User's Manual

AUTOSAR MCAL R4.0.3

User's Manual

FLS Driver Component Ver.1.0.2

Embedded User's Manual

Target Device:
RH850/P1x-C

Renesas Electronics
www.renesas.com

Rev.1.00 Feb 2017

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
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-C 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:

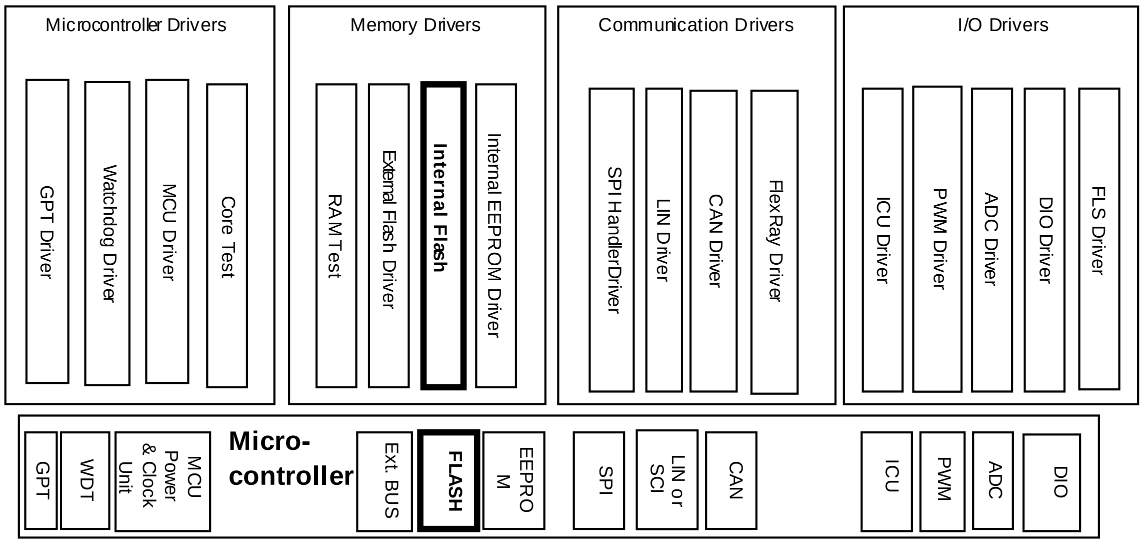

Figure 1-1 System Overview of FLS Driver Component in AUTOSAR MCAL Layer

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.

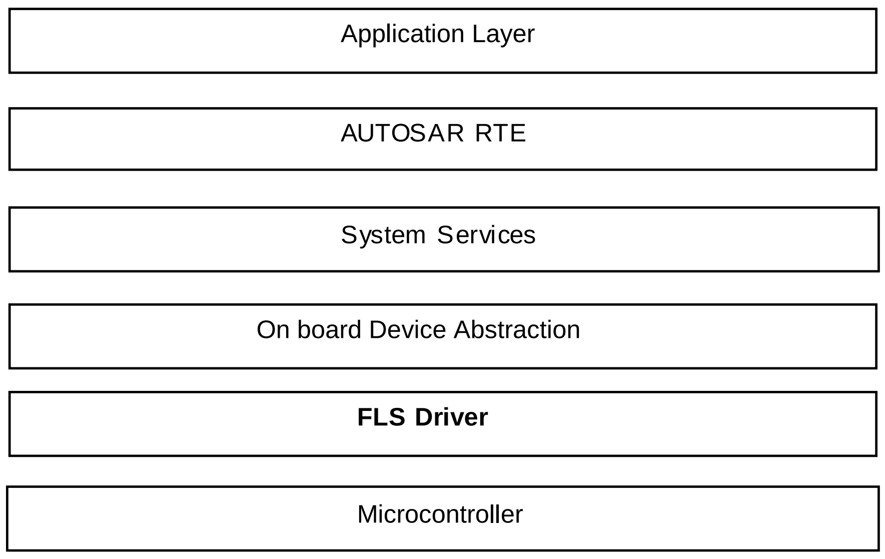

Figure 1-2 System Overview of AUTOSAR Architecture

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	Contents
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 (P1x-C Specific Information)	This section provides the P1x-C 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.	RH850/P1H-C Document User’s Manual: Hardware (r01uh0517ej0100_rh850p1x-c_Open.pdf)	1.0
2.	Autosar R4.0 Specification of FLS Driver (AUTOSAR_SWS_FlashDriver.pdf)	3.2.0
3.	AUTOSAR BUGZILLA (http://www.autosar.org/bugzilla) Note: AUTOSAR BUGZILLA is a database, which contains concerns raised against information present in AUTOSAR Specifications.	-
4.	Specification of Compiler Abstraction (AUTOSAR_SWS_CompilerAbstraction.pdf)	3.2.0
5.	Specification of Memory Mapping (AUTOSAR_SWS_MemoryMapping.pdf)	1.4.0
6.	Specification of Platform Types (AUTOSAR_SWS_PlatformTypes.pdf)	2.5.0

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 “R20UT3642EJ0100 - 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

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_Private_Fcu.h
                                       \Fls_RegWrite.h

X1X\ P1x-C
\modules\fls\sample_application\<SubVariant>\make\<Complier>
                                   \App_FLS_P1X-C_Sample.mak

X1X\P1x-C\modules\fls\sample_application\make\
                         <Complier>\App_FLS_P1x-C_Sample.ld

X1X\P1x-C\modules\fls\Sample_application\<SubVariant>\obj
```

X1X\P1x-C\modules\fls\generator\R403_FLS_P1x-C_BSWMDT.arxml.

X1X\P1x-C\modules\fls\user_manual

(User manuals will be available in this folder)

Note:
1. <Complier> can be ghs.
2. <AUTOSAR_version> should be 4.0.3.
3. <SubVariant> can be P1H-C, P1H-CE, P1M-C.

Chapter 4 Forethoughts

4.1. General

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

- AUTOSAR FLS driver supports Data Flash access only. Code Flash access is out of scope.
- 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.
- The FLS Driver Component's job processing function (Fls_MainFunction) is a polled function.
- 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_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.
- 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_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.
- 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).
- 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.
- 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.
- 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 processing of blank check operation is applicable for Data flash only.
- 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.
- 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.
- User application shall not program Code Flash in the application mode. Code Flash shall only be programmed in safe environment in the boot mode. User application shall not write safety related data into Code Flash or Data Flash during driving cycle for safety critical applications.

- 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.
- 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 P1x-C 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.
- 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.

- Fls_Read API performs the reading of the flash memory with a blank check operation before read but Fls_ReadImmediate performs the reading of the flash memory without a blank check operation before read.
- Fls_BlankCheck API used to perform the blank check of flash memory before reading the flash memory, Fls_Suspend API performs the suspend of the ongoing write, read or erase job, Fls_Resume API resumes the suspended job.
- The time-out implementation for erase/write operations will not done by FLS module and it needs to be carried out by the upper layer.

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:

- 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.
- 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 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.
- 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.
- 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.
- Fls_Cancel will stop the Flash programming hardware synchronously, thus, the ongoing Flash modification operation (Erase/Write) will be interrupted. This can result in undefined state of Flash block(s) the same way as general interruptions mentioned above.
- 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.
- 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.
- Fls_Init function temporarily disables Code Flash. During this time, since the Code Flash is not available, the FLS code is executed from internal RAM (allocated space on stack). 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.
- 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.
- 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.
- 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.

- 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_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 Erase, Write, and Blank Check operations only.
- 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.
- If a cancel request is accepted, during an ongoing 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 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.
- 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.
- 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–isn't possible
- 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.
- Any internal error occurred due to hardware failure during mode switching or issuing forced stop command shall set the driver status to UNINIT and job status to JOB_FAILED.
- The user shall configure the exact Module Short Name Fls in configurations as specified in config.xml file and the same shall be given in command line.
- The user should configure FACIn Unit properly to avoid hardware resource conflict.
- FLS initialization failure may happen in the system runtime due to transient hardware faults. The User shall enable DET in order to get FLS_E_UNINIT in case of initialization failure. If Fls_GetStatus API is used, upper layer can use this API to get MEMIF_UNINIT in case of initialization failure.

4.3. Data Consistency

To support the reentrancy and interrupt services, the FLS Software component will ensure the data consistency while accessing their own RAM storage or hardware registers.

```
#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 for shared resources:

- FLS_DRIVERSTATE_DATA_PROTECTION
- FLS_REGISTER_PROTECTION
- FLS_CODE_FLASH_DISABLED

These functions can be disabled by disabling the configuration parameter ‘FlsCriticalSectionProtection’.

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
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected : Fls_GstVar.GulJob StartAddress Fls_GstVar.GulJob EndAddress
Fls_Write	FLS_REGISTER_PROTECTION	HW Registers: FSADDR FEADDR
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected : Fls_GstVar.GulJob StartAddress Fls_GstVar.GulJob EndAddress
Fls_MainFunction	FLS_REGISTER_PROTECTION	HW Registers: DFERSTC DFERSTR DFERRINT
	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected : Fls_GstVar.GulJob StartAddress Fls_GstVar.pBuffer Address
Fls_Resume	FLS_REGISTER_PROTECTION	HW Registers: DFERSTC DFERSTR DFERRINT
Fls_Read	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected : Fls_GstVar.GulRea dAddress

API Name	Exclusive Area Type	Protected Resources
Fls_Compare	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during compare operation : Fls_GstVar.GulRequestedLength Fls_GstVar.GucOffset Fls_GstVar.GulReadAddress Fls_GstVar.pTempBufferAddress Fls_GstVar.pBufferAddress Fls_GstVar.GulCurrentLength Fls_GstVar.GucGenCommand Fls_GenState Fls_GenJobResult
Fls_Cancel	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during cancel operation : Fls_GenState Fls_GenJobResult Fls_GstVar.GucGenCommand
Fls_BlankCheck	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during blank check operation : Fls_GstVar.GucGenCommand Fls_GenJobResult Fls_GenState
Fls_ReadImmediate	FLS_DRIVERSTATE_DATA_PROTECTION	Driver state data is protected during read immediate operation : Fls_GstVar.GulReadAddress

Note:
The highest measured duration of a critical section was 140.725 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 parameters 'FlsMaxReadFastMode' and 'FlsMaxWriteFastMode' of the container 'FlsConfigSet are unused.	-
2.	The parameters 'FlsAcLoadOnJobStart' and 'FlsUseInterrupts' of the container 'FlsGeneral' is unused.	-
3.	The parameters 'FlsDefaultMode' and 'FlsProtection', FlsAcWrite' and 'FlsAcErase' of the container 'FlsConfigSet' are unused.	-

4.	The parameters 'FlsAcLocationErase', 'FlsAcLocationWrite', 'FlsAcSizeErase' and 'FlsAcSizeWrite' of the container 'FlsPublishedInformation' are unused.	-
5.	The component will support only the on-chip flash memory. External flash is not in the scope of this implementation.	-
6.	FLS_E_READ_FAILED_DED error code will be reported to DEM if read job is failed when double bit ECC error is generated.	-
7.	**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.	-
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.	

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 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. 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	-

Sl. No	API Name	User Mode	Supervisor Mode	Known limitation in User mode
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:

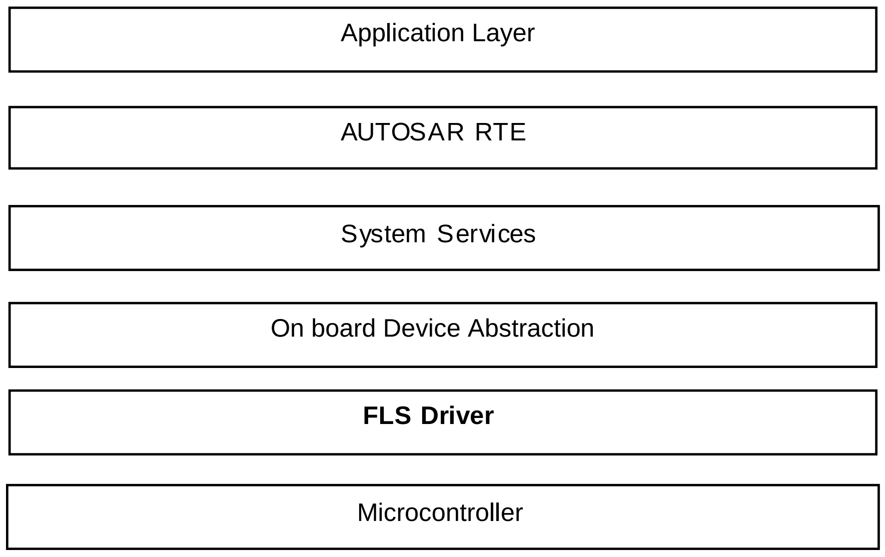

Figure 5-1 FLS Driver Component Architecture

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

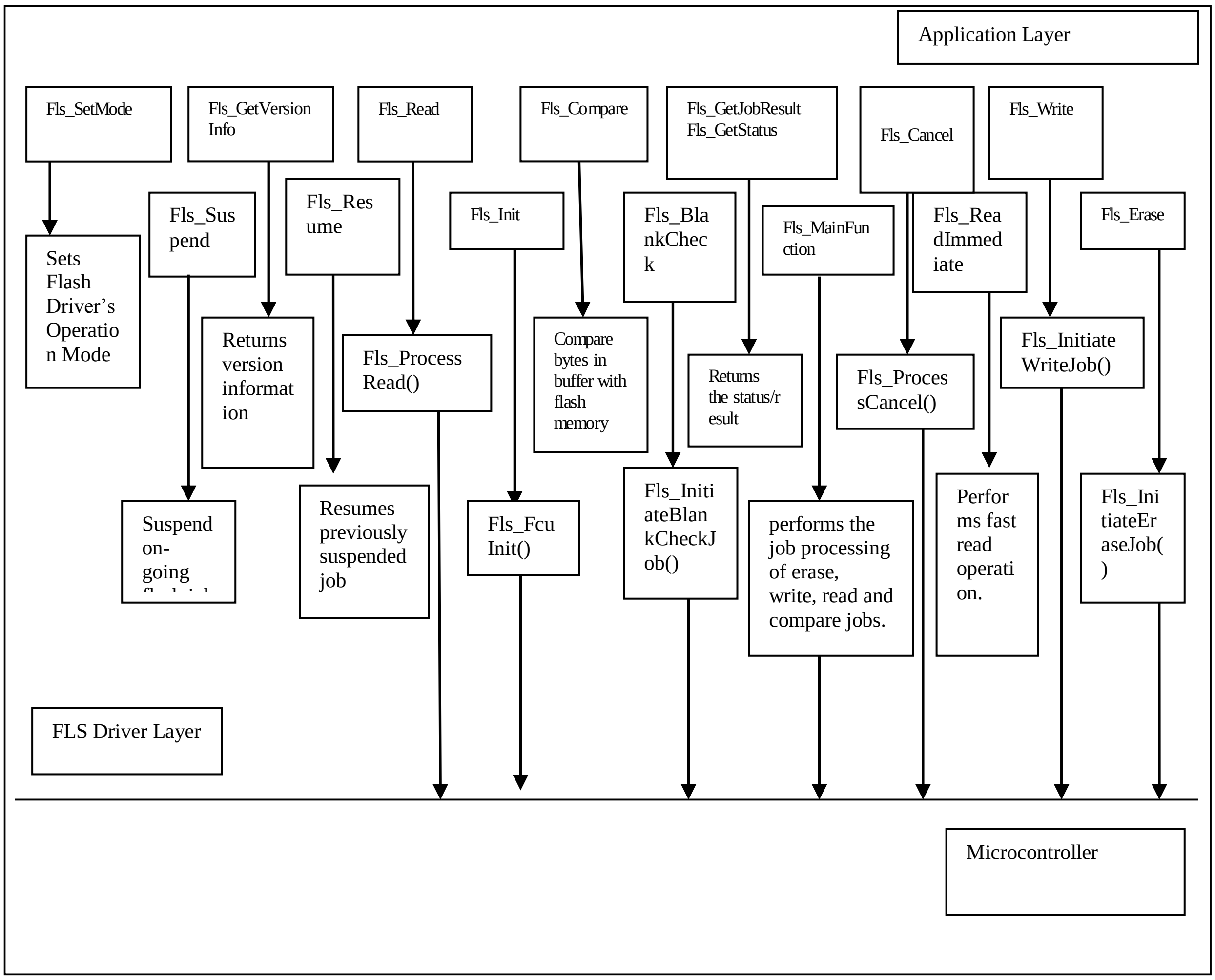
Application Layer
Fls_SetMode
Fls_GetVersion Info
Fls_Read
Fls_Compare
Fls_GetJobResult Fls_GetStatus
Fls_Cancel
Fls_Write
Sets Flash Driver's Operation Mode
Fls_Suspend
Fls_Resume
Fls_Init
Fls_BlankCheck
Fls_MainFunction
Fls_ReadImmediate
Fls_Erase
Returns version information
Fls_ProcessRead()
Compare bytes in buffer with flash memory
Returns the status/result
Fls_ProcessCancel()
Fls_InitiateWriteJob()
Suspend on-going
Resumes previously suspended job
Fls_FcuInit()
Fls_InitiateBlankCheckJob()
performs the job processing of erase, write, read and compare jobs.
Performs fast read operation.
Fls_InitiateEraseJob()
FLS Driver Layer
Microcontroller

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 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. 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. 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 canceling 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. 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 ongoing 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 ongoing 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
	FRAMMCR	16	RW	-	FLS_FCU_REGBIT_FRAMMCR_DUAL
	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	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	8	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 Param eter	Macro/Variable
Fls_MainFunction	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
	FBCSTAT	8	R	-	LulRegValue
	FHVE3	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
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
	FHVE3	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	8	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
Fls_Read	-	-	-	-	-
Fls_Compare	-	-	-	-	-
Fls_ReadImmediat e	-	-	-	-	-

API Name	Registers Used	Register Access 8/16/32 bits	Register Access R/W/RW	Config Parameter	Macro/Variable
Fls_Erase	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
	FHVE3	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
Fls_Write	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
	FHVE3	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
Fls_BlankCheck	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
	FHVE3	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
	FHVE15	8	RW	-	FLS_FLASH_PROTECTION_OFF FLS_FLASH_PROTECTION_ON
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
- Validate flash contents comparing with the application memory
- Cancel the ongoing erase, write, read or compare requests.
- Read the result of the last job
- Read the status of the FLS Driver Component.
- Flash Memory Blank Checking Module
- Flash Memory Immediate Reading Module
- 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 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 Data Flash access

- Fls_Cbk.h
- Fls_Cfg.h
- Fls_Hardware.h

The C source file generated by FLS Driver Generation Tool:

- Fls_PBcfg.c
- Fls_Hardware.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_RegWrite.h

The FLS Driver Component source files:

- Fls.c
- Fls_Internal.c
- Fls_Ram.c
- Fls_Version.c
- Fls_Private_Fcu.c

The Stub C header files:

- Compiler.h
- Compiler_Cfg.h
- MemMap.h
- Platform_Types.h

- SchM_Fls.h
- Dem.h
- Dem_Cfg.h
- Dem_IntErrId.h
- Det.h
- rh850_Types.h
- Std_Types.h
- MemIf.h
- Os.h
- MemIf_Types.h
- Rte.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 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.
Fls_Hardware.h	This file contains the #define macros for the hardware registers to be used by the driver.
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_Hardware.c	This file contains the reference objects for the structures of hardware register which is defined in device header file.
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.

File	Details
Fls_Version.h	This file contains the macros of AUTOSAR version numbers of all modules that are interfaced to FLS.
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_Internal.c	This file contains the Internal functions implementations of flash wrapper component.
Fls_Private_Fcu.c	This file contains FCU related API implementations.
Fls_Version.c	This file contains the code for checking version of all modules that are interfaced to FLS.
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.
Fls_PBTypes.h	This file contains the type definitions of post build parameters. It also contains the macros used by the FLS Driver Component.
SchM_Fls.h	This file is a stub for Fls SchM Component
Dem.h	This file is a stub for DEM Component
Dem_Cfg.h	This file contains the stub values for Dem_Cfg.h
Dem_IntErrId.h	This file is a stub for DEM Component
Det.h	This file is a stub for DET 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.
Std_Types.h	This file is a stub file which contains the standard type definitions.
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

Chapter 9 Generation Tool Guide

For information on the FLS Driver Code Generation Tool, please refer R20UT3642EJ0100-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_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_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		

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.	

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 Function Definitions

Sl. No	API's
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.	Fls_BlankCheck
13.	Fls_ReadImmediate
14.	Fls_Suspend
15.	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_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
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_MainFunction
Source of Error	When the Erase API service is invoked and the erase job fails, error will be reported by the job processing function.
Sl. No.	**2**
Error Code	FLS_E_WRITE_FAILED
Related API(s)	Fls_MainFunction
Source of Error	When the Write API service is invoked and the erase job fails, error will be reported by the job processing function.
Sl. No.	**3**
Error Code	FLS_E_READ_FAILED
Related API(s)	Fls_MainFunction
Source of Error	When the Read API service is invoked and the internal reading of the data flash memory fails, error will be reported by the job processing function.

Sl. No.	**4**
Error Code	FLS_E_COMPARE_FAILED
Related API(s)	Fls_MainFunction
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 by the job processing function.
Sl. No.	**5**
Error Code	FLS_E_READ_FAILED_DED
Related API(s)	Fls_MainFunction
Source of Error	During any read operation in the data flash memory, if any double bit error is detected, error will be reported by the job processing function.
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_MainFunction, Fls_ReadImmediate, Fls_BlankCheck, Fls_Suspend, Fls_Resume
Source of Error	If any write operation on the protection register fails, error shall be reported.
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
Sl. No.	**8**
Error Code	FLS_E_HW_FAILURE
Related API(s)	Fls_Init, Fls_Erase, Fls_Write, Fls_Read, Fls_Cancel, Fls_MainFunction, Fls_BlankCheck, Fls_Suspend,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

Chapter 12 Memory Organization

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

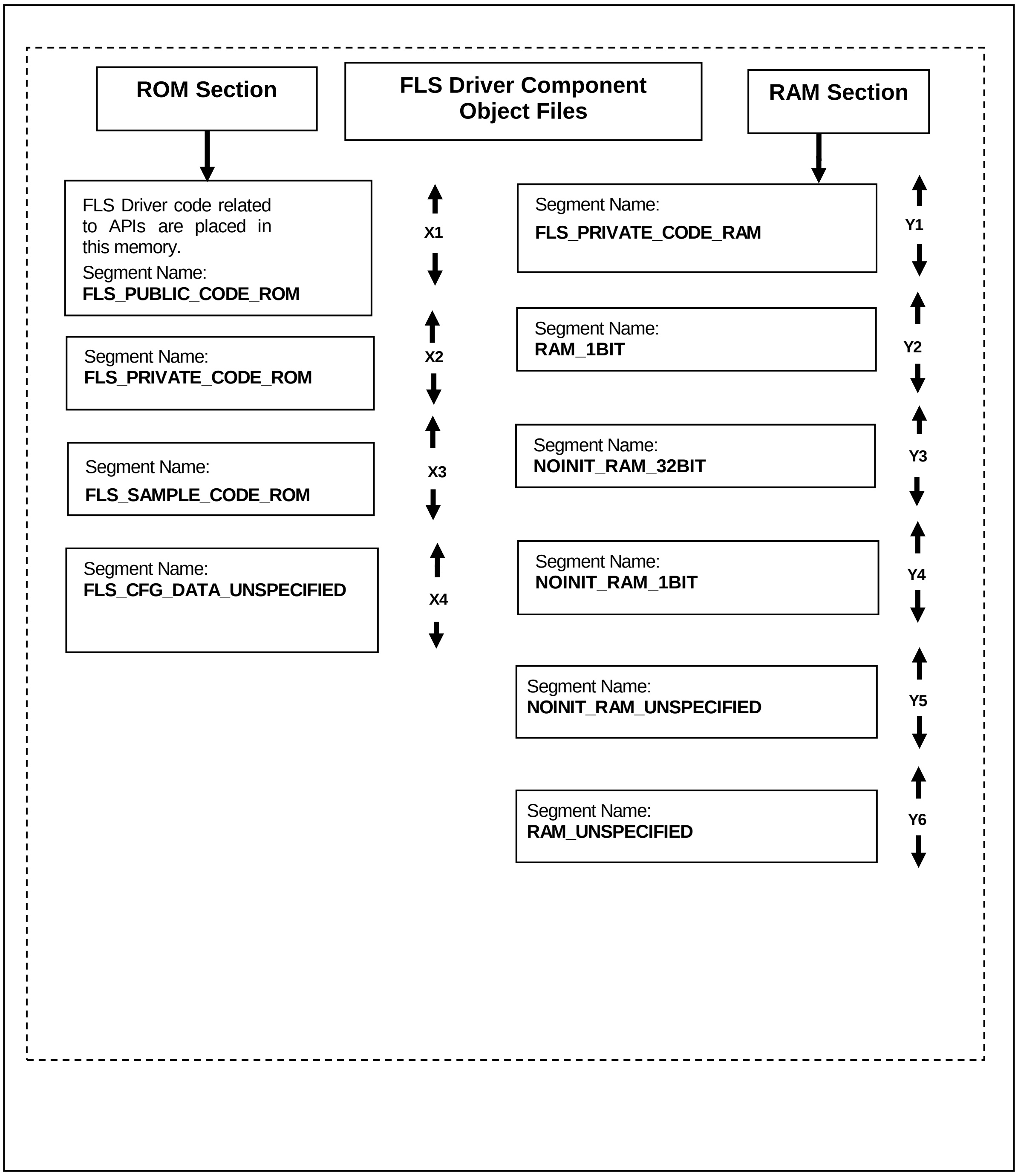

Figure 12-1 FLS Driver Component Memory Organization

ROM Sections:

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

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

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

FLS_CFG_DATA_UNSPECIFIED (X4): This section consists of FLS Driver Component database table of contents generated by the FLS Driver Component Generation Tool.

RAM Sections: Following are the Ram sections mapped.

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.

NOINIT_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.

NOINIT_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.

NOINIT_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.

RAM_UNSPECIFIED (Y6): This section consists of the global RAM variables 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.

Chapter 13 P1x-C Specific Information

P1x-C supports following devices:

- R7F701370A(CPU1(PE1)),
- R7F701371(CPU1(PE1)),
- R7F701372(CPU1(PE1)),
- R7F701373,
- R701374

13.1. Sample Application

13.1.1. 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.

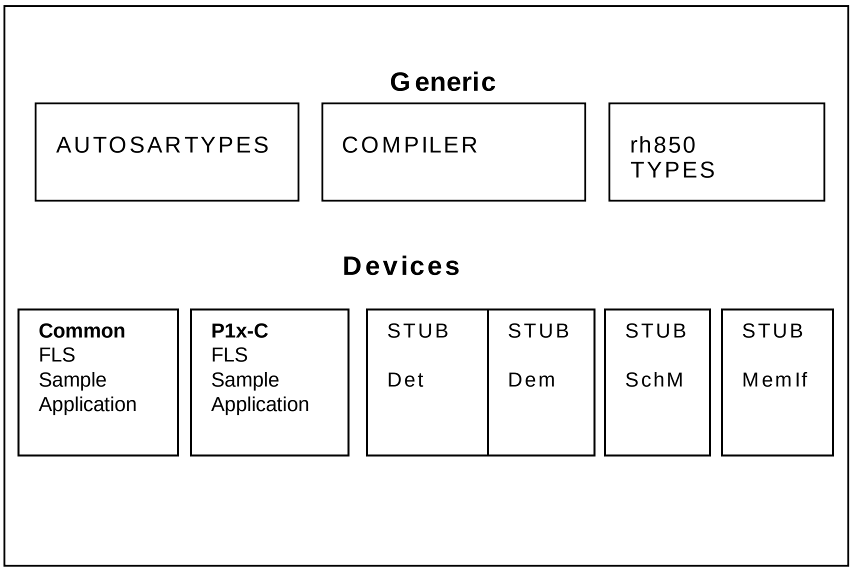

Figure 13-1 Overview Of FLS Driver Sample Application

The Sample Application of the P1X-C is available in the path

X1X\P1x-C\modules\fls\sample_application

The Sample Application consists of the following folder structure

X1X\P1x-C\modules\fls\definition\<AUTOSAR_version>\<SubVariant>

\ R403_FLS_P1X-C.arxml

X1X\P1x-C\modules\fls\sample_application\<SubVariant>\<AUTOSAR_version>

\src\Fls_PBcfg.c
\src\Fls_Hardware.c
\include\Fls_Hardware.h

\include\Fls_Cfg.h
\include\Fls_Cbk.h

\config\App_FLS_ P1x-C_701372_Sample.arxml.
\config\App_FLS_ P1x-C_701372_Sample.one
\config\App_FLS_ P1x-C_701372_Sample.html

\config\App_FLS_ P1x-C_701371_Sample.arxml.
\config\App_FLS_ P1x-C_701371_Sample.one
\config\App_FLS_ P1x-C_701371_Sample.html

\config\App_FLS_ P1x-C_701373_Sample.arxml.
\config\App_FLS_ P1x-C_701373_Sample.one
\config\App_FLS_ P1x-C_701373_Sample.html

\config\App_FLS_ P1x-C_701374_Sample.arxml.
\config\App_FLS_ P1x-C_701374_Sample.one
\config\App_FLS_ P1x-C_701374_Sample.html

\config\App_FLS_ P1x-C_701370A_Sample.arxml.
\config\App_FLS_ P1x-C_701370A_Sample.one
\config\App_FLS_ P1x-C_701370A_Sample.html

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 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 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.
- The API Fls_Suspend, suspends the on-going job.
- The API Fls_Resume, resumes the previous suspended job.

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.

13.1.2. Building Sample Application

13.1.2.1. 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_ P1x-C_<Device_name>_Sample.html

For P1x-C <Device_name> can be 701370A, 701372, 701373, 701374, 701371.

13.1.2.2. 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:
"X1X/P1x-C/common_family/make/<compiler>"

Now execute batch file SampleApp.bat with following parameters:

SampleApp.bat Fls <Device_name>

After this, the tool output files will be generated with the configuration as mentioned in App_FLS_ P1x-C_<Device_name> _Sample.html file is available in the path:
"X1X\P1x-C\modules\fls\sample_application\<SubVariant>\<AUTOSAR_version>\config\App_FLS_ P1x-C_<Device_name> _Sample.html"

- After this, all the object files, map file and the executable file App_FLS_ P1x-C_<Device_name> _Sample.out will be available in the output folder ("X1X\P1x-C\modules\fls\sample_application\<SubVariant>\obj\<Complier>").

- 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 file

"X1X\P1x-C\modules\fls\sample_application\<SubVariant>

\<AUTOSAR_version>\config\App_FLS_ P1x-C_<Device_name>

_Sample.arxml"

App_FLS_ P1x-C_<Device_name> _Sample.arxml" the database alone can be generated by using the following commands.

```
      make –f App_FLS_ P1x-C_<Device_name> _Sample.mak
generate_fls_config
      make –f App_FLS_ P1x-C_<Device_Number>_Sample.mak
App_FLS_ P1x-C_<Device_name>_Sample.run
```

- After this, a flash able Motorola S-Record file App_FLS_ P1x-C_<Device_name> _Sample.run is available in the output folder.

Note 1.For P1x-C <Device_name> can be 701370A, 701371, 701372, 701373, 701374.
2. <compiler> for example can be "ghs".
3. <SubVariant> can be P1H-C, P1H-CE, P1M-C.
4. <AUTOSAR_version> can be 4.0.3.

13.2. Memory and Throughput

13.2.1. ROM/RAM Usage

The details of memory usage for the typical configuration, with DET enabled is provided in this section.

Typical FLS configuration

DET OFF
All other Pre-Compile Settings ON
Number of bytes read in Fls_MainFunction shall be 256 bytes
The Flash erasure for 4 KB(Data Flash).

Table 13-1 ROM/RAM Details with DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701372
1.	ROM	FLS_PUBLIC_CODE_ROM	1780
		FLS_PRIVATE_CODE_ROM	3820
		FLS_CFG_DATA_UNSPECIFIED	630
		ROM.FLS_PRIVATE_CODE_RAM	178

2.	RAM	FLS_PRIVATE_CODE_RAM NOINIT_RAM_UNSPECIFIED NOINIT_RAM_32_BIT RAM_1_BIT NOINIT_RAM_1_BIT RAM_UNSPECIFIED	178 100 404 2 4 3

The details of memory usage for the typical configuration, with DET disabled is provided in this section.

Table 13-2 ROM/RAM Details without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701372
1.	ROM	FLS_PUBLIC_CODE_ROM FLS_PRIVATE_CODE_ROM FLS_CFG_DATA_UNSPECIFIED ROM.FLS_PRIVATE_CODE_RAM	1364 3590 1634 178
2.	RAM	FLS_PRIVATE_CODE_RAM NOINIT_RAM_UNSPECIFIED NOINIT_RAM_32_BIT RAM_1_BIT NOINIT_RAM_1_BIT RAM_UNSPECIFIED	178 100 404 2 3 3

13.2.2. Stack Depth

The worst-case stack depth for FLS Driver Component is 48 bytes.

13.2.3. Throughput Details

The throughput details of the APIs is mentioned below.
The clock frequency used to measure the throughput is 160 MHz for all APIs.

Table 13-3 Throughput Details Of The APIs

Sl. No.	API Name	Throughput in microseconds for device 701372	Remarks
1.	Fls_Init	334.687	-
2.	Fls_Erase	2.312	-
3.	Fls_Write	2.337	-
4.	Fls_Read	0.6	-

5.	Fls_GetStatus	0.125	-
6.	Fls_GetJobResult	0.125	-
7.	Fls_Compare	0.575	-
8.	Fls_GetVersionInfo	0.125	-
9.	Fls_SetMode	0.275	This API does not provide any functionality
10.	Fls_Cancel	0.25	-
11	Fls_ReadImmediate	0.725	-
12.	Fls_BlankCheck	2.125	-
13.	Fls_Erase Operation	3704.812	-
14.	Fls_BlankCheck Operation	175.37	-
15.	Fls_Write Operation	6516.862	-
16.	Fls_Read Operation	1328.862	-
17.	Fls_ReadImmediate Operation	43.162	-
18.	Fls_Compare Operation	74.225	-
19.	Fls_Suspend	0.237	-
20.	Fls_Resume	2.25	-

Chapter 14 Release Details

FLS Driver Software
Version: 1.0.2

Revision History

Sl.No.	Description	Version	Date
1.	Initial Version	1.0.0	12-Aug-2015
2.	1. Introduction Updated 2. Chapter 3, Section 3.1.1 updated 3. Chapter 4, Forethoughts updated 4. Chapter 5, Architecture Details updated 5. Chapter 8, FLS Component Header And Source File Description updated 6. Chapter 11, Table 11.1 and Table 11.2 updated 7. Chapter 12, memory Organization updated 8. Chapter 13, Section 13.2 Sample Application updated 9. Release details updated 10. R number added to User manual	1.0.1	11-May-2016
3.	The following changes are made: 1. In chapter 4, section 4.2 Preconditions points are revised. 2. Table 4-2 is updated with Known Limitation in User Mode. 3. Table 4-1 is added to list protected resources in FLS driver. 4. Chapter 8 is updated with Stub files and Table 8-1 is updated. 5. In Chapter 6, Table 6-1 is updated with Register Files. 6. In chapter 13, added references for device 701371. 7. Chapter 12 and chapter 13.2 are updated with memory sections 8. In Chapter 4, section 4.3 Data Consistency is updated. 9. In Chapter 4.4 Deviation list updated. 10. Updated Chapter 13.2.3 added Throughput for main function and updated with Fls_BlankCheck, Fls_Suspend, Fls_Resume API details in chapter 4.1. 11. Updated Chapter 12 Memory Organization 12. Updated Chapter 6 with details of the register as per individual API 13.Chapter 13, Added Processor name along with Device variants 14. In section 4.5, Note added. 15. In Chapter 12 memory organization updated and In chapter 13.2.1 memory usage updated. 16. In chapter 4.1 General section updated with time timeout monitoring details and chapter 4.4 with timeout monitoring deviation details.	1.0.2	28-Feb-2017

AUTOSAR MCAL R4.0.3 User's Manual
FLS Driver Component Ver.1.0.2
Embedded User's Manual

Publication Date: Rev.1.00, February 28, 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

AUTOSAR MCAL R4.0.3
User's Manual

Renesas Electronics Corporation

R20UT3641EJ0100