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.0.01 Apr 2015

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of 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. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may 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.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

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

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti- crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its 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 be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be 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, or if you have any other inquiries.

(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
API	Application Programming Interface
AUTOSAR	AUTomotive Open System ARchitecture
BSW	Basic SoftWare
DEM	Diagnostic Event Manager
DET/Det	Development Error Tracer
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read Only Memory
FCL	Code Flash Library
FDL	Data Flash Library
FLS	FLaSh Driver
GNU	GNU's Not Unix
HW	HardWare
ID/Id	Identifier
MCAL	Microcontroller Abstraction Layer
NA	Not Applicable
RAM	Random Access Memory
ROM	Read Only Memory
RTE	Run Time Environment
SCHM/SchM	Scheduler Manager
SW	SoftWare

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:

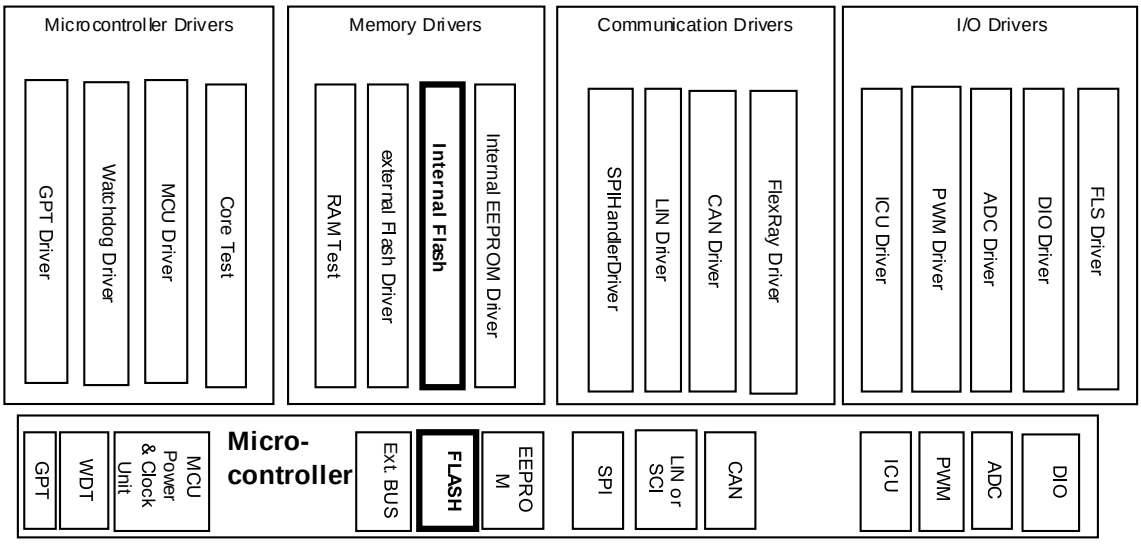

Figure 1-1 System Overview of FLS Driver Component in AUTOSAR MCAL Layer

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, this module uses a underlying FCL and FDL SW libraries for accessing and programming of flash.

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.

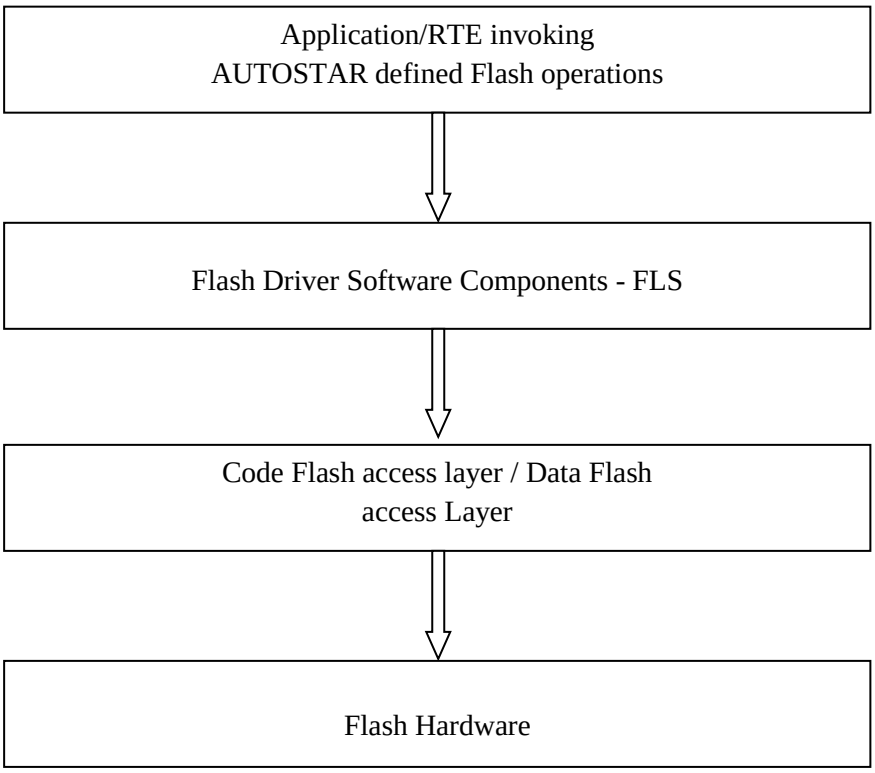

Figure 1-2 System Overview Of AUTOSAR Architecture

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 code flash and data flash memories.

The FLS component makes use of the FCL and FDL, which is an underlying software library contains the FCL and FDL APIs to perform the activities like accessing and programming the on-chip code flash and data flash hardware. This means FCL and FDL offers all functions and commands necessary to reprogram the application in a user friendly C language interface.

The FLS Component layer provides the wrapper for the Code Flash and Data Flash Library, which comprises of API for erase/write data to on-chip code flash and data flash memory of the device. The FLS Component conforms to the AUTOSAR standard and is implemented mapping to the AUTOSAR FLS Software Specification.

FCL and FDL acts as a programming interface between the Flash memory HW and higher level user applications; in this case it is the AUTOSAR FLS module. The FCL and FDL offers all required functions to handle code flash and data flash programming, that means programming the flash memory without programming tools and during program execution. FCL and FDL offer an easy- to-use interface to the internal firmware functionality. By calling the FCL and FDL library functions from user program, the contents of the flash memory can easily be rewritten in the field.

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.	r01uh0436ej0070_rh850p1x.pdf	0.70
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_BSW_MakefileInterface.pdf	0.3
7.	AUTOSAR BUGZILLA (http://www.autosar.org/bugzilla) Note: AUTOSAR BUGZILLA is a database, which contains concerns raised against information present in AUTOSAR Specifications.	-
8.	Code Flash Library for RH850 devices (FCL Library)	V2.00
9.	Data Flash Library for RH850 devices (FDL Library)	V2.00

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 "AUTOSAR_FLS_Tool_UserManual.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_Irq.c

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

X1X\P1x\modules\fls\src\fdl_descriptor.c
\fcl_descriptor.c
\r_fcl_hw_access.c
\r_fcl_hw_access_asm.850
\r_fcl_user_if.c
\r_fdl_hw_access.c
\r_fdl_user_if.c

X1X\P1x\modules\fls\include
\fdl_cfg.h
\r_fcl.h
\r_fcl_env.h
\r_fcl_global.h
\r_fcl_types.h
\r_fdl.h
\r_fdl_env.h
\r_fdl_global.h
\r_fdl_mem_map.h
\r_fdl_types.h
\r_typedefs.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.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, 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:

- 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 is 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.
- Program execution from Flash ROM is prohibited during flash programming.
  Therefore all FLS Components are located in RAM. The FLS components will be copied from Flash ROM to RAM during the startup. The FLS user has to assure that the application for programming control is also located to
  RAM area during ongoing flash programming operations.
- The FLS Driver Component's job processing function (Fls_MainFunction) is a polled function.
- Fls_SetMode does not provide any functionality to the user. Since there are no different flash memory access modes available. This API shall only be a dummy function.
- The configurations provided for fast mode operation are ignored by the Generation Tool and only configurations for normal mode operations are accepted as the underlying device and the FCL and FDL doesn't provide any functionality.
- 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 reFLSed if the length of the bytes to be erased is not in multiples of flash sector size.
- The configuration parameter FlsMaxEraseNormalMode which specifies the maximum data can be erased in one cycle of Fls_MainFunction() for data flash. The value for the parameter FlsMaxEraseNormalMode should be in multiples of data flash sector size.
- Fls_CF_read_memory_u08() and R_FDL_FCUFct_ReadOperation() will read the data from the flash memory depending on configuration of parameters FlsMaxReadNormalMode and FlsMaxCFReadNormalMode which specifies maximum data can be read in one cycle of Fls_MainFunction().
- Maximum value of FlsMaxReadNormalMode and FlsMaxCFReadNormalMode parameters specifies the size of a temporary buffer in RAM which is used when Fls_Read and Fls_Compare are called. The resulting RAM consumption has to be considered.
- R_FCL_I_write_memory_u32() and R_FDL_I_write_memory_u32() writes the data from target buffer to flash addresses depending on configuration of parameters FlsMaxWriteNormalMode and

FlsMaxCFWriteNormalMode which specifies maximum data can be written in one cycle of Fls_MainFunction().

- 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.
- The normal write verification using the direct memory read access is performed when DET is enabled.
- The processing of blank check operation will be applicable for Data flash only since no supporting APIs are in Code Flash Library.
- The component will support only the user mode of flash memory. Internal mode is not in the scope of this implementation.
- During activated flash environment, the access to 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 activated flash environment, 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.
- The FLS Driver Component will only invoke the call back notification functions. However, the implementation of the call back functions is the responsibility of the upper layer.
- The configuration parameter ‘FlsFclRamAddress’ minimum range is 0xFEDE0000 and the maximum range is ‘0xFEDFF4C8’ instead of ‘0xFEDFFFFF’ (RAM end address) as per device specification. Since the FCL routines are copied to RAM location during initialization. The RAM size required for FCL routines is 0xB38 bytes. The maximum range is provided with consideration of RAM size required for FCL routines.
- The user should ensure while configuring the parameter ‘FlsFclRamAddress’ value that the RAM area should not be effected the RAM area used for FLS driver RAM memory sections.
- When the parameter ‘FlsTimeoutMonitoring’ is configured as true then the timeout values for Erase, Write, Read and blank check are generating based on the parameters ‘FlsCFEraseTime’, ‘FlsCFWriteTime’ and ‘FlsCFReadTime’ and the values configured for ‘FlsMaxCFEraseNormalMode’, ‘FlsMaxCFWriteNormalMode’ and ‘FlsMaxCFReadNormalMode’ for code flash. Time out values are generating based on the parameters ‘FlsEraseTime’, ‘FlsWriteTime’ and ‘FlsReadTime’, ‘FlsBlankCheckTime’ and the values configured for ‘FlsMaxEraseNormalMode’, ‘FlsMaxWriteNormalMode’, ‘FlsMaxReadNormalMode’ for data flash.
- FLS driver supports three flash programming modes: Code Flash only (CF), both Code Flash and Data Flash (CFDF) and Data Flash only (DF). The flash programming mode can be configured via parameter "FlsAccess". The first two programming modes (CF, CFDF) are relevant for flash bootloader only. User application shall not program Code Flash during system runtime. From safety point of view FLS module in AUTOSAR BSW shall not include Code Flash programming functionality and shall supFLS Data Flash access only. Note: Flash bootloader is so far out of scope of AUTOSAR. User is responsible to verify and use FLS driver with proper configurations according to use-cases.
- Fls_Cancel Api will not affect/cancel the Fls_Suspend or Fls_Resume operations.
- In Fls_Suspend the timeout value for R_FDL_Handler will be 300 microseconds at 200MHz.

- 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.
- 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.

4.2. Preconditions

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

- The user should ensure that FLS Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- 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, fcl_descriptor.h, fcl_cfg.h, Fls_Cbk.h, fdl_descriptor.h, and Fls_PBcfg.c generated using FLS Generation Tool have to be linked along with FLS Driver Component source files.
- The FLS Driver Component needs to be initialized by calling Fls_Init() before calling any other Fls functions.
- Values for production code Event Ids should be assigned externally by the configuration of the DEM.
- 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.
- It is prohibited to call user code in ROM or FCL functions, which need ROM execution (i.e. Fls_Init()) during activated flash environment, this means during code flash programming operations. In case of ROM execution during code flash programming fatal error occurs.
- 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. FLS022, 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_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.

- 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 Fls_Erase, Fls_Write APIs on Data Flash only.
- The watchdog timer does not stop during the execution of the FCL.
- It is not possible to change the content of the request structure during command operation. If request data is changed during command operation, the library will crash.
- Before executing a write operation, please make sure the given address range is erased.
- 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.
- Cancel and suspend/resume operations are not allowed in case of two library instances as the effect is not evaluated.
- Standby is allowed but both instances have to consider that wakeup is required before continuing.
- Correct frequency configuration is essential for Flash programming quality and stability. Wrong configuration could lead to loss of data retention or Flash operation fail. If the CPU frequency is a fractional value, round up the value to the nearest integer. The clock reference of FLS driver is taken from the CPU clock.Do not change the CPU frequency during operation. If the frequency has to be changed, reinitialize the FLC with proper CPU frequency.
- All functions are not re-entrant. So, re-entrant calls of any FCL function must be avoided.
- 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.
- If a cancel request is accepted, during an on-going write, erase, or blank check operation and aprevious operation is already suspended, then both operations will be cancelled.
- It is not always possible to nest suspend and/or stand-by.

  E.g: Any operation ► suspend ► suspend – is not possible.
  Any operation ► stand-by ► stand-by – is not possible.
  Any operation ► stand-by ► 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 – is not possible

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.

```
#if (FLS_AR_VERSION == FLS_AR_HIGHER_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()

#elif (FLS_AR_VERSION == FLS_AR_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)

#endif

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

4.4. Deviation List

Table 4-1 FLS Driver Component Deviation List

Sl. No.	Description	AUTOSAR Bugzilla
1.	The fast mode parameters 'FlsMaxReadFastMode' and 'FlsMaxWriteFastMode' of the container 'FlsConfigSet are 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.	The flash access code is not loaded to RAM on job start.	-
5.	The parameters 'FlsDefaultMode' and '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.	FLS_E_READ_FAILED_DED error code will be reported to DEM if read job is failed when double bit ECC error is generated.	-
9.	The API Fls_GetVersionInfo is implemented as macro without DET error FLS_E_PARAM_POINTER.	-

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 4-2 User mode and Supervisor mode details when Data Flash enabled

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 because the Library initialization R_FDL_Init is failing while executing the API's R_FDL_IFct_ExeCodeInRAM which is located in r_fdl_hw_access.c file. This function will execute from the RAM and is fails due to ICCTRL have access permission in only supervisor mode.
2	Fls_Read	x	x	-
3	Fls_SetMode			
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	-
12.	Fls_Suspend	x	x	-
13.	Fls_Resume	x	x	-

Table 4-3 User mode and Supervisor mode details When Code Flash enabled

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 because the Library initialization R_FCL_Init is failing while executing the library functions in RAM. This is because the function “R_FCL_FCUFct_PrepareEnvironment" and internally calls the function "R_FCL_FCUFct_Clear_Cache" which clears the flash cache. The "R_FCL_FCUFct_Clear_Cache" function will execute STSR instruction (store contents of system register) for storing contents of ICCTRL (instruction cache control) to system register. Since the ICCTRL have the access permission in only supervisor mode and is fails in user mode.
2	Fls_Read	x	x	-
3	Fls_SetMode			
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	The Fls_MainFunction is failing in User mode because it will process all internal functions which will execute the R_FCL_Handler and _R_FCL_Execute functions in RAM. This is because the function "R_FCL_FCUFct_HandleMultiOperation" and internally calls the function "R_FCL_FCUFct_Clear_Cache" which clears the flash cache. The "R_FCL_FCUFct_Clear_Cache" function will execute STSR instruction (store contents of system register) for storing contents of ICCTRL (instruction cache control) to system register. Since the ICCTRL have the access permission in only supervisor mode and is fails in user mode.

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:

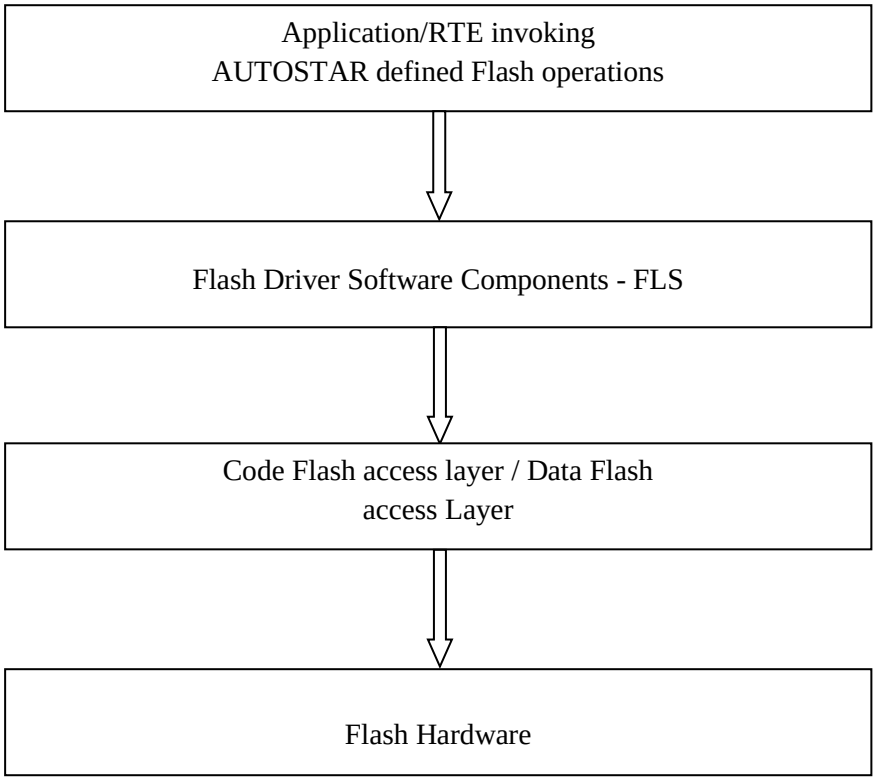

Figure 5-1 FLS Driver Component Architecture

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

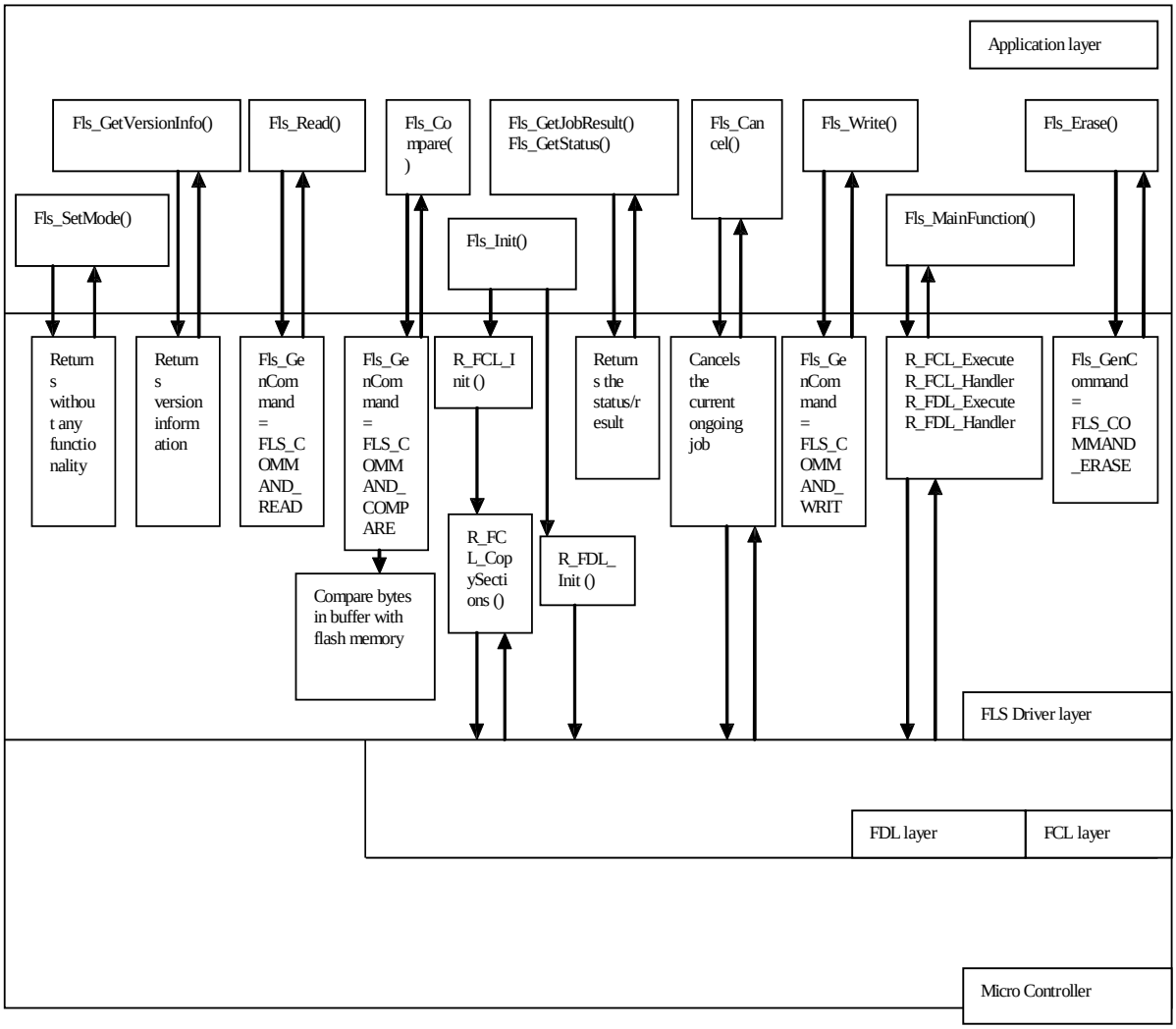

Figure 5-2 Component Overview Of FLS Driver Component

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

The 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 ongoing 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. FCL initialization API (R_FCL_Init) will be used for successful initialization of internal code flash programming environment and internal variables. After successful FCL initialization, R_FCL_Copysections function will be called for copying the FCL routines to RAM. FDL initialization API (R_FDL_Init) will be used for successful initialization of internal data flash programming environment and internal variables.

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(). In this job processing function the FCL library functions R_FCL_Execute and R_FCL_Handler are called to erase the requested code flash memory blocks. The FDL library functions R_FDL_Execute and R_FDL_Handler are called to erase the requested data flash memory blocks. In single cycle of Fls_MainFunction() call, R_FCL_Handler() erase the number of code flash memory blocks of flash memory depending on configuration of parameter FlsMaxCFEraseNormalMode and R_FDL_Handler() erase the number of data flash memory blocks of flash memory depending on configuration of parameter FlsMaxEraseNormalMode. The job is processed till the requested numbers of blocks are erased in the flash memory.
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 will be initiated first. If the cell is blank then the application buffer will 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 will be initiated by calling the FCL or FDL library function. This function reads the specified number of words from consecutive Flash addresses starting at the specified address and writes it into a buffer. In single cycle of Fls_MainFunction() call, R_FDL_FCUFct_ReadOperation will read the data from the data flash memory and Fls_CF_read_memory_u08 will read byte data from code flash memory depending on configuration of parameter FlsMaxReadNormalMode for data flash and FlsMaxCFReadNormalMode foe code flash. 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 will be processed by the job processing function Fls_MainFunction(). In this job processing function the writing of specified number of data bytes from buffer to flash memory will be initiated by calling either the FCL or FDL library function. These functions write the specified number of words from buffer to consecutive Flash addresses starting at the specified address. In single cycle of Fls_MainFunction() call, either R_FCL_Handler() or R_FDL_Handler() writes the data from target buffer to flash addresses depending on configuration of parameter FlsMaxWriteNormalMode for data

flash and FlsMaxCFWriteNormalMode for code flash. 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 will be processed by the job processing function Fls_MainFunction (). This compare operation will be implemented by calling either FCL or FDL library function. These functions initiate reading of defined words in flash and store it in the temporary buffer. Then actual data in application buffer will be compared with data in temporary buffer. Here data will be compared in terms of bytes. In single cycle of Fls_MainFunction() call, either R_FCL_Handler() or R_FDL_Handler() will read the data from the flash memory depending on configuration of parameter FlsMaxReadNormalMode for data flash and FlsMaxCFReadNormalMode for code 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 Cancellation Module

This sub-module provides the service for cancelling an ongoing memory request. After aborting the current ongoing 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.

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. All these services will be done by evaluating either FCL or FDL functions status and error codes from FCL or FDL library.

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 (Fls_MainFunction) that is invoked cyclically by the scheduler. This function will invoke the status check of the FCL or FDL library 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.

Fls_BlankCheck
This sub-module provides the service for performing blank check of the flash memory words. The request will be processed by the job processing function

Fls_MainFunction(). This function is invoked to perform the blank check of the single word. The FDL library function R_FDL_Handler is called to perform the requested data flash memory word blank check. 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().This API is applicable for Data Flash only.

Fls_ReadImmediate

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 (). This function reads the specified number of words from consecutive Flash addresses starting at the specified address and writes it into a buffer. In single cycle of Fls_MainFunction() call, R_FDL_Handler will 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 (). This API is applicable for Data Flash only.

Fls_Suspend

This sub-module provides the service of suspending the ongoing job.
Fls_Suspend is synchronous API. Fls_Suspend will block CPU (by calling FDL handler) for certain of time to perform suspend operation (R_FDL_SuspendRequest) and confirm the suspended status of the FDL library.

Fls_Resume

This sub-module provides the service of resume of previous suspended job.
Fls_Resume is synchronous API. Fls_Resume acknowledges the resume request by calling R_FDL_ResumeRequest command and it returns immediately.

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_SetFLMD0	FLMDCNT	32	RW	-	FLS_FLMDCNT
	FLMDPCMD	32	RW	-	FLS_FLMDPCMD
Fls_SetFHVE	FHVE3	32	RW	-	FLS_FHVE3
	FHVE15	32	RW	-	FLS_FHVE15

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:

- Programming of code flash
- 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 ongoing 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 ongoing job.
- Fls_Resume performs the resume of previous suspended job.

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 Software Generation Tool:

For only Code Flash access

- Fls_Cbk.h
- Fls_Cfg.h
- fcl_descriptor.h
- fcl_cfg.h

For only Data Flash access
- Fls_Cbk.h
- Fls_Cfg.h
- fdl_descriptor.h

For both Code Flash and Data Flash access
- Fls_Cbk.h
- Fls_Cfg.h
- fcl_descriptor.h
- fcl_cfg.h
- fdl_descriptor.h

The C source file generated by FLS Driver 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_Irq.h

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

The FLS specific C header files:
- Compiler.h
- Compiler_Cfg.h
- MemMap.h
- Platform_Types.h

The FCL and FDL library header and source files:
- r_fcl.h
- r_fcl_env.h
- r_fcl_global.h

- r_fcl_types.h
- r_fdl.h
- r_fdl_env.h
- r_fdl_global.h
- r_fdl_types.h
- r_fdl_mem_map.h
- fdl_cfg.h
- r_typedefs.h
- r_fdl_user_if.c
- r_fdl_hw_access.c
- r_fcl_user_if.c
- r_fcl_hw_access_asm.850
- r_fcl_hw_access.c
- fcl_descriptor.c
- fdl_descriptor.c

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 FLS Software Generation 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.
fcl_cfg.h	This file contains the device specific parameter that needs to be configured for different devices.
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 prototypes for internal functions of Flash Wrapper Component.
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.c	This file contains the implementation of all APIs.
Fls_Ram.c	This file contains the global variables used by FLS Driver Component.
Fls_Internal.c	This file contains the Internal functions implementations of flash wrapper component.

File	Details
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.
Platform_Types.h	This file provides provision for defining platform and compiler dependent types.
fdl_descriptor.h	This file contains FDL run-time configuration descriptor variable related defines.
fdl_cfg.h	This file contains FDL pre-compile definitions.
r_fcl.h	Header file containing FCL user interface function prototypes.
r_fcl_types.h	This File contains the FCL user interface type definitions.
r_fcl_global.h	This file contains FCL Flash programming global type defines, function and variables.
r_fcl_env.h	This file contains FCL Flash programming hardware related definitions.
r_fdl.h	Header file containing FDL user interface function prototypes.
r_fdl_types.h	This File contains the FDL user interface type definitions.
r_fdl_global.h	This file contains FDL Flash programming global type defines, function and variables.
r_fdl_env.h	This file contains FDL Flash programming hardware related definitions.
r_typedefs.h	This file contains renesas standard type definitions.
fcl_descriptor.h	This file contains FCL run-time configuration descriptor variable related defines.
r_fcl_user_if.c	This file contains the FCL user interface functions.
r_fcl_hw_access.c	This file contains the FCL hardware interface functions.
fcl_descriptor.c	This file contains the Descriptor variable definition.
r_fdl_user_if.c	This file contains the FDL user interface functions.
r_fdl_hw_access.c	This file contains the FDL hardware interface functions.
fdl_descriptor.c	This file contains the Descriptor variable definition.
r_fcl_hw_access_asm.850	This file contains the FCL hardware interface functions.
Fls_PBTypes.h	This file contains the type definitions of post build parameters. It also contains the macros used by the FLS Driver Component.
r_fdl_mem_map.h	This file contains FDL section mapping definitions.
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.

Chapter 9 Generation Tool Guide

For information on the FLS Driver Code Generation Tool, please refer “AUTOSAR_FLS_Tool_UserManual.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_VersionInfoType

10.1.2. Other Module Types

In this section all types included from the Dem.h are listed.
• Dem_EventIdType
• Dem_EventStatusType
• Memif_JobResultType
• Memif_StatusType

10.2. Type Definitions

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

Table 10-1 Fls_CommandType

Name:	Fls_CommandType	
Type:	Enumeration	
Range:	FLS_COMMAND_NONE	Used to Set the command to none.
	FLS_COMMAND_ERASE	Command used for Erase.
	FLS_COMMAND_WRITE	Command used for Write.
	FLS_COMMAND_READ	Command used for Read.
	FLS_COMMAND_COMPARE	Command used for Compare.
	FLS_COMMAND_BLANKCHECK	Command used for Blank check.
	FLS_COMMAND_READ_IMM	Command used for fast Read.
Description:	Enumeration for driver commands.	

Table 10-2 Fls_FlashType

Name:	Fls_FlashType	
Type:	Enumeration	
Range:	FLS_DF_ACCESS	Indicate the operations to be performed for data flash.
	FLS_CF_ACCESS	Indicate the operations to be performed for code flash.
Description:	Enumeration for flash access type.	

10.3. Function Definitions

Table 10-3 Function Definitions

Sl. No	API's	API's specific
1.	Fls_Init	-
2.	Fls_Erase	-
3.	Fls_Write	-
4.	Fls_Cancel	-
5.	Fls_GetStatus	-
6.	Fls_GetJobResult	-
7.	Fls_Read	-
8.	Fls_Compare	-
9.	Fls_SetMode	-
10.	Fls_GetVersionInfo	-
11.	Fls_MainFunction	-
12.	FlsJobEndNotification	-
13.	FlsJobErrorNotification	-
14.	Fls_BlankCheck	-
15.	Fls_ReadImmediate	-
16.	Fls_Suspend	-
17.	Fls_Resume	-

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
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_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
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 read, write, erase or compare 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

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_CFProcessEraseCommand and Fls_DFProcessEraseCommand
Source of Error	When the Erase API service is invoked and the FCL or FDL returns the job result as failed. Error will be reported by the job processing function.
Sl. No.	**2**
Error Code	FLS_E_WRITE_FAILED
Related API(s)	Fls_CFProcessWriteCommand and Fls_DFProcessWriteCommand
Source of Error	When the Write API service is invoked and the FCL or FDL returns the job result as failed. Error will be reported by the job processing function.
Sl. No.	**3**
Error Code	FLS_E_READ_FAILED
Related API(s)	Fls_CFProcessReadCommand and Fls_DFProcessReadCommand
Source of Error	When the Read API service is invoked and the FCL or FDL returns the job result as failed. Error will be reported by the job processing function.
Sl. No.	**4**
Error Code	FLS_E_COMPARE_FAILED
Related API(s)	Fls_CFProcessCompareCommand and Fls_DFProcessCompareCommand

Source of Error	When the Compare API service is invoked and the FCL or FDL returns the job result as failed. Error will be reported by the job processing function.
Sl. No.	**5**
Error Code	FLS_E_READ_FAILED_DED
Related API(s)	Fls_DFProcessReadCommand
Source of Error	When the Read API service is invoked and the FDL returns the job result as failed when double bit ECC error is generated. Error will be reported by the job processing function.

Chapter 12 Memory Organization

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

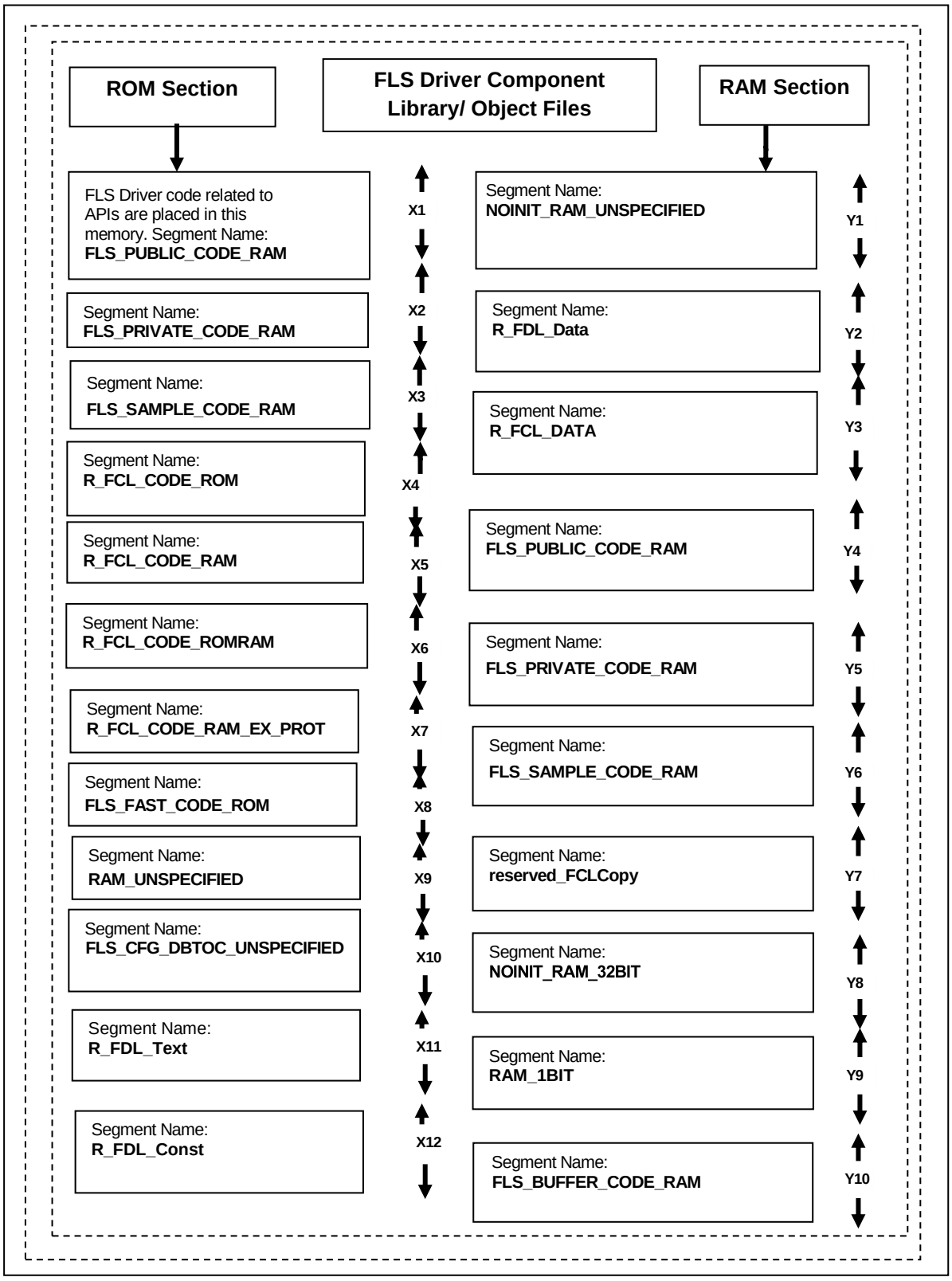

Figure 12-1 FLS Driver Component Memory Organization

ROM Sections:

FLS_PUBLIC_CODE_RAM (X1): This section consists of FLS Driver Component internal functions and scheduler function that can be located in code memory. This section is copied on to RAM by the GHS start-up routines.

FLS_PRIVATE_CODE_RAM (X2): This section consists of FLS Driver Component APIs and FCL functions that can be located in code memory. This section is copied on to RAM by the GHS start-up routines.

FLS_SAMPLE_CODE_RAM (X3): This section needs to be aligned at the end of FLS code sections in RAM, for exception protection.

R_FCL_CODE_ROM (X4): This section needs to be aligned at the end of FCL code sections in RAM, for exception protection. This section is copied to RAM by FCL library internal mechanism.

R_FCL_CODE_RAM (X5): This section contains the code executed at the beginning of self-programming. This code is executed at the original location, e.g. internal Flash. The library initialization is part of this section

R_FCL_CODE_ROMRAM (X6): This section contains the FCL library. This section is copied to RAM by FCL library internal mechanism.

R_FCL_CODE_RAM_EX_PROT (X7): This section contains the FCL library. This section is copied to RAM by FCL library internal mechanism or remains in the ROM depending on FCL mode setting.

FLS_FAST_CODE_ROM (X8): Interrupt functions of FLS Driver Component code that can be located in code memory.

R_FDL_Text (X11): This section consists of the FDL code. This can be located in code memory.

R_FDL_Const (X12): This section consists of the constants in ROM that are used by FDL software component. This can be located in code memory.

RAM Sections: Following are the Ram sections mapped.

NOINIT_RAM_UNSPECIFIED (Y1): 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.

R_FDL_Data (Y2): This section consists of the global RAM pointer variables that are used by FDL software component. This can be located in data memory.

R_FCL_DATA (Y3): This section consists of the global RAM pointer variables that are used by FCL software component. This can be located in data memory

FLS_PUBLIC_CODE_RAM (Y4): This section consists of FLS Driver

Component. These sections are copied on to RAM by the GHS start-up routines.

FLS_PRIVATE_CODE_RAM (Y5): This section consists of FLS internal software component. These sections are copied on to RAM by the GHS start-up routines.

FLS_SAMPLE_CODE_RAM (Y6): This section needs to be aligned at the end of the FLS code sections in RAM, for exception protection. These sections are copied on to RAM by the GHS start-up routines

reserved_FCLCopy (Y7): This section is required for locating the underlying FCL library component. It must be assured to locate this section at the RAM start. This needs to be in-line with FLS configuration parameter “FclRamAddress”.

NOINIT_RAM_32BIT (Y8): 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.

RAM_1BIT (Y9): 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.

FLS_BUFFER_CODE_RAM(Y10): This section consists of the global RAM variables used for temporary buffer that are initialized by start-up code and used internally by FLS Driver Component and other software components. The specific sections of respective software components will be merged into this RAM section accordingly.

RAM_UNSPECIFIED(X9): This section consists of the global RAM variables that are generated by FLS Driver Component Generation Tool. This can be located in data memory.

FLS_CFG_DBTOC_UNSPECIFIED(X10): This section consists of FLS Driver Component database table of contents generated by the FLS Driver Component Generation Tool. This can be located in code memory.

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 layers users and the mapping of the channels to the hardware units is provided in the following sections:

13.1.1. Translation header File

The translation header file 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:

- Programming of code flash
- Erase memory sectors
- Read flash contents to the application memory
- Fast read immediate to the application memory without blankcheck.

- Validate flash contents comparing with the application memory
- Cancel the ongoing 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, write and read operation.
- Resume the erase, write and read operation.

13.1.3. Parameter Definition File

Table 13-1 PDF information for P1M

PDF files	Devices supported
R403_FLS_P1M_04_05.arxml	701304, 701305
R403_FLS_P1M_10_to_15.arxml	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

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 of only data flash or only code flash or for both code flash and data flash supported. Depending on the configured use-case, the Sample Application is built based on setting of the flag 'FLS_ACCESS_FLAG' in ‘App_Fls_P1M_Sample.mak’ file to either ‘CODEFLASH_ACCESS’ or ‘DATAFLASH_ACCESS’ or ‘CFDF_ACCESS’ by the user during compile time in order to compile corresponding library source files.

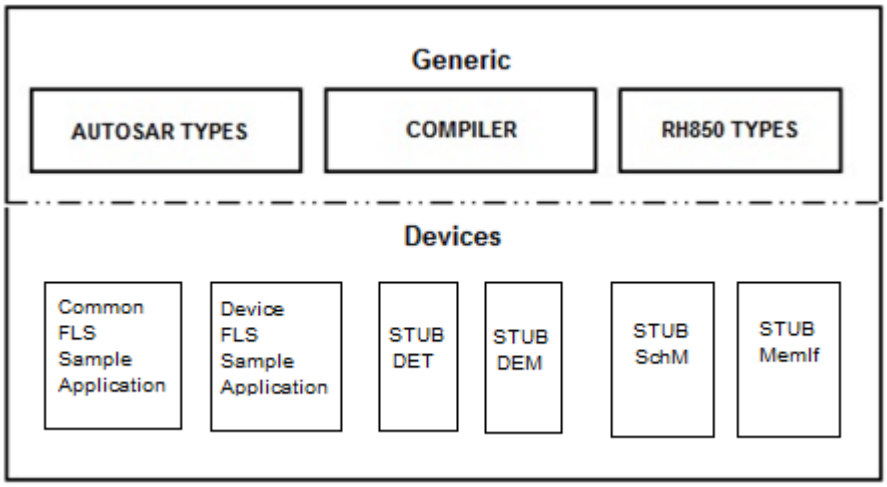

Figure 13-1 Overview Of FLS Driver Sample Application

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.arxml

\R403_FLS_P1M_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\fdl_descriptor.h
\include\Fls_Cfg.h
\include\Fls_Cbk.h

\config\App_FLS_P1M_701304_Sample.arxml
\config\App_FLS_P1M_701304_Sample.html
\config\App_FLS_P1M_701304_Sample.one

\config\App_FLS_P1M_701305_Sample.arxml
\config\App_FLS_P1M_701305_Sample.html
\config\App_FLS_P1M_701305_Sample.one

\config\App_FLS_P1M_701310_Sample.arxml
\config\App_FLS_P1M_701310_Sample.html
\config\App_FLS_P1M_701310_Sample.one

\config\App_FLS_P1M_701311_Sample.arxml
\config\App_FLS_P1M_701311_Sample.html
\config\App_FLS_P1M_701311_Sample.one

\config\App_FLS_P1M_701312_Sample.arxml
\config\App_FLS_P1M_701312_Sample.html
\config\App_FLS_P1M_701312_Sample.one

\config\App_FLS_P1M_701313_Sample.arxml
\config\App_FLS_P1M_701313_Sample.html
\config\App_FLS_P1M_701313_Sample.one

\config\App_FLS_P1M_701314_Sample.arxml
\config\App_FLS_P1M_701314_Sample.html
\config\App_FLS_P1M_701314_Sample.one

\config\App_FLS_P1M_701315_Sample.arxml
\config\App_FLS_P1M_701315_Sample.html
\config\App_FLS_P1M_701315_Sample.one

\config\App_FLS_P1M_701318_Sample.arxml
\config\App_FLS_P1M_701318_Sample.html
\config\App_FLS_P1M_701318_Sample.one

\config\App_FLS_P1M_701319_Sample.arxml
\config\App_FLS_P1M_701319_Sample.html
\config\App_FLS_P1M_701319_Sample.one

\config\App_FLS_P1M_701320_Sample.arxml
\config\App_FLS_P1M_701320_Sample.html
\config\App_FLS_P1M_701320_Sample.one

\config\App_FLS_P1M_701321_Sample.arxml
\config\App_FLS_P1M_701321_Sample.html
\config\App_FLS_P1M_701321_Sample.one

\config\App_FLS_P1M_701322_Sample.arxml
\config\App_FLS_P1M_701322_Sample.html
\config\App_FLS_P1M_701322_Sample.one

\config\App_FLS_P1M_701323_Sample.arxml
\config\App_FLS_P1M_701323_Sample.html
\config\App_FLS_P1M_701323_Sample.one

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 will in turn calls R_FCL_Init() and R_FCL_Copysections() which will initialize FCL internal variables. 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 ongoing 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 does not provide any functionality.
- 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 read the byte data from code flash memory.

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. Since neither the interrupt vector table nor the interrupt handler routines, which are normally located in the flash memory, are accessible while self-programming is active, the timer interrupt is not used for this purpose. In order to do so, interrupt acknowledges have to be re-routed to non-flash memory. This can be achieved by suitably modifying the start-up code to access the system registers (SW_CFG/SW_BASE respectively EH_CFG/ EH_BASE) to reroute the interrupt vector of the timer interrupt to the RAM area.

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, 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 13.3.2.1 *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 701312
1.	ROM	ROM.FLS_PUBLIC_CODE_RAM	3040
		ROM.FLS_PRIVATE_CODE_RAM	2312
		ROM.FLS_APPL_CODE_RAM	1986
		R_FCL_CODE_ROM	0
		R_FCL_CODE_RAM	0
		R_FCL_CODE_ROMRAM	0
		R_FCL_CODE_RAM_EX_PROT	32
		FLS_FAST_CODE_ROM	212

2.	RAM	reserved_FCLCopy	9216
		FLS_PUBLIC_CODE_RAM	3040
		FLS_PRIVATE_CODE_RAM	2312
		FLS_APPL_CODE_RAM	1986
		R_FCL_DATA	0
		R_FDL_Data	84
		NOINIT_RAM_UNSPECIFIED	4
		RAM_UNSPECIFIED	44
		FLS_BUFFER_CODE_RAM	100

Table 13-4 ROM/RAM Details Without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701312
1.	ROM	ROM.FLS_PUBLIC_CODE_RAM	3040
		ROM.FLS_PRIVATE_CODE_RAM	2312
		ROM.FLS_APPL_CODE_RAM	2012
		R_FCL_CODE_ROM	0
		R_FCL_CODE_RAM	0
		R_FCL_CODE_ROMRAM	0
		R_FCL_CODE_RAM_EX_PROT	32
		FLS_FAST_CODE_ROM	212

2.	RAM	reserved_FCLCopy	9216
		FLS_PUBLIC_CODE_RAM	3040
		FLS_PRIVATE_CODE_RAM	2312
		FLS_APPL_CODE_RAM	2012
		R_FCL_DATA	0
		R_FDL_Data	84
		NOINIT_RAM_UNSPECIFIED	4
		RAM_UNSPECIFIED	44
		FLS_BUFFER_CODE_RAM	100

Remark The section “reserved_FCLCopy” might not be the actual RAM area, but only the ‘reserved’ area.

13.3.2 Stack Depth

The worst-case stack depth for FLS Driver Component is for the typical configuration provided in Section 13.3.2.1 *Configuration Example*.

Table 13-5 Stack Depth Table

Sl. No	Device Name	Stack Depth (in Bytes)
1.	R7F701312	384

13.3.3 Throughput Details

The throughput details of the APIs for the configuration mentioned in the Section 13.3.2.1*Configuration Example* are listed here. The clock frequency used to measure the throughput is 80MHz for all APIs.

Table 13-6 Throughput Details Of The APIs

Sl. No.	API Name	Throughput in µ seconds for 701312	Remarks
1.	Fls_Init	487.80	-
2.	Fls_Erase	9.450	-
3.	Fls_Write	9.630	-
4.	Fls_Cancel	1.440	-
5.	Fls_GetStatus	0.630	-
6.	Fls_GetJobResult	0.630	-
7.	Fls_Read	2.520	-
8.	Fls_Compare	2.430	-

Sl. No.	API Name	Throughput in µ seconds for 701312	Remarks
9.	Fls_SetMode	NA	This API does not provide any functionality
10.	Fls_GetVersionInfo	0.540	-
11.	Fls_BlankCheck	5.940	-
12.	Fls_ReadImmediate	2.430	-
13.	Erase Operation	9.090	This is the time taken for the complete erase operation of 256 bytes data length.
14.	Write Operation	9.990	This is the time taken for the complete write operation of 256 bytes data length.
15.	Fls_BlankCheck operation	6.120	This is the time taken for performing blank check operation of 256 bytes data length.
16.	Fls_ReadImmediate	2.610	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	2.340	This is the time taken for the complete read operation of 256 bytes data length.
18.	Compare Operation	2.520	This is the timetaken for the complete compare operation of 256 bytes data length.

Sl. No.	API Name	Throughput in µ seconds for 701312	Remarks
19.	FLENDNM_ISR operation	8.550	This is the time taken for the complete Erase of 1 block data length.
		9.360	This is the time taken for the complete Write of 1 word data length.
20	Fls_Suspend	38.430	-
21	Fls_Resume	6.587	-

Chapter 14 Release Details

FLS Driver Software

Version: 1.3.1

Revision History

Sl.No.	Description	Version	Date
1.	Initial Version	1.0.0	28-Oct-2013
2.	As per CR 066, below changes are made. 1. The Figure "Component Overview of FLS Driver Component “is alignment corrected. 2. FLS driver component version information is updated. 3. In chapter 6 Register Details are updated. 4. Chapter 2 is updated for referenced documents version. 5. Section 4.1 is updated for removing information about “FlsWriteInternalVerify”. 6. Section 4.1 and 5 are updated to replace API name ‘R_FCL_I_read_memory_u08’ by ‘Fls_CF_read_memory_u08’. 7. Section 4.1, General is updated for adding information about time out values of erase, read, write and blank check. 8. Section 4.2 is updated for removing Fls_LengthType restriction on size as precondition. 9. Section 4.5 is updated for adding supervisor and user mode details for added devices. 10. Section 13.1.1 is updated for adding the device names. 11. Section 13.2 is updated for assembler and linker details. 12. Section 13.3 is updated for naming convention change of parameter definition files. 13. “FLS_E_PARAM_POINTER” is removed from Table11-1. 14. Section13.4 is updated for RAM/ROM usage details.	1.0.1	22-Jan-2014
3.	As per CR 107 Following changes are made: 1. Chapter 2: ‘Reference document’ is updated. 2. Chapter 4: ‘Forethoughts’ is updated for Following: • Section 4.1 ‘General’ is updated for description. • Section 4.2 ‘Preconditions is updated for description to add Fls_BlankCheck and Fls_ReadImmediate API. • Section 4.3 ‘Data consistency’ is updated for description to add macro for version. • Section 4.4 ‘deviation List’ is updated to add Fls_GetVersionInfo API. • Section 4.4 ‘User mode and supervisor mode’: Table 4-2 is updated for ‘Fls_Cancel API’ and to add ‘Fls_BlankCheck’ and ‘Fls_ReadImmediate’ API. • Section 4.4: Table 4.3 is updated to add ‘Fls_Suspend’ and ‘Fls_Resume’ API. 3. Chapter 5: ‘Architecture Details’ is updated to add ‘Fls_Suspend’, ‘Fls_Resume’, ‘Fls_BlankCheck’ and ‘Fls_ReadImmediate’ API. 4. Chapter 6: ‘Register Details’ is updated for register access. 5. Chapter 7 and Section 13.1.2 is updated to add ‘Fls_Suspend’, ‘Fls_Resume’, ‘Fls_BlankCheck’ and ‘Fls_ReadImmediate’ services. 6. Chapter 8 is updated to add ‘Fls_Irq.h’ and ‘Fls_Irq.c’ 7. Chapter 10 is updated for following: • Section 10.2 is updated to remove Fls_‘VerifyType’ type definition. • Section 10.3 is updated to add ‘Fls_Suspend’, ‘Fls_Resume’, ‘Fls_BlankCheck’ and ‘Fls_ReadImmediate’ API.	1.0.2	02-Sep-2014

Sl.No.	Description	Version	Date
	8. Chapter 12: 'Memory Organization':Figure 12-1 and desrcription is updated to add 'FLS_FAST_CODE_ROM', 'RAM_UNSPECIFIED','FLS_CFG_DBTOC_UNSPECIFIED' and 'FLS_BUFFER_CODE_RAM'. 9. Chapter 13 is updated for Following: • Chapter 13 is updated to remove R7F701300-R7F701303, R7F701306- R7F701309 and to add R7F701318- R7F701323 P1M supported devices. • Section 13.1.3 and section 13.1.4 are added for parameter definition files and for ISR function respectevely. • Section 13.2.1 is updated for compiler, Linker and Assembler options. • Section 13.3 is updated for Sample Application path and description to add 'Fls_ReadImmediate' and 'Fls_BlankCheck' API. • Section 13.4: 'Memory and throughput' is updated. 10. Chapter 14: 'Release Details' is updated for FLS Driver Version.		
4	As per CR 31 Following changes are made: 1. Table 11-1 DET Errors of FLS Driver Component is updated. 2. Section 13.3.1 Sample Application Structure is updated. 3. Section 13.4.1 RAM/ROM details is updated. 4. Chapter 14 Release Details is updated to correct Chapter heading. 5. Section 4.1 is updated to change the description from 'F1L' to 'P1M'.	1.0.3	14-Oct-2014
5	As per CR 82 Following changes are made: 1. Section 4.5 is updated for user and supervisory mode. 2. Section 13.2.1 is updated for removal –c option. 3. Chapter 3 is updated for update in trxml file path of sample application. 4. Page 64 is updated for header correction. 5. Section 13.1.2 is updated for suspend and resume services. 6. Chapter 5 is updated for page number and header. 7. Section 13.4.3 is updated for throughput.	1.0.4	02-Dec-2014
6	The following changes are made: 1. Chapter 2: 'Reference document' is updated 2. As part of device support activity for R7F701304, R7F701305, R7F701313, R7F701315, R7F701318 to R7F701323 updated sections 3.1.1, 13.1.1, 13.1.2, 13.3.1. 3. Updated version number and copyright year. 4. Updated section 13.4 for memory and throughput. 5. Removed section Compiler,Linker and Assembler in Chapter13. 6. Removed Test_Application_P1x.trxml path from Section 3.1.1. 7. Section 4.1 is updated for adding note in Forethoughts. 8. Chapter 12 is updated. 9. Section 4.2. Preconditions is updated. 10. Updated Tables 4-2 and 4-3 in Section 4.5. 11. Chapter 14: 'Release Details' is updated for FLS Driver Version	1.0.5	24-Apr-2015

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

Publication Date: Rev.0.01, April 24, 2015

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0

AUTOSAR MCAL R4.0.3
User's Manual

Renesas Electronics Corporation