Cancel Operation

- If a cancel request is accepted, during an on-going write or erase operation and a previous operation is already suspended, then both operations will be cancelled.

FLS Suspend operation

- Suspend operation shall not be performed in between atomic operations of the job. i.e., in between 64 bytes of erase and 4 bytes of write, suspension is not possible. The job can be suspended only after completion of one atomic operation.
- When an erase job is suspended, calling a write job at the same address of that of erase job and then resuming the previously suspended erase job shall report DET indicating failure of erase verification.

4.3. Data Consistency

To support FLS the reentrancy and interrupt services, the FLS Software component will ensure the data consistency while accessing their own RAM storage or hardware registers
The FLS module will use below macro for respective higher and lower version.

#define FLS_ENTER_CRITICAL_SECTION (Exclusive_Area)
SchM_Enter_Fls_##Exclusive_Area ()

#define FLS_EXIT_CRITICAL_SECTION (Exclusive_Area)
SchM_Exit_Fls_##Exclusive_Area ()

The following exclusive areas along with scheduler services are used to provide data integrity and register protection for shared resources:

- FLS_DRIVERSTATE_DATA_PROTECTION
- FLS_REGISTER_PROTECTION
- FLS_CODE_FLASH_DISABLED

Note: The data buffer provided by the application will not be validated for data consistency. It would be the responsibility of the application to ensure consistency of the flash data during flash read and write operations.

Table 4-1 FLS Driver Protected Resources List

API Name	Exclusive Area Type	Protected Resources
Fls_Init	FLS_REGISTER_PROTECTION	HW Registers: FRAMMCR FCURAME FPCKAR
	FLS_CODE_FLASH_DISABLED	Firmware storage area switching is protected
Fls_Erase	FLS_REGISTER_PROTECTION	HW Registers: FSADDR FEADDR IMR
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected

API Name	Exclusive Area Type	Protected Resources
Fls_Write	FLS_REGISTER_PROTECTION	HW Registers: FSADDR FEADDR IMR
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected
Fls_MainFunction	FLS_REGISTER_PROTECTION	HW Registers: DFERSTC DFERSTR DFERRINT IMR
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected
Fls_Resume	FLS_REGISTER_PROTECTION	HW Registers: DFERSTC DFERSTR DFERRINT IMR
Fls_Read	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected
Fls_Compare	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during compare operation
Fls_Cancel	FLS_REGISTER_PROTECTION	HW Registers: IMR
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during cancel operation
Fls_BlankCheck	FLS_REGISTER_PROTECTION	HW registers: IMR
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during blank check operation
Fls_ReadImmediate	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during read immediate operation
FLS_FLENDNM_ISR	FLS_REGISTER_PROTECTION	IMR

Note: The highest measured duration of a critical section is 128.537 micro seconds, measured for **Fls_Init** API.

4.4. Deviation List

Table 4-2 FLS Driver Component Deviation List

Sl. No.	Description	AUTOSAR Bugzilla
1.	The fast mode parameter 'FlsMaxWriteFastMode' of the container 'FlsConfigSet is unused.	-
2.	The parameters 'FlsAcLoadOnJobStart' and 'FlsUseInterrupts' of the container 'FlsGeneral' is unused.	-
3.	The flash access routines are not placed into a separate C-module like 'Fls_ac.c'.	-
4.	**FLS140** and **FLS141** are not fulfilled, because the FLS module does not load the flash access code for erase/write operation to the location in RAM on job start.	-
5.	The parameters 'FlsProtection', FlsAcWrite' and 'FlsAcErase' of the container 'FlsConfigSet' are unused.	-
6.	The parameters 'FlsAcLocationErase', 'FlsAcLocationWrite', 'FlsAcSizeErase' and 'FlsAcSizeWrite' of the container 'FlsPublishedInformation' are unused.	-
7.	The component will support only the on-chip flash memory. External flash is not in the scope of this implementation.	-
8.	**FLS272, FLS359, FLS360 and FLS361** from AUTOSAR Specification of Flash Driver are not fulfilled here because timeout monitoring can be configured independent of DET setting. However only when both timeout monitoring and DET are enabled, FLS_E_TIMEOUT will be reported in case of detected timeout error.	-
9.	The timeout monitoring can be configured independent of DET setting in FLS. **FLS272**, **FLS359**, **FLS360**, **FLS361** can only be fulfilled, when both timeout monitoring and DET are enabled, i.e., FLS_E_TIMEOUT will be reported for the respective flash operations in case of detected timeout error.	-
10.	**FLS201_Conf** from AUTOSAR Specification of Flash Driver is not fulfilled here because FlsSectorList is limited to one sector with fixed sector size. User shall not configure multiple sectors. Since data flash is a monolithic on-chip NV memory with homogeneous block size, it is not required to have multiple sectors with the same sector sizes. Important is that FLS driver shall support possible usage of "user pool" (private data flash area that cannot be accessed by FLS driver). This can be done by proper configuration of FlsSectorStartaddress and FlsNumberOfSectors.	-

4.5. User mode and supervisor mode

The below table specifies the APIs which can run in user mode, supervisor mode or both modes

Table 4-3 User mode and Supervisor mode details

Sl. No	API Name	User Mode	Supervisor Mode	Known limitation in User mode
1	Fls_Init	-	x	The Fls_Init is failing in User mode. This is because inside Fls_Init function STSR instruction (to store contents of system register) is called for storing contents of ICCTRL (instruction cache control) to system register. Since the ICCTRL have the access permission in only supervisor mode, Fls_Init fails in user mode.
2	Fls_Read	x	x	-
3	Fls_SetMode	x	x	-
4	Fls_Write	x	x	-
5	Fls_Cancel	x	x	-
6	Fls_GetStatus	x	x	-
7	Fls_GetJobResult	x	x	-
8	Fls_Erase	x	x	-
9	Fls_Compare	x	x	-
10	Fls_GetVersionInfo	x	x	-
11	Fls_MainFunction	x	x	-
12	Fls_BlankCheck	x	x	-
13	Fls_ReadImmediate	x	x	-
14	Fls_Suspend	x	x	-
15	Fls_Resume	x	x	-

Note: Implementation of critical section is not dependent on MCAL. Hence critical section is not considered to the entries for user mode in the above table.

Chapter 5 Architecture Details

The FLS Software architecture is shown in the following figure. The FLS user shall directly use the APIs to configure and execute the FLS conversions:

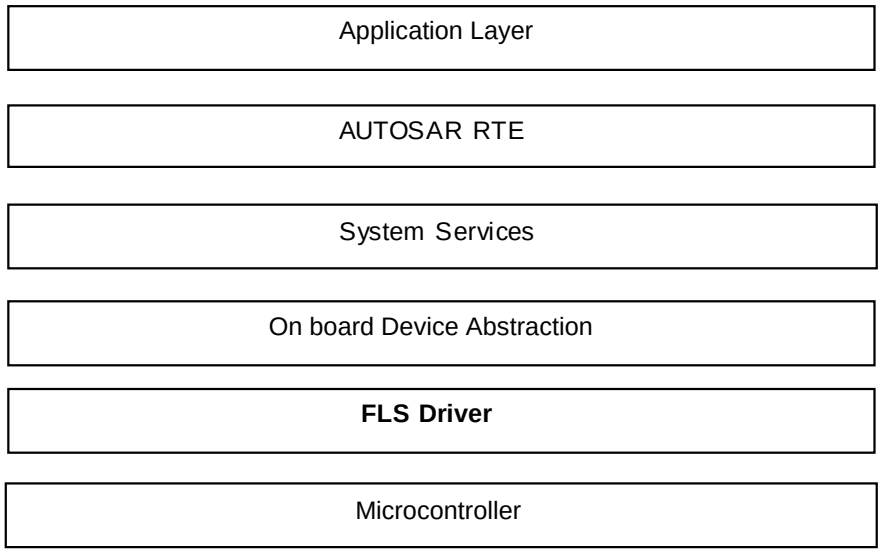

Figure 5-1 FLS Driver Component Architecture

The basic architecture of the FLS Driver Component is illustrated in the following Figure:

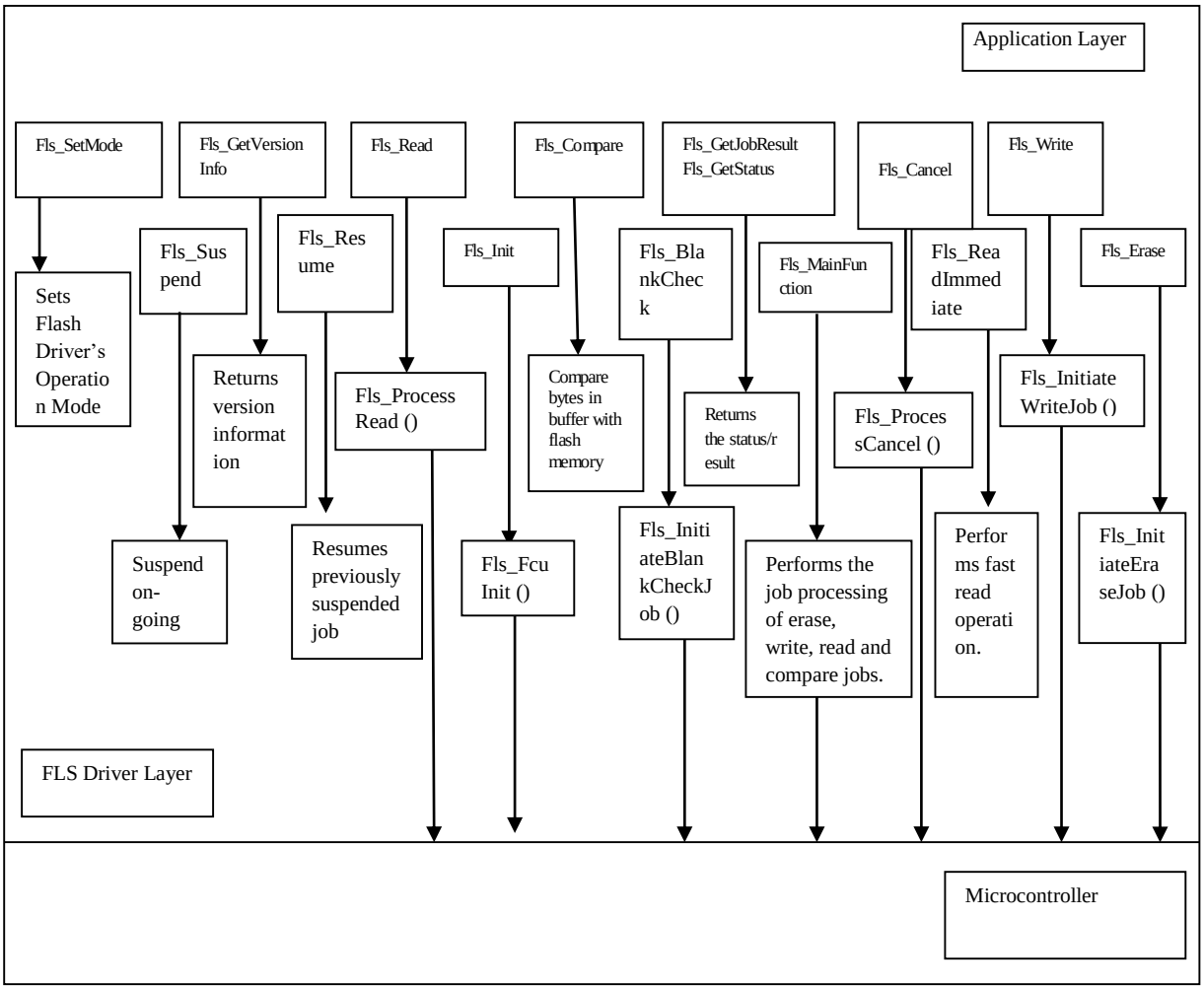

Figure 5-2 Component Overview of FLS Driver Component

The internal architecture of FLS Driver Component is shown in the above figure. The FLS Driver Component Software Component provides services for the following processes:

The FLS Driver Component is divided into the following sub modules based on the functionality required:

- Initialization
- Erasing the flash memory
- Writing to the flash memory
- Reading the flash memory
- Fast Read to the application memory without performing blank check
- Validating contents of flash memory
- Cancellation of Request
- Reading result and status information
- Module version information
- Blank check of flash memory
- Job Processing
- Fls_Suspend suspends the on-going job.
- Fls_Resume performs the resume of previous suspended job.

Initialization

The initialization sub-module provides the service for initialization of the flash driver and initializes the global variables used by the FLS Component. FCU initialization API initializes FCU Global Variable Structure and prepares the environment. After that firmware code is copied to the RAM and FACI frequency is set. The function also resets the FCU and initialize the hardware registers to default values.

The API related to this sub-module is Fls_Init.

Flash Memory Erasing Module

This sub-module provides the service for erasing the blocks of the flash memory.

The request will be processed by the job processing function Fls_MainFunction. The First round of erase operation is initiated from within the API itself. Fls_MainFunction is then called to erase the remaining requested data flash memory blocks. The job is processed till the requested numbers of blocks are erased in the flash memory. Blank Check shall be done to ensure that the blocks are completely erased.

The API related to this sub-module is Fls_Erase.

Flash Memory Reading Module

This sub-module provides the service for reading the contents of the flash memory. The request will be processed by the job processing function Fls_MainFunction.

In this job processing function, blank check for the specified words shall be performed first. If the cell is blank, then the application buffer shall be filled with the value specified by the parameter 'FlsErasedValue'. If the cell is not blank, then reading of the specified words from the Flash memory shall be performed. This sub-module reads the specified number of words from consecutive Flash addresses starting at the specified address and writes it into a buffer. Read operation shall be initiated within the sub-module itself. Single cycle of Fls_MainFunction shall read the maximum number of bytes configured depending on the parameters 'FlsMaxReadNormalMode' (if default mode is MEMIF_MODE_SLOW) and 'FlsMaxReadFastMode' (if default mode is MEMIF_MODE_FAST). The job is processed till the requested bytes of length are copied into the application buffer.

The API related to this sub-module is Fls_Read.

Flash Memory Writing Module

This sub-module provides the service for writing to the flash memory.

The request shall be processed by the job processing function Fls_MainFunction. The First round of write operation is initiated from within the API itself. In this job processing function, the writing of specified number of data bytes from buffer to flash memory shall be performed. The function writes the specified number of words from buffer to consecutive Flash addresses starting at the specified address. Single cycle of Fls_MainFunction shall write 4 bytes of

data from target buffer to flash addresses. The job is processed till the requested number of bytes is written to the flash memory
The API related to this sub-module is Fls_Write.

Flash Memory Contents Validating Module

This sub-module provides the service for comparing the contents of the flash memory with the application buffer.
The request shall be processed by the job processing function Fls_MainFunction.

This sub-module shall read the defined number of words in flash and store it in the temporary buffer. Then actual data in application buffer shall be compared with data in temporary buffer. Here data shall be compared in terms of bytes. Single cycle of Fls_MainFunction shall read the data from the flash memory depending on configuration of parameter 'FlsMaxReadNormalMode' for data flash. The job is processed till the requested number of bytes are read and compared with the application buffer.

The API related to this sub-module is Fls_Compare.

Request Set Mode Module

This sub-module sets the flash driver operation mode.

The API related to this sub-module is Fls_SetMode.

Request Cancellation Module

This sub-module provides the service for cancelling an on-going memory request.

After aborting the current on-going memory operations this sub- module prepares internal variables to accept the next Read/Write/Erase/ Compare command. The cancel request will be synchronous and a new job can be requested immediately after the return from this function. A suspended job is also cancelled.

The API related to this sub-module is Fls_Cancel.

Result Reading and Status Information Providing Module

This sub-module provides the services for getting the current status of the module or results of the initiated job request or the response to previously issued command and return the current status of the current job execution.

The APIs related to this sub-module are Fls_GetStatus, Fls_GetJobResult.

Software Component Version Info Module

This module provides API for reading Module Id, Vendor Id and vendor specific version numbers.

The API related to this sub-module is Fls_GetVersionInfo.

Job Processing Module

The command requests are always processed by the main function that is invoked cyclically by the scheduler. This function will perform the status check while processing the flash operations requests. This API derives the internal driver status. Completion of the flash operation needs to be checked in order to continue the reprogramming flow. A Time-out feature is available with the help of time-out counter operation in this API.

The API related to this sub-module is Fls_MainFunction.

Flash Memory Blank Check Module

This sub-module provides the service for performing blank check of the flash memory words. The request shall be processed by the job processing function Fls_MainFunction. This function is invoked to perform the blank check of the single word. The job is processed till the requested numbers of words are performed with the blank check in the flash memory.

The API related to this sub-module is Fls_BlankCheck.

Flash Memory Fast Read Module

This sub-module provides the service for reading the contents of the flash memory. The request shall be processed by the job processing function Fls_MainFunction. This function reads the specified number of words from consecutive Flash addresses starting at the specified address and writes it into a buffer. Single cycle of Fls_MainFunction, shall read the data from the data flash memory. The data from flash memory (source address) is read to the data buffer (Target address) of application without performing blank check before read. The job is processed till the requested bytes of length are copied into the application buffer.

The API related to this sub-module is Fls_ReadImmediate.

Job Suspend Module

This sub-module provides the service of suspending the on-going job. The driver goes into idle state after the job is suspended. Fls_Suspend is asynchronous API. Fls_Suspend shall reject any unacceptable request of suspension such as issuing suspend request for operations other than erase and write and if no on-going job is present.

The API related to this sub-module is Fls_Suspend.

Job Resume Module

This sub-module provides the service for performing the resume of the previous suspended job. Fls_Resume is synchronous API. Fls_Resume acknowledges the resume request and it returns immediately.

The API related to this sub-module is Fls_Resume.

Chapter 6 Registers Details

This section describes the register details of FLS Driver Component.

Table 6-1 Register Details

API Name	Registers Used	Register Access 8/16/32 bits	Register Access R/W/RW	Config Parameter	Macro/Variable
Fls_Init	FSADDR	32	RW	-	LulStartAddr FLS_FCU_ADDR_REG_RESET
	FEADDR	32	RW	-	LulEndAddr FLS_FCU_ADDR_REG_RESET
	FSTATR	32	R	-	LulRegValue LulReturnValue
	FENTRYR	16	RW	-	LddMode FLS_FCU_REGBIT_FENTRY_KEY LusModeRegVal
	FASTAT	8	RW	-	FLS_FCU_REGBIT_FASTAT_CMDLK
	FCURAME	16	RW	-	FLS_FCU_REGBIT_FCURAME_FCRME FLS_FCU_REGBIT_FCURAME_KEY FLS_FCU_REGBIT_FCURAME_RESET FLS_FCU_REGBIT_FCURAME_FRAMTRAN
	FPCKAR	16	RW	-	FLS_FCU_REGBIT_FPCKAR_KEY LusFaciFreq
	FRTEINT	8	RW	-	FLS_FACI_FRTEINT_RESET_VAL
	FCUFAREA	8	RW	-	LucModeVal
	ICCTRL	32	RW	-	FLS_FCU_SYSTEM_REGISTER_ICCTRL
	CDBCR	32	RW	-	FLS_FCU_SYSTEM_REGISTER_CDBCR
	DFECCCTL	16	RW	-	FLS_DFECCCTL_RESET_VAL
	DFERRINT	8	RW	-	FLS_ DFERRINT _RESET_VAL
	DFTSTCTL	16	RW	-	FLS_ DFTSTCTL _RESET_VAL
	FHVE3	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON

API Name	Registers Used	Register Access 8/16/32 bits	Register Access R/W/RW	Config Parameter	Macro/Variable
Fls_Erase, Fls_MainFunction, Fls_Resume	FSADDR	32	RW	-	LulCurrentStartAddr FLS_FCU_ADDR_REG_RESET LulStartAddr
	FEADDR	32	RW	-	LulCurrentEndAddr FLS_FCU_ADDR_REG_RESET LulEndAddr
	FSTATR	32	R	-	LulRegValue LulReturnValue
	FENTRYR	16	RW	-	LddMode FLS_FCU_REGBIT_FENTRY_KEY LusModeRegVal
	ICFLENDNM	16	RW	-	LusRegvalue
	IMRn	16	RW	-	pFlEndImrAddress, usFlEndImrMask
	FHVE3	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
Fls_Write, Fls_MainFunction, Fls_Resume	FSADDR	32	RW	-	LulCurrentStartAddr FLS_FCU_ADDR_REG_RESET
	FEADDR	32	RW	-	LulCurrentStartAddr + FLS_FCU_WRITE_SIZE) - FLS_FCU_ONE FLS_FCU_ADDR_REG_RESET
	FSTATR	32	R	-	LulRegValue LulReturnValue
	FENTRYR	16	RW	-	LddMode FLS_FCU_REGBIT_FENTRY_KEY LusModeRegVal
	ICFLENDNM	16	RW	-	LusRegvalue
	IMRn	16	RW	-	pFlEndImrAddress, usFlEndImrMask
	FHVE3	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
Fls_Cancel	FENTRYR	16	RW	-	LddMode FLS_FCU_REGBIT_FENTRY_KEY LusModeRegVal
	FASTAT	8	RW	-	FLS_FCU_REGBIT_FASTAT_CMDLK
	FSTATR	32	R	-	LulReturnValue
	ICFLENDNM	16	RW	-	LusRegvalue
	IMRn	16	RW	-	pFlEndImrAddress, usFlEndImrMask

API Name	Registers Used	Register Access 8/16/32 bits	Register Access R/W/RW	Config Parameter	Macro/Variable
Fls_Read, Fls_MainFunction, Fls_Resume	FSADDR	32	RW	-	LulStartAddr FLS_FCU_ADDR_REG_RESET
	FEADDR	32	RW	-	LulEndAddr FLS_FCU_ADDR_REG_RESET
	DFERSTC	8	W	-	FLS_FCU_REGBIT_DFERSTC_ERRCLR
	DFERRINT	8	RW	-	LucRegValue FLS_FCU_REGVAL_DFERRINT_NOINT
	DFERSTR	8	R	-	LulErrorRegValue
	FSTATR	32	R	-	LulReturnValue LulRegValue
	ICFLENDNM	16	RW	-	LusRegvalue
	FBCSTAT	8	R	-	LulRegValue
	FENTRYR	16	RW	-	LddMode FLS_FCU_REGBIT_FENTRY_KEY LusModeRegVal
Fls_ReadImmediate, Fls_MainFunction	DFERSTC	8	W	-	FLS_FCU_REGBIT_DFERSTC_ERRCLR
	DFERRINT	8	RW	-	LucDFERRInt FLS_FCU_REGVAL_DFERRINT_NOINT
	DFERSTR	8	R	-	LulDFERStatus
Fls_Compare, Fls_MainFunction	DFERSTC	8	W	-	FLS_FCU_REGBIT_DFERSTC_ERRCLR
	DFERRINT	8	RW	-	LucRegValue FLS_FCU_REGVAL_DFERRINT_NOINT
	DFERSTR	8	R	-	LulErrorRegValue
	ICFLENDNM	16	RW	-	LusRegvalue
Fls_BlankCheck, Fls_MainFunction Fls_Resume	FSADDR	32	RW	-	LulStartAddr
	FEADDR	32	RW	-	LulEndAddr
	FSTATR	32	R	-	LulReturnValue LulRegValue
	FBCSTAT	8	R	-	LulRegValue
	FENTRYR	16	RW	-	LddMode FLS_FCU_REGBIT_FENTRY_KEY LusModeRegVal
	ICFLENDNM	16	RW	-	LusRegvalue
	IMRn	16	RW	-	pFlEndImrAddress, usFlEndImrMask
	FHVE3	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	32	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON

API Name	Registers Used	Register Access 8/16/32 bits	Register Access R/W/RW	Config Parameter	Macro/Variable
Fls_GetStatus	-	-	-	-	-
Fls_GetJobResult	-	-	-	-	-
Fls_Suspend	-	-	-	-	-
Fls_GetVersionInfo	-	-	-	-	-

Chapter 7 Interaction between the User and FLS Driver Component

The details of the services supported by the FLS Driver Component to the upper layers users and the mapping of the channels to the hardware units is provided in the following sections:

7.1. Services Provided by FLS Driver Component to the User

The FLS Driver Component provides the following functions to upper layers:

- Writing contents to data flash memory
- Erase flash memory sectors
- Read flash contents to the application memory
- Fast Read to the application memory without performing blank check
- Validate flash contents comparing with the application memory
- Cancel the on-going erase, write, read or compare requests.
- Read the result of the last job
- Blank check of flash memory
- Read the status of the FLS Driver Component.
- Fls_Suspend suspends the on-going job.
- Fls_Resume performs the resume of previous suspended job.

Caution:

- If other software components in BSW are accessing data flash or FACI registers, then the synchronization between FLS and other software components shall be handled by user application to ensure data consistency.

- Please pay attention that many FLS APIs are non-reentrant. This means it is not allowed to call a non-reentrant API function from a different program context (e.g. interrupt service routines, other threads) while another or the same non-reentrant API function is already running.
  In particular, when calling Fls_MainFunction, user application shall avoid collision with other non-reentrant FLS APIs.

Chapter 8 FLS Driver Component Header and Source File Description

This section explains the FLS Driver Component's C Source and C Header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by FLS Driver Code Generation Tool:

- Fls_Cbk.h
- Fls_Cfg.h

The C source file generated by FLS Driver Code Generation Tool:

- Fls_PBcfg.c

The FLS Driver Component C header files:
- Fls.h
- Fls_Debug.h
- Fls_Internal.h
- Fls_Types.h
- Fls_PBTypes.h
- Fls_Version.h
- Fls_Ram.h
- Fls_Private_Fcu.h
- Fls_Irq.h
- Fls_RegWrite.h

The FLS Driver Component source files:
- Fls.c
- Fls_Internal.c
- Fls_Ram.c
- Fls_Private_Fcu.c
- Fls_Version.c
- Fls_Irq.c

The Stub C header files:
- Compiler.h
- Compiler_Cfg.h
- MemMap.h
- Platform_Types.h
- rh850_Types.h
- Dem.h
- Dem_Cfg.h
- Det.h
- MemIf.h
- MemIf_Types.h
- Os.h
- Rte.h
- Std_Types.h
- SchM_Fls.h

The description of the FLS Driver Component files is provided in the table below:

Table 8-1 Description of the FLS Driver Component Files

File	Details
Fls_Cfg.h	This file is generated by the Renesas Unified Code Generator Tool for various FLS Driver Component pre-compile time parameters. The macros and the parameters generated will vary with respect to the configuration in the input ECU Configuration description file. This file also contains the handles for Fls Pin configuration set.
Fls_Cbk.h	This file contains declarations of notification functions to be used by the application. The notification function name can be configured.
Fls_PBcfg.c	This file contains post-build configuration data. The structures related to FLS Initialization are provided in this file. Data structures will vary with respect to parameters configured.
Fls.h	This file provides extern declarations for all the FLS Driver Component APIs. This file provides service Ids of APIs, DET Error codes and type definitions for FLS Software initialization structure. This header file shall be included in other modules to use the features of FLS Driver Component.
Fls_Debug.h	This file provides Provision of global variables for debugging purpose.
Fls_Internal.h	This file contains the declarations of the internally used functions.
Fls_Types.h	This file contains the common macro definitions and the data types required internally by the FLS software component.
Fls_Ram.h	This file contains the extern declarations for the global variables that are defined in Fls_Ram.c file and the version information of the file.
Fls_Version.h	This file contains the macros of AUTOSAR version numbers of all modules that are interfaced to FLS.
Fls_Irq.h	This file contains the external declaration for the interrupt functions used by FLS Driver Module.
Fls_Private_Fcu.h	This file contains API Declarations of Flash Control Unit specific functions
Fls_RegWrite.h	This file is to have macro definitions for the registers write and verification.
Fls.c	This file contains the implementation of all APIs.
Fls_Ram.c	This file contains the global variables used by FLS Driver Component.
Fls_Private_Fcu.c	This file contains FCU related API implementations
Fls_Internal.c	This file contains the definition of the internal functions that access the hardware registers.
Fls_Version.c	This file contains the code for checking version of all modules that are interfaced to FLS.
Fls_Irq.c	This file contains the implementation of all the interrupt functions used by FLS Driver Module.
Compiler.h	Provides compiler specific (non-ANSI) keywords. All mappings of keywords, which are not standardized, and/or compiler specific are placed and organized in this compiler specific header.
Compiler_Cfg.h	This file contains the memory and pointer classes.
MemMap.h	This file allows to map variables, constants and code of modules to individual memory sections. Memory mapping can be modified as per ECU specific needs.

File	Details
Platform_Types.h	This file provides provision for defining platform and compiler dependent types.
Fls_PBTypes.h	This file contains the type definitions of post build parameters. It also contains the macros used by the FLS Driver Component.
rh850_Types.h	This file provides macros to perform supervisor mode (SV) write enabled Register ICxxx and IMR register writing using OR/AND/Direct operation.
Dem.h	This file is a stub for DEM Component.
Dem_Cfg.h	This is a stub file used for defining dem event parameters used in the configuration.
Det.h	This file is a stub for DET Component.
MemIf.h	This file is a stub for MEMIF Module.
MemIf_Types.h	This file is a stub for MemIf component.
Os.h	This file is a stub for Os Component.
Rte.h	This file is a stub for Rte Component.
Std_Types.h	This file is a stub file which contains the standard type definitions.
SchM_Fls.h	This file is a stub file which is used to get the support of critical section protection.

Chapter 9 Generation Tool Guide

For information on the FLS Driver Code Generation Tool, please refer “R20UT3711EJ0101-AUTOSAR.pdf” document.

Chapter 10 Application Programming Interface

This section explains the Data types and APIs provided by the FLS Driver Component to the Upper layers.

10.1. Imported Types

This section explains the Data types imported by the FLS Driver Component and lists its dependency on other modules.

10.1.1. Standard Types

In this section all types included from the Std_Types.h are listed:

- Std_ReturnType
- Std_VersionInfoType

10.1.2. Other Module Types

In this section all types included from the Dem.h and MemIf_Types.h are listed.

- Dem_EventIdType
- Dem_EventStatusType
- Memif_JobResultType
- Memif_StatusType
- MemIf_ModeType

10.2. Type Definitions

This section explains the type definitions of FLS Driver Component according to AUTOSAR Specification

10.2.1 Fls_ConfigType

Table 10-1 Fls_ConfigType

Name:	Fls_ConfigType		
Type:	Structure		
	Type	**Name**	**Explanation**
	unit32	ulStartOfDbToc	Database start value
	void*	pJobEndNotificationPointer	Pointer to job end callback notification
	void*	pJobErrorNotificationPointer	Pointer to job error callback notification
	void*	pEccSEDNotificationPointer	Pointer to ECC SED callback notification
	void*	pEccDEDNotificationPointer	Pointer to ECC DED callback notification
	uint32	ulFlsSlowModeMaxReadBytes	Maximum number of Read bytes in Normal Mode
	uint32	ulFlsFastModeMaxReadBytes	Maximum number of Read bytes in fast Mode

	uint16*	pFlEndImrAddress	Address for error IMR registers
	uint16	usFlEndImrMask	Mask for IMR register
Element:	volatile Fls_FACIRegType	pFACIRegPtr	Base Address for FACI Registers
	volatile Fls_ECCRegType	pECCRegPtr	Base Address for ECC Registers
	MemIfModeType	ddDefaultMode	Default Mode value
Description:	Structure to hold the flash driver configuration set. The contents of the initialisation data structure are specific to the flash memory hardware		

10.2.2 Fls_AddressType

Table 10-2 Fls_AddressType

Name:	Fls_AddressType	
Type:	uint	
Range:	8/16/32 bits	Size depends on target platform and flash device.
Description:	Used as address offset from the configured flash base address to access a certain flash memory area.	

10.2.3 Fls_LengthType

Table 10-3 Fls_LengthType

Name:	Fls_LengthType	
Type:	uint	
Range:	Same as Fls_AddressType	Shall be the same type as Fls_AddressType because of arithmetic operations. Size depends on target platform and flash device.
Description:	Specifies the number of bytes to read/write/erase/compare.	

10.3. Function Definitions

Table 10-4 API Provided by FLS Driver Component

Sl. No	API's name
1.	Fls_Init
2.	Fls_Erase
3.	Fls_Write
4.	Fls_Cancel
5.	Fls_GetStatus
6.	Fls_GetJobResult

Sl. No	API's name
7.	Fls_Read
8.	Fls_Compare
9.	Fls_SetMode
10.	Fls_GetVersionInfo
11.	Fls_MainFunction
12.	Fls_BlankCheck
13.	Fls_ReadImmediate
14.	Fls_Suspend
15.	Fls_Resume

10.3.1. Fls_Init

Name:	Fls_Init		
Prototype:	FUNC(void, FLS_PUBLIC_CODE) Fls_Init(P2CONST(Fls_ConfigType, AUTOMATIC, FLS_APPL_CONST) ConfigPtr)		
Service ID:	0x00		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Fls_ConfigType	ConfigPtr	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	void	NA	
Description:	This service performs initialization of the FLS Driver component.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.2. Fls_Erase

Name:	Fls_Erase		
Prototype:	FUNC(Std_ReturnType, FLS_PUBLIC_CODE) Fls_Erase (Fls_AddressType TargetAddress, Fls_LengthType Length)		
Service ID:	0x01		
Sync/Async:	Asynchronous		
Reentrancy:	Non-Reentrant		
	Type	**Parameter**	**Value/Range**

Parameters In:	Fls_AddressType	TargetAddress	Target address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: FLS_SIZE - 1
	Fls_LengthType	Length	Number of bytes to erase Min.: 1 Max.: FLS_SIZE - TargetAddress
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: If Erase command has been accepted. E_NOT_OK: If Erase command has not been accepted.	
Description:	This API will erase the one or more complete flash sectors.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.3. Fls_Write

Name:	Fls_Write		
Prototype:	FUNC(Std_ReturnType, FLS_PUBLIC_CODE) Fls_Write(Fls_AddressType TargetAddress,P2CONST(uint8, AUTOMATIC, FLS_APPL_CONST) SourceAddressPtr, Fls_LengthType Length)		
Service ID:	0x02		
Sync/Async:	Asynchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Fls_AddressType	TargetAddress	Target address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: FLS_SIZE - 1
	uint8	SourceAddressPtr	Pointer to source data buffer
	Fls_LengthType	Length	Number of bytes to write Min.: 1 Max.: FLS_SIZE - TargetAddress
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: write command has been accepted E_NOT_OK: write command has not been accepted	
Description:	Writes one or more complete flash pages		
Configuration Dependency:	None		

Preconditions:	None

10.3.4. Fls_Cancel

Name:	Fls_Cancel		
Prototype:	FUNC(void, FLS_PUBLIC_CODE) Fls_Cancel(void)		
Service ID:	0x03		
Sync/Async:	synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	NA	
Description:	Cancel an ongoing flash read, write, erase or compare job.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.5. Fls_GetStatus

Name:	Fls_GetStatus		
Prototype:	FUNC(MemIf_StatusType, FLS_PUBLIC_CODE) Fls_GetStatus(void)		
Service ID:	0x04		
Sync/Async:	synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	MemIf_StatusType	Type definition for MEMIF status types MEMIF_UNINIT, MEMIF_IDLE, MEMIF_BUSY, MEMIF_BUSY_INTERNAL.	
Description:	Return the FLS module state synchronously		
Configuration Dependency:	None		
Preconditions:	None		

10.3.6. Fls_GetJobResult

Name:	Fls_GetJobResult		
Prototype:	FUNC(MemIf_JobResultType, FLS_PUBLIC_CODE) Fls_GetJobResult(void)		
Service ID:	0x05		
Sync/Async:	synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	MemIf_JobResultType	Type definition for MEMIF job result types MEMIF_JOB_OK, MEMIF_JOB_FAILED, MEMIF_JOB_PENDING, MEMIF_JOB_CANCELED, MEMIF_BLOCK_INCONSISTENT, MEMIF_BLOCK_INVALID	
Description:	Return the result of the last job synchronously		
Configuration Dependency:	None		
Preconditions:	None		

10.3.7. Fls_MainFunction

Name:	Fls_MainFunction		
Prototype:	FUNC(void, FLS_PUBLIC_CODE) Fls_MainFunction(void)		
Service ID:	0x06		
Timing:	FIXED_CYCLIC		
Sync/Async:	NA		
Reentrancy:	NA		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	NA	
Description:	Performs the processing of jobs.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.8. Fls_Read

Name:	Fls_Read		
Prototype:	FUNC(Std_ReturnType, FLS_PUBLIC_CODE) Fls_Read(Fls_AddressType SourceAddress, P2VAR(uint8, AUTOMATIC, FLS_APPL_CONST) TargetAddressPtr, Fls_LengthType Length)		
Service ID:	0x07		
Sync/Async:	Asynchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Fls_AddressType	SourceAddress	Source address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: FLS_SIZE - 1
	Fls_LengthType	Length	Number of bytes to read Min.: 1 Max.: FLS_SIZE - TargetAddress
Parameters InOut:	None	NA	NA
Parameters out:	uint8	TargetAddressPtr	Pointer to target data buffer
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: read command has been accepted E_NOT_OK: read command has not been accepted	
Description:	Read from flash memory		
Configuration Dependency:	None		
Preconditions:	None		

10.3.9. Fls_Compare

Name:	Fls_Compare		
Prototype:	FUNC(Std_ReturnType, FLS_PUBLIC_CODE) Fls_Compare (Fls_AddressType SourceAddress, P2CONST(uint8, AUTOMATIC, FLS_APPL_CONST) TargetAddressPtr, Fls_LengthType Length)		
Service ID:	0x08		
Sync/Async:	Asynchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Fls_AddressType	SourceAddress	Source address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: FLS_SIZE - 1
	Fls_LengthType	Length	Number of bytes to compare Min.: 1 Max.: FLS_SIZE - SourceAddress
	uint8	TargetAddressPtr	Pointer to target data buffer

Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: compare command has been accepted E_NOT_OK: compare command has not been accepted	
Description:	Compares the contents of an area of flash memory with that of an application data buffer.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.10. Fls_SetMode

Name:	Fls_SetMode		
Prototype:	FUNC(void, FLS_PUBLIC_CODE) Fls_SetMode(MemIf_ModeType LenMode)		
Service ID:	0x09		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	MemIf_ModeType	Mode	MEMIF_MODE_SLOW: Slow read access / normal SPI access. MEMIF_MODE_FAST: Fast read access / SPI burst access.
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	NA	
Description:	Sets the flash driver's operation mode.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.11. Fls_GetVersionInfo

Name:	Fls_GetVersionInfo		
Prototype:	FUNC(void, FLS_PUBLIC_CODE) Fls_GetVersionInfo (P2VAR(Std_VersionInfoType, AUTOMATIC, FLS_APPL_DATA) versioninfo)		
Service ID:	0x10		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA

Parameters out:	Std_VersionInfoType	VersioninfoPtr	Pointer to where to store the version information of this module.
Return Value:	**Type**	**Possible Return Values**	
	None	NA	
Description:	Returns the version information of this module.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.12. Fls_ReadImmediate

Name:	Fls_ReadImmediate		
Prototype:	FUNC(Std_ReturnType, FLS_PUBLIC_CODE) Fls_ReadImmediate (Fls_AddressType SourceAddress, P2VAR(uint8, AUTOMATIC, FLS_APPL_CONST) TargetAddressPtr, Fls_LengthType Length)		
Service ID:	0x11		
Sync/Async:	Asynchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Fls_AddressType	SourceAddress	Source address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: FLS_SIZE - 1
	Fls_LengthType	Length	Number of bytes to read Min.: 1 Max.: FLS_SIZE - TargetAddress
Parameters InOut:	None	NA	NA
Parameters out:	uint8	TargetAddressPtr	Pointer to target data buffer
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: read command has been accepted E_NOT_OK: read command has not been accepted	
Description:	Performs the reading of the flash memory. The data from flash memory (source address) is read to the data buffer (Target address) of application without performing blank check before read.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.13. Fls_BlankCheck

Name:	Fls_BlankCheck
Prototype:	FUNC(Std_ReturnType, FLS_PUBLIC_CODE) Fls_BlankCheck (Fls_AddressType TargetAddress, Fls_LengthType Length)
Service ID:	0x12

Sync/Async:	Asynchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Fls_AddressType	TargetAddress	Target address in flash memory. This address offset will be added to the flash memory base address. Min.: 0 Max.: FLS_SIZE - 1
	Fls_LengthType	Length	Number of bytes to be blank checked Min.: 1 Max.: FLS_SIZE - TargetAddress
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: Blank check command has been accepted E_NOT_OK: Blank check command has not been accepted	
Description:	Performs the blank check of flash Memory		
Configuration Dependency:	None		
Preconditions:	None		

10.3.14. Fls_Suspend

Name:	Fls_Suspend		
Prototype:	FUNC(Std_ReturnType , FLS_PUBLIC_CODE) Fls_Suspend(void)		
Service ID:	0x13		
Sync/Async:	Asynchronous		
Reentrancy:	Non Re-entrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK: Suspend job has been accepted E_NOT_OK: Suspend job has not been accepted	
Description:	Performs the suspension of the ongoing job		
Configuration Dependency:	None		
Preconditions:	None		

10.3.15. Fls_Resume

Name:	Fls_Resume		
Prototype:	FUNC(void, FLS_PUBLIC_CODE) Fls_Resume(void)		
Service ID:	0x14		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	NA	
Description:	Resumes the suspended job		
Configuration Dependency:	None		
Preconditions:	None		

10.3.16. Fls_CallSwitchBFlashErrorNotification

Name:	Fls_CallSwitchBFlashErrorNotification		
Prototype:	void Fls_CallSwitchBFlashErrorNotification(void)		
Service ID:	NA		
Sync/Async:	Synchronous		
Reentrancy:	NA		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	NA	
Description:	This callback function is invoked when failure occurs while re-enabling of Code Flash during FCU initialization procedure.		
Configuration Dependency:	None		
Preconditions:	None		

Chapter 11 Development and Production Errors

In this section the development errors that are reported by the FLS Driver Component are tabulated. The development errors will be reported only when the pre compiler option FlsDevErrorDetect is enabled in the configuration. The production code errors are not supported by FLS Driver Component.

11.1 FLS Driver Component Development Errors

The following table contains the DET errors that are reported by FLS Driver Component. These errors are reported to Development Error Tracer Module when the FLS Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 11-1 DET Errors of FLS Driver Component

Sl. No.	**1**
Error Code	FLS_E_UNINIT
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_Compare, Fls_Cancel, Fls_GetStatus, Fls_GetJobResult, Fls_MainFunction, Fls_Init, Fls_ReadImmediate, Fls_BlankCheck, Fls_Suspend, Fls_Resume, Fls_SetMode
Source of Error	When the API service is invoked before initialization.
Sl. No.	**2**
Error Code	FLS_E_PARAM_ADDRESS
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_Compare, Fls_ReadImmediate, Fls_BlankCheck
Source of Error	When the API service is invoked with a wrong address.
Sl. No.	**3**
Error Code	FLS_E_PARAM_LENGTH
Related API(s)	Fls_Erase, Fls_Write, Fls_Read, Fls_Compare, Fls_ReadImmediate, Fls_BlankCheck
Source of Error	When the API service is invoked with a wrong length.
Sl. No.	**4**
Error Code	FLS_E_PARAM_DATA
Related API(s)	Fls_Write, Fls_Read, Fls_Compare, Fls_ReadImmediate
Source of Error	When the API service is invoked with a NULL buffer address.
Sl. No.	**5**
Error Code	FLS_E_BUSY
Related API(s)	Fls_Init, Fls_Erase, Fls_Write, Fls_Read, Fls_Compare, Fls_SetMode , Fls_ReadImmediate, Fls_BlankCheck
Source of Error	When the API service is invoked when the driver is still busy.
Sl. No.	**6**
Error Code	FLS_E_VERIFY_ERASE_FAILED
Related API(s)	Fls_MainFunction
Source of Error	When the erase verification fails.

Sl. No.	7
Error Code	FLS_E_VERIFY_WRITE_FAILED
Related API(s)	Fls_MainFunction
Source of Error	When the write verification fails.
Sl. No.	**8**
Error Code	FLS_E_PARAM_CONFIG
Related API(s)	Fls_Init, Fls_SetMode
Source of Error	API initialization service invoked with wrong parameter.
Sl. No.	**9**
Error Code	FLS_E_TIMEOUT
Related API(s)	Fls_MainFunction
Source of Error	API service invoked when time out supervision of a write, erase or blank check job failed
Sl. No.	**10**
Error Code	FLS_E_INVALID_DATABASE
Related API(s)	Fls_Init
Source of Error	API service Fls_Init called without/with a wrong database is reported using following error code
Sl. No.	**11**
Error Code	FLS_E_PARAM_POINTER
Related API(s)	Fls_GetVersionInfo
Source of Error	API service Fls_GetVersionInfo invoked with a null pointer

11.2 FLS Driver Component Production Errors

The following table contains the DEM errors that are reported by FLS Driver Component. These are the hardware errors reported during runtime.

Table 11-2 DEM Errors of FLS Driver Component

Sl. No.	1
Error Code	FLS_E_ERASE_FAILED
Related API(s)	Fls_Erase
Source of Error	When the Erase API service is invoked and the erase job fails, error will be reported to DEM module. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**2**
Error Code	FLS_E_WRITE_FAILED
Related API(s)	Fls_Write
Source of Error	When the Write API service is invoked and the erase job fails, error will be reported to DEM module. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**3**
Error Code	FLS_E_READ_FAILED

Related API(s)	Fls_Read
Source of Error	When the Read API service is invoked and the internal reading of the data flash memory fails, error will be reported to DEM module. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**4**
Error Code	FLS_E_COMPARE_FAILED
Related API(s)	Fls_Compare
Source of Error	When the Compare API service is invoked and when the comparison between the data in the application buffer and the data flash memory fails, error will be reported to DEM module. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**5**
Error Code	FLS_E_READ_FAILED_DED
Related API(s)	Fls_Read, Fls_ReadImmediate, Fls_Compare
Source of Error	When the Read/ReadImmediate/Compare API service is invoked, if any double bit error is detected, error will be reported to DEM module. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**6**
Error Code	FLS_E_REG_WRITE_VERIFY
Related API(s)	Fls_Init,Fls_Erase, Fls_Write, Fls_Read, Fls_Compare, Fls_Cancel, Fls_ReadImmediate, Fls_BlankCheck, Fls_Suspend, Fls_Resume
Source of Error	If any write operation on the protection register fails, error shall be reported to DEM module. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**7**
Error Code	FLS_E_ECC_FAILED
Related API(s)	Fls_Init
Source of Error	During initialization, FLS module shall read FRTEINT register and check if any ECC error has occurred. If any errors are there, DEM shall be reported. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**8**
Error Code	FLS_E_HW_FAILURE
Related API(s)	Fls_Init, Fls_Erase, Fls_Write, Fls_Cancel, Fls_BlankCheck, Fls_Resume
Source of Error	If any failure has occurred due to mode switch or forced stop or clear status command processing failure, DEM shall be reported. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.
Sl. No.	**9**
Error Code	FLS_E_INT_INCONSISTENT
Related API(s)	FLS_FLENDNM_ISR

Source of Error	If any failure has occurred due to interrupt inconsistency has been identified from the unknown source, DEM shall be reported. The Dem module shall provide the interface Dem_ReportErrorStatus to the BSW modules, to report BSW events which are processed.

Chapter 12 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of FLS Driver Component software.

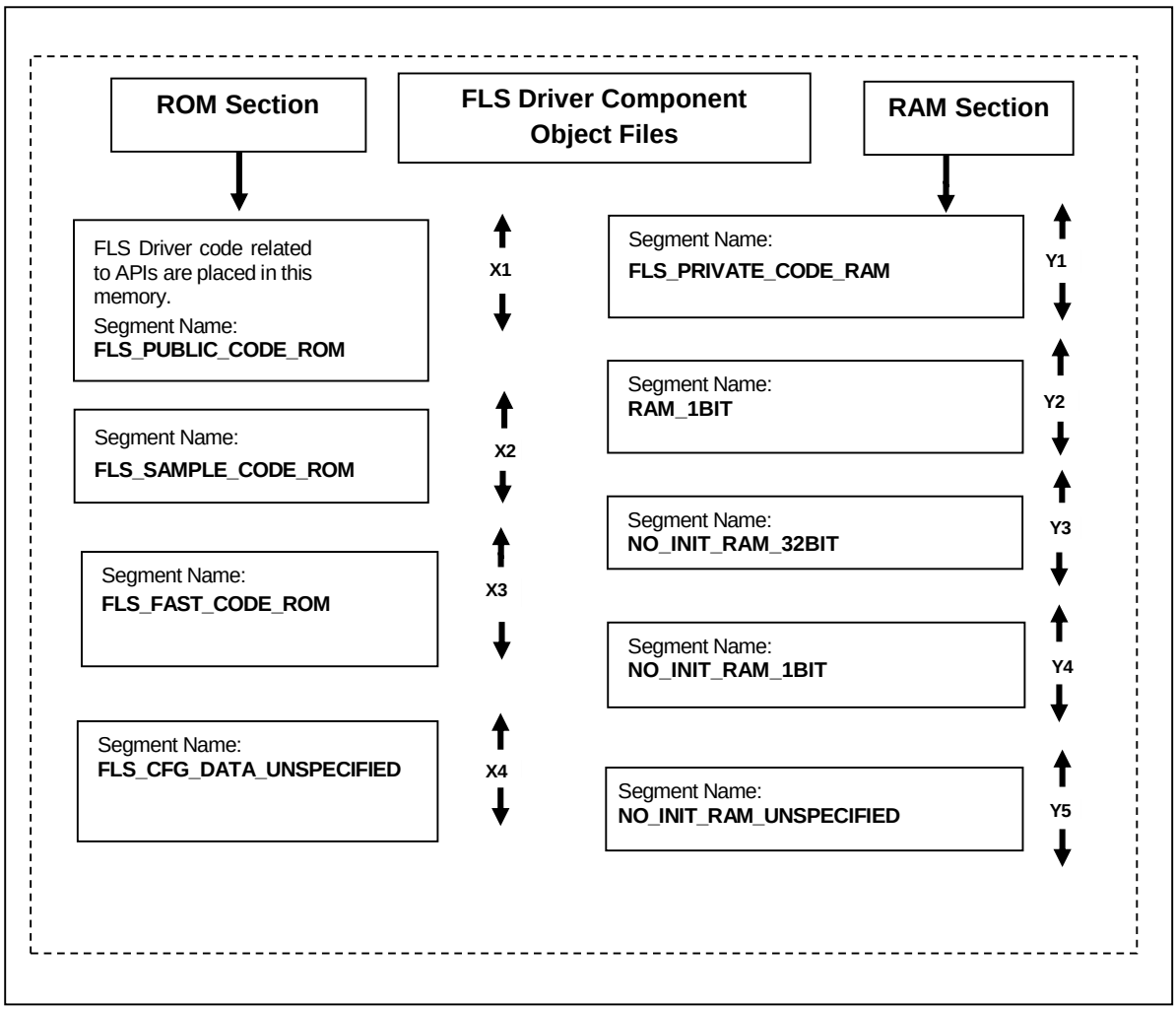

Figure 12-1 FLS Driver Component Memory Organization

ROM Sections:

FLS_PUBLIC_CODE_ROM (X1): This section consists of FLS Driver Component public APIs that can be located in code memory.

FLS_SAMPLE_CODE_ROM (X2): This section consists of FLS sample application that can be located in code memory.

FLS_FAST_CODE_ROM (X3): Interrupt functions of FLS Driver Component code that can be located in code memory.

FLS_CFG_DATA_UNSPECIFIED(X4): The const section in the file Fls_PBcfg.c is placed in this memory.

RAM Sections:

FLS_PRIVATE_CODE_RAM (Y1): This section in RAM is copied from ROM section (X1) by the GHS start-up routines.

RAM_1BIT (Y2): This section consists of the global RAM variables of 1-bit size that are initialized by start-up code and used internally by FLS software component and other software components. The specific sections of respective software components will be merged into this RAM section accordingly.

NO_INIT_RAM_32BIT (Y3): This section consists of the global RAM variables of 32-bit size that are used internally by FLS software component and other software components. The specific sections of respective software components will be merged into this RAM section accordingly.

NO_INIT_RAM_1BIT (Y4): This section consists of the global RAM variables of 1-bit size that are used internally by FLS software component and other software components. The specific sections of respective software components will be merged into this RAM section accordingly.

NO_INIT_RAM_UNSPECIFIED (Y5): This section consists of the global RAM variables that are used internally by FLS software component and other software components. The specific sections of respective software components will be merged into this RAM section accordingly.

Chapter 13 P1M Specific Information

P1M supports following devices:

- R7F701304
- R7F701305
- R7F701310
- R7F701311
- R7F701312
- R7F701313
- R7F701314
- R7F701315
- R7F701318
- R7F701319
- R7F701320
- R7F701321
- R7F701322
- R7F701323

13.1. Interaction between the User and FLS Driver Component

The details of the services supported by the FLS Driver Component to the upper layer users and the mapping of the channels to the hardware units is provided in the following sections:

13.1.1. Translation header File

P1x_translation.h supports following devices:

- R7F701304
- R7F701305
- R7F701310
- R7F701311
- R7F701312
- R7F701313
- R7F701314
- R7F701315
- R7F701318
- R7F701319
- R7F701320
- R7F701321
- R7F701322
- R7F701323

13.1.2. Services Provided by FLS Driver Component to the User

The FLS Driver Component provides the following functions to upper layers:

- Erase memory sectors
- Read flash contents to the application memory

- Fast read immediate to the application memory without blank check.
- Validate flash contents comparing with the application memory
- Cancel the on-going erase, write, read or compare requests.
- Read the result of the last job
- Blank check of flash memory sector.
- Read the status of the FLS Driver Component.
- Suspend the erase and write operation.
- Resume the erase and write operation.

13.1.3. Parameter Definition File

Table 13-1 PDF information for P1M

PDF files	Devices supported
R403_FLS_P1M_04_05_10_to_15.arxml	701304, 701305, 701310, 701311, 701312, 701313, 701314, 701315
R403_FLS_P1M_18_to_23.arxml	701318, 701319, 701320, 701321, 701322, 701323

13.1.4. ISR Functions for FLS module

The table below provides the list of handler addresses corresponding to the hardware unit ISR(s) in FLS Driver Component. The user should configure the ISR functions mentioned below:

Table 13-2 Interrupt Functions for FLS Module

Interrupt Source	Name of the ISR Function
FLENDNM_ISR	FLS_FLENDNM_ISR
	FLS_FLENDNM_CAT2_ISR

Note: The functions with “INTERRUPT “as pilot tag, provides an indication to the compiler that the function following this tag is an interrupt function type. The tag name can vary according to the compiler. User should take care of the tag name with respect to compiler used.

13.1.5. Data Flash Address Space

The Data Flash address space of the RH850/P1x is as below:
Data Flash Address:
512-KB device: FF20 0000_H to FF20 $7FFF_H$
1-MB device: FF20 0000_H to FF20 $7FFF_H$
2-MB device: FF20 0000_H to FF20 $FFFF_H$

13.2. Sample Application

13.2.1. Sample Application Structure

The Sample Application is provided as reference to the user to understand the method in which the FLS APIs can be invoked from the application. The Sample Application is provided for three use-cases.

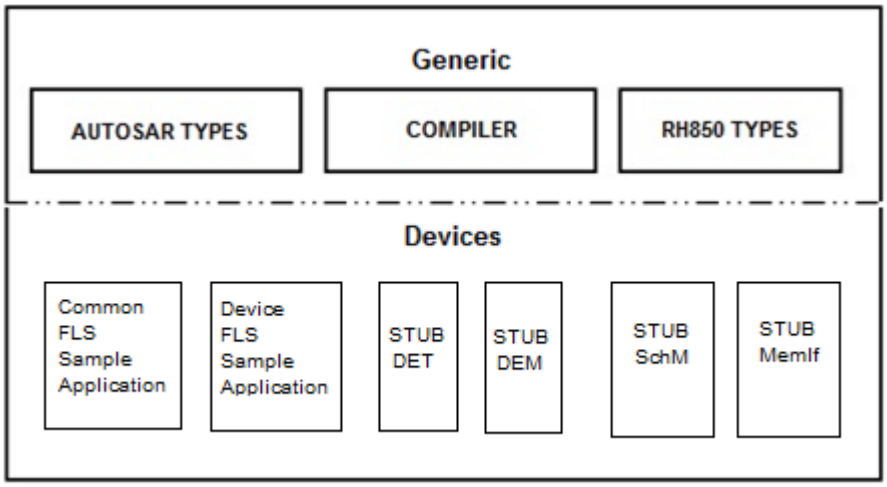

Figure 13-1 Overview of FLS Driver Sample Application

The Sample Application of the P1M is available in the path

X1X\P1x\modules\fls\sample_application

X1X\P1x\modules\fls\definition\<AUTOSAR_version>\<SubVariant>\
\R403_FLS_P1M_04_05_10_to_15.arxml

\R403_FLS_P1M_18_to_23.arxml

X1X\P1x\modules\fls\sample_application\<SubVariant>\
<AUTOSAR_version>\
\src\Fls_PBcfg.c
\include\Fls_Cfg.h
\include\Fls_Cbk.h

\config\App_FLS_P1M_701304_Sample.arxml

\config\App_FLS_P1M_701305_Sample.arxml

\config\App_FLS_P1M_701310_Sample.arxml

\config\App_FLS_P1M_701311_Sample.arxml

\config\App_FLS_P1M_701312_Sample.arxml

\config\App_FLS_P1M_701313_Sample.arxml

\config\App_FLS_P1M_701314_Sample.arxml

\config\App_FLS_P1M_701315_Sample.arxml

\config\App_FLS_P1M_701318_Sample.arxml

\config\App_FLS_P1M_701319_Sample.arxml

\config\App_FLS_P1M_701320_Sample.arxml

\config\App_FLS_P1M_701321_Sample.arxml

\config\App_FLS_P1M_701322_Sample.arxml

\config\App_FLS_P1M_701323_Sample.arxml

In the Sample Application all the FLS APIs are invoked in the following sequence:

- The API Fls_GetVersionInfo is invoked to get the version Information of FLS component with a variable of Std_VersionInfoType type, after the call of this API the passed parameter will get updated with the FLS Driver Component version details.
- The API Fls_Init is invoked with config pointer. This API performs the initialization of the FLS Driver Component. This API initializes all the elements (Global Variables) of Global structure.
- The API Fls_Erase is invoked to erase one or more complete Flash Sectors.
- The API Fls_Write is invoked to write the one or more complete flash pages to the flash device from the application data buffer
- The API Fls_Read is invoked to read the requested length of flash memory and stores it in the application data buffer.
- The API Fls_Compare is invoked to compare the contents of an area of flash memory with that of an application data buffer.
- The API Fls_Cancel is invoked to cancel an on-going flash operations like read, write, erase or compare job.
- The API Fls_Getstatus returns the FLS module state synchronously.
- The API Fls_GetJobResult returns the result of the last job synchronously.
- The API Fls_Setmode, this API sets the flash driver operation mode.
- The API Fls_Mainfunction is invoked performs processing of the flash Read, Erase, write or compare jobs. It's a scheduled function. The Fls_Mainfunction accepts only read, write, erase or compare job at a time.
- The API Fls_ReadImmediate is invoked for reading of the flash memory. The data from flash memory (source address) is read to the data buffer (Target address) of application without performing blank check before read.
- The API Fls_BlankCheck is invoked to verify whether the memory is properly erased before doing a write operation.

Remark The API Fls_MainFunction needs to be called in a certain time interval configured using the parameter "FlsCallCycle". Hence, the sample application invokes the API 'Fls_MainFunction' periodically in a loop with sufficient software delay. Calling Fls_MainFunction in the interrupt mode of FLS will not perform the substantial Flash operations; it will be executed as dummy function, except that the timeout supervision will be performed within the Fls_MainFunction call if timeout monitoring is enabled.

13.2.2. Building Sample Application

13.2.2.1. Configuration Example

This section contains the typical configuration which is used for measuring RAM/ROM consumption, stack depth and throughput details.
Configuration Details:
App_FLS_<SubVariant>_<Device_Name>_Sample.arxml

13.2.2.2. Debugging the Sample Application

Remark GNU Make utility version 3.81 or above must be installed and available in the path as defined by the environment user variable “GNUMAKE” to complete the build process using the delivered sample files.

Open a Command window and change the current working directory to “make” directory present as mentioned in below path:
“external/X1X/P1x/common_family/make/<compiler>”

Now execute batch file SampleApp.bat with following parameters:

SampleApp.bat fls <AUTOSAR_version> <Device_Name>

After this, the tool output files will be generated with the configuration as mentioned is available in the path:
“X1X\P1x\modules\fls\sample_application\<SubVariant>\<AUTOSAR_version>\config”

- After this, all the object files, map file and the executable file App_FLS_P1M_Sample.out will be available in the output folder (“X1X\P1x\modules\fls\sample_application\<SubVariant>\obj\<compiler>” in this case).
- The executable can be loaded into the debugger and the sample application can be executed.

Remark Executable files with ‘*.out’ extension can be downloaded into the target hardware with the help of Green Hills debugger.

If any configuration changes (only post-build) are made to the ECU Configuration Description files

“X1X\P1x\modules\fls\sample_application\<SubVariant>\<AUTOSAR_version>\config\ App_FLS_P1M_<Device_name>_Sample.arxml”

App_FLS_P1M_<Device_name>_Sample.arxml” the database alone can be generated by using the following commands.

make –f App_FLS_<SubVariant>_Sample.mak generate_fls_config

make –f App_FLS_<SubVariant>_Sample.mak App_FLS_<SubVariant>_Sample.s37

- After this, a flash able Motorola S-Record file App_FLS_ App_FLS_<SubVariant>_Sample.s37_Sample.s37 is available in the output folder.

Note: 1**.** <compiler> for example can be “ghs”.

2. <Device_Name> indicates the device to be compiled, which can be 701304, 701305, 701310, 701311, 701312, 701313, 701314, 701315, 701318, 701319, 701320, 701321, 701322 or 701323.
3. <SubVariant> can be P1M.
4. <AUTOSAR_version> can be 4.0.3.

13.3. Memory and Throughput

13.3.1. ROM/RAM Usage

The details of memory usage for the typical configuration, with DET enabled as provided in Section 13.2.2.1 *Configuration Example* are provided in this section.

Table 13-3 ROM/RAM Details with DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701318
1.	ROM	FLS_PUBLIC_CODE_ROM	1834
		FLS_PRIVATE_CODE_ROM	5468
		FLS_FAST_CODE_ROM	170
		FLS_CFG_DATA_UNSPECIFIED	48
		ROM.FLS_PRIVATE_CODE_RAM	362
2.	RAM	FLS_PRIVATE_CODE_RAM	362
		NO_INIT_RAM_UNSPECIFIED	100
		NO_INIT_RAM_32BIT	516
		RAM_1BIT	4
		NO_INIT_RAM_1BIT	4

Table 13-4 ROM/RAM Details without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701318
1.	ROM	FLS_PUBLIC_CODE_ROM	1350
		FLS_PRIVATE_CODE_ROM	4352
		FLS_FAST_CODE_ROM	144
		FLS_CFG_DATA_UNSPECIFIED	48
		ROM.FLS_PRIVATE_CODE_RAM	334
2.	RAM	FLS_PRIVATE_CODE_RAM	334
		NO_INIT_RAM_UNSPECIFIED	100
		NO_INIT_RAM_32BIT	516
		RAM_1BIT	4
		NO_INIT_RAM_1BIT	3

13.3.2. Stack Depth

The worst-case stack depth for FLS Driver Component is for the typical configuration provided in Section 13.2.2.1 *Configuration Example*.

Table 13-5 Stack Depth Table

Sl. No	Device Name	Stack Depth (in Bytes)
1.	R7F701318	36

13.3.3. Throughput Details

The throughput details of the APIs for the configuration mentioned in the Section 13.2.2.1*Configuration Example* are listed here. The clock frequency used to measure the throughput is 160MHz for all APIs.

Table 13-6 Throughput Details of the APIs

Sl. No.	API Name	Throughput in µ seconds for 701318 with Interrupt OFF	Throughput in µ seconds for 701318 with Interrupt ON	Remarks
1.	Fls_Init	415.112	416.650	-
2.	Fls_Erase	3.812	4.687*	-
3.	Fls_Write	3.862	4.675	-
4.	Fls_Cancel	0.237	0.287	-
5.	Fls_GetStatus	0.125	0.125	-
6.	Fls_GetJobResult	0.137	0.137	-
7.	Fls_Read	1.225	1.562	-
8.	Fls_Compare	0.962	0.862	-
9.	Fls_GetVersionInfo	0.15	0.125	-
10.	Fls_BlankCheck	2.862	3.737*	-
11.	Fls_SetMode	0.312	0.287	
12.	Fls_ReadImmediate	1.437	1.4	-
13.	Erase Operation	5320.237	5220.937	This is the time taken for the complete erase operation of 256 bytes data length.

Sl. No.	API Name	Throughput in µ seconds for 701318 with Interrupt OFF	Throughput in µ seconds for 701318 with Interrupt ON	Remarks
14.	Write Operation	6525.620	6543.675	This is the time taken for the complete write operation of 256 bytes data length.
15.	Blank Check Operation	176.35	178.887	This is the time taken for performing blank check operation of 256 bytes data length.
16.	Read Immediate Operation	45.375	48.375	This is the time taken for the complete fast read operation of 256 bytes data length without performing blank check before read.
17.	Read Operation	1498.237	1501.475	This is the time taken for the complete read operation of 256 bytes data length.
18.	Compare Operation	236.712	239.55	This is the time taken for the complete compare operation of 256 bytes data length.
19.	FLENDNM_ISR operation	NA	4.812	This is the time taken for the complete Erase of 1 block data length.
		NA	4.962	This is the time taken for the complete Write of 1 word data length
20.	Fls_Suspend	NA	0.225	Suspend and Resume throughput are taken for interrupt mode as per the requirement

Sl. No.	API Name	Throughput in µ seconds for 701318 with Interrupt OFF	Throughput in µ seconds for 701318 with Interrupt ON	Remarks
21.	Fls_Resume	NA	4.3	Suspend and Resume throughput are taken for interrupt mode as per the requirement
22.	Fls_MainFunction - for single Read Operation	1498.237	NA	This is the time taken for the complete read operation of 256 bytes data length.
23.	Fls_MainFunction - for single Erase Atomic Operation	5.137	NA	This is the time taken for Fls scheduler to complete the single erase atomic operation
24.	Fls_MainFunction - for single Write Atomic Operation	3.137	NA	This is the time taken for Fls scheduler to complete the single write atomic operation

NOTE: The Throughput time for a Single Fls_Erase and Fls_BlankCheck operation is more with Interrupt ON than with Interrupt OFF.
This is because in Interrupt ON, Throughput time includes the normal operation time along with the Interrupt ISR execution time since the interrupt is generated immediately after the command processing is complete and is serviced accordingly.

Chapter 14 Release Details

FLS Driver Software

Version: 1.0.5

Revision History

Sl.No.	Description	Version	Date
1.	Initial Version	1.0.0	15-Feb-2016
2.	Following changes are made: 1. Chapter 2 ‘Reference Documents’ updated to correct the device manual name and version and corrected version of 'AUTOSAR_SWS_MemoryMapping.pdf' and 'AUTOSAR_SWS_CompilerAbstraction.pdf'. 2. Updated Chapter 4 ‘ForeThoughts’ to add the precaution about accessing hardware register in section 4.1 ‘General’. 3. Updated Chapter 6 ‘Register Details’ to add ICFLENDNM register wherever applicable and removed register details of Fls_SetFHVE function. 4. Updated Chapter 5 ‘Architecture Details’, to correct the description of Fls_Suspend API. 5. Chapter 12 ‘Memory Organization’ has been updated to remove the sections FLS_BUFFER_CODE_RAM, FLS_USER_BUFFER_CODE_RAM and FLS_CFG_DBTOC_UNSPECIFIED and added NOINIT_RAM_32BIT and FLS_CFG_DATA_UNSPECIFIED. 6. Updated Chapter 13.3 ‘Memory and Throughput’. 7. Updated Chapter 14 ‘Release Details’ to correct the driver version.	1.0.1	24-Mar-2016
3.	Following changes are made: 1. Added precondition items about critical protection and transient hardware faults in chapter 4.2 ‘Precondition’. 2. Updated Chapter 14 ‘Release Details’. 3. Added a ‘Note’ below the table 'Supervisor mode and User mode details'. 4. Updated Chapter 13.3 ‘Memory and Throughput’. 5. In chapter 8, heading changed to "The Stub C header files:" and missing stub files are added. 6. Table 4-1 is added to list protected resources in FLS driver 7. Section 13.2 reference to .one and .html files are removed 8. Note added in section 13.2.1 ISR function. 9. Description added for the FLS Driver Component Files. 10. Updated Chapter 6 ‘Register Details’ to add IMR register details. 11. Merged parameter definition files in sections 13.1.3 and 13.2.1 12. Added new header file ‘Fls_RegWrite.h’ in section 3.1.1 and section 8. 13. Updated R-Number 14. Chapter 12 is updated for the modification of Memory sections	1.0.2	16-Jul-2016
4.	Following changes are made: 1. Updated section 4 ‘Forethoughts’ 2. Updated the table 11-1 ‘Development and Production Errors’ to add Fls_SetMode API under DET ‘FLS_E_PARAM_CONFIG’ 3. Updated Chapter 13.3 ‘Memory and Throughput’ 4. Updated chapter 12 Memory Organization 5. Updated section 13.1.2 Services Provided by FLS Driver Component to the User	1.0.3	21-Oct-2016

Sl.No.	Description	Version	Date
5.	Following changes are made: 1. Updated Table for Abbreviations and Acronyms to remove unused Abbreviations/ Acronyms. 2. Updated Chapter 2 Reference Documents. 3. Updated section 4.1 'General' to add information about Read, Read Immediate, Suspend and Resume operations. 4. Updated section 4.2 'Preconditions' to add information about BlankCheck operation. 5. Updated Chapter 3 and Chapter 9 to rename the FLS Driver Generation Tool reference document. 6. Updated Section 13.3.1 with ROM/RAM usage values. 7. Updated Section 13.3.3 'Throughput Details' to add Note under Table 13-6 Throughput Details of the APIs and updated the Throughput details. 8. Updated section 12 'Memory Organization' to remove unused memory segments and their descriptions. 9. Updated Chapter 14 for FLS Driver Software version. 10. Updated notice, address and copyright information's. 11. Version of R-number is updated at the end of document. 12. Updated Section 7.1 to add Cautions. 13. Updated Section 10.3 with the details of Fls_Resume API.	1.0.4	17-Feb-2017
6.	Following changes are made: 1. Updated section 4.2 and section 10.3 with details of newly added notification function Fls_CallSwitchBFlashErrorNotification. 2. Updated section 13.3 'Memory and Throughput'. 3. Updated Chapter 14 for FLS Driver Software version. 4. Version of R-number is updated at the end of document. 5. Updated section name from FLS_START_SEC_PRIVATE_CODE and FLS_STOP_SEC_PRIVATE_CODE to FLS_START_SEC_PRIVATERAM_CODE and FLS_STOP_SEC_PRIVATERAM_CODE respectively in section 4.2.	1.0.5	04-May-2017

AUTOSAR MCAL R4.0.3 User's Manual
FLS Driver Component Ver.1.0.5
Embedded User's Manual

Publication Date: Rev.1.02, May 04, 2017

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.1

AUTOSAR MCAL R4.0.3
User's Manual

Renesas Electronics Corporation

R20UT3710EJ0102