User´s Manual

32

RH850 Family

Data Flash Library, Type T01

RENESAS MCU RH850 Family

Installer: RENESAS_FDL_RH850_T01E_V2.xx (xx>=10)

Renesas Electronics
www.renesas.com

R01US0079ED0213
Jun 10, 2019

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

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

(Rev.4.0-1 November 2017)

How to use this document

(1) Purpose and Target Readers

This manual is designed to provide the user with an understanding of the functions and characteristics of the Self-Programming Library. It is intended for users designing application systems incorporating the library. A basic knowledge of embedded systems is necessary in order to use this manual. The manual comprises an overview of the library, API description, usage notes and cautions.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Cautions section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions.

Refer to the text of the manual for details.

(2) List of Abbreviations and Acronyms

Abbreviation	Full form
Code Flash	Embedded Flash where the application code or constant data is stored.
Data Flash	Embedded Flash where mainly the data of the EEPROM emulation are stored.
Dual operation	Dual operation is the capability to access flash memory during reprogramming of another flash memory range. Dual operation is available between Code Flash and Data Flash. Between different Code Flash macros dual operation depends on the device implementation.
ECC	Error Correction Code
EEL	Abbreviation for a software library representing any EEPROM emulation concept (see also EEPROM emulation)
EEPROM	Electrically erasable programmable read-only memory
EEPROM emulation	In distinction to a real EEPROM, EEPROM emulation uses the Flash memory (or a part of it) to emulate EEPROM behavior. To gain a similar behavior some side parameters have to be taken in account.
FCL	Code Flash Library (Code Flash access layer)
FDL	Data Flash Library (Data Flash access layer)
FHVE	Software protection of flash memory against programming and erasure. Not present in all devices.
Flash	Electrically erasable and programmable non-volatile memory. The difference to ROM is, that this type of memory can be re-programmed several times.
Flash block	A flash block is the smallest erasable unit of the flash memory.
Power save mode	Device modes to consume less power than during normal operation. In the device documentation also called “stand-by modes”
RAM	“Random access memory” - volatile memory with random access
ROM	“Read only memory” - non-volatile memory. The content of that memory cannot be changed.

Table of Contents

Chapter 1 Introduction

This user manual describes the internal structure, the functionality and the application programming interface (API) of the Renesas RH850 Data Flash Access Library (FDL) Type 01, designed for RH850 flash devices based on a common flash technology.

The libraries are delivered in source code. However, it has to be considered carefully to do any changes, as not intended behavior and programming faults might be the result.

The Renesas RH850 Data Flash Access Library Type 01 (from here on referred to as FDL) is provided for the Green Hills, IAR and Renesas compiler environments. The library and application programs are distributed using an installer tool allowing selecting the appropriate environment.
The IAR environment is supported for the Europe and America regions only.

The libraries are delivered together with device dependent application programs, showing the implementation of the libraries and the usage of the library functions.

This manual is based on the assumption that the device will operate in supervisor mode.

Please ensure to always use the latest release of the library in order to take advantage of improvements and bug fixes.

If you are located in Europe:

The Data Flash Access library, the latest version of this user manual and other device dependent information can be downloaded from the following URL:
http://www.renesas.eu/update

If you require Flash library related support, please contact our European support team using the following mail address:
application_support.flash-eu@lm.renesas.com

If you are located in other regions:

The FDL and this user manual always recommend use of the latest version.

Note:
Please read all chapters of this user manual carefully. Much attention has been put to proper description of usage conditions and limitations. Anyhow, it can never be completely ensured that all incorrect ways of integrating the library into the user application are explicitly forbidden. So, please follow the given sequences and recommendations in this document exactly in order to make full use of the library functionality and features and in order to avoid malfunctions caused by library misuse.

Flash Infrastructure

Besides the Code Flash, many devices of the RH850 microcontroller family are equipped with a separate flash area — the Data Flash. This flash area is meant to be used exclusively for data. It cannot be used for instruction execution (code fetching).

Flash Granularity

The Data Flash of RH850 device is separated into blocks of 64 bytes. While erase operations can only be performed on complete blocks, data writing can be done on a granularity of one word (4 bytes). Reading from an erased flash word will return undefined data. The number of available Data Flash blocks varies between the different RH850 devices. Please refer to the corresponding user's manual for the hardware for detailed information.

Chapter 2 Architecture

This chapter introduces the basic software architecture of the FDL and provides the necessary background for application designers to understand and effectively use the library. Please read this chapter carefully before moving on to the details of the API description.

2.1 Layered architecture

This chapter describes as an example the layered architecture and functional blocks that may belong to an EEPROM Emulation System (EES). Even though this manual describes the functional block FDL, a short description of the other functional blocks and their relationship can be beneficial for the general understanding.

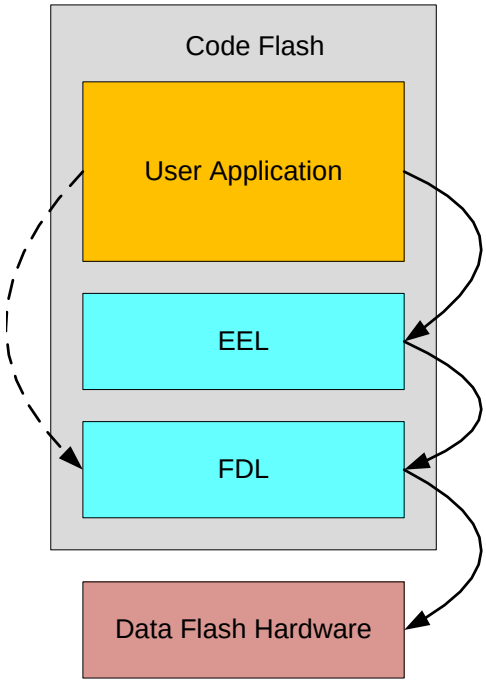


Figure 1: Symbolic relationship between the EES functional blocks

As depicted in the figure above, the software architecture of the EEPROM Emulation System is built up of several blocks:

- **User Application:** This functional block will use functions offered by the FDL and EEL. The user shall take care for synchronization between EEL operations and possibly executed direct FDL accesses by the application.
- **EEPROM Emulation Library (EEL):** This functional block represents an example of an EEPROM emulation concept which offers all functions and commands that the “User Application” block can use in order to handle its EEPROM data.
- **Data Flash Access Library (FDL):** The Data Flash Access Library is the subject of this manual. It should offer an access interface to any user-defined flash area, the so called “FDL-pool” (described in next chapter). Beside the initialization function, the FDL allows the execution of access-commands like write/blank check as well as suspend-able erase command.
- **Data Flash Hardware:** This functional block represents the Flash Programming Hardware (the flash sequencer) controlled by the FDL.

2.2 Pool definitions

The *FDL pool* defines the Flash blocks, the user application and a potential EEPROM emulation (EEL) may use for FDL Flash access. The limits of the *FDL pool* are taken into consideration by any of the FDL Flash access commands. The user can define the size of the *FDL pool* freely at project run-time during library initialization.

The *FDL pool* provides the space for the *EEL Pool* which is allocated by the EEL inside the *FDL pool*. The *EEL Pool* provides the Flash space for the EEL to store the emulation data and management information.

All *FDL pool* space not allocated by the *EEL Pool* is freely usable by the user application, so is called the *User Pool*.

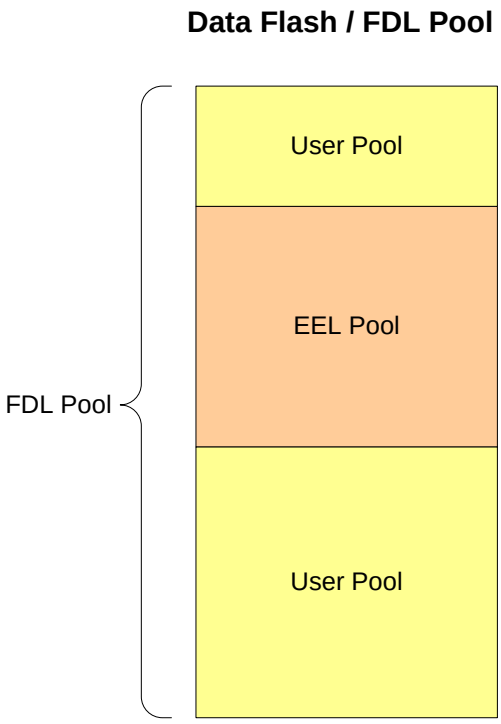


Figure 2: Pools overview

2.3 Architecture related notes

- All Data Flash related operations are executed by the FDL. Thus, the application cannot access (erase, write ...) the Data Flash directly. An exception is reading the Flash contents. As the Flash is mapped to the CPU address space, it can be directly read by the CPU. The FDL provides an additional read operation that will take care of possible ECC (error correction code) errors to allow error polling.
- The FDL allows accessing the Data Flash only.
- Parallel Flash operations (except reading by the CPU) on Data Flash and Code Flash are not possible.

Chapter 3 Functional specifications

3.1 Supported functions, commands and Flash operations

For a better understanding of the flows and mechanisms required for an FDL usage, the basic functions of the FDL are introduced in the following. The API of the FDL is thereby, on the one hand based on functions used to manage the operation of the library itself, on the other hand it offers so-called commands to access and control the content of the FDL pool.

The following table lists all functions that the library will support. Please refer to the chapter 4.4 “Functions” for detailed descriptions.

Table 1: FDL Functions

Function	Description
R_FDL_Init	Initialize the library and Flash hardware
R_FDL_Execute	Initiate a Flash operation
R_FDL_Handler	Control an initiated Flash operation and forward the status.
R_FDL_SuspendRequest	Request suspending an on-going Flash operation
R_FDL_ResumeRequest	Resume a suspended Flash operation
R_FDL_CancelRequest	Request cancelling an on-going Flash operation
R_FDL_StandBy	Suspend an on-going Flash operation from an asynchronous context
R_FDL_WakeUp	Wake-up the FDL from stand-by state
R_FDL_GetVersionString	Return a pointer to the library version string

Commands are used to manage the FDL pool. Commands are initiated via `R_FDL_Execute` and the further progress is controlled by regular execution of `R_FDL_Handler`.

The following commands can be used to execute the following Flash operations:

Table 2: FDL commands and operations

Command	Initiated Flash operation	Description
R_FDL_CMD_ERASE	Flash erase	Erase one or more Flash blocks
R_FDL_CMD_WRITE	Flash write	Write one or more Flash words
R_FDL_CMD_BLANKCHECK	Flash blank check	Blank Check one or more Flash words. Return the fail address in case some Flash word is not blank
R_FDL_CMD_READ	Flash data read	Read one or more Flash words to a buffer. Return a possible ECC (Error correction code) error to the application together with the address of the error
R_FDL_CMD_PREPARE_ENV	-	Prepare Flash environment

The following picture shows the basic flow of Flash operations as an example of erasing two Flash blocks. While the Flash hardware can only erase or write one unit (erase one block, write one word), the FDL will manage handling multiple units. Blank Check is executed on word basis but internally it is split in multiple units at each multiple of 0x1000 bytes boundary.

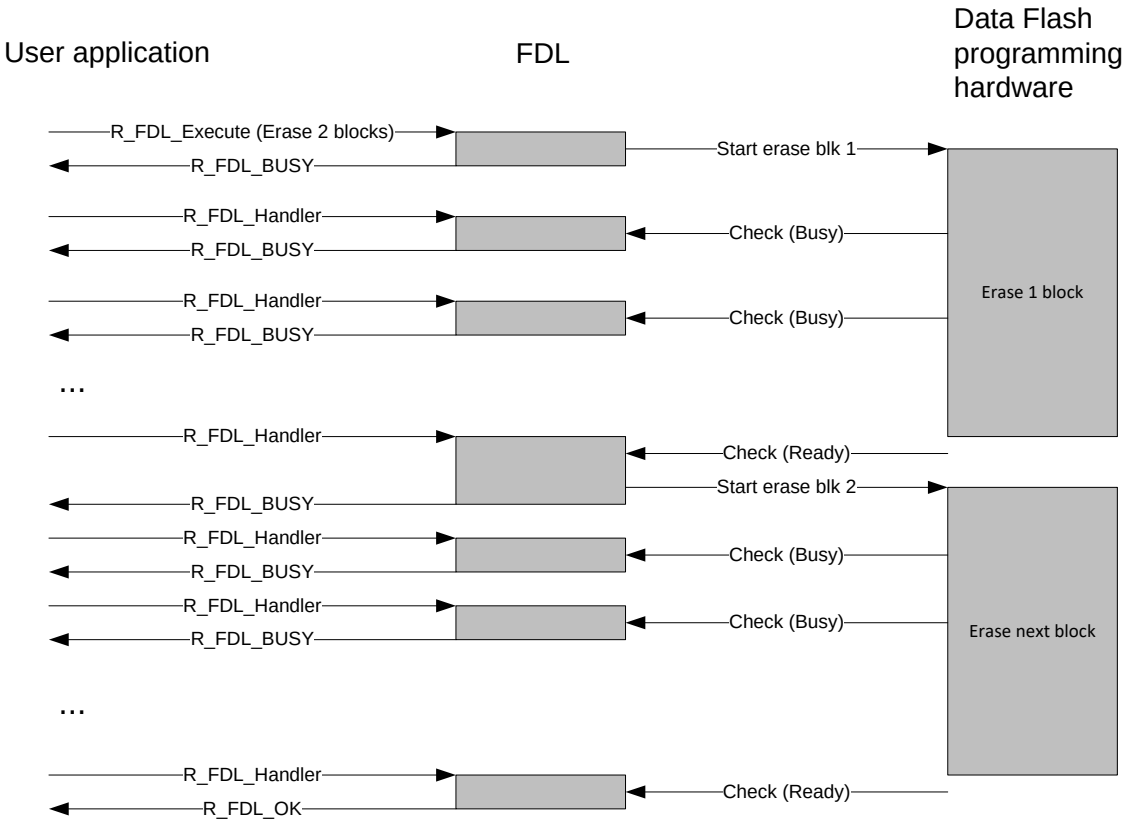


Figure 3: Flash erase sequence

3.2 Request-response oriented dialog

The FDL utilizes request-response architecture in order to initiate the commands. This means any "requester" (any tasks in the user application) has to fill a request structure and pass it by reference to the Data Flash Access Library using `R_FDL_Execute` function. The FDL interprets the content of the request variable, checks its plausibility and initiates the execution. The feedback is reflected immediately to the requester via the status member (`status_enu`) of the same request structure. The completion of an accepted request/command is done by calling `R_FDL_Handler` periodically as long as the request remains "busy".

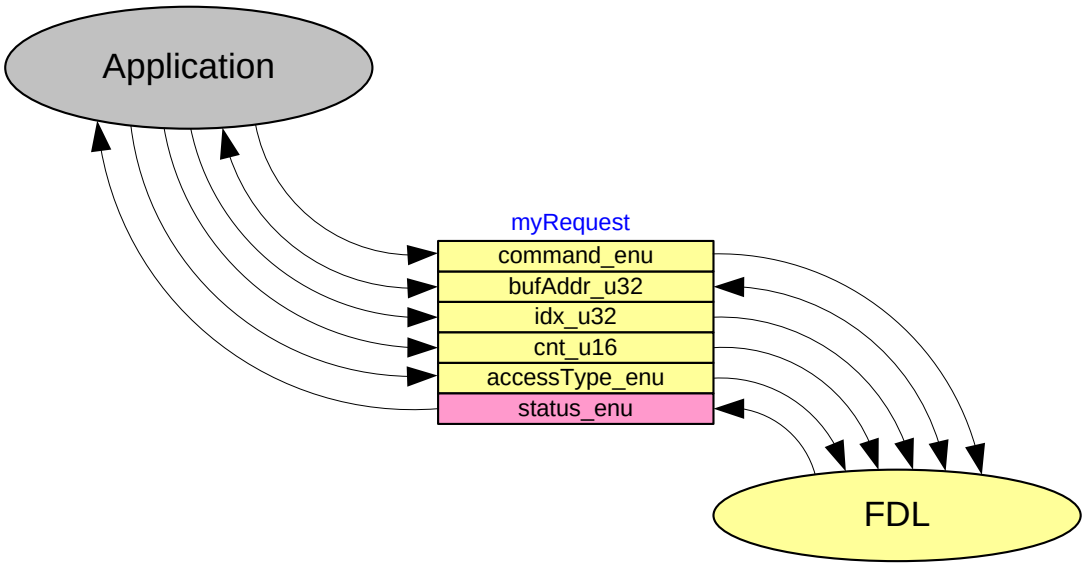


Figure 4: Usage of the request structure

Details on the request variable structure and its members are given later in section 4.3.3 “r_fdl_request_t”. Please also note that not all structure members are required for all commands. The individual command descriptions in section 4.5 “Commands” provide the corresponding detailed information.

3.3 Background operation

The flash technology provided by Renesas enables the application to write/erase the Data Flash in parallel to the CPU execution. In order to satisfy the operation in concurrent or distributed systems, the command execution is divided into two steps:

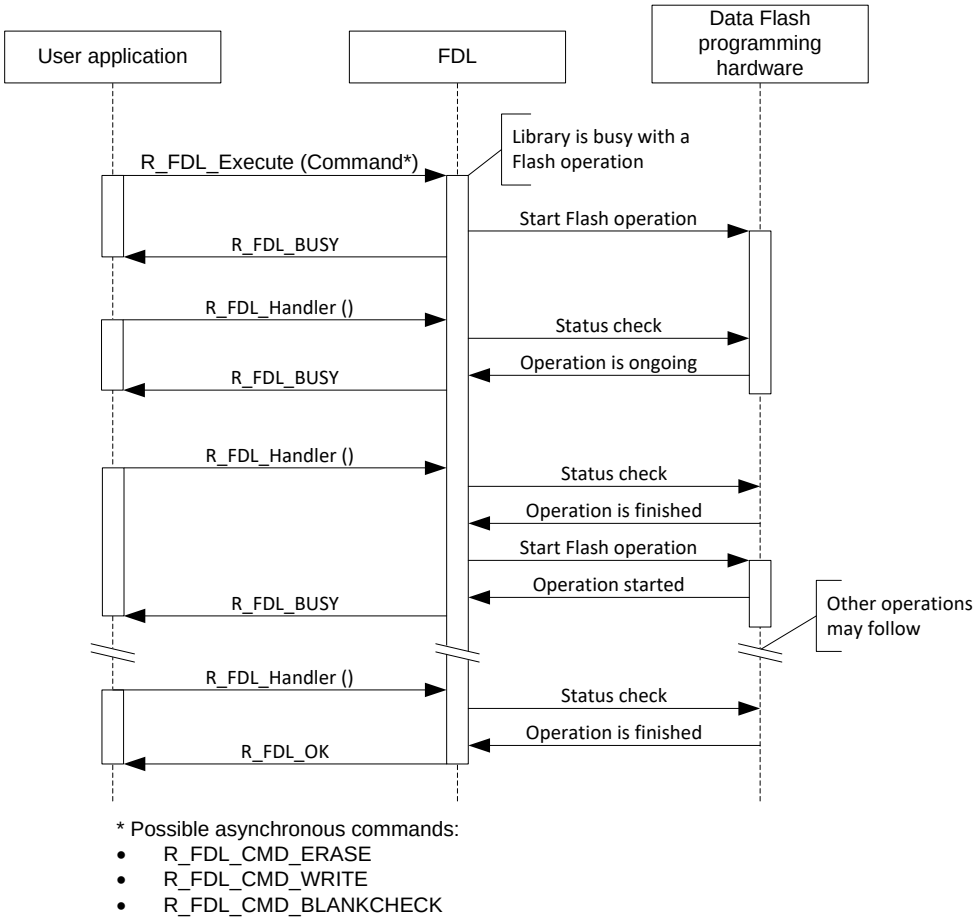
1. Initiation of the command execution using `R_FDL_Execute`
2. Processing of the requested command state by using `R_FDL_Handler`

This approach comes with one important advantage: Command processing can be done centrally at one place in the target system (normally the idle-loop or the scheduler loop), while the status of the requests can be polled locally within the requesting tasks.

Please note that `R_FDL_Execute` only initiates the command execution and returns immediately with the request-status "busy" after execution of the first internal state (or an error in case the request cannot be accepted).

The device flash hardware is responsible for executing the operation in the background. The device hardware operation might be divided into multiple operations, each performed on a separate occasion, depending on the number of blocks and data items. The first operation is conducted by calling the `R_FDL_Execute` function. The second and subsequent operations are triggered by calling the `R_FDL_Handler` function. Thus, there is a need to call the `R_FDL_Handler` function multiple times. Processing is suspended from the time each separate operation is completed until the next one is triggered. Therefore, as the time interval between `R_FDL_Handler` functions call increases, so does the overall processing time.

An exception to this background operation is `R_FDL_CMD_READ` command that is executed synchronously during `R_FDL_Execute` function.



* Possible asynchronous commands:

- R_FDL_CMD_ERASE
- R_FDL_CMD_WRITE
- R_FDL_CMD_BLANKCHECK

Figure 5: Background operation

3.4 Flash access protection

The FDL Flash Access Protection shall protect Flash accesses to unintended addresses. The protection distinguishes EEL-Pool Flash blocks from User-Pool blocks (refer to chapter 2.2 “Pool definitions” for more information). An access as user application will be allowed to all configured Flash blocks outside the EEL-Pool, while an access from EEL will be allowed to the EEL-Pool only.

Generally, on any Data Flash operation initiation, the access type must be defined in the operation request structure variable. Setting this variable enables the access either to the EEL-Pool or to the Data Flash blocks outside the EEL-Pool (User-Pool). If the variable is not initialized appropriately or if the wrong pool shall be accessed, a protection error is returned.

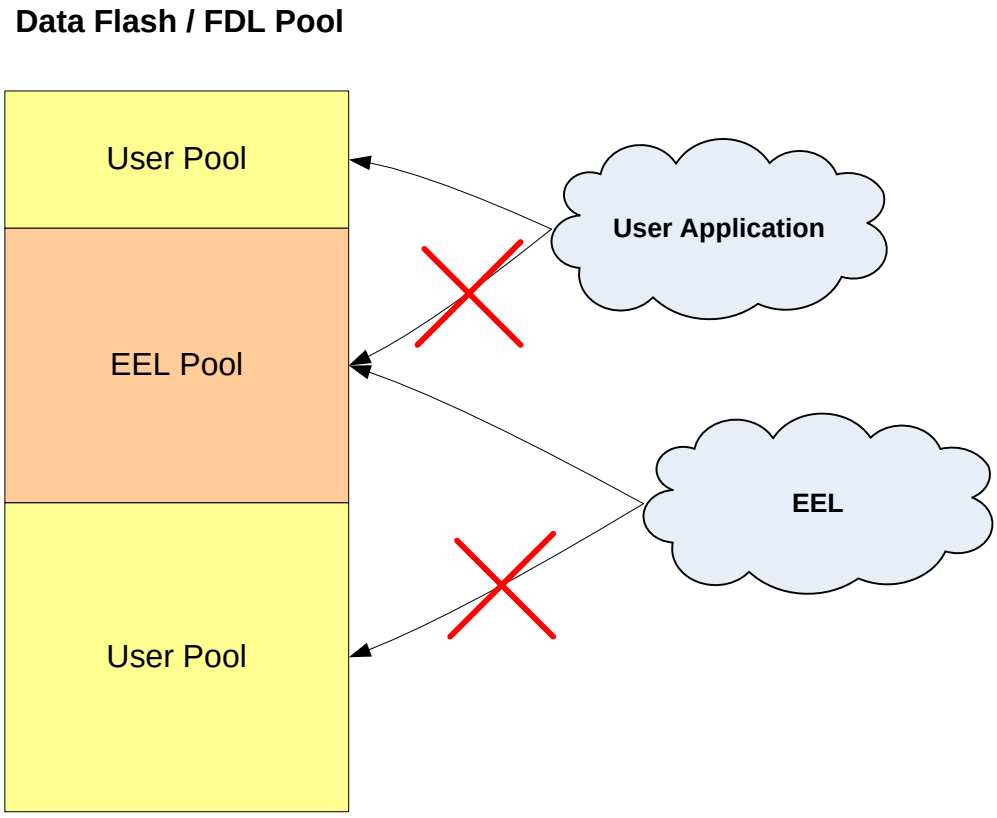


Figure 6: Flash Access Rights

3.5 Suspend / Resume mechanism

Some Data Flash operations can last a long time especially multiple erase and write. The user application cannot always wait for the operation end because other operations have higher priority. So, from user point of view current operation is suspend-able and can be resumed after finishing the other Flash accesses.

From software point of view an on-going operation always ends in suspended state. In case the Flash hardware has already finished an operation but its end result has not already been processed by the library, the library returns the suspended status. The final operation result is returned after successful resume request.

The FDL contains special functions to suspend and resume an ongoing operation. Please refer to chapter 4.4.3.1 “R_FDL_SuspendRequest”.

Suspend restrictions:

- Erase operation ► suspend ► Erase operation – is not possible
- Write operation ► suspend ► Erase/Write operation – is not possible
- Any operation ► suspend ► other operation ► suspend – is not possible

Suspend permissions:

- Blank Check operation ► suspend ► Erase/Write/Blank Check/Read operation – is possible
- Erase operation ► suspend ► Write/Blank Check/Read operation – is possible
- Write operation ► suspend ► Blank Check/Read – is possible

Notes:

- New Flash operations after suspending a Flash operation are only allowed on Flash areas not affected by the suspended operation.
- It is recommended to avoid nesting as much as possible.
- When Erase processing is suspended and resumed, this is not considered as an additional erase with respect to the specified Flash erase endurance.

Examples of Erase or Write Suspend-Resume flow:

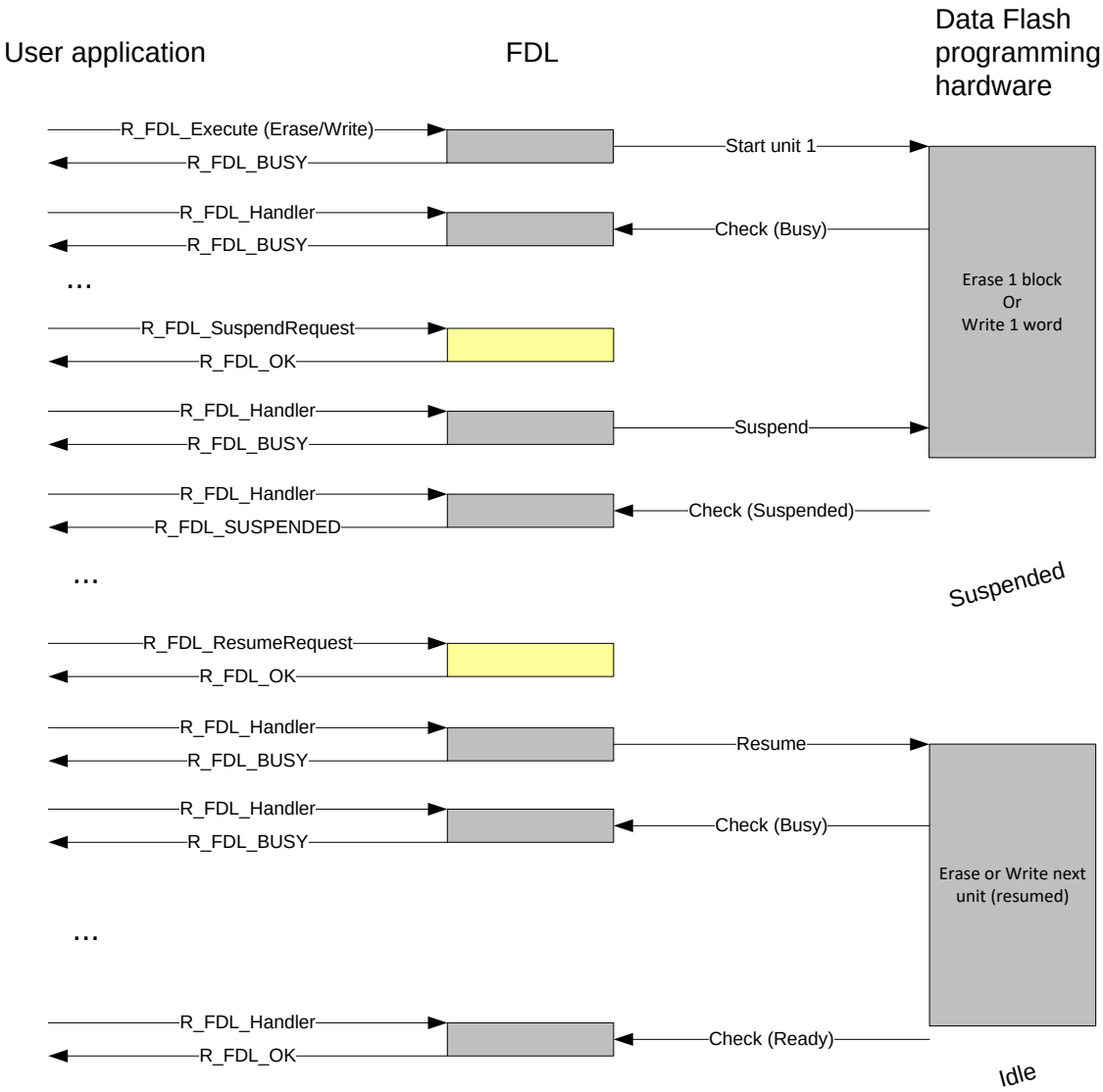


Figure 7: Erase/Write Suspend Resume Flow

Blank check operation will not be interrupted by a suspend request unless the operation reaches a Flash Macro boundary (any multiple of 0x1000 bytes) or it will be finished:

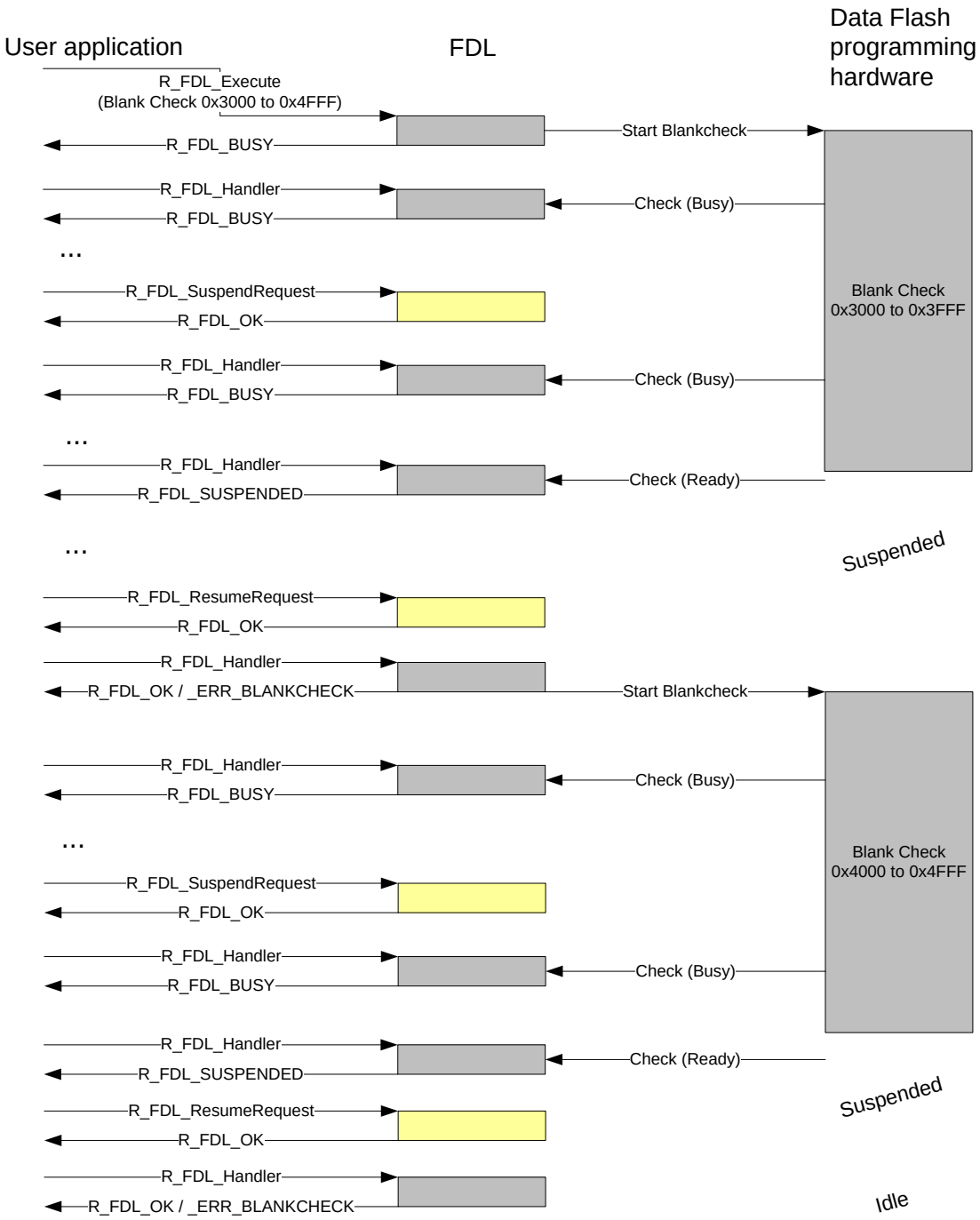


Figure 8: Suspend/Resume a blank check operation

3.6 Stand-by and Wake-up functionality

Entering a device power save (stand-by) mode is not allowed, when a Data Flash operation is on-going. Due to that, especially Data Flash Erase operation can delay entering a power save mode significantly. In order to allow fast entering of such mode, the functions R_FDL_StandBy and R_FDL_WakeUp have been introduced. The main functionality of the functions is to suspend a possibly on-going Data Flash Erase or Write operation (R_FDL_StandBy) and resume it after waking up from power save mode (R_FDL_WakeUp).

Once started, stand-by processing must always end in stand-by status. Calling the R_FDL_StandBy does not necessarily immediately suspend any Data Flash operation, as suspend might be delayed by the device internal hardware or might not be supported at all (only Erase and Write are suspend-able). In this case, the R_FDL_StandBy function must be called repeatedly until the stand-by status is reached.

Blank Check and Read Data Flash operations are suspendable from software point of view, but the library will wait for the operation to be finished by hardware while suspend is processed and the result will be presented after resuming. This wait, however, is not that important because blank check and read operations are much faster than erase or write.

Calling the R_FDL_WakeUp function may not immediately make the device resume operation from the stand-by state (in which case R_FDL_BUSY will be returned). Unless an error is reported, call the R_FDL_WakeUp function repeatedly until it returns R_FDL_OK.

Note that the behavior of stand-by and wake-up operations may differ between versions of the FDL.

<Using FDL V2.12 or an earlier version>

In case the FDL is in an idle state (no data flash operations in progress), the FDL will immediately enter the stand-by state when the R_FDL_StandBy function is called. Calling the R_FDL_WakeUp function will cause the FDL to return to its previous state (in this case the idle state).

<Using FDL V2.13 or a later version>

In case the FDL is in an idle state (no data flash operations in progress), the FDL will return R_FDL_BUSY when the R_FDL_StandBy function is called. Call the R_FDL_StandBy function repeatedly until the FDL enters the stand-by state. Calling the R_FDL_WakeUp function will cause the FDL to return to its previous state (in this case the idle state).

The following pictures illustrate examples of the library's behavior when a stand-by request is issued during FDL operation:

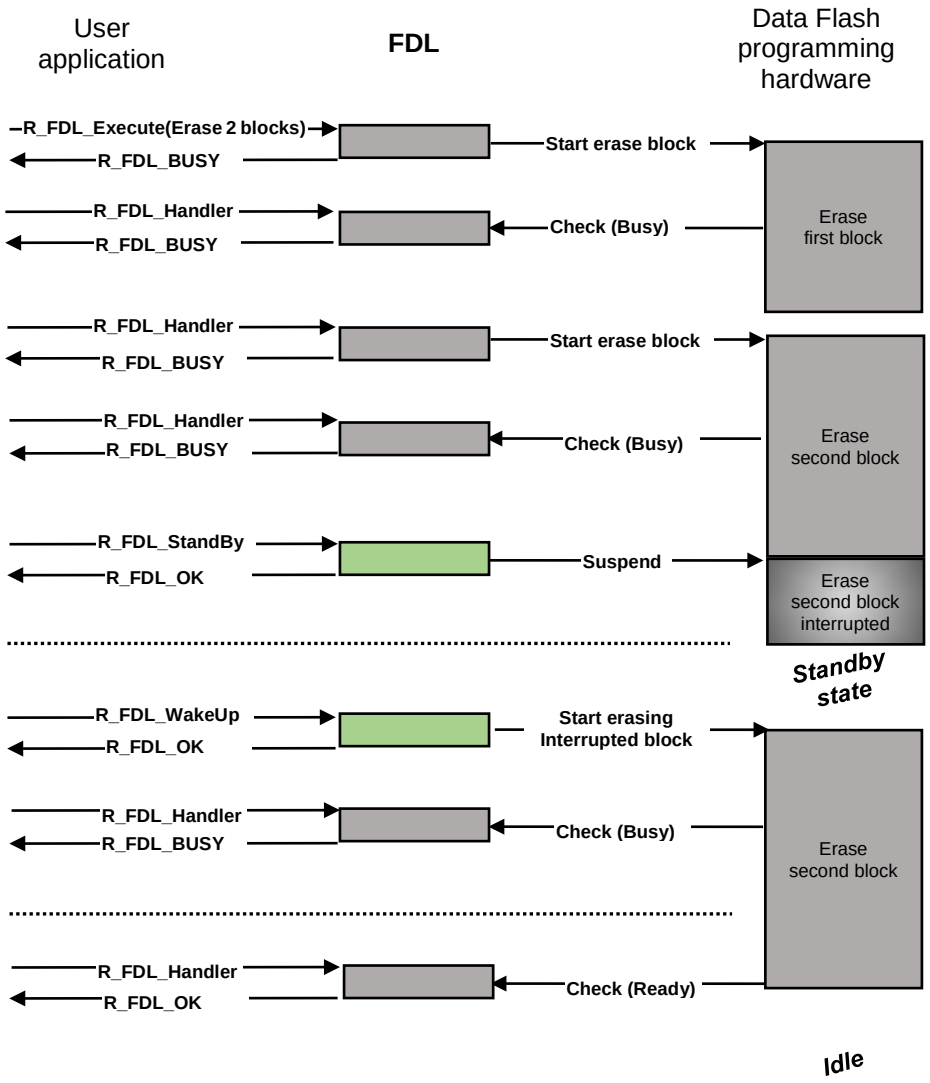


Figure 9: Stand-by processing on a Data Flash Erase operation

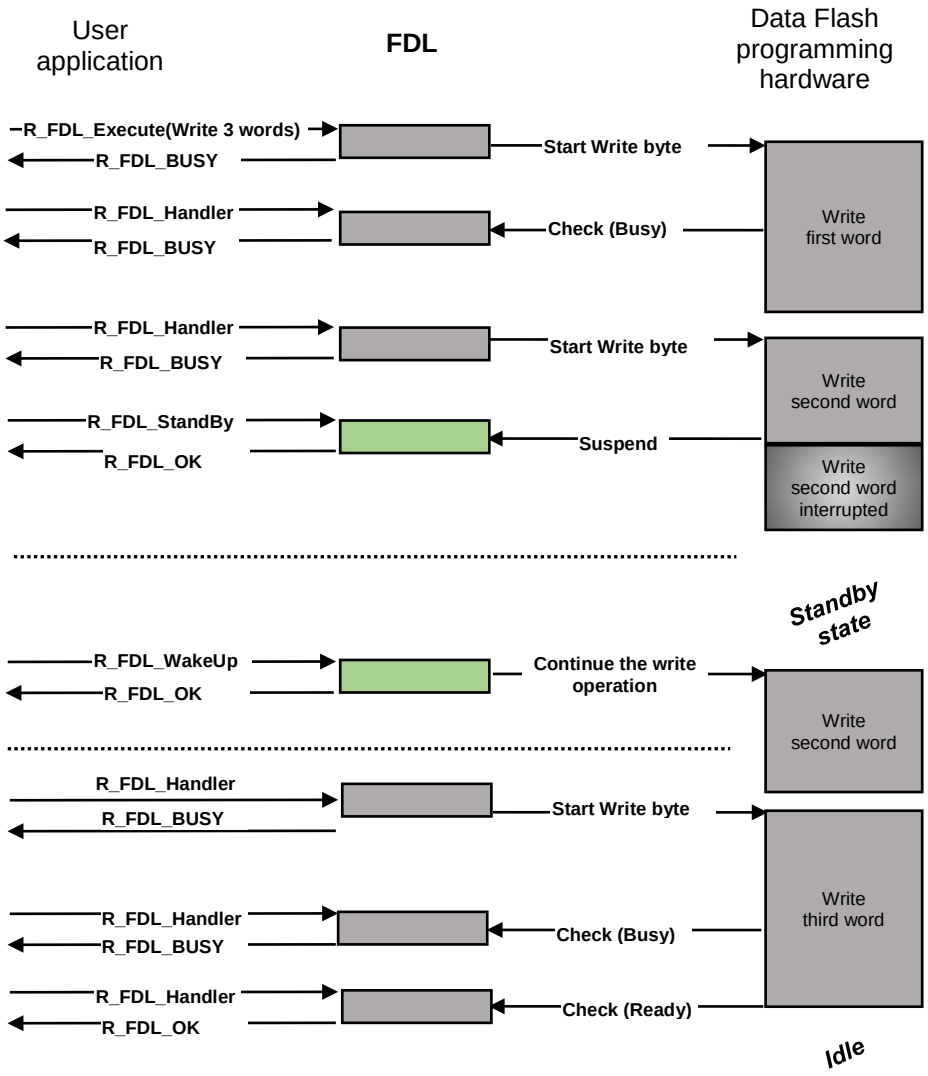


Figure 10: Stand-by processing on a Data Flash Write operation

3.7 Cancel mechanism

The Flash Erase, Write and Blank Check are long lasting operations. The user application cannot always wait for the operation end. Under certain conditions, the user application cannot wait for the end of a long lasting Flash operation. So, such operation should be cancel-able.

The FDL contains a special function to cancel the Erase, Write and the Blank Check operation. Please refer to chapter 4.4.3.5 “R_FDL_CancelRequest”.

Examples of Erase, Write or Blank Check cancel flow:

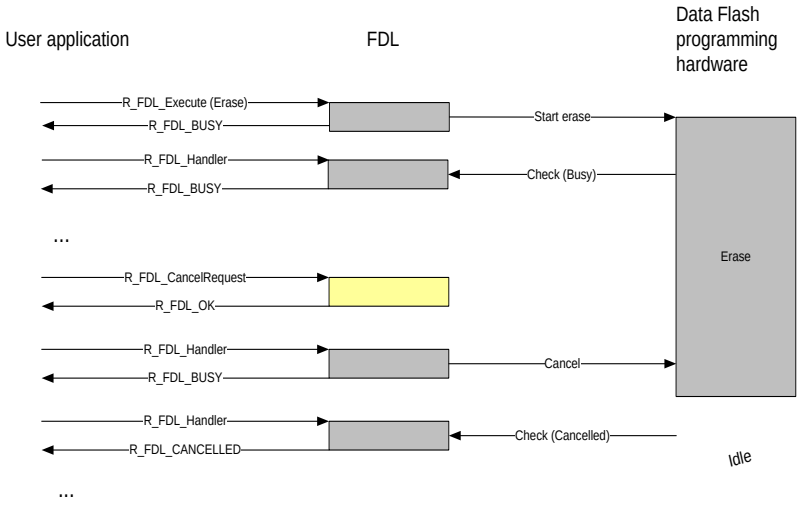


Figure 11: Cancel a normal erase operation

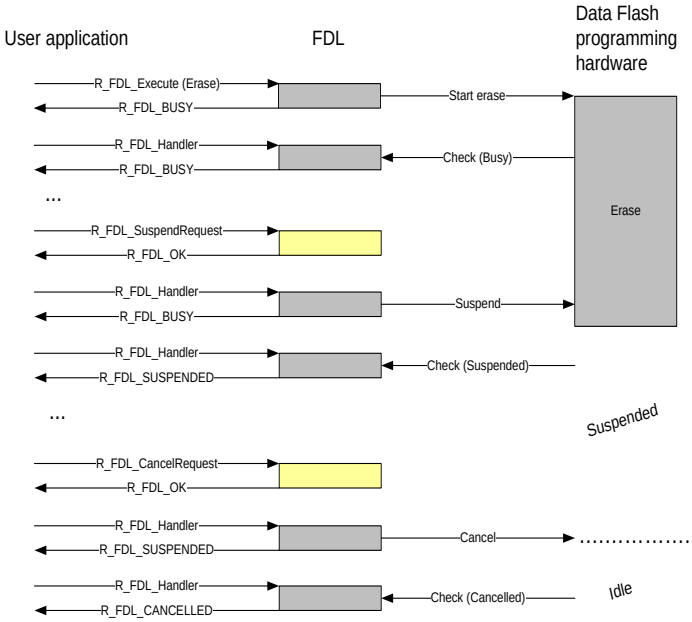


Figure 12: Cancel an erase operation in suspended library state

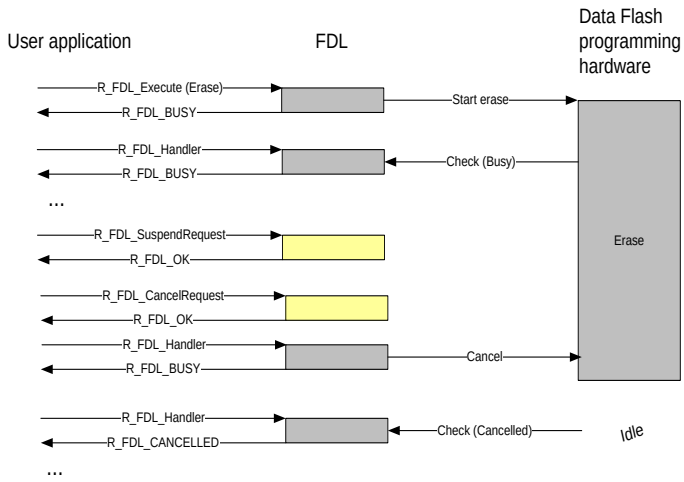


Figure 13: Cancel an erase operation after the suspend request is accepted

From software point of view an on-going or suspended erase/write/blank check operation always ends in cancelled if request is accepted. If a cancel request is accepted, during an on-going write, erase or blank check operation and a previous operation is already suspended, then both operations will be cancelled.

3.8 Loop supervision

All FDL commands except `R_FDL_CMD_READ` have internal polling loops to check for hardware status. These loops are time supervised by FDL software to avoid locking the CPU in an infinite loop. If a hardware error occurs the FDL will abort after a certain timeout and report `R_FDL_ERR_INTERNAL` on the current command.

The timeout depends on the latency of the pooling loop but no later than 800 microseconds provided that the device CPU is running at a frequency in the range permitted for FDL normal operation.

3.9 Internal error

When FDL detects abnormal behavior it will report command status `R_FDL_ERR_INTERNAL`. Further commands are rejected with status `R_FDL_ERR_REJECTED`.

Following steps can be taken to recover after this error is encountered:

a. reinitialize the library (execute `R_FDL_Init` function then run `R_FDL_CMD_PREPARE_ENV` command)

b. if a. fails again then reset the device and proceed to re-initialization

c. if b. fails again then replace the device

Please note that depending on the hardware error, Data Flash can remain in programming mode and in such case only remedy b. and c. can be applied. Data flash may still be available for reading in programming mode but data integrity is not guaranteed.

Chapter 4 User interface (API)

This chapter provides the formal description of the application programming interface of the Flash Data Library Type T01 (FDL). It is strongly advised to read and understand the previous chapters presenting the concepts and structures of the library before continuing with the API details.

4.1 Pre-compilation configuration

The pre-compilation configuration has a direct impact on the object file generated by the compiler. Hence it is used for conditional compilation (e.g. solve device dependencies of the code).

The configuration is done in the module `fdl_cfg.h`. The user has to configure all parameters and attributes by adapting the related constant definitions in that header-file.

The following configuration options are available:

1. Critical section

   One configuration element is the critical section handling of the library. The command `R_FDL_CMD_PREPARE_ENV` needs to activate the device internal special memory for a short time in order to have access to certain data. This results in disabling the Code Flash. During that time, code from Code Flash cannot be executed as well as data cannot be read. The library provides the possibility to execute call-back routines in order for the user to handle the implications of disabling the Code flash (for the impact on the application, please refer to Chapter 6 “Cautions”). The call-back routines are executed at the begin and end of the critical section. The defines to set the call back routines are described in the following:

   `FDL_CRITICAL_SECTION_BEGIN`: Possibility to execute a call back routine at critical section start (e.g. disable interrupts and exceptions)

   `FDL_CRITICAL_SECTION_END`: Possibility to execute a call back routine at critical section end (e.g. enable interrupts and exceptions)

   Implementation in the sample application:

   ```
   #define FDL_CRITICAL_SECTION_BEGIN FDL_User_CriticalSetionBegin();
   #define FDL_CRITICAL_SECTION_END   FDL_User_CriticalSetionEnd();
   ```

2. Location of RAM code

   During execution of command `R_FDL_CMD_PREPARE_ENV` when Code Flash is not available certain FDL functions are executed from RAM. The RAM location used can be selected to be in a buffer located within a special FDL section `R_FDL_CODE_RAM` or in a buffer located on stack. The advantage of using the stack buffer is that RAM is reused instead of being permanently allocated. However, if executing code from stack is not allowed by the security concept then a permanent buffer has to be allocated within special section `R_FDL_CODE_RAM`.

   The default option is to permanently allocate the buffer in special section unless `R_FDL_EXE_INIT_CODE_ON_STACK` is defined.

3. Disabling switching of the FCU firmware area (supported by V2.11 and later versions)

   During execution of command `R_FDL_CMD_PREPARE_ENV`, Code Flash needs to be switched off several times in order to read data from the firmware area and prepare FDL environment. There are some devices that need less switching in order to perform this preparation. If the FDL is running on such device then `R_FDL_NO_BFA_SWITCH` must be defined. When the FDL is built with this option defined, do not define R_FDL_MIRROR_FCU_COPY or R_FDL_NO_FCU_COPY.

   This is required for the F1K, F1KM, F1KH device group but not for other devices.

4. Compatibility mode

   The library API changed from version V1.03 to version V2.00. For all differences between versions V1.03 and V2.00, please refer to installer Release.txt file. One major change is adding a new command `R_FDL_CMD_PREPARE_ENV` to prepare the flash environment, functionality handled by the `R_FDL_Init` function in library version V1.03. By adding the new command, the library basic reprogramming flow changed with direct impact on the user application code (updates are needed). For users who do not want to update their application code, the library provides the possibility to be complaint with the old reprogramming flow (used for library version V1.03), while keeping all the updates made for the new version.

   The compatibility mode can be enabled by defining `R_FDL_LIB_V1_COMPATIBILITY` symbol in `fdl_cfg.h` configuration file.

   If the compatibility mode is disabled, it is user's responsibility to update its code according to the reprogramming flow as described in chapter 5.7 "Basic programming flow".

5. Copying FCU firmware without switching the FCU firmware area (supported by V2.12 and later versions)

   When `R_FDL_MIRROR_FCU_COPY` is defined, the FDL does not switch between the user area and FCU firmware area.

   When the FDL is built with this option defined, do not define R_FDL_NO_BFA_SWITCH or R_FDL_NO_FCU_COPY.

6. Disabling copying of FCU firmware (supported by V2.12 and later versions)

   When `R_FDL_NO_FCU_COPY` is defined, the FCU firmware transfer function is not executed.

   This option is set and built for RH850/D1M1A, D1M1-V2, and D1S1.

   When the FDL is built with this option defined, do not define R_FDL_NO_BFA_SWITCH or R_FDL_MIRROR_FCU_COPY.

Note:

The pre-compilation definitions `R_FDL_NO_FCU_COPY`, `R_FDL_MIRROR_FCU_COPY`, and `R_FDL_NO_BFA_SWITCH` supported by V2.12 and later versions of RH850 FDL Type01 are only applicable to specific device groups.

The following table shows the correspondence between the definitions and device groups.

Pre-compilation definition	F1L/F1M/ F1H	D1L/ D1M1/D1M1H/ D1M2/D1M2H	D1M1A/ D1M1-V2/ D1S1	F1K/F1KM/ F1KH	Future products
R_FDL_NO_BFA_SWITCH	-	-	-	✓	-
R_FDL_MIRROR_FCU_COPY	-	-	-	-	✓
R_FDL_NO_FCU_COPY	-	-	✓	-	-

Note: For the device groups supported by the version of the FDL you are using, see the support.txt file that came with the FDL.

4.2 Run-time configuration

The FDL configuration can be changed dynamically at runtime. It contains important FDL related information (e.g. CPU frequency, number of blocks used by library) and EEL information (e.g. EEL pool size and EEL starting block number).

The run-time configuration is stored in a descriptor structure (see `r_fdl_descriptor_t`), which is declared in `r_fdl_types.h`, but defined in the user application and passed to the library by the function `R_FDL_Init`.

The file `fdl_descriptor.c` shall show an example of the descriptor structure definition and filling, while the `fdl_descriptor.h` shall show an example of the definitions required to fill in the structure.

In fact, the file `fdl_descriptor.h` might be modified according to the user applications needs and might be added to the user application project together with the `fdl_descriptor.c`. The descriptor files (.c and .h) are part of the library installation package.

The following settings should be configured by user:

1. **CPU_FREQUENCY_MHZ:** This defines the internal CPU frequency in MHz unit, rounded up to the nearest integer, e.g. for 24.3 MHz set `CPU_FREQUENCY_MHZ` to 25. Please check the device user's manual for limit values.
2. **FDL_POOL_SIZE:** It defines the number of blocks to be accessed by the FDL for user access and EEL access. Usually it is set to the total number of blocks physically available on the device. For example, if the device is equipped with 32 KB of Data Flash and the block size is 64 bytes, then FDL_POOL_SIZE can be any value up to 512.
3. **EEL_POOL_START:** It defines the starting block of the EEL-Pool. If FDL is used without EEL on top, the value should be set to 0.
4. **EEL_POOL_SIZE:** It defines the number of blocks used for the EEL-Pool. If FDL is used without EEL on top, the value should be set to 0.

FDL block size is always equal to the physical block size of Data Flash.

Example of descriptor when FDL is used alone:

```
/* default access code */
#define CPU_FREQUENCY_MHZ       (80)
/* FDL pool will use 512 blocks * 64 bytes = 32KB, no EEL pool */
#define FDL_POOL_SIZE           (512)
#define EEL_POOL_START          (0)
#define EEL_POOL_SIZE           (0)
```

Example of descriptor when EEL is used:

```
/* default access code */
#define CPU_FREQUENCY_MHZ       (80)
/* FDL pool will use 32KB, EEL pool occupies fist 16 KB */
#define FDL_POOL_SIZE           (512)
#define EEL_POOL_START          (0)
#define EEL_POOL_SIZE           (256)
```

4.3 Data types

This section describes all data definitions used and offered by the FDL. In order to reduce the probability of type mismatches in the user application, please make strict usage of the provided types.

Definitions are similar to those in the standard C99 stdint.h header, but please carefully check that there are no size or endianness mismatches if you are using other definitions in your project.

Compiling V2.13 or a later version of the RH850 FDL Type 01

When you are using V2.13 or a later version of the RH850 FDL Type 01, it is assumed that all data definitions will be compiled according to C99* or a later standard defined by the ISO.

If you select C99 or a later standard for the GHS compiler, Renesas compiler or IAR compiler you are using, the compiler provides the “stdint.h” header file containing data definitions.

If you select a standard earlier than C99 or no standard, on the other hand, data definitions in "r_typedefs.h” (equivalent to stdint.h) will be used.

*Note: The formal name of C99 is ISO/IEC 9899:1999.

Compiling V2.12 or an earlier version of the RH850 FDL Type 01

When you are using V2.12 or an earlier version of the RH850 FDL Type 01, change the directive `#include "r_typedefs.h"`, which is in the following two files, to `#include "stdint.h"`.

- r_fdl_global.h
- r_fdl_user_if_init.c

<Example of a change>

```
#include "stdint.h"
/* #include "r_typedefs.h" */
```

Similarly, when using the attached sample files, change the directive `#include "r_typedefs.h"` to `#include "stdint.h"`.

The sample files for the RH850 FDL Type 01 are as follows.

- fdlapp_control.c
- fdlapp_main.c
- fdl_descriptor.c
- fdl_user.c

4.3.1 Library specific simple type definitions

Type definitions in cases where C99 or a later standard is selected for the compiler:

See “stdint.h” provided with the compiler.

Type definitions (extracted from "r_typedefs.h") in cases where a standard earlier than C99 or no standard is selected for the compiler:

Type definition:

```
typedef signed char         int8_t;
typedef unsigned char       uint8_t;
typedef signed short        int16_t;
typedef unsigned short      uint16_t;
typedef signed long         int32_t;
typedef unsigned long       uint32_t;
typedef unsigned char       rBool;
```

Description: These simple types are used throughout the library API. All library specific simple type definitions can be found in file `r_typedefs.h`, which is part of the library installation package.

4.3.2 r_fdl_descriptor_t

Type definition:

```
typedef struct R_FDL_DESCRIPTOR_T
{
    uint16_t    cpuFrequencyMHz_u16;
    uint16_t    fdlPoolSize_u16;
    uint16_t    eelPoolStart_u16;
    uint16_t    eelPoolSize_u16;
} r_fdl_descriptor_t;
```

Description: This type is the run-time configuration (see chapter 4.2 “Run-time configuration”). A variable of this type is read during initialization phase then hardware and internal variables are set according to the configuration.

Member / Value:

Member / Value	Description
cpuFrequencyMHz_u16	CPU frequency in MHz
fdlPoolSize_u16	FDL pool size in number of blocks
eelPoolStart_u16	Number of first block of the EEL pool
eelPoolSize_u16	Last block of the EEL pool

4.3.3 r_fdl_request_t

Type definition:

```
typedef volatile struct R_FDL_REQUEST_T
{
    r_fdl_command_t     command_enu;
    uint32_t            bufAddr_u32;
    uint32_t            idx_u32;
    uint16_t            cnt_u16;
    r_fdl_accessType_t  accessType_enu;
    r_fdl_status_t      status_enu;
} r_fdl_request_t;
```

Description: This structure is the central type for the request-response-oriented dialog for the command execution (see section 3.2 “Request-response oriented dialog”). Not every element of this structure is required for each command. However, all members of the request variable must be initialized once before usage. Please refer to section 4.5 “Commands” for a more detailed description of the structure elements command-specific usage.

For simplification, `idx_u32` structure member is a virtual address that starts at `0x0` and not at the address at which Data Flash is mentioned in the hardware user manual.

Member / Value:

Member / Value	Description
command_enu	User command to execute: • R_FDL_CMD_ERASE • R_FDL_CMD_WRITE • R_FDL_CMD_BLANKCHECK • R_FDL_CMD_READ • R_FDL_CMD_PREPARE_ENV
bufAddr_u32	Source/Destination buffer address for Write/Read operations
idx_u32	Bidirectional: • start block number when starting block based commands (erase) or • start word address when starting address based commands (write, blank check, read) or • failure address in case of blank check (1st not blank Flash address) or read commands (1st read address with ECC error)
cnt_u16	Number of blocks (64 bytes) to operate in case of erase command. Number of words (4 bytes) to operate for all the other commands.
accessType_enu	Data Flash access originator: • R_FDL_ACCESS_USER or • R_FDL_ACCESS_EEL
status_enu	Status/Error codes returned by the library, see 4.3.6 "r_fdl_status_t"

4.3.4 r_fdl_command_t

Type definition:

```
typedef enum R_FDL_COMMAND_T
{
    R_FDL_CMD_ERASE,
    R_FDL_CMD_WRITE,
    R_FDL_CMD_BLANKCHECK,
    R_FDL_CMD_READ,
    R_FDL_CMD_PREPARE_ENV
} r_fdl_command_t;
```

Description: User command to execute. This type is used within the structure `r_fdl_request_t` (see section 4.3.3 "r_fdl_request_t") in order to specify which command shall be executed via the function `R_FDL_Execute`. A detailed description of each command can be found in section 4.5 “Commands”.

Member / Value:

Member / Value	Description
R_FDL_CMD_ERASE	Erase Data Flash block(s)
R_FDL_CMD_WRITE	Write Data Flash word(s)
R_FDL_CMD_BLANKCHECK	Blank check certain Data Flash area
R_FDL_CMD_READ	Read from Data Flash and return data and possible ECC errors
R_FDL_CMD_PREPARE_ENV	Prepare Flash environment

4.3.5 r_fdl_accessType_t

Type definition:

```
typedef enum R_FDL_ACCESS_TYPE_T
{
    R_FDL_ACCESS_NONE,
    R_FDL_ACCESS_USER,
    R_FDL_ACCESS_EEL
} r_fdl_accessType_t;
```

Description: In order to initiate a Data Flash operation, the access type to the Data Flash must be set depending on the configured pool that will be accessed. The pool ranges are defined in the FDL descriptor, passed to the `R_FDL_Init` function (please check Figure 6: Flash Access Rights").

After each operation the access right is reset to `R_FDL_ACCESS_NONE` to prevent accidental access.

Member / Value:

Member / Value	Description
R_FDL_ACCESS_NONE	FDL internal value. Not used by the application
R_FDL_ACCESS_USER	Application wants to execute an FDL operation in the User-pool Data Flash area
R_FDL_ACCESS_EEL	Application wants to execute an FDL operation in the EEL-pool Data Flash area

4.3.6 r_fdl_status_t

Type definition:

```
typedef enum R_FDL_STATUS_T
{
    R_FDL_OK,
    R_FDL_BUSY,
    R_FDL_SUSPENDED,
    R_FDL_ERR_CONFIGURATION,
```

```
    R_FDL_ERR_PARAMETER,
    R_FDL_ERR_PROTECTION,
    R_FDL_ERR_REJECTED,
    R_FDL_ERR_WRITE,
    R_FDL_ERR_ERASE,
    R_FDL_ERR_BLANKCHECK,
    R_FDL_ERR_COMMAND,
    R_FDL_ERR_ECC_SED,
    R_FDL_ERR_ECC_DED,
    R_FDL_ERR_INTERNAL,
    R_FDL_CANCELLED
} r_fdl_status_t;
```

Description: This enumeration type defines all possible status and error-codes that can be generated by the FDL. Some error codes are command specific and are described in detail in section 4.5 “Commands”.

Member / Value:

Member / Value	Description
R_FDL_OK	FDL operation successfully finished
R_FDL_BUSY	FDL operation is still ongoing
R_FDL_SUSPENDED	Data Flash operation is suspended
R_FDL_ERR_CONFIGURATION	The FDL configuration (descriptor) was wrong
R_FDL_ERR_PARAMETER	An error was found in the given parameter(s)
R_FDL_ERR_PROTECTION	FDL operation stopped due to hardware error, wrong access rights or wrong conditions
R_FDL_ERR_REJECTED	A flow error occurred (e.g. library not initialized, other operation on-going)
R_FDL_ERR_WRITE	Data Flash write error
R_FDL_ERR_ERASE	Data Flash erase error
R_FDL_ERR_BLANKCHECK	The blank check command was stopped because the specified area is not blank
R_FDL_ERR_COMMAND	Unknown command
R_FDL_ERR_ECC_SED	Single bit error detected by ECC
R_FDL_ERR_ECC_DED	Double bit error detected by ECC
R_FDL_ERR_INTERNAL	The current FDL command stopped due to a library internal error (e.g. hardware errors that should never occur or library errors which were not expected and might result from library data manipulation by the application)
R_FDL_CANCELLED	Data Flash operation is cancelled

4.4 Functions

The API functions, grouped by their role in the interface:

Initialization:

- R_FDL_Init

Flash operations:

- R_FDL_Execute
- R_FDL_Handler

Operation control:

- R_FDL_SuspendRequest
- R_FDL_ResumeRequest
- R_FDL_StandBy
- R_FDL_WakeUp
- R_FDL_CancelRequest

Administration:

- R_FDL_GetVersionString

The following sub-chapters describe the Flash operations that can be initiated and controlled by the library. The operations are initiated by a library function R_FDL_Execute and later on controlled by the library function R_FDL_Handler.

All FDL interface functions are prototyped in the header file r_fdl.h.

4.4.1 Initialization

4.4.1.1 R_FDL_Init

Outline: Initialization of the Data Flash Access Library.

Interface: C Interface

```
r_fdl_status_t R_FDL_Init (const r_fdl_descriptor_t * descriptor_pstr);
```

Arguments: Parameters

Argument	Type	Access	Description
descriptor_pstr	r_fdl_descriptor_t *	R	FDL configuration descriptor (see section 4.3.2 “r_fdl_descriptor_t”)

Return value

Type	Description
r_fdl_status_t	• R_FDL_OK Operation finished successfully. • R_FDL_ERR_CONFIGURATION Wrong parameters have been passed to the FDL: ◦ Descriptor address is NULL ◦ FDL-pool is zero ◦ EEL-pool ends beyond FDL-pool edge ◦ Specified CPU clock is outside limits for this device[1] • R_FDL_ERR_INTERNAL[1] Initialization failed due to various factors (insufficient stack space, unknown hardware or software issues)

[1] compatibility mode enabled, please refer to chapter 4.1 “Pre-compilation configuration” for details

Pre-conditions: *Compatibility mode enabled:*
Interrupt execution shall be disabled for a brief time during execution of this function. This must either be done in advance by the user, or the user must properly configure provided callback macro functions in `fdl_cfg.h` (see description and example below).

Compatibility mode disabled:
None

Post-conditions: None

Description: This function is executed before any execution of FDL Flash operation.

Function checks the input parameters and initializes the hardware and software.

Note:

In case the compatibility mode is enabled, this function will temporarily disable Code Flash. Please refer to Chapter 6 Cautions for limitations that must be considered.

Example:

```
const r_fdl_descriptor_t  sampleApp_fdlConfig_enu =
{
    CPU_FREQUENCY_MHZ,
    FDL_POOL_SIZE,
    EEL_POOL_START,
    EEL_POOL_SIZE
};

r_fdl_status_t ret;

ret = R_FDL_Init (&sampleApp_fdlConfig_enu);

if (ret != R_FDL_OK)
{
    /* Error handler */
}
```

Example: for setting the protected section with callbacks provided in the sample application

```
#define FDL_CRITICAL_SECTION_BEGIN     FDL_User_CriticalSetionBegin();
#define FDL_CRITICAL_SECTION_END       FDL_User_CriticalSetionEnd();
```

4.4.2 Flash operations

4.4.2.1 R_FDL_Execute

Outline: Initiate a Data Flash operation.

Interface: C Interface

```
void R_FDL_Execute (r_fdl_request_t * request_pstr);
```

Arguments: Parameters

Argument	Type	Access	Description
request_pstr	r_fdl_request_t *	RW	This argument points to a request structure defining the command, command parameters and also the execution results. A more detailed description of request structure can be found in section 4.3.3 “r_fdl_request_t”.

Return value

Type	Description
none	

Pre-conditions: `R_FDL_Init` must have been executed successfully.

Post-conditions: Call `R_FDL_Handler` until the Flash operation is finished. This is reported by the request structure status return value (value changes from `R_FDL_BUSY` to a different value).

The user application must not modify members of the request structure while the command is in operation.

Description: The execute function initiates all Flash modification operations. The operation type and operation parameters are passed to the FDL by a request structure, the status and the result of the operation are returned to the user application also by the same structure. The required parameters as well as the possible return values depend on the operation to be started.

This function only starts a hardware operation according to the command to be executed. The command processing must be controlled and stepped forward by the handler function `R_FDL_Handler`.

Possible commands, parameters and return values are described into chapter 4.5 “Commands”.

Example: Erase blocks 0 to 3.

```
r_fdl_request_t myRequest;

myRequest.command_enu      = R_FDL_CMD_ERASE;
myRequest.idx_u32          = 0;
myRequest.cnt_u16          = 4;
myRequest.accessType_enu   = R_FDL_ACCESS_USER;

R_FDL_Execute (&myRequest);
while (myRequest.status_enu == R_FDL_BUSY)
{
    R_FDL_Handler ();
}

if (myRequest.status_enu != R_FDL_OK)
{
    /* Error handler */
}
```

Example: Write 8 bytes starting from addresses 0x10.

```
r_fdl_request_t myRequest;

uint32_t data[]            = { 0x11223344, 0x55667788 };

myRequest.command_enu      = R_FDL_CMD_WRITE;
myRequest.idx_u32          = 0x10;
myRequest.cnt_u16          = 2;
myRequest.bufAddr_u32      = (uint32_t)&data[0];
myRequest.accessType_enu   = R_FDL_ACCESS_USER;

R_FDL_Execute (&myRequest);
while (myRequest.status_enu == R_FDL_BUSY)
{
    R_FDL_Handler ();
}

if (myRequest.status_enu != R_FDL_OK)
{
    /* Error handler */
}
```

Example: Blank Check addresses from 0x10 to 0x17.

```
r_fdl_request_t myRequest;

myRequest.command_enu      = R_FDL_CMD_BLANKCHECK;
myRequest.idx_u32          = 0x10;
myRequest.cnt_u16          = 2;
myRequest.accessType_enu   = R_FDL_ACCESS_USER;

R_FDL_Execute(&myRequest);

while (myRequest.status_enu == R_FDL_BUSY)
{
    R_FDL_Handler();
}

if (myRequest.status_enu != R_FDL_OK)
{
    /* Error handler */
```

```
}
```

Example: Read two words starting from address 0x10.

```
r_fdl_request_t myRequest;

uint32_t data[2];

myRequest.command_enu      = R_FDL_CMD_READ;
myRequest.idx_u32          = 0x10;
myRequest.cnt_u16          = 2;
myRequest.bufAddr_u32      = (uint32_t)&data[0];
myRequest.accessType_enu   = R_FDL_ACCESS_USER;

R_FDL_Execute(&myRequest);

if (myRequest.status_enu != R_FDL_OK)
{
    /* Error handler */
}
```

Example: Prepare the Data Flash environment.

```
r_fdl_request_t myRequest;

myRequest.command_enu  = R_FDL_CMD_PREPARE_ENV;

R_FDL_Execute(&myRequest);

while (myRequest.status_enu == R_FDL_BUSY)
{
    R_FDL_Handler();
}

if (myRequest.status_enu != R_FDL_OK)
{
    /* Error handler */
}
```

4.4.2.2 R_FDL_Handler

Outline: This function needs to be called repeatedly in order to drive pending commands and observe their progress.

Interface: C Interface

```
void R_FDL_Handler (void);
```

Arguments: Parameters

Argument	Type	Access	Description
None			

Return value

Type	Description
None	

Pre-conditions: `R_FDL_Init` and `R_FDL_Execute` must have been executed successfully.

Execution of the `R_FDL_CMD_PREPARE_ENV` command in case of compatibility mode disabled:
Interrupt execution shall be disabled for a brief time during execution of this function. This must either be done in advance by the user application (for the complete duration of the command execution), or the user must properly configure provided callback macro functions in `fdl_cfg.h`.

See chapter 4.1 “Pre-compilation configuration”

Post-conditions: The status of a pending FDL command may be updated, i.e. the `status_enu` member of the corresponding request structure is written.

Description: The function needs to be called regularly in order to drive pending commands and observe their progress. Thereby, the command execution is performed state by state. When a command execution is finished, the request status variable (structural element `status_enu` of `r_fdl_request_t`) is updated with the status/error code of the corresponding command execution.

Note:

When no command is being processed, `R_FDL_Handler` consumes few CPU cycles.

Example:

```
while (true)
{
    R_FDL_Handler();
    User_Task_A();
    User_Task_B();
    User_Task_C();
    User_Task_D();
}
```

4.4.3 Operation control

4.4.3.1 R_FDL_SuspendRequest

Outline: This function requests suspending a Flash operation in order to be able to do other Flash operations.

Interface: C Interface

```
r_fdl_status_t R_FDL_SuspendRequest (void);
```

Arguments: Parameters

Argument	Type	Access	Description
None			

Return value

Type	Description
r_fdl_status_t	• R_FDL_OK Operation finished successfully • R_FDL_ERR_REJECTED Wrong library handling flow: ◦ No operation is ongoing ◦ FDL is already in suspended state ◦ FDL is processing a cancel request

Pre-conditions: A Flash operation must have been started and not yet finished (request structure status value is `R_FDL_BUSY`). The FDL must not be processing another suspend or a cancel request.

Post-conditions: Call `R_FDL_Handler` until the library is suspended (status `R_FDL_SUSPENDED`)

If the function returned successfully, no further error check of the suspend procedure is necessary, as a potential error is saved and restored on `R_FDL_ResumeRequest`.

The request structure used before suspend shall not be modified by the command(s) issued during suspended state.

Description: This function requests suspending a Flash operation in order to be able to do other Flash operations.

Example:

```
r_fdl_status_t  srRes_enu;
r_fdl_request_t myReq_str_str;
uint32_t        i;

/* Start Erase operation */
myReq_str_str.command_enu     = R_FDL_CMD_ERASE;
myReq_str_str.idx_u32         = 0;
myReq_str_str.cnt_u16         = 4;
myReq_str_str.accessType_enu  = R_FDL_ACCESS_USER;

R_FDL_Execute (&myReq_str_str);

/* Now call the handler some times */
i = 0;
while ( (myReq_str_str.status_enu == R_FDL_BUSY) && (i < 10) )
{
    R_FDL_Handler ();
    i++;
}

/* Suspend request and wait until suspended */
srRes_enu = R_FDL_SuspendRequest ();
```

```
if (R_FDL_OK != srRes_enu)
{
    /* error handler */
    while (1)
        ;
}

while (R_FDL_SUSPENDED != myReq_str_str.status_enu)
{
    R_FDL_Handler ();
}


/* Now the FDL is suspended and we can handle other operations or read the Data
Flash ... */

/* Erase resume */
srRes_enu = R_FDL_ResumeRequest();

if (R_FDL_OK != srRes_enu)
{
    /* Error handler */
}

/* Finish the erase */
while (myReq_str_str.status_enu == R_FDL_SUSPENDED)
{
    R_FDL_Handler();
}
while (myReq_str_str.status_enu == R_FDL_BUSY)
{
    R_FDL_Handler();
}

if (R_FDL_OK != myReq_str_str.status_enu)
{
    /* Error handler */
}
```

4.4.3.2 R_FDL_ResumeRequest

Outline: This function requests to resume the FDL operation after suspending.

Interface: C Interface

```
r_fdl_status_t R_FDL_ResumeRequest (void);
```

Arguments: Parameters

Argument	Type	Access	Description
None			

Return value

Type	Description
r_fdl_status_t	• R_FDL_OK Operation finished successfully • R_FDL_ ERR_REJECTED Wrong library handling flow: • FDL is not in suspended state • FDL is processing a cancel request

Pre-conditions: The library must be in suspended state. The FDL must not be processing a cancel request.

Post-conditions: Call `R_FDL_Handler` until the library operation is resumed.

Description: This function requests to resume the FDL operation after suspending. The resume is just requested by this function. Resume handling is done by the `R_FDL_Handler` function.

Example: See `R_FDL_SuspendRequest`.

4.4.3.3 R_FDL_StandBy

Outline: This function suspends an ongoing flash operation.

Interface: C Interface

```
r_fdl_status_t R_FDL_StandBy (void);
```

Arguments: Parameters

Argument	Type	Access	Description
None			

Return value

Type	Description
r_fdl_status_t	• R_FDL_OK FDL operation finished successfully • R_FDL_BUSY The started Flash operation is still on-going • R_FDL_ ERR_REJECTED Flow error: • Library is not initialized • Library is already in stand-by mode

Pre-conditions: R_FDL_Init must have been executed successfully.

FDL is not in stand-by mode.

Post-conditions: Repeat the execution of the R_FDL_StandBy function until the state indicated by the function changes from R_FDL_BUSY.

Do not execute functions R_FDL_Execute, R_FDL_SuspendRequest, R_FDL_ResumeRequest,R_FDL_CancelRequest or R_FDL_StandBy when FDL is in stand-by state.

Description: This function suspends an ongoing flash operation and brings FDL into stand-by state. The system can then change to special states (e.g. enter HALT mode, reduce clock speed...).

Function does not necessarily immediately suspend any Flash operation, as suspend might be delayed by the device internal hardware or might not be supported at all (only Erase and Write are suspendable). So, the function R_FDL_StandBy tries to suspend the Flash operation and returns R_FDL_BUSY as long as a Flash operation is on-going. If suspend was not possible (e.g. blank check operation), R_FDL_BUSY is returned until the operation is finished normally.

So, in order to be sure to have no Flash operation on-going, the function must be called continuously until the function does no longer return R_FDL_BUSY or until a timeout occurred.

After stand-by, it is mandatory to call R_FDL_WakeUp to resume normal FDL operation again. The prescribed sequence in case of using R_FDL_StandBy/R_FDL_WakeUp is:

- any FDL command is in operation
- call R_FDL_StandBy until it does no longer return R_FDL_BUSY
- put device in power save (stand-by) mode
- device wake-up
- call R_FDL_WakeUp
- continue with initial FDL command

Note:

Please do not enter a power save mode which resets/alters the Flash hardware or memory required for library operation - e.g. stack, library data or library operation related data (such as request variable). This need to be considered, because resuming the previous operation is not possible otherwise. The library is not able to detect such failure. A possible power save mode is HALT.
If entering other modes, the FDL need to be re-initialized by R_FDL_Init.

Example:

```
r_fdl_status_t  fdlRet_enu;
r_fdl_request_t myReq_str_str;

/* Start Erase operation */
myReq_str_str.command_enu     = R_FDL_CMD_ERASE;
myReq_str_str.idx_u32         = 0;
myReq_str_str.cnt_u16         = 4;
myReq_str_str.accessType_enu  = R_FDL_ACCESS_USER;

R_FDL_Execute (&myReq_str_str);

...
```

```
do
{
    fdlRet = R_FDL_StandBy ();
}
while (R_FDL_BUSY == fdlRet);
if (R_FDL_OK != fdlRet)
{
    /* error handler */
}

...
/* device enters power save mode */
...

...
/* device recovers from power save mode */
...
do
{
   fdlRet = R_FDL_WakeUp();
}
While(R_FDL_BUSY==fdlRet);
if (R_FDL_OK != fdlRet)
{
    /* error handler */
}

/* Finish erase command */

while (myReq_str_str.status_enu == R_FDL_BUSY)
{
    R_FDL_Handler ();
}

if (R_FDL_OK != myReq_str_str.status_enu)
{
    /* Error handler */
    while (1)
        ;
}
```

4.4.3.4 R_FDL_WakeUp

Outline: This function wakes up the library from stand-by state.

Interface: C Interface

```
r_fdl_status_t R_FDL_WakeUp (void);
```

Arguments: Parameters

Argument	Type	Access	Description
None			

Return value

Type	Description
r_fdl_status_t	• R_FDL_OK Operation finished successfully • R_FDL_BUSY The started Flash operation is still on-going • R_FDL_ ERR_REJECTED Wrong library handling flow: FDL is not in stand-by state

Pre-conditions: The library must be in stand-by mode.

The hardware conditions (CPU frequency, voltage, etc...) must be restored to the state before issuing the stand-by request.

Post-conditions: While the FDL returns R_FDL_BUSY in response to a call of the R_FDL_WakeUp function,call the R_FDL_WakeUp function repeatedly.

Description: The main purpose of this function is to wake up the library from the stand-by mode and resume Flash hardware. For more information see chapter 3.6 “Stand-by and Wake-up functionality”.

Example: See `R_FDL_StandBy`.

4.4.3.5 R_FDL_CancelRequest

Outline: This function requests cancelling an on-going or suspended Erase, Write or Blank check Flash operation.

Interface: C Interface

```
r_fdl_status_t R_FDL_CancelRequest (void);
```

Arguments: Parameters

Argument	Type	Access	Description
None			

: Return value

Type	Description
r_fdl_status_t	• R_FDL_OK Operation finished successfully • R_FDL_ERR_REJECTED Wrong library handling flow: ◦ No operation is ongoing or suspended ◦ FDL is already processing another cancel request • R_FDL_ERR_INTERNAL A library internal error occurred, which could not happen in case of normal application execution • R_FDL_ERR_PROTECTION Code Flash or Data Flash is in programming mode

Pre-conditions: A Flash operation must have been started and not yet finished (request structure status value is `R_FDL_BUSY`) and/or suspended. The FDL must not be processing another cancel request.

Post-conditions: Call `R_FDL_Handler` until the library is cancelled (status `R_FDL_CANCELLED`)

Description: This function requests cancelling a Flash Erase, Write or Blank Check operation. For more information see chapter 3.7 ”Cancel mechanism”.

Example:

```
/* Erase block 0,1,2 and 3 */
r_fdl_request_t myRequest ;
r_fdl_status_t  srRes_enu ;
uint32_t i ;

myRequest.command_enu     = R_FDL_CMD_ERASE
myRequest.idx_u32         = 0
myRequest.cnt_u16         = 4
myRequest.accessType_enu  = R_FDL_ACCESS_USER;

R_FDL_Execute(&myRequest);

/* call the handler some time */
i= 0;
while ((myRequest.status_enu == R_FDL_BUSY) && (i<10))
{
    R_FDL_Handler ();
    i++;
}

/* Cancel request and wait until cancelled */
srRes_enu = R_FDL_CancelRequest () ;
if (R_FDL_OK != srRes_enu)
{
    /* Error treatment */
    ...
)
```

```
while (R_FDL_CANCELLED != myRequest.status_enu)
{
    R_FDL_Handler ();
}
```

4.4.4 Administration

4.4.4.1 R_FDL_GetVersionString

Outline: This function returns the pointer to the null terminated library version string.

Interface: C Interface

```
(const uint8_t*) R_FDL_GetVersionString (void);
```

Arguments: Parameters

Argument	Type	Access	Description
None			

Return value

Type	Description
const uint8_t *	The library version is a string value in the following format: “DH850T01xxxxxYZabcD” Please check function description below for details.

Pre-conditions: None

Post-conditions: None

Description:

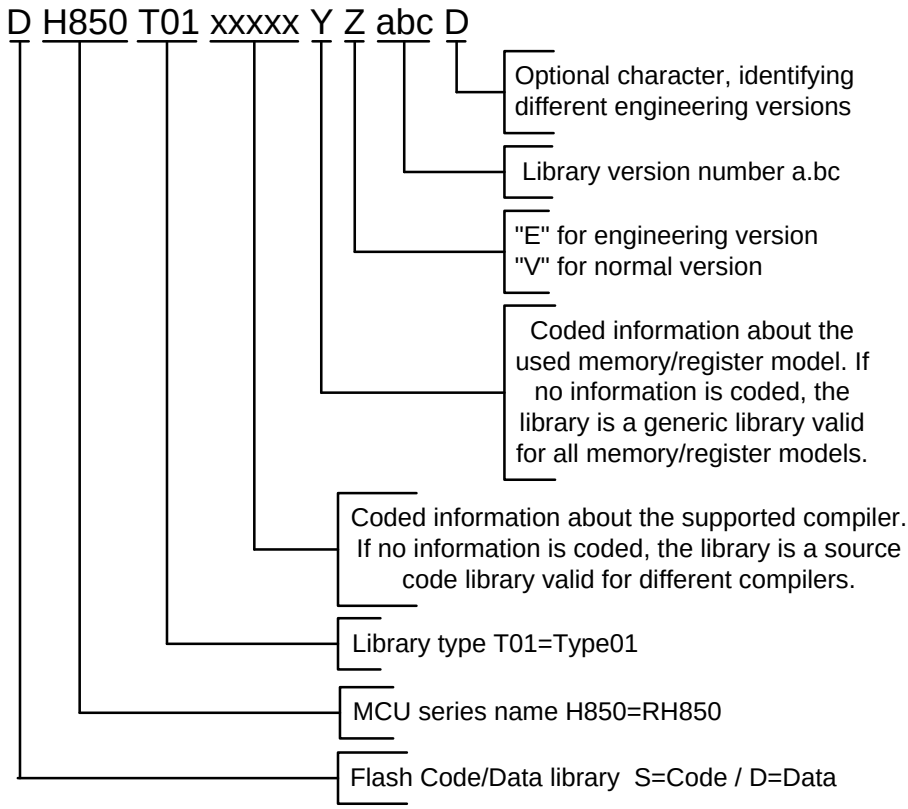


Figure 14: Version string

Example:

```
uint8_t * vstr = (uint8_t *)R_FDL_GetVersionString ();
```

4.5 Commands

The following sub-chapters describe the Flash operations that can be initiated and controlled by the library.

In general, all FDL commands can be handled in the same way as illustrated in Figure 15:

1. The requester fills up the private request variable `my_request` (command definition).
2. The requester tries to initiate the command execution by `R_FDL_Execute(&my_request)`.
3. The requester has to call `R_FDL_Handler` to proceed the FDL command execution as long the request is being processed (i.e. `my_request.status_enu == R_FDL_BUSY`).
4. After finishing the command (i.e. `my_request.status_enu != R_FDL_BUSY`) the requester has to analyse the status to detect potential errors.

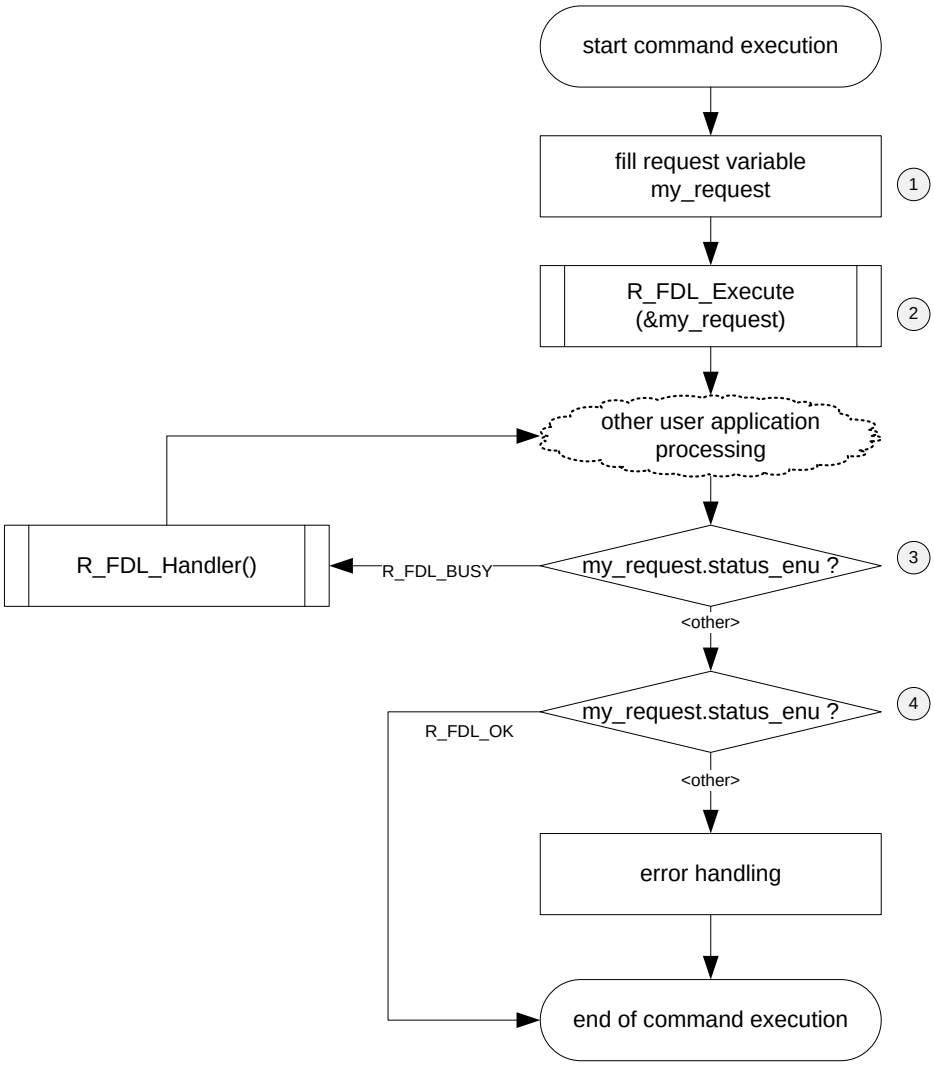


Figure 15: Generic command execution flow

4.5.1 R_FDL_CMD_ERASE

The erase command can be used to erase a number of Flash blocks defined by a start block and the number of blocks.

The members of the request structure given to `R_FDL_Execute` are described in the following table:

Table 3: Request structure usage for erase command

Structure member	Value	Description
command_enu	R_FDL_CMD_ERASE	Request a block erase operation
bufAddr_u32	-	Not used

Structure member	Value	Description
idx_u32	{uint32_t number}	Number of the first block to be erased. Flash blocks are defined by the erase granularity that is 64 bytes, e.g.: block 0: 0x00 ... 0x3F block 1: 0x40 ... 0x7F ...
cnt_u16	{uint16_t number}	Numbers of blocks to erase
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	Selects the Flash pool in which the command will be able to operate.
status_enu	-	This is an output member. It contains the status of the operation during and after the execution. Possible values are described in the next table.

The following table describes all possible status returns:

Table 4: Erase operation returned status

Status	Background and Handling	
R_FDL_BUSY	meaning	Operation started successfully
	reason	No problems during execution
	remedy	Call `R_FDL_Handler` until the Flash operation is finished, reported by the request structure status return value
R_FDL_OK(1)	meaning	Operation finished successfully
	reason	No problems during execution
	remedy	Nothing
R_FDL_SUSPENDED(1)	meaning	An on-going Flash operation was successfully suspended
	reason	Suspend processing successfully finished
	remedy	Start another operation or resume the suspended operation
R_FDL_CANCELLED(1)	meaning	An on-going or suspended Flash operation was successfully cancelled
	reason	Cancel processing successfully finished
	remedy	Start another operation
R_FDL_ERR_PARAMETER	meaning	Current command is rejected
	reason	Wrong command parameters: • access is made outside of physically available Data Flash • command shall operate in User-pool but `accessType_enu` is not `R_FDL_ACCESS_USER` • command shall operate in EEL-pool but `accessType_enu` is not `R_FDL_ACCESS_EEL` • `cnt_u16` is 0 or it is too big
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_PROTECTION	meaning	Current command is rejected

Status	Background and Handling	
	reason	• Activated device specific protection mechanisms prevent Flash operations (availability depending on the device, e.g. FHVE protection mechanism)
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_REJECTED(2)	meaning	Current command is rejected
	reason	Another operation is ongoing
	remedy	Request again the command when the preceding command has finished
R_FDL_ERR_ERASE(1)	meaning	Affected Flash area could not be completely erased
	reason	FDL was initialized with incorrect CPU frequency Hardware defect
	remedy	Re-initialize FDL with correct frequency A Flash block respectively the complete Data Flash should be considered as defect
R_FDL_ERR_INTERNAL	meaning	A library internal error occurred, which could not happen in case of normal application execution
	reason	An error occurred that cannot be determined by the library, such as caused by: • FDL code or data sections destruction, wrong program flow, Flash hardware modification • Hardware defect
	remedy	Please refer to section 3.9 “Internal error“

(1) `R_FDL_Execute` will never set this status code

(2) `R_FDL_Handler` will never set this status code

4.5.2 R_FDL_CMD_WRITE

The write command can be used to write a number of data words located in the RAM into the Data Flash at the location specified by the virtual target address.

Note:

It is not allowed to “overwrite” data, which means writing data to already partly or completely written Flash area. Please always erase the targeted area before writing into it.

The members of the request structure given to `R_FDL_Execute` are described in the following table:

Table 5: Request structure usage for write command

Structure member	Value	Description
command_enu	R_FDL_CMD_WRITE	Request a write operation
bufAddr_u32	{uint32_t number}	Address of the buffer containing the source data to be written.
idx_u32	{uint32_t number}	The virtual start address for writing in Data Flash aligned to word size (4 bytes).
cnt_u16	{uint16_t number}	Number of words to write.
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	Selects the Flash pool in which the command will be able to operate.

Structure member	Value	Description
status_enu	-	This is an output member. It contains the status of the operation during and after the execution. Possible values are described in the next table.

The following table describes all possible status returns:

Table 6: Write operation returned status

Status	Background and Handling	
R_FDL_BUSY	meaning	Operation started successfully
	reason	No problems during execution
	remedy	Call `R_FDL_Handler` until the Flash operation is finished, reported by the request structure status return value
R_FDL_OK(1)	meaning	Operation finished successfully
	reason	No problems during execution
	remedy	Nothing
R_FDL_SUSPENDED(1)	meaning	An on-going Flash operation was successfully suspended
	reason	Suspend processing successfully finished
	remedy	Start another operation or resume the suspended operation
R_FDL_CANCELLED(1)	meaning	An on-going or suspended Flash operation was successfully cancelled
	reason	Cancel processing successfully finished
	remedy	Start another operation
R_FDL_ERR_PARAMETER	meaning	Current command is rejected
	reason	Wrong command parameters: • access is made outside of physically available Data Flash • command shall operate in User-pool but `accessType_enu` is not `R_FDL_ACCESS_USER` • command shall operate in EEL-pool but `accessType_enu` is not `R_FDL_ACCESS_EEL` • `cnt_u16` is 0 or it is too big • flash writing address is not aligned with granularity (4 bytes)
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_PROTECTION	meaning	Current command is rejected
	reason	Other device specific protection mechanisms (e.g. security unit like ICU or FHVE protection mechanisms) prevent Flash operations.
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_REJECTED(2)	meaning	Current command is rejected
	reason	Another operation is ongoing

Status	Background and Handling	
	remedy	Request again the command when the preceding command has finished
R_FDL_ERR_WRITE(1)	meaning	Data could not be written correctly
	reason	• User flow issue: write on not completely erased Flash area • FDL was initialized with incorrect CPU frequency • Hardware defect
	remedy	• Erase write area before writing • Re-initialize FDL with correct frequency • A Flash block respectively the complete Data Flash should be considered as defect
R_FDL_ERR_INTERNAL	meaning	A library internal error occurred, which could not happen in case of normal application execution
	reason	An error occurred that cannot be determined by the library, such as caused by: • FDL code or data sections destruction, wrong program flow, Flash hardware modification • Hardware defect
	remedy	Please refer to section 3.9 “Internal error“

(1) `R_FDL_Execute` will never set this status code

(2) `R_FDL_Handler` will never set this status code

4.5.3 R_FDL_CMD_BLANKCHECK

The blank check command can be used by the requester to check whether a specified amount of memory starting from a specified address is written. This command will stop at the first memory location that is not erased with status `R_FDL_ERR_BLANKCHECK`.

Notes:

1. On blank check fail, the cells are surely not blank. This might result from successfully written cells, but also from interrupted execution of erase or write commands (resulting in partially written or erased cells).
2. On blank check pass, the cells are surely not written. This might result from successfully erased cells, but also from interrupted execution of erase or write commands (resulting in partially written or erased cells).
3. On blank check pass, there is a theoretical chance that a further write command will end with write error if the cells level is very near to the blank check level.
4. Depending on the Flash operations use case (e.g. EEPROM emulation) it may be necessary to log the Flash operations results in order to be sure that Flash cells are correctly written or erased. The way of logging depends on the use case (e.g. as part of an EEPROM emulation concept)
5. Internally blank check operation is split into smaller operations every time the operation crosses a 0x1000 bytes boundary. This means that time to suspend is not going to exceed the time to fully perform a blank check on 0x1000 bytes.

The members of the request structure given to `R_FDL_Execute` are described in the following table:

Table 7: Request structure usage for blank check command

Structure member	Value	Description
command_enu	R_FDL_CMD_BLANKCHECK	Request a blank check operation
bufAddr_u32	-	Not used
idx_u32	{uint32_t number}	Input: The virtual start address for performing blank check in data flash. Must be word (4 bytes) aligned. Output: Fail address in case of blank check error, unchanged if the operation finishes with `R_FDL_OK`.
cnt_u16	{uint16_t number}	Number of words (4 bytes) to check
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	Selects the Flash pool in which the command will be able to operate.
status_enu	-	This is an output member. It contains the status of the operation during and after the execution. Possible values are described in the next table.

The following table describes all possible status returns:

Table 8: Blank check operation returned status

Status	Background and Handling	
R_FDL_BUSY	meaning	Operation started successfully
	reason	No problems during execution
	remedy	Call `R_FDL_Handler` until the Flash operation is finished, reported by the request structure status return value
R_FDL_OK(1)	meaning	Operation finished successfully
	reason	No problems during execution
	remedy	Nothing
R_FDL_SUSPENDED(1)	meaning	An on-going Flash operation was successfully suspended
	reason	Suspend processing successfully finished
	remedy	Start another operation or resume the suspended operation
R_FDL_CANCELLED(1)	meaning	An on-going or suspended Flash operation was successfully cancelled
	reason	Cancel processing successfully finished
	remedy	Start another operation
R_FDL_ERR_PARAMETER	meaning	Current command is rejected

Status	Background and Handling	
	reason	Wrong command parameters: • access is made outside of physically available Data Flash • command shall operate in User-pool but `accessType_enu` is not `R_FDL_ACCESS_USER` • command shall operate in EEL-pool but `accessType_enu` is not `R_FDL_ACCESS_EEL` • `cnt_u16` is 0 or it is too big • flash blank check address is not aligned with granularity (4 bytes)
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_PROTECTION	meaning	Current command is rejected
	reason	Other device specific protection mechanisms (e.g. security unit like ICU or FHVE protection mechanisms prevent Flash operations)
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_REJECTED(2)	meaning	Current command is rejected
	reason	Another operation is ongoing
	remedy	Request again the command when the preceding command has finished
R_FDL_ERR_BLANKCHECK(1)	meaning	Affected Flash area is not completely blank (See notes above regarding interpretation of the check result!)
	reason	• (Partly) written Flash area is checked • Not completely erased Flash area is checked
	remedy	Remedy depends on the expected result of the Blank Check: • `R_FDL_ERR_BLANKCHECK` was the expected result: Nothing to do, the Flash contains data • `R_FDL_OK` was the expected result: Perform a Flash Erase operation
R_FDL_ERR_INTERNAL	meaning	A library internal error occurred, which could not happen in case of normal application execution
	reason	An error occurred that cannot be determined by the library, such as caused by: • FDL code or data sections destruction, wrong program flow, Flash hardware modification • Hardware defect
	remedy	Please refer to section 3.9 “Internal error“

(1) `R_FDL_Execute` will never set this status code

(2) `R_FDL_Handler` will never set this status code

4.5.4 R_FDL_CMD_READ

The read operation will read a certain address range in the Data Flash and copy the data to the specified target buffer.

Additionally, ECC errors in the read data will be signalled to the user application by the request structure (error status and first error address). In case of single bit ECC error, the data read will be continued and the 1st occurrence of the ECC error will be returned. In case of double bit error, the read operation is stopped and the fail address is returned. In case of a previous single bit error detected, the fail address of the single bit error is overwritten.

Read command execution is synchronous to execution of `R_FDL_Execute` function. Therefore this command cannot be suspended and does not need to be processed by `R_FDL_Handler` function.

The members of the request structure given to `R_FDL_Execute` are described in the following table:

Table 9: Request structure usage for read command

Structure member	Value	Description
command_enu	R_FDL_CMD_READ	Request a read operation
bufAddr_u32	{uint32_t number}	Data destination buffer address in RAM. **Note:** The buffer must be 32-bit aligned!
idx_u32	{uint32_t number}	Data Flash virtual address from where to read. Must be word (4 bytes) aligned.
cnt_u16	{uint16_t number}	Numbers of words (4 bytes) to read
accessType_enu	R_FDL_ACCESS_USER / R_FDL_ACCESS_EEL	Selects the Flash pool in which the command will be able to operate.
status_enu	-	This is an output member. It contains the status of the operation during and after the execution. Possible values are described in the next table.

The following table describes all possible status returns:

Table 10: Read operation returned status

Status	Background and Handling	
R_FDL_OK	meaning	Operation finished successfully
	reason	No problems during execution
	remedy	Nothing
R_FDL_ERR_PARAMETER	meaning	Current command is rejected

Status	Background and Handling	
	reason	Wrong command parameters: • access is made outside of physically available Data Flash • command shall operate in User-pool but `accessType_enu` is not `R_FDL_ACCESS_USER` • command shall operate in EEL-pool but `accessType_enu` is not `R_FDL_ACCESS_EEL` • `cnt_u16` is 0 or it is too big • flash read address is not 4-byte aligned • buffer address is 0x0 or not 4-byte aligned
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_PROTECTION	meaning	Current command is rejected
	reason	Other device specific protection mechanisms (e.g. security unit like ICU or FHVE protection mechanisms prevent Flash operations.
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_REJECTED	meaning	Current command is rejected
	reason	Another operation is ongoing
	remedy	Request again the command when the preceding command has finished
R_FDL_ERR_ECC_SED	meaning	The read operation detected single bit ECC error(s) in the read data. Single bit errors are automatically corrected by the ECC logic. The address of the first error occurrence is returned in the request structure.
	reason	• Not completely written or erase Flash • Cell level degradation by time • Hardware defect
	remedy	Remedy depends on the use case: • In case of reading possibly not completely written Flash is intended, reaction depends on the usage concept • In case of successfully written Flash is expected, try to refresh the data (Erase the Flash and write the data again) or refrain from further Flash operations and investigate in the root cause of the error
R_FDL_ERR_ECC_DED	meaning	The read operation detected a double bit ECC error in the read data. This error cannot be corrected by the ECC logic. The read operation will stop at the failing address and the fail address is returned.
	reason	• Not completely written or erase Flash • Hardware defect

Status	Background and Handling	
	remedy	Remedy depends on the use case: • In case of reading possibly not completely written Flash is intended, reaction depends on the usage concept. • In case of successfully written Flash is expected, refrain from further Flash operations and investigate in the root cause of the error.
R_FDL_ERR_INTERNAL	meaning	A library internal error occurred, which could not happen in case of normal application execution
	reason	An error occurred that cannot be determined by the library, such as caused by: • FDL code or data sections destruction, wrong program flow, Flash hardware modification • Hardware defect
	remedy	Please refer to section 3.9 “Internal error“

The following figure shows the handling of ECC error registers during read command execution:

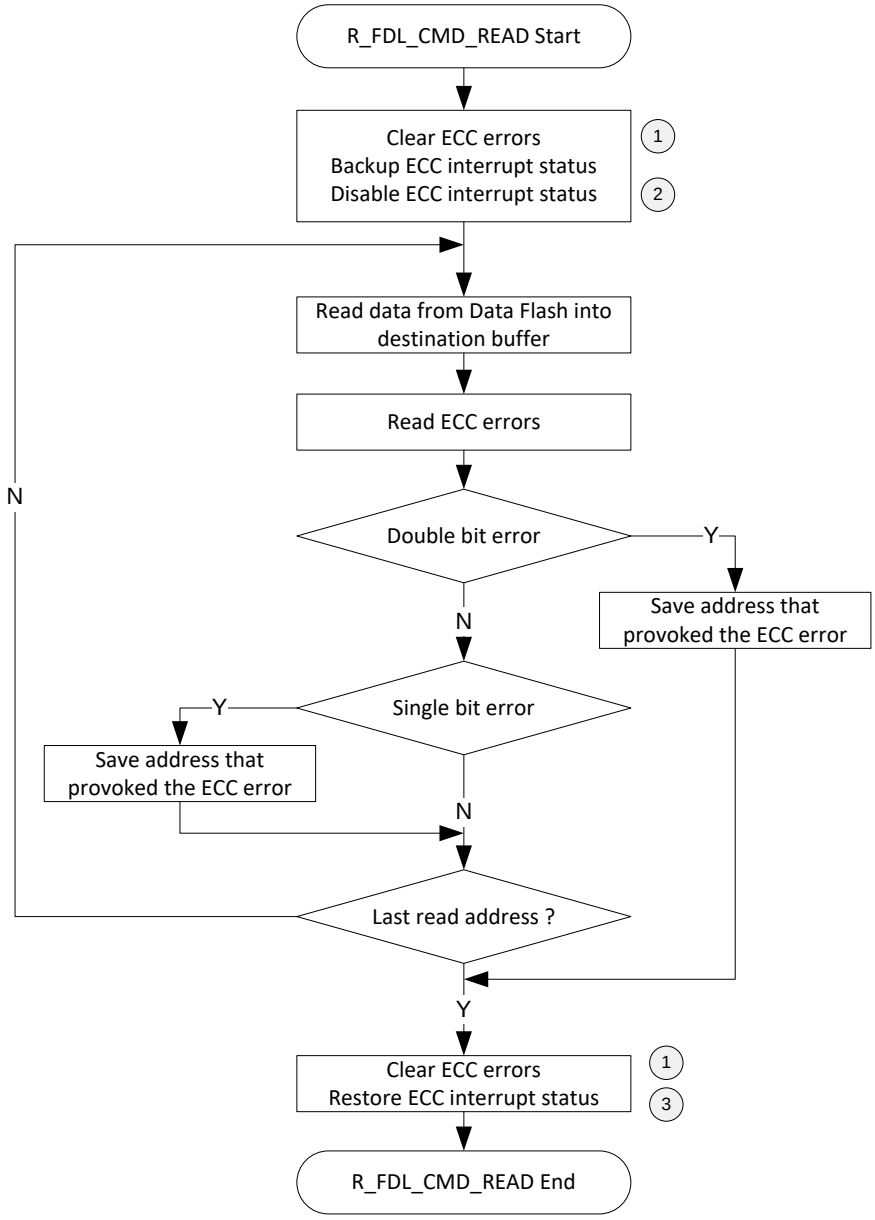


Figure 16: Handling of ECC error registers during read command

The user shall take into consideration that the following registers are modified:

1. DFERSTC register is written to clear any errors in DFFSTERSTR
2. DFERRINT register is backed up and cleared
3. DFERRINT register is restored

4.5.5 R_FDL_CMD_PREPARE_ENV

The prepare environment command is used to copy used firmware code to the RAM. After copy process is finished, the command will perform calculations and setting of FACI frequency. This command resets the FCU and initializes the hardware registers to default values.

Note, however, that library's internal functions will not be copied to RAM when the RH850/F1K, F1KM, F1KH, D1M1A, D1M1-V2, or D1S1 is in use.

Note:

The Code Flash might become inaccessible during command execution. Please refer to Chapter 6 Cautions for limitations that must be considered.

If the compatibility mode (refer to chapter 4.1 “Pre-compilation configuration” for details) is disabled, the command shall be executed after the R_FDL_Init and before any other command.

If the compatibility mode (refer to chapter 4.1 “Pre-compilation configuration” for details) is enabled, the command may not be executed since it is part of the `R_FDL_Init` function. In this case, if the command is still executed then the library will return the status `R_FDL_ERR_REJECTED`.

Table 11: Request structure usage for prepare environment command

Structure member	Value	Description
command_enu	R_FDL_CMD_PREPARE_ENV	Prepare Flash environment
bufAddr_u32	-	Not used
idx_u32	-	Not used
cnt_u16	-	Not used
accessType_enu	-	Not used
status_enu	-	This is an output member. It contains the status of the operation during and after the execution. Possible values are described in the next table.

The following table describes all possible status returns:

Table 12: Prepare environment operation returned status

Status	Background and Handling	
R_FDL_BUSY	meaning	Operation started successfully
	reason	No problems during execution
	remedy	Call `R_FDL_Handler` until the Flash operation is finished, reported by the request structure status return value
R_FDL_OK(1)	meaning	Operation finished successfully
	reason	No problems during execution
	remedy	Nothing
R_FDL_ERR_CONFIGURATION(1)	meaning	Current command is rejected
	reason	Wrong parameters have been passed to the FDL by descriptor: • frequency outside the allowed range • FDL pool size value is higher than physical Data Flash size
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_REJECTED(2)	meaning	Current command is rejected
	reason	Another operation is ongoing

Status	Background and Handling	
	remedy	Request again the command when the preceding command has finished
R_FDL_ERR_PROTECTION	meaning	Flash hardware operation protected
	reason	Other device specific protection mechanisms (e.g. security unit like ICU or FHVE protection mechanisms) prevent Flash operations.
	remedy	Refrain from further Flash operations and investigate in the root cause
R_FDL_ERR_INTERNAL(1)	meaning	A library internal error occurred, which could not happen in case of normal application execution
	reason	An error occurred that cannot be determined by the library, such as caused by: • FDL code or data sections destruction, wrong program flow, Flash hardware modification • Hardware defect
	remedy	Please refer to section 3.9 “Internal error“

(1) `R_FDL_Execute` will never set this status code
(2) `R_FDL_Handler` will never set this status code

Chapter 5 Library setup and usage

This chapter contains important information about how to put the FDL into operation and how to integrate it into your application. Please read this chapter carefully — and also especially Chapter 6 “Cautions” — in order to avoid problems and malfunction of the library. Before integrating the library into your project, however, please make sure that you have read and understood how the FDL works and which basic concepts are used (see Chapter 2 “Architecture” and Chapter 3 “Functional specifications”).

5.1 Obtaining the library

The FDL is provided by means of an installer via the Renesas homepage at

http://www.renesas.eu/update

Please follow the instructions of the installer carefully. Please ensure to always work on the latest version of the library.

5.2 File structure

The library is delivered as a complete compilable sample project which contains the FDL and in addition an application sample to show the library implementation and usage in the target application.

The delivery package contains dedicated directories for the library, containing the source and the header files.

5.2.1 Overview

The following picture contains the library and the application related files:

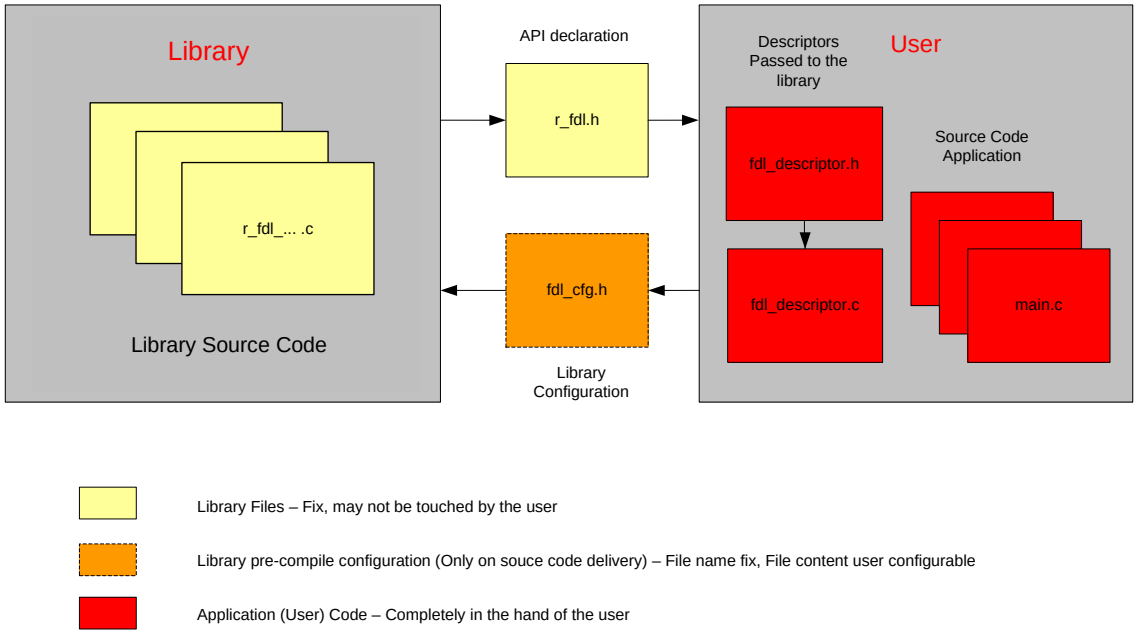


Figure 17: File structure of library and sample application

The library must be configured for compilation. The file `fdl_cfg.h` contains defines for that. As it is included by the library source files, the file contents may be modified by the user, but the file name may not.

These files reflect an example, how the library descriptor variable can be built up and passed to the function `R_FDL_Init` for run-time configuration. The structure of the descriptor is defined in `r_fdl_types.h` which needs to be included in the user application. The value definition should be done in the file `fdl_descriptor.h`. The constant variable definition and value assignment should be done in the file `fdl_descriptor.c`. If adding the files `r_fdl_ descriptor.c`/`h` to the application, only the file `fdl_descriptor.h` needs to be adapted by the user, while `fdl_descriptor.c` may remain unchanged. For usage please refer to chapter 4.2 “Run-time configuration”.

5.2.2 Delivery package directory structure and files

The following table contains all files installed by the library installer:

- Files in red belong to the build environment, controlling the compile, link and target build process
- Files in blue belong to the sample application
- Files in green are description files only
- Files in black belong to the FDL

Table 13: File structure of the FDL package

File		Description
<installation_folder>/FDL		
Release.txt		Library package release notes
support.txt		List of supported devices
<installation_folder>/FDL /<compiler>/<device_name>		
Build.bat		Batch file to build the FDL sample application
Clean.bat		Batch file to clean the FDL sample application
Makefile		Make file that controls the build and clean process
<installation_folder>/FDL /<compiler>/<device_name>/Sample(1)		
dr7f70xxxx_startup.850(3)	<for GHS compiler>	Device and compiler specific start-up code
cstart.asm	<for REC compiler>	
cstartup.s	<for IAR compiler>	
dr7f70xxxx.ld(3)	<for GHS compiler>	Compiler specific linker directives
dr7f70xxxx.dir(3)	<for REC compiler>	
layout.icf lnkr7f70xxxxxafp.icf(3)	<for IAR compiler>	
dr7f70xxxx.dvf.h (3) dr7f70xxxx_irq.h(3)	<for GHS compiler>	Definitions of IO registers, interrupt and exceptions vector table, for RH850 devices. <for GHS compiler>: Use dr7f70xxxx.dvf.h(3) or dr7f70xxxx_0.h(3), and io_macros_v2.h. <for REC compiler>: Use "boot.asm" or "vecttbl.asm".
iodefine.h boot.asm	<for REC compiler>	
ior7f70xxxxxafp.h(2) (3)	<for IAR compiler>	
app.h		Sample application code
fdlapp_control.c		
fdlapp_main.c		
target.h		Initialization code for target microcontroller

File	Description
fdl_cfg.h	FDL pre-compilation definitions
fdl_descriptor.c	FDL descriptor used in the sample application
fdl_descriptor.h	
fdl_user.c	User defined code for handling interrupts and library pre-initialization
fdl_user.h	
<installation_folder>/FDL /<compiler>/FDL	
r_fdl.h	FDL API definitions
r_fdl_types.h	User interface type definitions, error and status codes
<installation_folder>/FDL /<compiler>/FDL/lib	
r_typedefs.h	C types used by FDL library
r_fdl_mem_map.h	Section mapping definitions
r_fdl_env.h	Internal FDL definitions
r_fdl_global.h	Global variables and settings
r_fdl_hw_access.c	FDL main source code
r_fdl_user_if.c	
r_fdl_user_if_init.c	

(1) File names are dependent on the chosen device. Shown filenames are valid for F1L devices.

(2) This file is not included in the library installer.
Please obtain this file from IAR development environment.
e.g.) C:\Program Files\IAR Systems\Embedded Workbench 7.3\rh850\inc

(3) xxxx is a number (e.g. dr7f701007).

(4) The make.exe file which is run from the batch files that come with the RH850 data flash library (FDL) Type01 is an external tool, and requires downloading from a Web site that distributes make.exe. As stated in Release.txt, GNU Make was used to confirm operation of the sample application. If you wish to use a GNU Make environment, download make.exe from the Web site of GNU Make and install it. Execute the batch files after that.

5.3 Library resources

5.3.1 Linker sections

The following sections are related to the Data Flash Access Library and need to be defined in the linker file (please see sample linker directive file for an example):

Data sections:

- `R_FDL_DATA` - This section contains all FDL internal variables. It can be located either in internal or external RAM.

Code sections:

- `R_FDL_CONST` - This section contains library internal constant data. It can be located anywhere in the code flash.
- `R_FDL_TEXT` - FDL code section containing the library code. It can be located anywhere in the code flash.

RAM code sections (optional, if `R_FDL_EXE_INIT_CODE_ON_STACK` is not defined, see 4.1 “Pre-compilation configuration”):

- R_FDL_CODERAM - FDL code to be executed from RAM when Code Flash is not available during environment preparation. It can be located in any internal RAM. If initialization is done from stack buffer this section is empty otherwise it has a size of 256 bytes.

5.3.2 Stack and data buffer

The FDL utilizes the same stack as specified in the user application. It is the developer's duty to reserve enough stack for the operation of both, user application and FDL. With source code library it is not possible to give an exact value for stack consumption. However, an estimate value for the FDL library is: 384 bytes for GHS compiler and 416 bytes for Renesas compiler when stack is used for code RAM execution during R_FDL_Init function. If reserved section R_FDL_CodeRam is used, then stack consumption is reduced with 252 bytes.

The data buffer used by the FDL refers to the RAM area in which data is located that is to be written into the data flash. This buffer needs to be allocated and managed by the user.

Note:

In order to allocate the stack and data buffer to a user-specified address, please utilize the link directives of your framework.

5.4 MISRA compliance

The FDL code has been tested regarding MISRA™ compliance.

The used tool is the QA C™ Source Code Analyzer which tests against the MISRA C™ 2004 standard rules.

Note:

"MISRA" and "MISRA C" is a registered trademark of HORIBA MIRA Ltd, held on behalf of the MISRA Consortium. "QA C" is a registered trademark of Programming Research Ltd.

All MISRA related rules have been enabled. Remaining findings are commented in the code while the QAC checker machine is set to silent mode in the concerning code lines.

5.5 Sample application

It is very important to have theoretic background about the Data Flash and the FDL in order to successfully implement the library into the user application. Therefore it is important to read this user manual in advance. The best way, after initial reading of the user manual, will be testing the FDL application sample.

After a first compile run, it will be worth playing around with the library in the debugger. By that you will get a feeling for the source code files and the working mechanism of the library. After this exercise it might be easier to understand and follow the recommendations and considerations of this document.

Note:

Before the first compile run, the compiler path must be configured in the "Makefile" of the sample application: set the variable COMPILER_INSTALL_DIR to the correct compiler directory.

5.6 Library configuration

Before using the Data Flash Access library, the library has to be configured and adapted to a certain degree in order to fit the requirements of the user application. For information about configuration settings and handling, please refer to chapter 4.2 “Run-time configuration”.

5.7 Basic programming flow

The following flow chart shows the basic reprogramming flow for a certain Data Flash range.

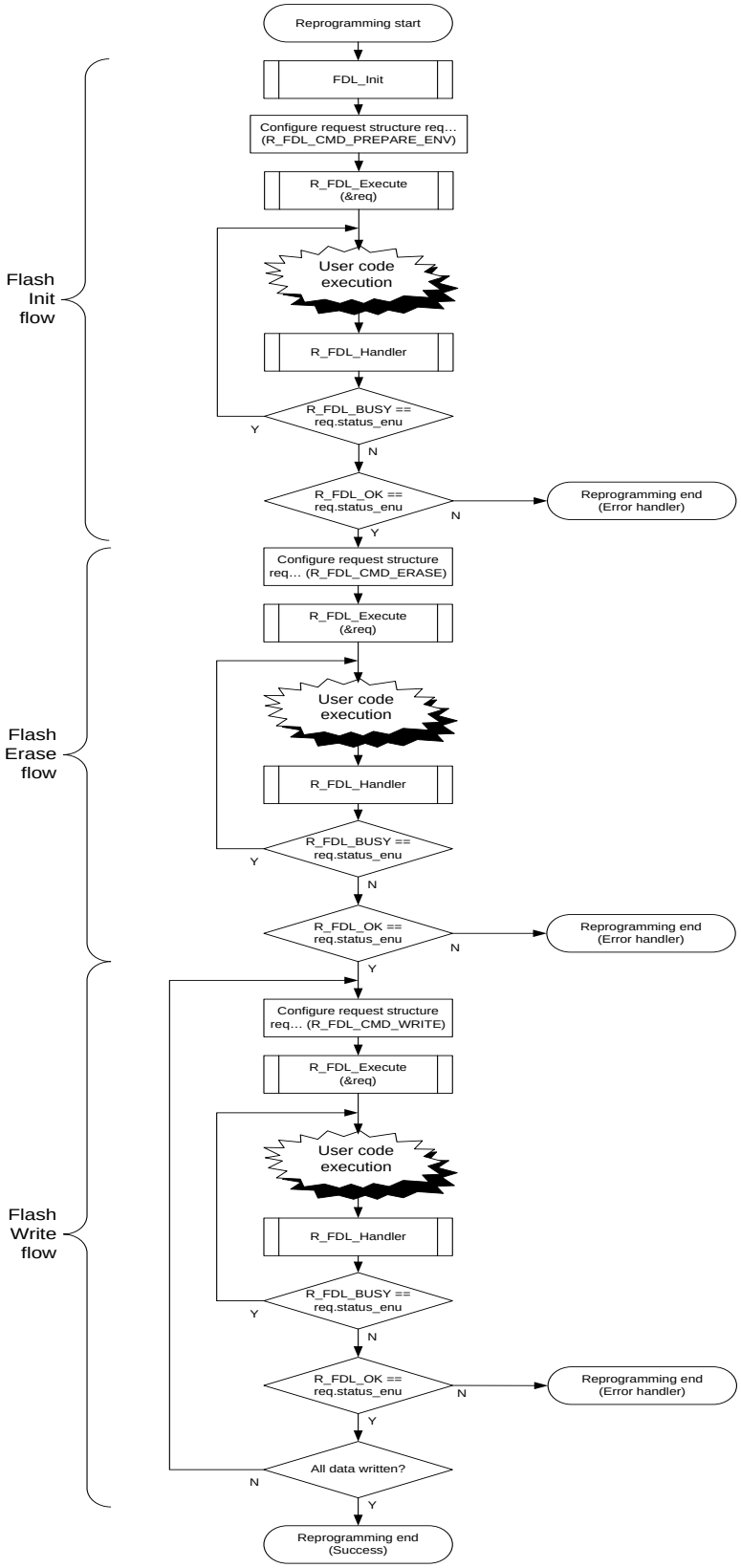


Figure 18: Basic programming flow

Error treatment of the FDL functions themselves is not detailed in the flow charts for simplification reasons.

For details on enabling or disabling access to the Data Flash, refer to the user's manual for the hardware. An example is given by the sample application, file `sample_app_main.c`, functions `FDL_Open` and `FDL_Close`.

Note:

If the compatibility mode is enabled (refer to chapter 4.1 “Pre-compilation configuration” for details) then execution of command `R_FDL_CMD_PREPARE_ENV` is not required in the programming flow.

5.8 R_FDL_Handler calls

Once initiated FDL operations need to be driven forward by successive handler calls. The frequency of these handler calls does have an impact on the FDL operation performance and needs to be adapted to the target application.

In the following, different approaches for calling the `R_FDL_Handler` are compared with respect to their advantages and disadvantages:

- Calling `R_FDL_Handler` repeatedly after starting an operation execution: This approach is also utilized in most of the code examples you can find in this manual. Typically realized in a loop waiting for the operation status not to be busy anymore, this approach results in the best FDL operation performance. However, the CPU is fully loaded and blocked for other tasks as long as the FDL operation is being executed.
- Calling `R_FDL_Handler` in a timed task: By calling the `R_FDL_Handler` periodically, FDL commands can be driven forward while other tasks are processed by the CPU. The period between the status check calls can have significant impact on the FDL operation performance. Shorter calling intervals result in better FDL performance, but also increase the CPU load by the FDL. Due to this trade-off, a general advice for the calling interval cannot be given. It needs to be analysed and tailored individually for each target application.
- Calling `R_FDL_Handler` in the idle task: If it is ensured that the idle task is called often enough, this method might result in a good FDL performance, as the handler can be called continuously. However, this approach is not deterministic in case of a high CPU load by the application itself.

Due to the individual requirements of each application, a general advice for selecting a strategy to call the `R_FDL_Handler` cannot be given. Please also consider that mixtures of the above-mentioned approaches can be meaningful depending on the target scenario.

Note:

When evaluating concepts for calling the `R_FDL_Handler`, please be aware that all FDL functions are not re-entrant. That means it is not allowed to call an FDL function from interrupt service routines while another FDL function is already running.

Chapter 6 Cautions

Before starting the development of an application using the FDL, please carefully read and understand the following cautions:

1. CPU operating frequency configuration:

   Correct frequency configuration is essential for Flash programming quality and stability. Wrong configuration could lead to Flash operation fail.

   The limits for CPU frequency are device dependent. Please consult the device user's manual for correct range.

   If the CPU frequency is a fractional value, round up the value to the nearest integer.

   If you are using an RH850/F1KM-S4 or RH850/F1KH-D8, the operating frequency of the flash sequencer controlled by the FDL is 1/8 of the CPU operating frequency (CPU clock at up to 240 MHz) by default.

   fPCLK = 1/8 fCPUCLK_H (with CKDIVMD= 1 and CPU operating frequency of up to 240 MHz)

   When the option byte setting of the RH850/F1KM-S4 or RH850/F1KH-D8 is CKDIVMD=0 (up to 120 MHz), however, the operating frequency of the flash sequencer controlled by the FDL must be changed to 1/4 of the CPU operating frequency.

   fPCLK = 1/4 fCPUCLK_H (with CKDIVMD=0 so max. CPU operating frequency is 120 MHz)

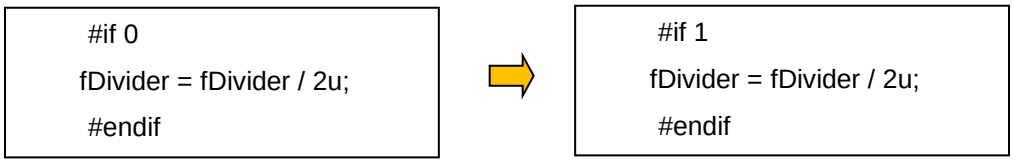
   If you are using V2.13 or a later version of the FDL, you can change the operating frequency of the flash sequencer to support CPU operating frequencies up to 120 MHz.

   To change the operating frequency of the flash sequencer, enable the line of source code that halves the denominator of the frequency ratio within the R_FDL_FCUFct_SetFrequency function in the "r_fdl_hw_access.c" file that comes with the FDL.

   Specifically, change "#if 0" to "#if 1" on the line following the comments after the "Changing CKDIVMD" keyword.

   <Keyword>

   ```
   /*********************************************************************
   * SAMPLE: Changing CKDIVMD
   *********************************************************************/
   ```

   Change "#if 0" to "#if 1" in cases where you wish to use the RH850/F1KM-S4 or RH850/F1KH-D8 with the CKDIVMD=0 (up to 120 MHz) setting.

   *V2.12 and earlier versions of the FDL do not support the CKDIVMD=0 (up to 120 MHz) setting.

2. CPU mode:

   The initialization command `R_FDL_CMD_PREPARE_ENV` must be executed in CPU supervisor mode. Please consult the device user's manual for details.

3. Function re-entrancy:

   All functions are not re-entrant. So, re-entrant calls of any FDL function must be avoided.

4. Task switch, context change, synchronization between functions:

   Each function depends on global available information and is able to modify this information. In order to avoid synchronization problems, it is necessary that at any time only one FDL or FCL function is executed. So, it is not allowed to start an FDL or FCL function, then switch to another task context and execute another FDL or FCL function while the last one has not finished.

5. Entering power save (stand-by) mode:

   Entering power save mode is not allowed at all during on-going Data Flash operations. Use `R_FDL_StandBy` or wait until operations are no longer busy.

6. Different power save (stand-by) modes:

   Please do not enter a power save mode which resets/alters the Flash hardware or memory required for library operation - e.g. stack, library data or library operation related data (such as request variable). This need to be considered, because resuming the previous operation is not possible otherwise. The library is not able to detect such failure. A possible power save mode is HALT.
   If entering other modes, the FDL need to be re-initialized by `R_FDL_Init`.

7. Initialization:

   The FDL library initialization by means of calling `R_FDL_Init` must be performed before calling most of the library functions. Exception is `R_FDL_GetVersionString` function that can be called anytime.

8. Critical section*1 handling:

   If the compatibility mode is enabled (refer to chapter 4.1 “Pre-compilation configuration” for details), the `R_FDL_Init` function temporarily disables Code Flash. If compatibility mode is disabled, execution of command `R_FDL_CMD_PREPARE_ENV` *2 temporarily disables Code Flash. During this time, since the Code Flash is not available, the library is executing code from internal RAM (allocated space on stack). Please ensure that:

   - Code execution is done from other locations (e.g. internal RAM).
   - No access to Code Flash is allowed, e.g. by jump to interrupt/exception functions, direct Code Flash Read/Execution from the CPU, DMA accesses to Code Flash. The user can configure the provided callback macro functions in `fdl_cfg.h`. , in order to handle e.g. interrupt & exception disable, DMA,... .The sample application provides examples on how to disable and restore interrupts and exceptions using the callback routines.

*1: For macro definitions related to critical sections, refer to 1, Critical section.

*2: While compatibility mode is enabled (as described in 4, Compatibility mode), calling the R_FDL_Init function temporarily disables the code flash memory.

9. Interrupted flash operations:

   In case of Flash modification operation (Erase / Write) interruption, 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. Furthermore, 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.

10. Write operation:

    Before executing a write operation, please make sure the given address range is erased.

11. Reading Data Flash:

    Data Flash on RH850 devices is based on a complementary read concept. Concept wise, reading erased Data Flash will show undefined data with a tendency to the previously written data. Additionally, most probably ECC errors are signalled.
    In case of reading the Data Flash directly by the CPU/DMA (not using the `R_FDL_CMD_READ` command), such ECC errors will result in ECC error interrupts/exceptions if the device is configured accordingly.

    DMA transfers from Data Flash are permitted, but need to be synchronized with the FDL.

    During command execution Data Flash is not available. Any direct read during command execution will result in invalid read data; therefore it must be avoided.

    The command `R_FDL_CMD_BLANK_CHECK` can be utilized to avoid reading blank Flash areas.

12. Dual operation / Parallel execution of FDL and FCL on a target device:

    - Both Flash libraries share the same Flash programming environment as resource. Thus, execution of any command must be synchronized on application level. It is not allowed to execute FDL and FCL functions or commands at the same time.
    - Cancel and suspend/resume operations are not allowed as the effect is not evaluated.
    - Standby is allowed but both instances have to consider that wakeup is required before continuing. Neither FDL nor FCL functions may be called before `R_FDL_WakeUp` execution.

13. Reusing the request command:

    Do not change the content of the request structure while an FDL command is operating because the library will not work correctly and/or data loss can occur. Multiple requests, each using different request structures, do not have these adverse effects.

14. Workload and supervision:

    It is recommended to supervise the FDL operations and functions execution by timeout supervision (e.g. timer, counter, watchdog, etc.). In addition, the user of the library should evaluate the time necessary to perform a certain operation and divide long lasting operations to meet real-time system specifications.

15. Suspend and stand-by restrictions:

    Suspend restrictions:

    - Erase operation ► suspend ► Erase operation – is not possible
    - Write operation ► suspend ► Erase/Write operation – is not possible
    - Any operation ► suspend ► other operation ► suspend – is not possible

    Suspend permissions:

    - Blank Check operation ► suspend ► Erase/Write/Blank Check/Read operation – is possible
    - Erase operation ► suspend ► Write/Blank Check/Read operation – is possible
    - Write operation ► suspend ► Blank Check/Read – is possible

    New Flash operations after suspending a Flash operation are only allowed on Flash areas not affected by the suspended operation.

    Standby restrictions:

    - Any operation ► stand-by ► only wake-up is possible

    It is recommended to avoid nesting as much as possible.

16. Stand-by:

    Do not continue FDL functions execution or start execution of any other function than R_FDL_GetVersionString, R_FDL_WakeUp or R_FDL_Init when the library is in stand-by mode.

17. Data alignment:

    Data Flash blocks are aligned to 64 bytes and Data Flash words are aligned to 4 bytes.

    While the source buffer of the command R_FDL_CMD_WRITE can be unaligned, the destination buffer of the command R_FDL_CMD_READ must be 32-bit aligned.

18. Pre-compilation options:

    The user must not use any pre-compilation configuration options that are not documented in this manual.

19. Supported devices:

    Please refer to the installer readme file to understand which device families are supported by the FDL.

20. Data Flash – Secure Erase:

    Based on the complementary read, the data flash has a tendency to show the previously written data even after a successful erase operation. To destroy the data in a secure way, the following flow is recommended:

    - Erase the intended area
    - Write a fixed pattern e.g. 0x00000000 to the complete area
    - Erase the same area again

    Using this flow, it is ensured that the read of the erased area will have a tendency towards the pattern, in the above example towards 0x00000000, and not to the previous, meaningful data.

21. Cancel suspended operation:

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

22. Only one library instance:

    More than one FDL instance can exist on a single device but only one library instance shall be executed at any given time.

23. Access protection check violation:

    The library performs access protection checks before starting any Flash operation. On violation of this check the library returns R_FDL_ERR_INTERNAL and remains in an undefined state. Re-initialization of the library is required.

    The conditions described above will not occur in case of correct library handling, but only in case of library misuse.

24. Protection error during resume request handling:

    During a resume request handling for a suspended operation, the error R_FDL_ERR_PROTECTION may appear. In this case, refrain from further Flash operations and investigate in the root cause (ongoing Data or Code Flash operation) and re-initialize the library in order to continue the reprogramming flow.

    The conditions described above will not occur in case of correct library handling, but only in case of library misuse.

25. Areas to be accessed when a pre-compilation definition is specified:

    Access by the FDL is to specific areas in accord with the pre-compilation definition setting during initialization processing.

    When you execute the `R_FDL_CMD_PREPARE_ENV` command of `R_FDL_Execute`, permit reading from the following areas.

Pre-compilation definition supported by FDL V2.12 and later versions	Areas to be accessed
R_FDL_NO_BFA_SWITCH	0x01030000 to 0x0103029F
R_FDL_MIRROR_FCU_COPY	0x01030000 to 0x0103029F, 0x01037000 to 0x01037FFF
R_FDL_NO_FCU_COPY	0x00010000 to 0x0001029F
None	0x00010000 to 0x0001029F, 0x00017000 to 0x00017FFF

Revision History

Chapter	Page	Description
		Rev. 1.03: Initial released document version
		Rev. 2.00:
3.1	14	Added cancel request and prepare environment command
3.5	19	Removed immediate resume figure
3.7	22	Added cancel mechanism handling
4.1	25	Removed Device family and added compatibility mode
4.2	26	Removed authentication ID
4.3.2	27	Removed authentication ID
4.3.3	28	Added R_FDL_CMD_PREPARE_ENV
4.3.4	29	Added R_FDL_CMD_PREPARE_ENV
4.3.6	30	Added R_FDL_CANCELLED status
4.4.1.1	32	Updated description
4.4.2.1	34	Added example for R_FDL_CMD_PREPARE_ENV
4.4.3.1	37	Updated return values
4.4.3.2	39	Updated return values
4.4.3.5	43	Added new interface function: R_FDL_CancelRequest
4.5.1	46	Added R_FDL_CANCELLED status
4.5.2	48	Added R_FDL_CANCELLED status
4.5.3	50	Added R_FDL_CANCELLED status
4.5.5	55	Added new command: R_FDL_CMD_PREPARE_ENV
5.2.2	59	Added IAR related files and r_fdl_user_if_init.c
5.7	62	Updated reprogramming flow figure
6	65	Added new cautions
		Rev. 2.11:
All		Minor description corrections and wording update
All		Document formatting update
All		Updated cross-references
-	1	Updated title and installer name
-	2	Updated Notice text
-	-	Deleted Regional information page
-	-	Deleted Preface page
-	3	Updated How to use this document page
1	6,7	Updated location specific text and Flash Granularity
2.1	8	Updated 2.1 Layered architecture
2.3	9	Updated 2.3 Architecture related notes
3.5	14	Updated 3.5 Suspend / Resume mechanism
3.8,3.9	21	Added new chapter 3.8 and 3.9
4.1	22,23	Added new static configuration options
4.2	24	Updated 4.2 Run-time configuration
4.3.3	27	Updated Member /Value for 4.3.3 r_fdl_request_t

Chapter	Page	Description
4.4.2.2	35	Updated Preconditions for 4.4.2.2 R_FDL_Handler
4.4.3.3	39	Updated 4.4.3.3 R_FDL_StandBy
4.5.1	47	Updated Table 4 for R_FDL_ERR_ERASE and R_FDL_ERR_INTERNAL
4.5.2	49	Updated Table 6 for R_FDL_ERR_WRITE and R_FDL_ERR_INTERNAL
4.5.3	49	Added new note to R_FDL_BLANKCHECK command
	51	Updated Table 8 for R_FDL_ERR_BLANKCHECK and R_FDL_ERR_INTERNAL
4.5.4	52	Updated 4.5.4 R_FDL_CMD_READ
	53,54	Updated Table 10 for R_FDL_ERR_PARAMETER and R_FDL_ERR_ECC_SED and R_FDL_ERR_ECC_DED and R_FDL_ERR_INTERNAL
4.5.5	57	Updated Table 12 for R_FDL_ERR_INTERNAL
5.2.1	58	Updated Figure 17: File structure of library and sample application
5.2.2	59,60	Updated Table 13: File structure of the FDL package
5.3.1	60	Updated sections
5.3.2	61	Updated stack resources
6	64-66	Updated cautions
		Rev. 2.12:
4.1	22,23	Updated 4.1 Pre-compile configuration
4.3	25	Updated 4.3 Data Types
4.4.3.5	42	Added R_FDL_ERR_INTERNAL to Return value
4.5.5	56	Updated 4.5.5 R_FDL_CMD_PREPARE_ENV
5.2.2	59,60	Updated Table 13: File structure of the FDL package
6	64,67	Updated No.1 of Chapter 6 Cautions and added No.25
		Rev. 2.13:
3.6	17	Updated 3.6 Stand-by and Wake-up functionality
4.1	23	Updated note for 4.1 Pre-compilation configuration
4.3	25	Updated 4.3 Data types
4.3.1	26	Updated 4.3.1 Library specific simple type definitions
4.4.3.3	41	Updated Example of 4.4.3.3 R_FDL_StandBy
4.4.3.4	42	Updated 4.4.3.4 R_FDL_WakeUp
4.5.5	56	Updated 4.5.5 R_FDL_CMD_PREPARE_ENV
5.4	62	Updated 5.4 MISRA compliance
6	66,67	Updated No.1 and No.8 of Chapter 6 Cautions

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

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

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

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

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 101-T01, Floor 1, Building 7, Yard No. 7, 8th Street, Shangdi, Haidian District, Beijing 100085, 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 200333, China
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 No 3A-1 Level 3A Tower 8 UOA Business Park, No 1 Jalan Pengaturcara U1/51A, Seksyen U1, 40150 Shah Alam, Selangor, Malaysia
Tel: +60-3-5022-1288, Fax: +60-3-5022-1290

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 6.0

RH850 Family User’s Manual: Data Flash Library Type T01

Publication Date: Rev.1.00 Nov 11, 2013
Rev.2.13 Jun 10, 2019

Published by: Renesas Electronics Corporation

Data Flash Library Type T01

Renesas Electronics Corporation

R01US0079ED0213