TLE985x

Microcontroller with FastLIN and Power Switches for Automotive Applications

Firmware User Manual

Revision 1.0

2019-03-05

Automotive Power

Revision History
Microcontroller with FastLIN and Power Switches for Automotive Applications

Page or Item	Subjects (major changes since last revision)
Revision 1.0, 2019-03-05	
	Initial release

Table of Contents

1 Introduction

This document specifies the BootROM firmware behavior for the TLE985x microcontroller family.

The specification is organized into the following major sections:

Table 1-1 Document Content Description

Topic	Description
Startup procedure	**BootROM Startup procedure**: An overview on the Startup procedure: the first steps executed by the BootROM after a reset,
FastLIN BSL features	**Boot Strap Loader (BSL)**: An overview on the BSL: the module used to download and to run code from NVM and RAM
	BSL commands - Protocol (Version 2.0): Details and Commands description
	BSL via FastLIN (UART via Local Interconnected Network)
NVM structure	**NVM**: An overview on the NVM: the module used to initialize and program the NVM sectors and pages
User Routines description	**User Routines**: User routines description

1.1 Purpose

The document describes the functionality of the BootROM firmware.

1.2 Scope

The BootROM firmware for the TLE985x family will provide the following features

- Startup procedure for stable operation of TLE985x chip
- Debugger connection for proper code debug
- BSL mode for users to download and run code from NVM and RAM
- NVM operation handling, e.g. program, erase and verify

1.3 Abbreviations and Special Terms

A list of terms and abbreviations used throughout the document is provided in **“Terminology” on Page 120**.

2 Overview

This specification includes the description of all firmware features including the operations and tasks defined to support the general startup behavior and various boot options.

2.1 Firmware Architecture

The BootROM in the TLE985x consists of a firmware image located inside the device's ROM. It consists of the startup procedure, the bootstrap loader via FastLIN, NVM user routines and NVM integrity handling routines.

The BootROM in TLE985x is located at the address 00000000_H, and so represents the standard reset handler routine. The BootROM firmware is executed in the ARM Cortex CPU core and uses the SRAM for variables and software stack.

Figure 2-1 shows the TLE985x components used during execution of the BootROM.

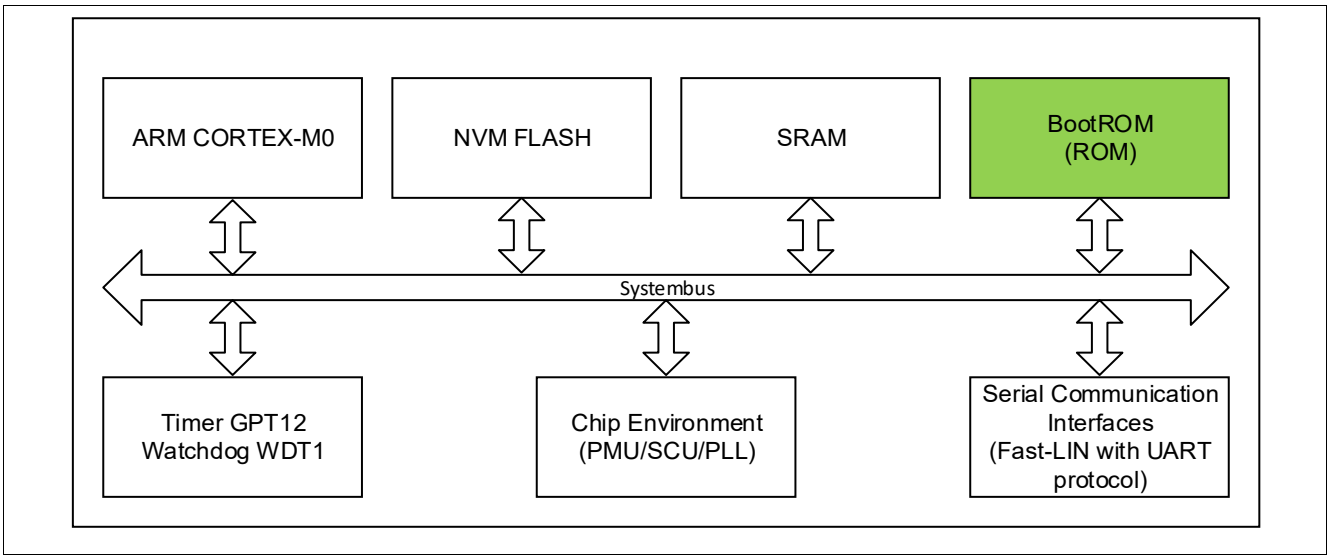


Figure 2-1 Block Diagram of the BootROM and its Interaction with other TLE985x Components

The startup procedure is the first software-controlled operation in the BootROM that is automatically executed after every reset. Certain startup submodules are skipped depending on the type of reset (more details are provided in **"Reset Types" on Page 15**) and the error which might occur (more details are provided in **"Startup Error Handling" on Page 18**).

The startup procedure includes the NVM initialization, PLL configuration, enabling of NVM protection, branching to the different modes and other startup procedure steps.

There are two operation modes in the BootROM :

- BSL mode
- User/Debug mode

The deciding factor will be on the latch values of TMS and P0.0 upon a reset. During reset, these signals are latched at the rising edge of RESET pin. Details are provided in **"Boot Modes" on Page 11**.

2.2 Program Structure

The different sections of the BootROM provide the following basic functionality.

Startup procedure

The startup procedure is the main control program in the BootROM. It is the first software-controlled operation in the BootROM that is executed after any reset.

User/Debug mode

It is used to support user code execution in the NVM address space. However, if the Bytes at address 11000004_H-11000007_H are erased ($FFFFFFFF_H$), then device enters sleep mode.

If a valid user reset vector was found at 11000004_H (values at 11000004_H - 11000007_H not equal to $FFFFFFFF_H$) and a proper No Activity Counter (NAC) value is found then the BootROM proceeds into user mode.

In case an invalid NAC value is found (see also **“NAC Definition” on Page 12**), the device waits indefinitely for a FastLIN BSL communication.

BSL mode

The BSL mode is used to support BSL via the FastLIN protocol. Downloading of code/data to RAM and NVM is supported in this mode.

3 BootROM Startup procedure

This chapter describes the BootROM startup procedure in TLE985x.

The startup procedure is the first software-controlled operation in the BootROM that is automatically executed after every reset.

There are 2 operation modes in the BootROM :

- User/BSL mode
- Debug Support mode

The operation modes get selected dependent on the latch values of two (2) pins upon reset. Details are provided in **“Boot Modes” on Page 11**.

For each HW module a HW abstraction layer (HAL) is implemented with its associated module specific firmware functions called by the BootROM startup procedure

Figure 3-1gives an overview by showing the startup code partitioning into firmware modules and the corresponding dataflow.

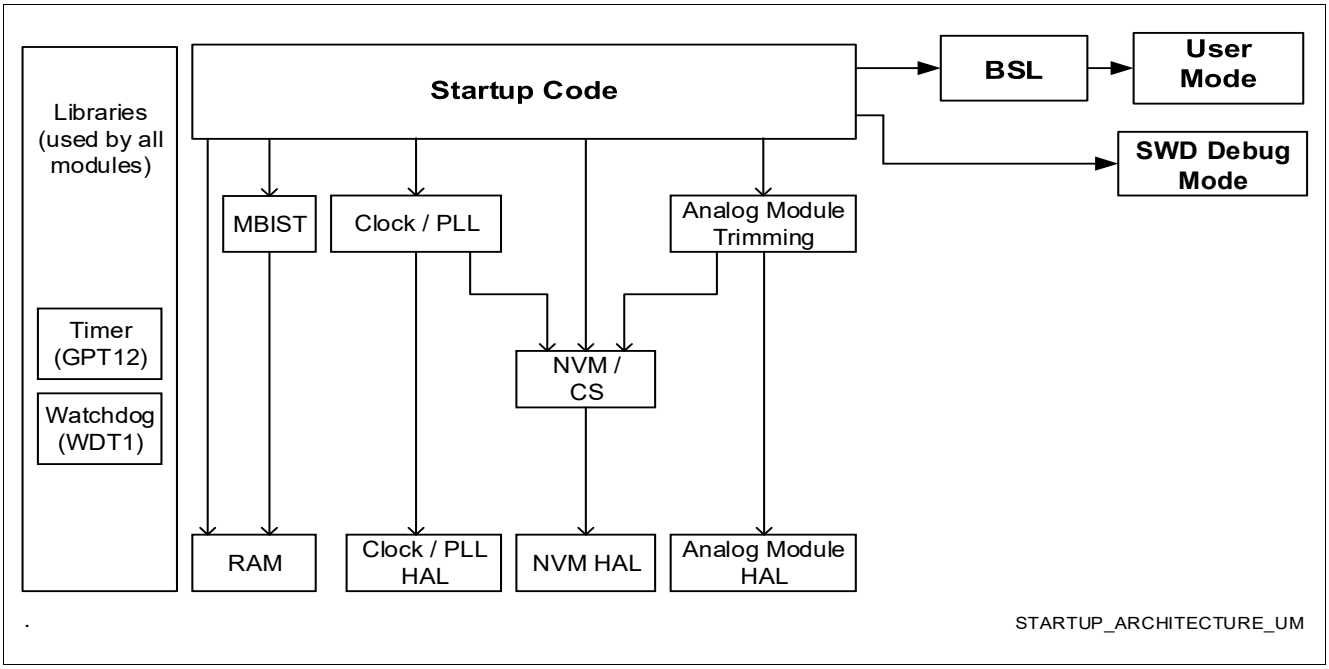


Figure 3-1 Startup Procedure Architecture Overview

The startup code performs different device initialization steps.

After initialization, the BootROM either starts BSL communication (according to configuration) or jumps to user mode code execution.

For user mode, bootROM will execute the startup procedure, redirect the vector table to the beginning of the NVM in user accessible space and jump to the customer defined reset handler routine (jump to the address pointed by the address 11000004_H) to execute the user program.

3.1 Startup Program Structure

The first task executed by the BootROM startup procedure is to check the reset type.

The BootROM also reads the logical state of certain external Pins (see **“Boot Modes” on Page 11**) to decide which initialization sub modules to be executed or to be skipped during the startup sequence.

A list of supported boot mode pin selections is given in **“Boot Modes” on Page 11**.

Many of the submodule initialization tasks require further configuration parameters which are stored in the NVM CS (Configuration Sector).

The initialization process differs slightly between each selected boot mode. Each boot mode has a different set of initialization steps to be performed. For instance, some initialization steps might be skipped for one mode but carried out for another mode. Some initialization steps are bypassed for hot reset.

The functional blocks are listed in **Table 3-1**.

Various flowcharts for the different boot modes are shown in **“Flowcharts for User BSL / Debug Modes” on Page 14**.

Table 3-1 Functional Blocks

Block	Description	Reference
Watchdog Disable	The WDT1 is disabled, depending on the boot mode.	**Section 3.8.1**
RAM MBIST	Performs RAM MBIST (MBIST range depends on RAM mode setting).	**Section 3.8.2**
RAM Init	Inits RAM to zero (Init range depends on RAM mode setting).	
MapRAM Init	Inits MapRAM based on MapBlock data. The dedicated service algorithm is applied (only executed in NVM data linear mode).	**Section 5.4**
Analog Module Trimming	Analog module NVM CS trimming values are configured in the hardware.	**Section 3.8.5**
PLL Init	Switch system clock to PLL	**Section 0.3.8**
Start NAC Timer	Start a timer which is dedicated to the user mode / BSL “no activity count timeout” calculation.	**Section 3.8.8**
BSL	BSL communication	**Chapter 4**

3.2 Boot Modes

The different BootROM-supported boot modes are listed in **Table 3-2 “BootROM Boot Modes” on Page 11**. Device enters into a specific bootmode based on pin configuration during reset release. The mode decides which initialization parts are to be executed by the BootROM.

Table 3-2 BootROM Boot Modes

TMS / SWD_IO	P0.0 / SWD_CLK	Mode / Comment
0	X	**USER_BSL_MODE** User Mode / BSL Mode
1	1	**SWD_DEBUG_MODE** Debug Support Mode with SWD port
All other values		Reserved for internal use

3.3 Debug Support Mode Entry (with SWD port)

Debug support mode is available for SWD interface. The BootROM starts the overall device initialization as described in **“Startup Program Structure” on Page 10**.

The BootROM then enters a waiting loop to synchronize with a debugger connected to the Serial Wire Debug (SWD) interface. After that, the BootROM finishes the boot process and starts to execute user code under debugger control.

Firmware ensures that jumping to user code in user- or debug mode is performed with the same RAM and SFR content, except for a WDT1 user code entry.

The watchdog is always disabled in debug support mode, except when the debug error loop is entered after a boot error.

3.4 NAC Definition

The No Activity Counter (NAC) value defines the time window after reset release, within which the firmware is able to receive BSL connection messages. If no BSL messages are received during the NAC window and NAC time has expired the firmware code proceeds to user mode.

The NAC value is a byte value which describes the timeout delay with a granularity of 5 ms. The NAC timeout supports a maximum of 140 ms, corresponding to NAC=$1C_H$. User API and the BSL command to write the NAC value check the provided NAC value and discard values too high (limit due to WDT) and too low (limit due to the fastest possible passphrase sequence). **Table 3-3** shows the valid NAC values.

Table 3-3 Valid NAC Value

NAC Value	Time out behavior
00_H	the BSL window is closed, no BSL connection is possible and the user mode is entered without delay.
01_H	report error (value not supported)
02_H-$1C_H$	time out delay of NAC*5ms before jumping to user code
$1D_H$-FE_H	report error (values not supported)
FF_H	no timeout is used, BootROM code will switch off WDT1 and wait indefinitely for a BSL connection attempt

After ending the start up procedure, the program will detect any activity on the FastLIN interface for the remaining NAC window. When no activity is detected, the program will jump to user mode. **“FastLIN Passphrase and Node Address for Diagnostic (NAD)” on Page 21**

In case a valid BSL passphrase is detected during the BSL window the firmware suspend the counting of the WDT1 in order to avoid that requested BSL communication is broken by a WDT1 reset. The firmware will then re-enable WDT1 before jumping to user code.

User mode is entered by jumping to the reset handler. This can happen directly from the startup routine, after the NAC waiting time for possible BSL communication, or as a result of BSL commands. In startup, a jump to user mode will only occur if the NVM content at 11000004H-11000007H is not FFFFFFFFH, otherwise, the BootROM executes an endless loop. In BSL execution commands, it jumps to the address specified as an input through the command itself.

3.4.1 Unlock BSL Communications

The BootROM locks the FastLIN communication after reset to avoid unexpected BSL communication on the customer side. The host needs to unlock the communication by sending a passphrase sequence to the BootROM.

Details about this passphrase and how it influences the NAC timeout are given in **"FastLIN Passphrase and Node Address for Diagnostic (NAD)" on Page 21*****.***

3.4.2 Post User Mode Entry Recommendations

Upon USER MODE entry, it is highly recommended to perform the following checks and actions:

Prior to any NVM operation, it is recommended to implement a test of the bit MRAMINITSTS in the register SCU_SYS_STRTUP_STS (SCU_SYS_STRTUP_STS.MRAMINITSTS).

If the bit is clear then the data flash mapping is consistent, NVM write/erase operation can be performed. To see if the Service Algorithm might have been active the user has to check the MEMSTAT register. If the Service Algorithm was active the user has to expect that expected logical data flash pages are not present anymore. The user has to take care of this and reconstruct any missing page. Furthermore it might be possible that the Service Algorithm (**Chapter 5.4, Data Flash Initialization**) reports an unrecoverable failure inside the Data Flash, then the same corrective actions shall be applied as described in the following paragraph for the case that SCU_SYS_STRTUP_STS.MRAMINITSTS is set.

If SCU_SYS_STRTUP_STS.MRAMINITSTS is set, then the data flash mapping is inconsistent, the mapping might not be complete and any NVM operation like write or erase is not safe and might cause further inconsistencies inside the data flash. As corrective actions the user might reset the device (cold reset) in order to give the Service Algorithm a chance to repair the data flash sector. If this attempt fails again, then a sector erase is needed to reinitialize the data flash sector and to remove any mapping inconsistency. After the data flash sector has been erased the user has to take care of reconstructing the expected logical data flash pages.

The reset source should get read from the PMU Reset Status Register (PMU_RESET_STS). Clearing PMU_RESET_STS is strongly recommended in the user startup code, as uncleared bits can cause a wrong reset source interpretation in the BootROM firmware after the next reset (e.g. handling a warm reset as a cold reset).

The system startup status register SCU_SYS_STRTUP_STS should get checked for any startup fails. See the TLE985xQX User's Manual for a detailed register description.

3.5 User and BSL Mode Entry

Entry to user mode is determined by the No Activity Count (NAC) value, see **"NAC Definition" on Page 12**.

After waiting the time defined by the current NAC value, the startup procedure sets the VTOR register to point to the beginning of the NVM (11000000_H) and starts user code execution at the address vector found at 11000004_H (Reset Vector). It is the responsibility of the user to provide a meaningful VTOR table at the beginning of the NVM.

If NVM double Bit error occurs when reading the NAC value, the system goes into an endless loop waiting for BSL communication. Before entering User mode (except for Hot Reset, see **Figure 3-3 "Flowchart – User BSL Mode (UM)" on Page 15**), the system clock frequency is switched to PLL output and to the max. frequency as stated in the datasheet. In case PLL has not locked within 1 ms, the clock source fINTOSC/4 (20 MHz) will be used. After every reset, the user shall check whether the system is running on the low precision clock or on the PLL output reading SYSCON0.SYSCLKSEL register and e.g. restart the PLL if necessary.

3.6 Flowcharts for User BSL / Debug Modes

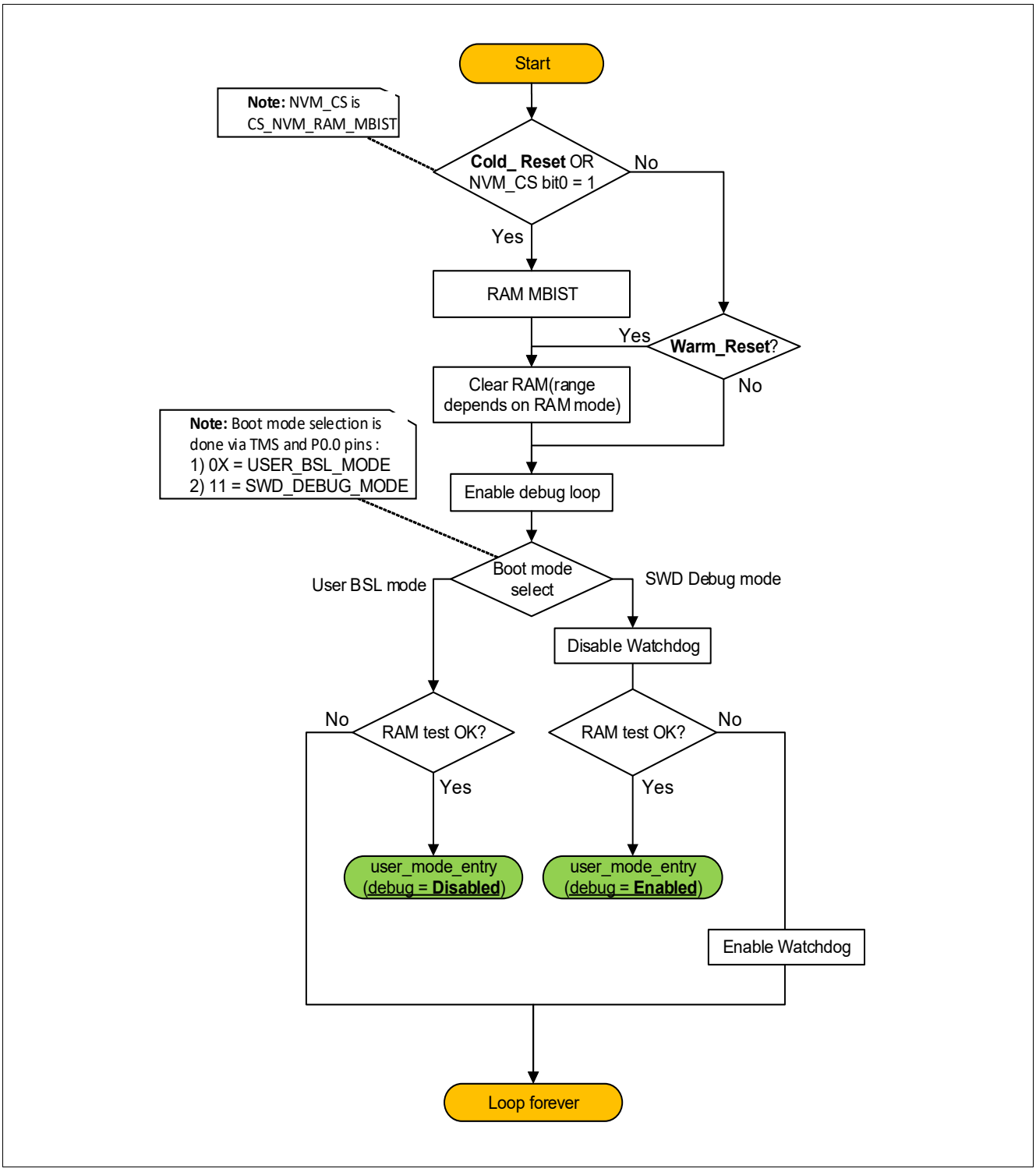


Figure 3-2 Flowchart – Reset (UM)

Figure 3-3 “Flowchart – User BSL Mode (UM)” on Page 15 shows the flow of User BSL mode. This is the default user entry mode, in which the BootROM configures the device with user device variants. If device init fails (e.g. due to NVM not available, NVM init error or trimming error), the bootROM protects the complete NVM

region and starts error handling. Otherwise, the BootROM executes a set of sequences before jumping to user code. BSL mode is entered in case of a cold/warm reset and returns to the main flow in case of a timeout.

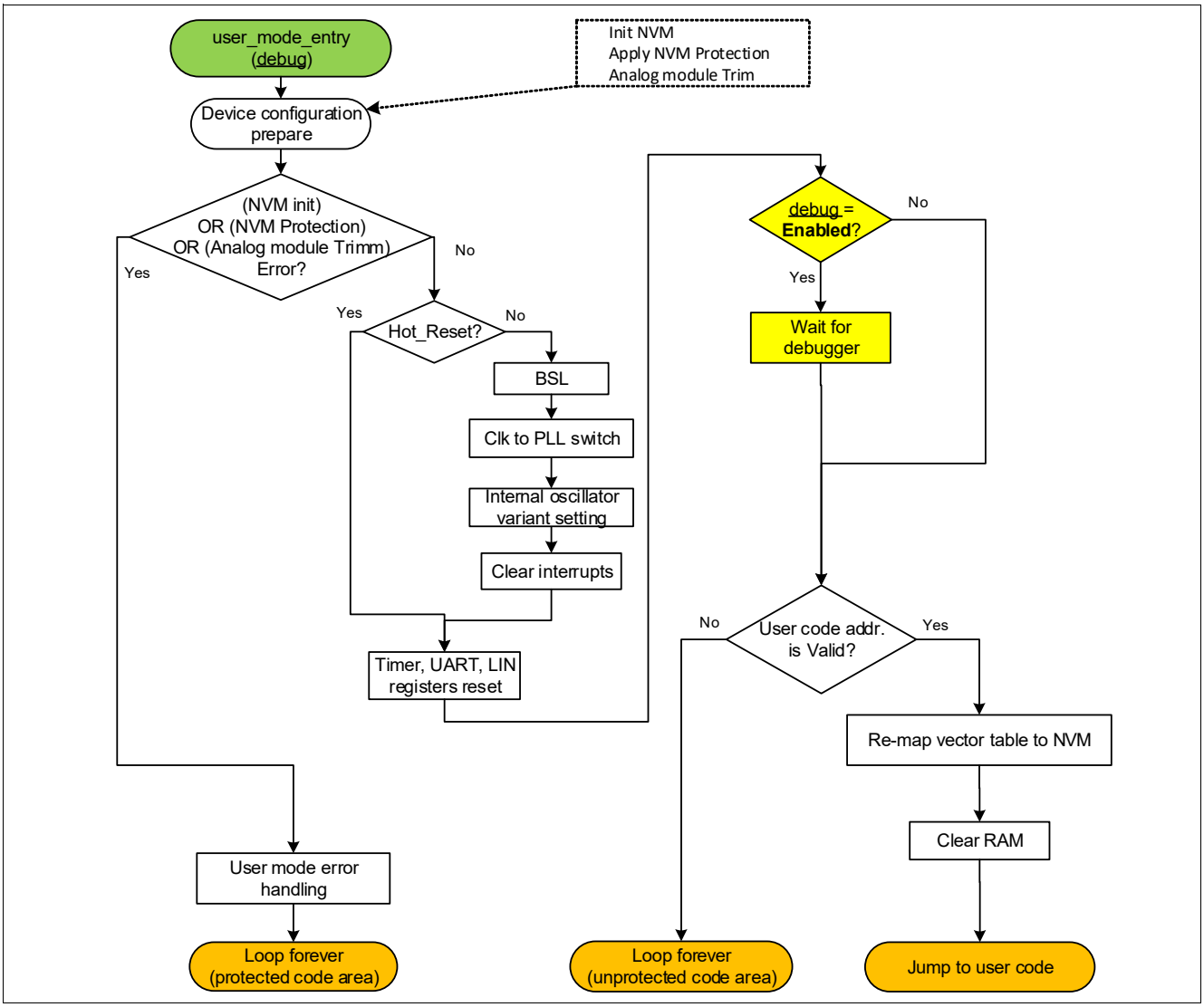


Figure 3-3 Flowchart – User BSL Mode (UM)

3.7 Reset Types

The BootROM classifies the different hardware resets according to the following reset types:

- Cold reset
- Warm reset
- Hot reset

Cold reset

The reset events generated from the following sources, are classified as cold resets:

- POR: Power-on reset
- Pin reset
- Watchdog reset

- System fail

After a cold reset, all initialization steps, listed in **Table 3-1 “Functional Blocks” on Page 11**, are processed in accordance with the boot mode. In user/debug mode, the RAM MBIST & Init range depends on RAM mode setting.

Warm Reset

The reset events generated from the following sources, are classified as Warm resets:

- Sleep-exit reset
- Stop-exit reset

After a warm reset, the initialization steps, listed in **Table 3-1**, are processed, except:

- RAM MBIST (only executed if forced by NVM CS configuration, as described in **“RAM MBIST and RAM Initialization” on Page 17**)

Hot Reset

The reset events generated from the following sources, are classified as Hot resets:

- Software triggered reset
- Lock-up reset

After a Hot reset, the initialization steps, listed in **Table 3-1**, are processed, except:

- RAM MBIST & RAM init- (only executed if forced by NVM CS configuration, as described in **“RAM MBIST and RAM Initialization” on Page 17**)
- Download of analog module trimming parameters (incl. oscillator and PLL settings)
- Switch system clock to PLL output

Reset priority

In case more than one reset event occur, the post reset initialization procedure with the highest priority type is executed. The priority is evaluated according to this priority order (where “1” is the highest priority):

1. Cold reset
2. Warm reset
3. Hot reset

Attention: ***The reset source is read from the PMU Reset Status Register (PMU_RESET_STS). Clearing PMU_RESET_STS is strongly recommended in the user startup code as uncleared bits can cause a wrong reset source interpretation in the BootROM firmware after the next reset (e.g. handling a warm reset as a cold reset).***

3.8 Startup Procedure Submodules

Startup submodules are described in this section.

3.8.1 Watchdog Configuration

After a reset, the watchdog WDT1starts with a long open window. WDT1 continues running while waiting for the first BSL frame. If host synchronisation is completed during the BSL waiting time (defined by NAC), WDT1 is disabled and its status is frozen.

WDT1 is re-enabled when entering user mode from BSL mode. The watchdog WDT1is disabled before entering into debug mode.

For all reset types, firmware startup in user mode enables WDT1 before jumping to user code, and the watchdog cannot be disabled while user code is being executed.

3.8.2 RAM MBIST and RAM Initialization

The firmware depending on the reset type performs test and/or initialization of the RAM. By default, the FW performs the RAM MBIST only for cold reset while the RAM initialization is executed for cold and warm reset

By default, no RAM MBIST and RAM initialization is performed for hot reset.

Specific user controllable configurations are offered to change the default behavior. By means of the **NVM CS Usage** configuration, user can enable the RAM MBIST also for warm and hot reset. .

In addition, user can partially disable the RAM MBIST at cold reset. Refer to **Section 3.8.4** for specific information.

The RAM MBIST consist of a linear write/read algorithm using alternating data on data and parity field and so it destroys the content and the parity integrity of the tested RAM. For this reason each time a RAM MBIST is executed, it is followed by a RAM initialization.

RAM initialization writes the target RAM region to zero with the proper parity thus preventing an ECC error during user code execution.

In case an error is detected in the RAM MBIST or RAM initialization, the appropriate error status is captured and the device enters an endless loop. As the watchdog is enabled when entering the endless error loop after a boot in user or debug mode, a WDT1 cold reset is asserted after timeout and the RAM MBIST or RAM initialization is re-executed.

After five (5) consecutive watchdog resets, the device enters SLEEP mode (by hardware function).

Note: The standard RAM interface is disabled during MBIST test execution.

Note: In case of RAM MBIST or initialization error, SCU_NVM_PROT_STS is untouched.

3.8.3 NVM CBSL Region Size Configuration

During startup, the CBSL configuration is read from the configuration sector (CS), the lower two bits get programmed into the NVM_PROT_STS register. See the register bit field NVM_PROT_STS.CUS_BSL_SIZE for a list of values defined. If the value read is FF_H or a ECC2 error has occurred, then a default value CS_CUST_BSLSIZE = 00_H is used.

3.8.4 RAM Mode Key and NVM Data Mode Key

The switching key feature provides a way to configure RAM mode (legacy or preserve mode) and NVM mode (data mapped or data linear mode). Each feature is configured by checking dedicated switch key locations in the NVM configuration space. In order to avoid mode switching failures due to e.g. read noise, the firmware evaluates during startup two key locations (one per feature) placed redundantly on 3 different pages. As long as at least one valid key is detected, the corresponding mode is switched on.The key values are predefined,

only the presence of that specific key value will enable the corresponding mode switch. The keys can only be written with a dedicated BSL command. The user can choose to write either a single RAM key, a single NVM key or both keys at once with one BSL command. The switching key set functionality is effective only when CS_SWITCH_KEY_CTRL_EN is enabled in the NVM configuration sector. For details about the BSL command for switch key programming, refer to **Chapter 4.3.9, Command $8A_H$ – NVM: Switch Keys Set**.

3.8.5 Analog Module Trimming

During analog module trimming, the trimming values of PMU, voltage regulators, FastLIN module, temperature sensor, oscillator, PLL, bridge driver and other analog modules are read from the NVM configuration sector and written into the respective SFRs. In case the 100 times programmable pages (100TP pages) 0 and 1 with data for the trimming process contain CRC errors (no user programming assumed), predefined SFR values from the third 100TP page are used.

- User 100TP Analog Trimming Data
  - The user has eight 100TP pages. The values of the first (page 0) and second (page 1) pages are automatically copied into the dedicated SFR registers after every cold or warm reset thus replacing the registers default reset values. The user can check them by reading the dedicated SFRs or by reading directly the content of the page.
  - This procedure allows the user to configure the ADC1 . The complete list of SFRs is provided in **“Analog Module Trimming (100TP Pages)” on Page 132**
  - In case the first and second 100TP NVM CS (Configuration Sector) pages do not contain valid trimming data (CRC failure), the BootROM reports error and copies alternative backup trimming values from the third (page 2) 100TP page.

3.8.6 ADC1 Core Offset Calibration

During device testing, a basic offset calibration step gets performed for each device. However, different temperatures, supply voltages, board layout or soldering stress in the customer application might impact ADC accuracy.
In cold or warm reset, the ADC1 core offset calibration can be enabled or disabled via a CS setting. Writing 0x01 in CS_ADC1_STARTUP_CALIBRATION enables ADC1 core calibration in startup. With all other values offset calibration is skipped.
Offset calibration is also available as a user API function, see **Chapter 10.9**, API:user_adc1_offset_calibration. The calibration process requires the measurement core module and ADC1 to be powered on.

3.8.7 Startup Error Handling

To ensure that the device is properly booted, error checking and error handling are added to the startup procedure.

For USER_BSL_MODE and SWD_DEBUG_MODE, any submodule failure(**Chapter 3.8, Startup Procedure Submodules**) brings the startup sequence into an endless loop with WDT1 enabled..

If a startup error occurs – except for double-bit errors for NVM reading – and the boot option is USER_BSL_MODE, the device is set to a safe mode with limited access to HW resources. The device reboots with each WDT1 timeout. If the errors persist after five (5) WDT1 triggered timeouts, the device enters SLEEP mode .

Regardless of the boot mode, the system enters an endless loop in the case of invalid user code address error, RAM MBIST error or NVM CS pages checksum error.

3.8.8 No Activity Counter (NAC) Configuration

A NAC timeout value is stored in a NVM CS page.

During user mode, this parameter is read from the NVM CS (Configuration Sector). This parameter is provided as an API parameter when calling the BSL module. For details, refer to **Section 3.4**.

If the NVM CS does not contain a valid NAC, a “wait forever” NAC (NAC=FF_H) is given to the BSL module.
A changed NAC value takes effect only after the next reset.

The BootROM offers 2 user API functions to read and write NAC parameter:

- **user_nac_get**
- **user_nac_set**

3.8.9 FastLIN Node Address for Diagnostics (NAD) Configuration

For FastLIN, a NAD value is stored in a NVM CS page.

During user mode, this parameter is read from the NVM CS (Configuration Sector). The parameter is provided as an API parameter when calling the FastLIN BSL module. For details, please refer to **“FastLIN Passphrase and Node Address for Diagnostic (NAD)” on Page 21**.

If the NVM CS (Configuration Sector) does not contain a valid NAD, a “broadcast” NAD (NAD=FF_H) is given to the FastLIN BSL module.

A changed NAD value takes effect only after the next reset.

The BootROM offers user APIs for reading and writing NAD parameter:

- **user_nad_get**
- **user_nad_set**

4 Boot Strap Loader (BSL)

The Boot Strap Loader (BSL) module supports handling of message-based command request and response communication over the FastLIN interface. The received command messages are parsed and executed according to the FastLIN protocol. The protocol message format is shared by theserial interface. Details about this message protocol are given in **"BSL commands - Protocol (Version 2.0)" on Page 29**.

Figure 4-1 shows the various software submodules in the BSL module. The BSL protocol is handled on a single protocol level that processes all messages described in **"BSL commands - Protocol (Version 2.0)" on Page 29**.

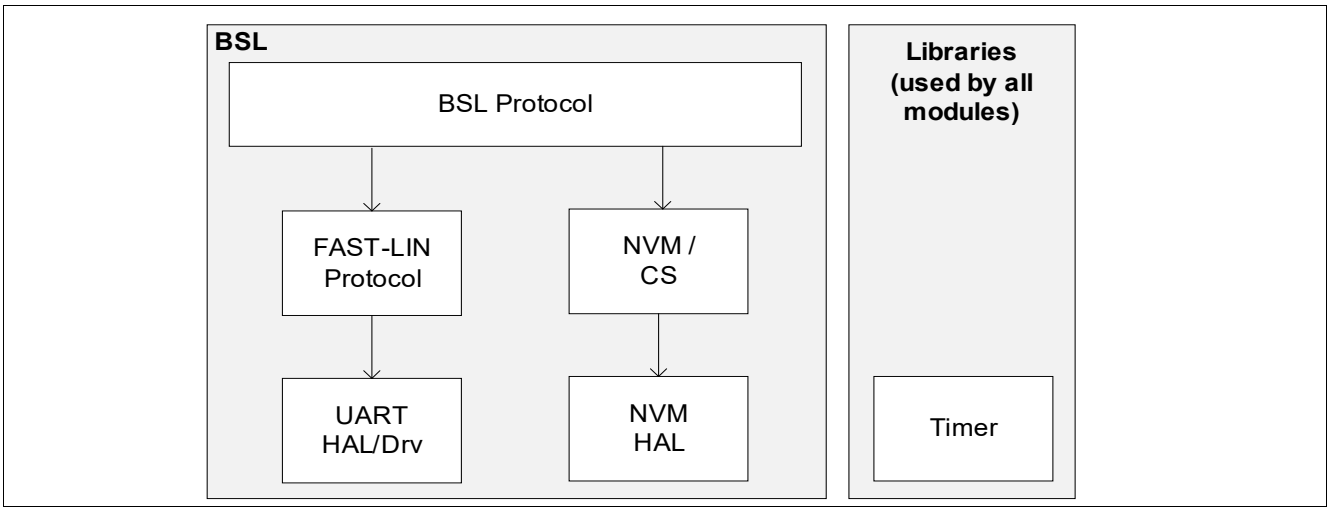


Figure 4-1 BSL Architecture

All command messages are encapsulated in an interface-specific frame format. This format includes specified parameters, such as a checksum calculation and overall message size. .

The BSL protocol layer performs the command execution based on the parsed BSL commands. This results in programming of the NVM, NVM CS (Configuration Sector), downloading to RAM or execution of NVM/RAM code. It also includes the aspect that some commands are blocked based on applied hardware protection or boot mode selection.

4.1 BSL Overview

In this chapter, more details about the BSL mechanisms are provided.

4.1.1 BSL Interframe Timeout

The interframe timeout is a configuration parameter read by BootROM startup code from the NVM CS (Configuration Sector).

The interframe timeout parameter has the same format as the NAC value (2 = 2x5ms).

The parameter value is set to 0x38, which results in a timeout value of 280ms (0x38 x 5ms).

4.1.2 NVM / RAM Range Access

BSL commands are available to access NVM and RAM.

In particular, BSL commands can access all user NVM, 100TP pages and the RAM area.

4.1.3 FastLIN Passphrase and Node Address for Diagnostic (NAD)

The BootROM locks the BSL FastLIN communication after reset to avoid unexpected BSL communication on the customer side. The host needs to unlock the communication by sending a passphrase sequence to the BootROM.

A passphrase consists of two consecutive frames, where each frame contains a fixed pattern. To unlock the BSL communication, both passphrase frames have to be sent by the host. Any other received message within the passphrase sequence stops the unlock sequence. The unlock procedure always restarts on receiving the first passphrase frame.

The contents of both passphrase frames are described in **Figure 4-2**.

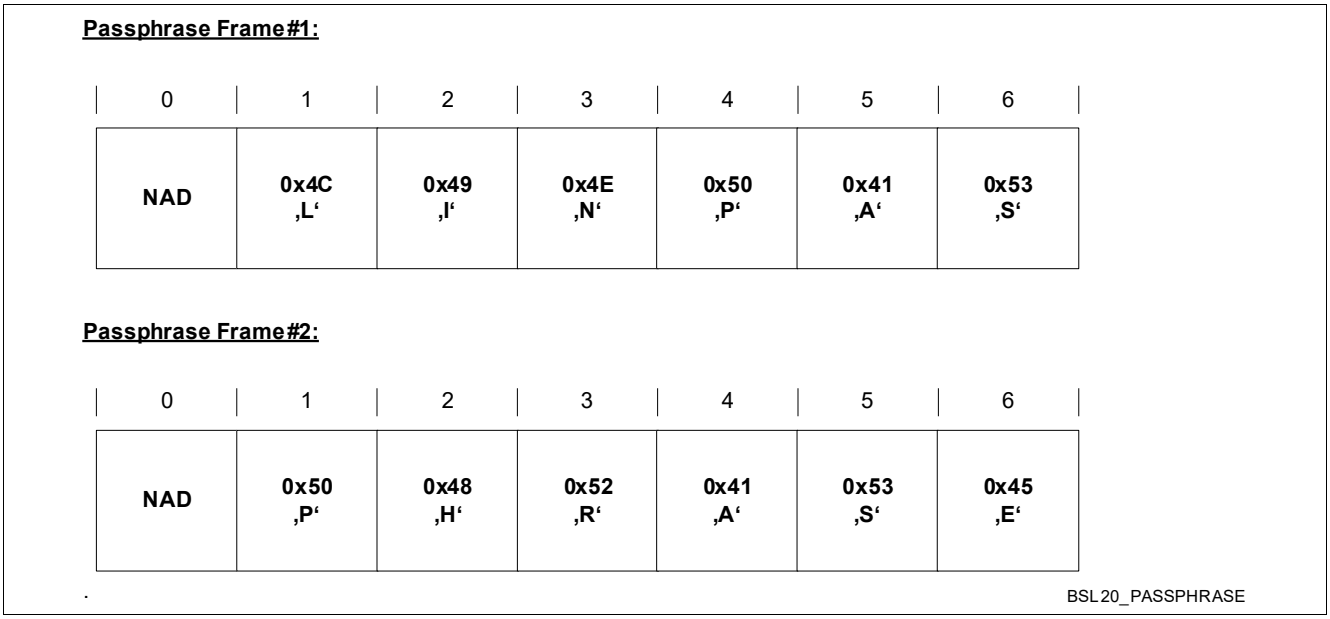


Figure 4-2 Passphrase Content

BSL communication supports node addressing (NAD).

The first byte of each passphrase frame carries the NAD field (1 byte, all value from 00_H to FF_H are valid), both NAD fields need to have the same value. The NAD field specifies the address of the active slave node (only slave nodes have a NAD address). **Table 4-1** lists the BootROM-supported NAD address ranges. The NAD address in place is set during device programming as a BSL API parameter (for details see also **“User and BSL Mode Entry” on Page 13**). The firmware treats a received BSL message with a NAD value of FF_H as a 'broadcast' message. The BSL responds to this no matter which NAD value is stored in the NVM CS.
With FastLIN communication, the passphrase frames are embedded in the BSL-via-FastLIN protocol and therefore get extended by a checksum byte field. Details about frame encapsulation are given in .
BSL communication only gets unlocked if both NAD fields and the passphrase matches and if the FastLIN frame checksum was received correctly.

In all other cases the BSL communication remains locked, all frames received get ignored, including FastLIN frames with valid checksum fields.

Regardless of the result, no response is sent back upon passframe reception.

The NAC timeout stops when the communication is unlocked after receiving the second valid passphrase frame. For more details about NAC timeout, refer to **Section 3.4**.

Table 4-1 NAD Address Range

NAD Value	Description
00_H to FE_H	Valid slave address range for individual slave addressing
FF_H	Broadcast address for concurrent slave addressing Default address if NAD value is not programmed

4.1.4 BSL Message Parsing & Responses

The BSL protocol provides single message commands and multimessage commands. A message state machine is implemented, which first collects all command-related messages before executing the command. It periodically polls the underlying interface protocol layer (e.g. FastLIN protocol layer) to collect all frames belonging to a BSL command.

Command Message State Machine

Figure 4-3 gives an overview of the BSL command message state machine.

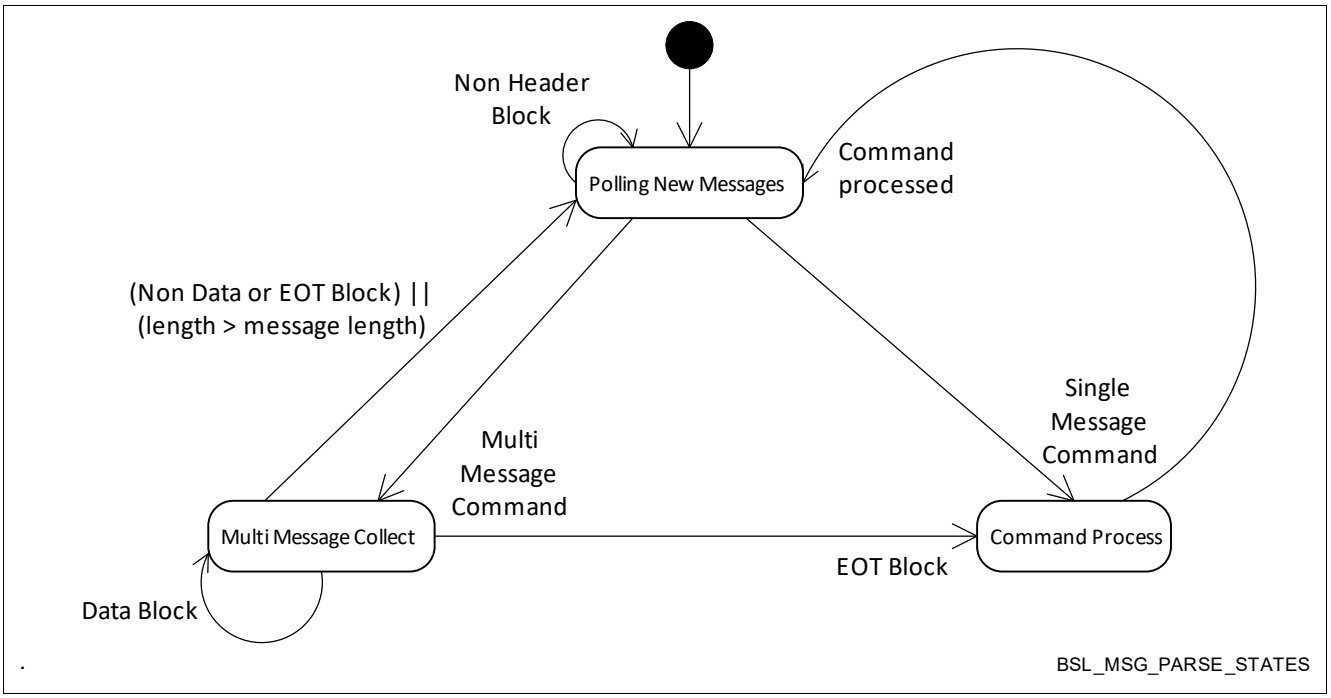


Figure 4-3 BSL Command Message State Machine

The state machine starts to wait for the header block. This could be either a command which consists of a single header block (the message type MSB bit is set) or a command that consists of multiple messages (the message type MSB bit is zero).

For multimessage commands, the first message is a header block. The second message data is always followed by an EOT block message.

The EOT block message reception initiates command parsing and execution.

The command processing includes message validation, where the message parameters are checked for boundaries, any hardware applied protection and if this message is supported for this boot mode.

The state machine aborts the multimessage collection if the overall data bytes of all collected messages have exceeded the maximum message data buffer length of 137 bytes (7 bytes in the header block message + 130 EOT data bytes).

For single message commands, all command-related information is already available in the header block message. The command parsing and execution start right after receiving the message.

After command execution and after a response has been sent, the state machine returns to the header block polling state in order to wait for the next command.

Any received message which does not fit the current state or state transmission leads to an exit from the current state and restarting of the whole state machine.

Response Message State Machine

The command response is specific to the used serial interface. . Further details are described in the interface specific protocol layer part.

Figure 4-4 gives an overview of the FastLIN response message state machine.

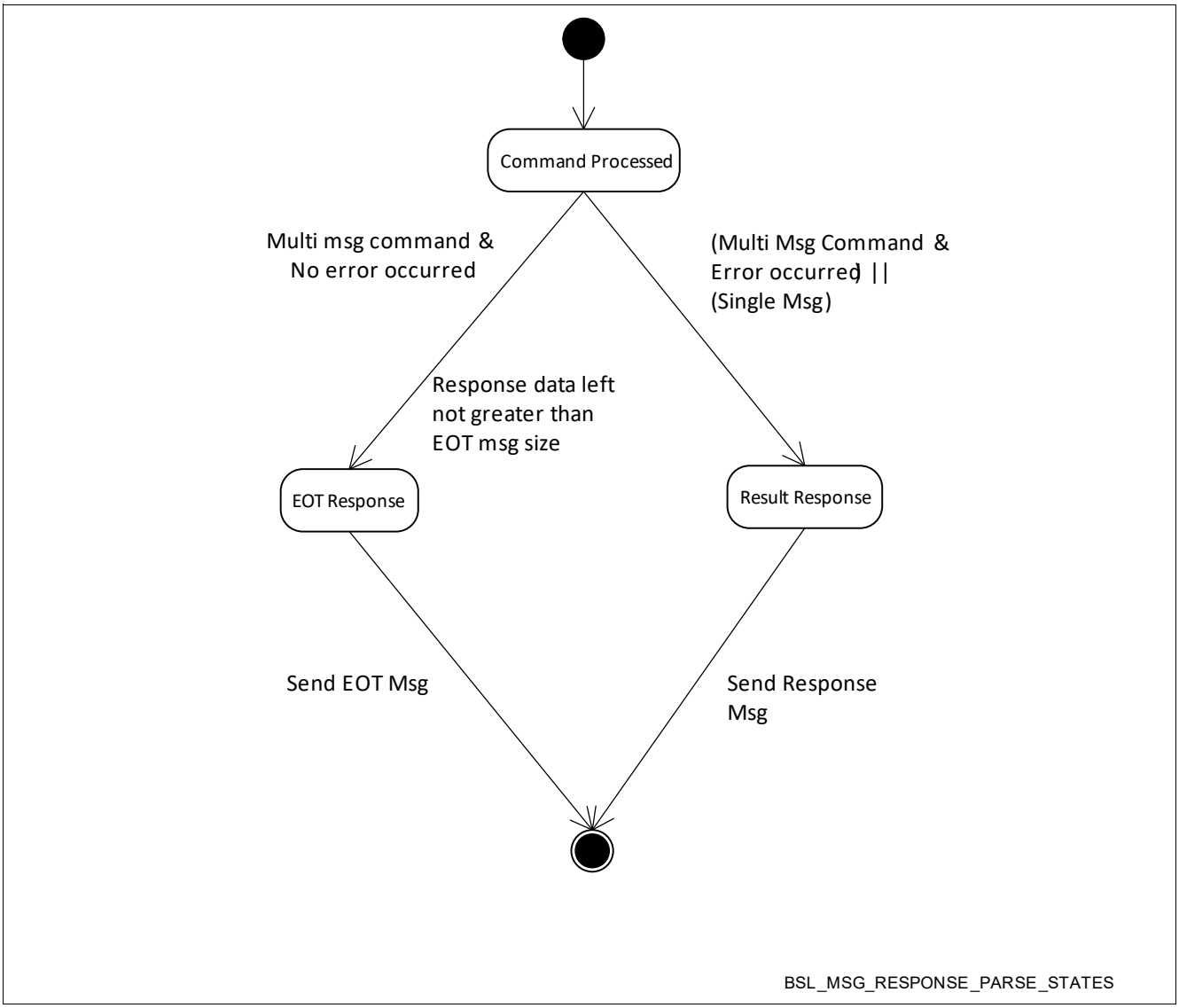


Figure 4-4 FastLIN Response Message State Machine

Some BSL messages request read-out of data from the device. These messages expect EOT block.

Other BSL messages download data or initiate code execution. They do not request reading out of any data. These messages only reply with a status response message.

A BSL command execution replies with a status response message in the event that the command execution fails.

Attention: ***The BootROM responds to each incoming command. The response is either the requested data or the response block (e.g. success or error code). Only the code execution command does not reply with a response message.***

4.1.5 Command Execution

The command data is checked and validated after all the message data is received. This includes that the message parameters are checked for boundaries, any hardware-applied protection (e.g. NVM protection) and if this message is supported for this boot mode.

The following command classes are supported:

- **RAM access** – RAM accesses are directly done by the BSL protocol without the use of any other submodule.
- **NVM access** – NVM write accesses are performed using the NVM API, NVM read access can be performed directly.
- **100TP access** – 100TP accesses are performed using the NVM CS (Configuration Sector) API.
-

4.1.6 Timing Constraints

The host needs to add a delay between all sent BSL command header and EOT messages.

The BootROM also requires an additional waiting time to process the full received BSL command. The BootROM is not able to provide the response messages or able to receive new commands before this period expires. The host must wait this length of time before sending a new command.

To give the BootROM time to process each byte in a CMD or EOT frame, the byte and frame timing must comply to the values shown in **Table 4-2**.

Table 4-2 BSL Byte and Frame Timing Limits and Highest Transfer Rate

Delay type	FastLIN (µs, min.)	
Between bytes	3.7	
Between the end of a CMD frame to the start of an EOT frame	20	
Interframe	User specified in NVM CS	
Host waiting time after response is received until a new frame can be sent	20	

* There are certain BSL commands that need longer processing time. These involve NVM write/erase operations. The host waiting time is longer before a command response can be requested or before a result is sent back. Changing a value in an already programmed NVM page, which happens if a setting is changed, requires the following NVM steps:

- Read the full page into the HW assembly buffer
- Update the HW buffer with new data
- Program the page from the HW assembly buffer
- Erase the old page

Total time: 8 ms

The processing time must always be taken into account.

4.1.7 BSL Interframe Timeout Behavior

To keep track of BSL frame transmission violations, interframe timeout is used (described also in **Chapter 4.1.1**). This chapter summarizes the different use-case scenarios where BSL frame timeouts are applied.

BSL frame transmission timeout is handled differently and depends on:

- BSL has not received any valid host synchronization yet. In this case NAC timeout value is used for all timeout calculations. If timeout is reached this means NAC timer expired.
- BSL has completed host synchronization. All timeouts are based on the interframe timeout value. This means wait forever for frame start and once frame reception has started, time measurement against interframe timeout is performed.

4.1.8 BSL Host Synchronization

Valid host synchronization: For FastLIN the full passphrase has been received before NAC expires.

4.2 BSL via FastLIN

4.2.1 FastLIN Protocol

FastLIN is a LIN enhancement supporting higher baudrate of up to 230.4 kBd. FastLIN is especially useful during back-end programming, where fast programming cycles are desirable.

The FastLIN BSL supports baudrate of 38.4 kBd, 115.2 kBd and 230.4 kBd via the internal LIN Tranceiver HW module.

The FastLIN protocol uses UART data link layer to parse all incoming FastLIN frames, it rejects frames with a failing checksum calculation result. All received frames are passed on to the BSL transport layer, which concatenates them to complete commands for BSL application layer. The checksum is always the last field of FastLIN command- and response frames.

For user mode, the default FastLIN baudrate is 115.2 kBd. The actual session baudrate can be changed with the BSL command **"Command 93_H – FastLIN: Set Session Baudrate" on Page 59**.

Figure 4-5 shows the FastLIN protocol architecture.

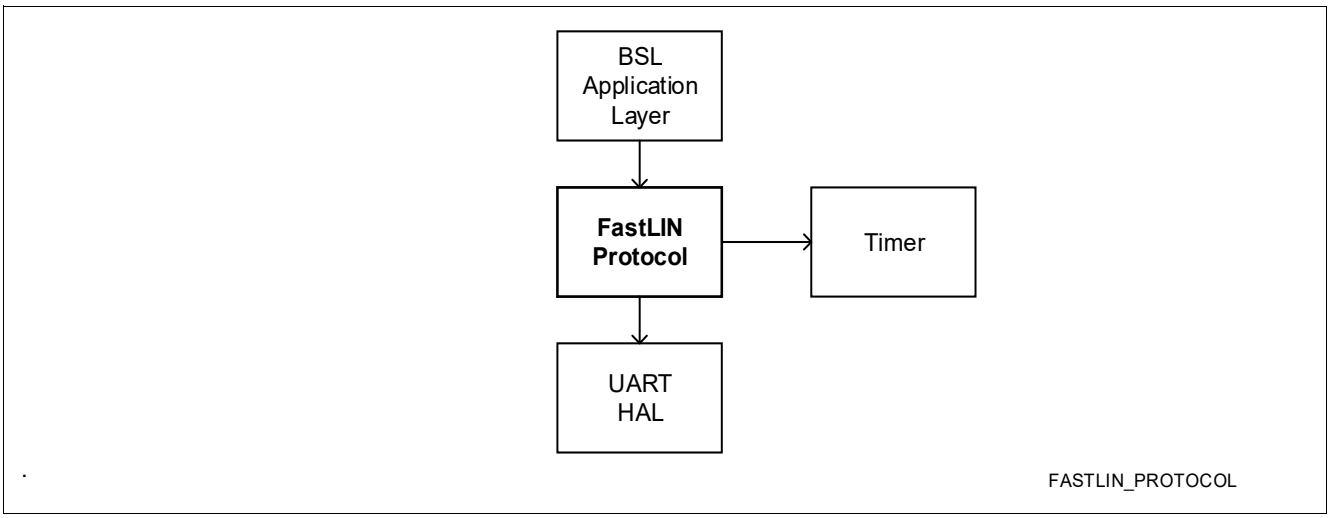


Figure 4-5 FastLIN Protocol Architecture

Figure 4-6 shows the interaction between Hardware and software layers of FastLIN BSL.

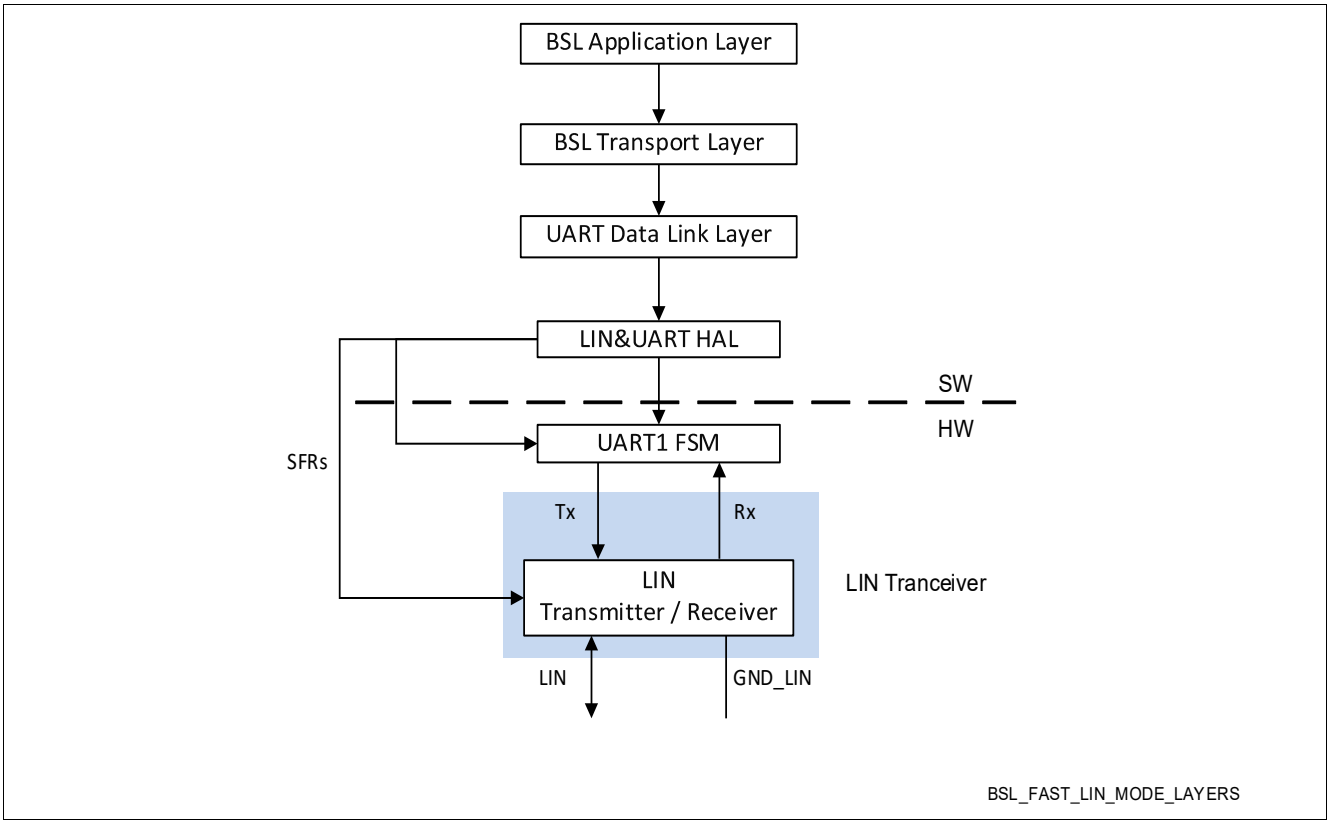


Figure 4-6 FastLIN BSL Software Hardware Layers

4.2.2 FastLIN

After successful synchronization with the host, the FastLIN communication with the host starts. This section describes the command frame format that is used for the host communication.

The communication between the host and the FastLIN data link layer is performed by a simple transfer protocol. The information is sent from the host to the bootROM in blocks. All the blocks follow the specified block structure.

4.2.2.1 Command Frame Format

This section describes command frames that are sent by the host to initiate a command. This frame format encapsulates the BSL commond messages, which are described in **Section 4.3**.

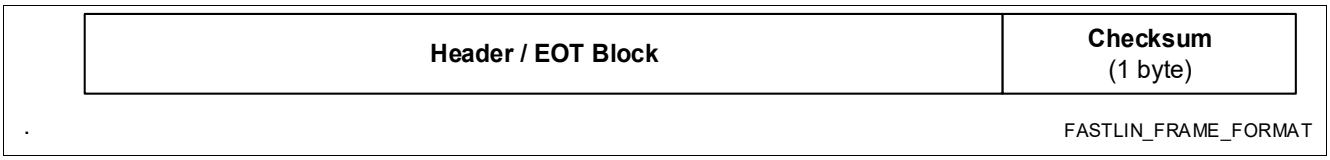


Figure 4-7 FastLIN Command Frame Format

Figure 4-7 shows the FastLIN frame format of frames sent by the host to the BootROM. It contains a BSL header block or EOT block and an additional checksum byte.

The length of the header / EOT block depends on the BSL messages, which are described in **Section 4.3**.

Each frame is terminated with the checksum byte. This checksum calculation includes all data bytes of the header / EOT block. Details are given in **“Checksum” on Page 28**.

4.2.2.2 Response Frame Format

This section describes response frames that the BootROM sends in reply to a command request. It contains a BSL EOT block or response block and an additional checksum byte. This frame format encapsulates the BSL response messages, which are described in **Section 4.3**.

The length of the EOT / response block depends on the information sent. Some commands request some data, which are sent by EOT blocks. Other commands do not request data and just the command execution result is reported. **Figure 4-8** shows the FastLIN response frame format:

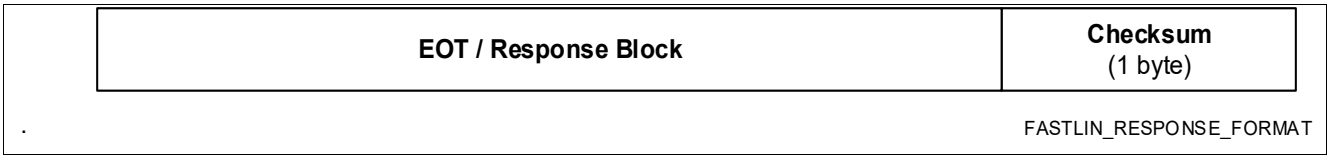


Figure 4-8 FastLIN Response Frame Format

Each frame is terminated with the checksum byte. This checksum calculation includes all data bytes of the BSL EOT / response block. Details are given in **"Checksum" on Page 28**.

4.2.2.3 Checksum

The checksum contains the inverted eight-bit sum with a carry over all data bytes.

Checksum calculation over the data bytes only is referred to as a classic checksum. An eight-bit sum with carry is equivalent to the sum of all values, subtracted by 255 every time the sum is greater than or equal to 256.

The checksum is the last field of Command and Response FastLIN frames.

4.3 BSL commands - Protocol (Version 2.0)

This section describes the boot strap loader messages that are used by the FastLINprotocol. The physical layer encapsulation of these messages is described in **“BSL via FastLIN” on Page 26**.

All commands support acknowledge response message, which contain an error code with the result of the executed command. Some messages return a response message with the result of the executed command and some messages also return requested data. For messages that return data, the data should be treated as a response with no errors and the acknowledge response message will in this case not be sent. The messages will either return an error code of a detected error or return the requested data. Data response messages are described together with the command messages. The response and error code messages are described in **“Acknowledge Response Message (81_H)” on Page 66**.

Some commands do not intend to send any response message. For instance, the code execution command messages directly jump to the requested code location. These messages will only send a response message if the requested command could not be executed.

Each incoming message is verified. Inconsistent frames (e.g. invalid checksum or length) are silently rejected. Unknown messages and message types are also rejected, with response message ERR_LOG_CODE_MSG_VALIDITY_FAIL.

Whether the host waiting time before response is sent back or whether a response can be asked for is defined for each of the messages if it deviates from the definition given in **Section 4.1.6**.

The following BSL commands are supported:

- **Command 02_H – RAM: Write Data/Program**
- **Command 83_H – RAM: Execute**
- **Command 84_H – RAM: Read Data**
- **Command 05_H – NVM: Write Data/Program**
- **Command 86_H – NVM: Execute**
- **Command 87_H – NVM: Read Data**
- **Command 88_H – NVM: Erase**
- **Command 89_H – NVM: Protection Password Set**
- **Command $8A_H$ – NVM: Switch Keys Set**
- **Command $8B_H$ – NVM: Page Checksum Check**
- **Command $0C_H$ – NVM: NVM Checksum Calculation**
- **Command $0D_H$ – NVM: 100TP Write**
- **Command $8E_H$ – NVM: 100TP Read**
- **Command $8F_H$ – BSL: NAC Set**
- **Command 90_H – BSL: NAC Get**
- **Command 91_H – FastLIN: NAD Set**
- **Command 92_H – FastLIN: NAD Get**
- **Command 93_H – FastLIN: Set Session Baudrate**
- **Command 97_H – NVM 100TP Erase**
- **Command 98_H – NVM: Reflash Prepare**
- **Command 99_H – NVM: Set CBSL Size**

Dummy Bytes

Depending on the BSL frame data fill level, some frame data bytes are not used. Those bytes are filled with dummy bytes, which are set to zero. The BootROM ignores dummy bytes, independent of their values.

Padding Bytes

If the customer adds padding bytes, although this is not regular it is still supported by the firmware. Padding bytes up to a data field size of 128 bytes are possible. The firmware will accept the real data and will find the checksum byte after the last padding byte.

RAM Access Limitation

Access to the RAM is limited for the BSL commands **Command 84_H – RAM: Read Data** and **Command 02_H – RAM: Write Data/Program**. In all boot modes, the full RAM range can be read but global variables/data and stack area cannot be written to. Trying to write to these areas will result in an error.

RAM and NVM Address Range Check

All BSL commands reading or writing the NVM or RAM check the address range and report an error if the memory region is exceeded. The number of bytes to be read or written must be greater than zero.

BSL Commands Protection Group

With any command, the BSL module checks NVM protection to determine if the command can be executed. Details are given specifically with each BSL command description. Basically, all the BSL commands downloading into NVM/RAM/NVM CS are blocked if any NVM region read protection is set or in case the write protection is set on the target region. The BSL commands reading from NVM or 100TP are blocked if the region addressed is read protected. An error is returned upon any access violation. **Table 0-1** states which NVM protection group is checked before a given BSL command is executed.

Definitions of NVM protection groups:

- **Group 1:** These commands are always allowed to execute, regardless of protection.
- **Group 2**: These commands are only blocked if the region addressed is read protected.
- **Group 3**: These commands are blocked if read protection on any region is set.
- **Group 4**:These commands are blocked if read protection on any region is set or the region addressed is write protected.

Table 4-3 NVM Protection Check for BSL Commands

NVM prot. group Group name	**BSL Command**
Group 1 Protection Ignored	**Command 89_H – NVM: Protection Password Set** **Command 98_H – NVM: Reflash Prepare** **Command 83_H – RAM: Execute** **Command 84_H – RAM: Read Data** **Command $8B_H$ – NVM: Page Checksum Check** **Command $0C_H$ – NVM: NVM Checksum Calculation** **Command 93_H – FastLIN: Set Session Baudrate** **Command 90_H – BSL: NAC Get** **Command 92_H – FastLIN: NAD Get**
Group 2 Read Protection	**Command 87_H – NVM: Read Data** **Command $8E_H$ – NVM: 100TP Read**
Group 3 IP Protection	**Command 86_H – NVM: Execute** **Command 02_H – RAM: Write Data/Program**
Group 4 IP- and Write Protection	**Command 05_H – NVM: Write Data/Program** **Command 88_H – NVM: Erase** **Command $0D_H$ – NVM: 100TP Write** **Command 97_H – NVM 100TP Erase** **Command $8F_H$ – BSL: NAC Set** **Command 91_H – FastLIN: NAD Set** **Command $8A_H$ – NVM: Switch Keys Set** 1)

1) Command assigned to IP- and Write protection group for ROM code size optimization although it shall be blocked by any write/read protection.

4.3.1 Command 02_H – RAM: Write Data/Program

Firmware supports downloading of data and code to the device's internal RAM via command 02_H.

The host initiates the RAM download by sending a header block message. This message contains information about the RAM location (offset address based on RAM start address). With FastLIN, data bytes are sent within the EOT block message.

This command does not support to write RAM locations which BootROM uses for global variable and stack storage. This command rejects the write operation if the offset is out of range, or offset plus count is out of range. .

Details about the NVM access protection are given in **"Command 89_H – NVM: Protection Password Set" on Page 44**. It returns an error code in the response message.

This message supports downloading of a maximum of 128 bytes into the RAM. Larger memory blocks need to be split into multiple **Command 02_H – RAM: Write Data/Program** messages.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

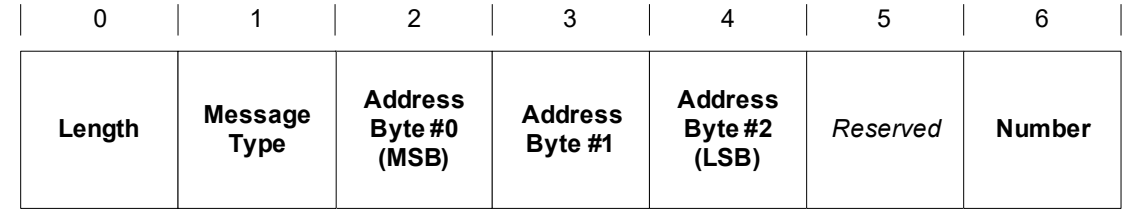


Table 4-4 "Command 02_H – RAM: Write Data/Program" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	RAM write command. Always set to 02_H
Address Byte #0 (MSB)	24-bit RAM address offset where to store the download data. The offset starts counting from the RAM start address 1800.0000_H
Address Byte #1	
Address Byte #2 (LSB)	
Number	8-bit number of data bytes to write with the whole message. The BootROM expects to receive these bytes in the EOT block. Maximum supported length: 128 bytes.

EOT Block

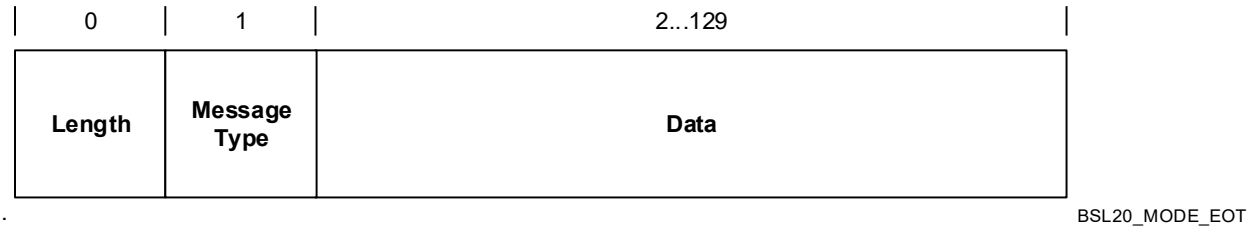


Table 4-5 "Command 02_H – RAM: Write Data/Program" EOT Block Field Description

Field	Description
Length	Number of bytes to follow (Message Type- and Data field)
Message Type	EOT block. Always set to 80_H
Data	Data to be written, minimum size 1 byte, maximum size 128 bytes

Returned error codes

The message can return the following error codes:

- ERR_LOG_SUCCESS
- ERR_LOG_CODE_NVM_IS_READ_PROTECTED
- ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID
- ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID

4.3.2 Command 83_H – RAM: Execute

Firmware triggers execution of a RAM user program by the Host via command 83_H. This code can be previously downloaded by the BSL **Command 02_H – RAM: Write Data/Program**.

The host initiates the RAM code execution by sending the header block message. This messages contains the RAM address offset (offset address based on RAM start address) where to jump for code execution.

This command does not check if any valid code is placed in RAM before jumping to the given code location.

Before jumping to RAM the following steps are done:

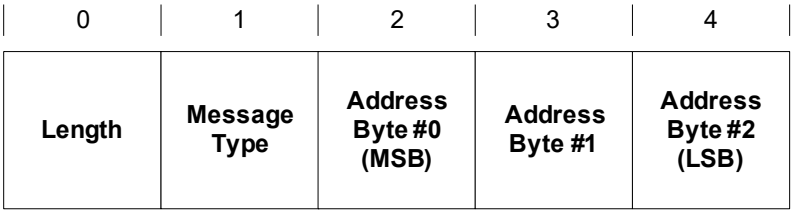
- The BootROM configures the stack pointer to 1800.0400_H. It is recommended that the RAM code adapts the stack pointer on demand.
- The system clock is switched to PLL at the device default or user defined frequency from NVM CS settings.
- All interrupts are cleared.
- reenable watchdog

It is not allowed for the RAM code to make a return call. ARM LR register has been set to zero when jumping to RAM. If BSL should be re-entered a system reset must be performed.

This command does not send any response back to the host, unless an error is detected. It performs the RAM code execution right after receiving the header block.

For potential command execution constraints see **“NVM Protection Check for BSL Commands” on Page 31**

Header Block



BSL20_MODE01_HEADER

Table 4-6 “Command 83_H – RAM: Execute” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 04_H
Message Type	RAM execute command. Always set to 83_H
Address Byte #0(MSB)	24-bit RAM address offset where to jump for code execution. The offset starts counting from the RAM start address 1800.0000_H Maximum supported offset: RAM size -4
Address Byte #1	
Address Byte #2(LSB)	

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID
- ERR_LOG_CODE_NVM_RAM_EXEC_PARAMS_INVALID

4.3.3 Command 84_H – RAM: Read Data

Firmware supports reading of data from the device's internal RAM via command 84_H.

The host initiates the RAM read by sending a header block message. This message contains information about the RAM location (offset address based on RAM start address) and the number of bytes read.

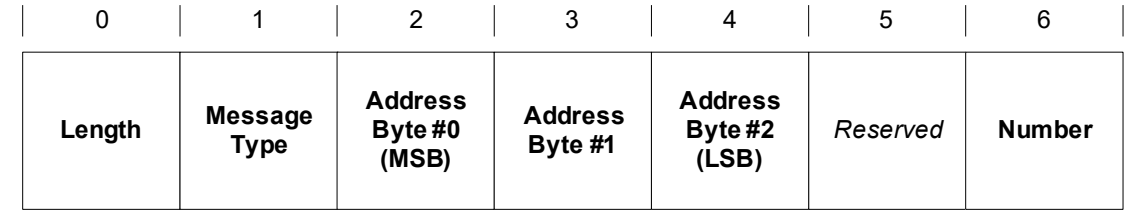
BootROM sends back the requested data within the terminating EOT block message.

This command rejects the read operation if the offset is out of range, or offset plus count is out of range. It returns an error code in the response message.

This message supports reading of a maximum of 128 bytes from the RAM. Larger memory blocks need to be split into multiple **Command 84_H – RAM: Read Data** messages.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

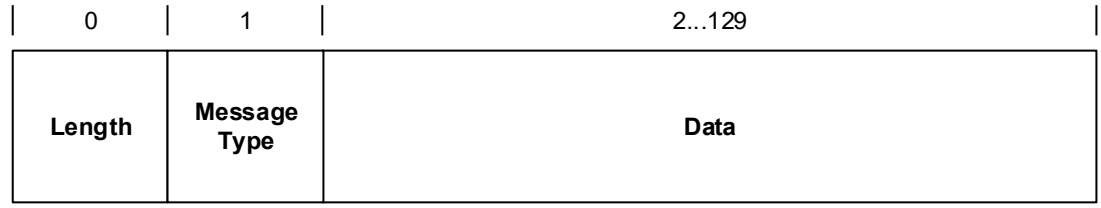


BSL20_MODE02_HEADER

Table 4-7 "Command 84_H – RAM: Read Data" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	RAM read data command. Always set to 84_H
Address Byte #0(MSB)	24-bit RAM address offset where to read the data. The offset starts counting from the RAM start address 1800.0000_H
Address Byte #1	
Address Byte #2(LSB)	
Number	8-bit number of data bytes to read. The BootROM will send these bytes in the EOT block. Maximum supported length: 128 bytes.

EOT Block



BSL20_MODE_EOT

Table 4-8 “Command 84_H – RAM: Read” Data EOT Block Field Description

Field	Description
Length	Number of bytes to follow (Message Type- and Data field)
Message Type	EOT block. Always set to 80_H
Data	Data to be written, minimum size 1 byte, maximum size 128 bytes

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID
- ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID

4.3.4 Command 05_H – NVM: Write Data/Program

Firmware supports programming of data and code to the device's internal NVM via command 05_H.

The host initiates NVM programming by sending a header block message. This message contains information about the NVM location (offset address based on NVM start address). The data bytes are sent within the EOT block message.

This command does not support NVM cross page boundary programming. It rejects the page write operation if the offset is out of range, or offset plus count is out of range. It returns an error code in the response message. No bytes are programmed if the data does not fit the page size. NVM write supports partial non-page-aligned programming, preserving the page data not passed as an input.

NVM write supports downloading flexible number of bytes, maximum of 128 bytes, into the NVM. Larger memory blocks need to be split into multiple **Command 05_H – NVM: Write Data/Program**.

The host waiting time before a response is sent back is 8 ms.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

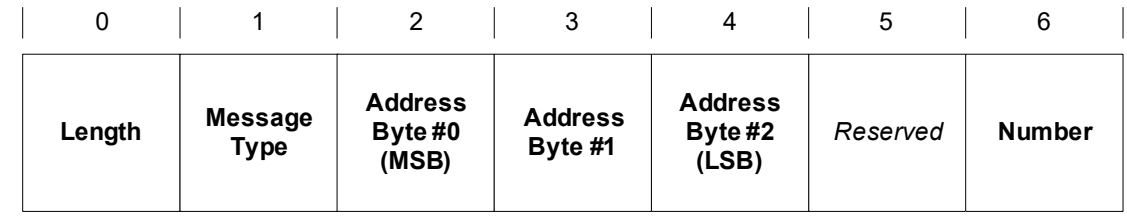


Table 4-9 "Command 05_H – NVM: Write Data/Program" Header Block Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM write data/program command. Always set to 05_H
Address Byte #0(MSB)	24-bit NVM address offset where to store the download data. The offset starts counting from the NVM start address 1100.0000_H
Address Byte #1	
Address Byte #2(LSB)	
Number	8-bit number of data bytes to write with the whole message. The BootROM expects to receive these bytes in the EOT block. Maximum supported length: 128 bytes.

EOT Block

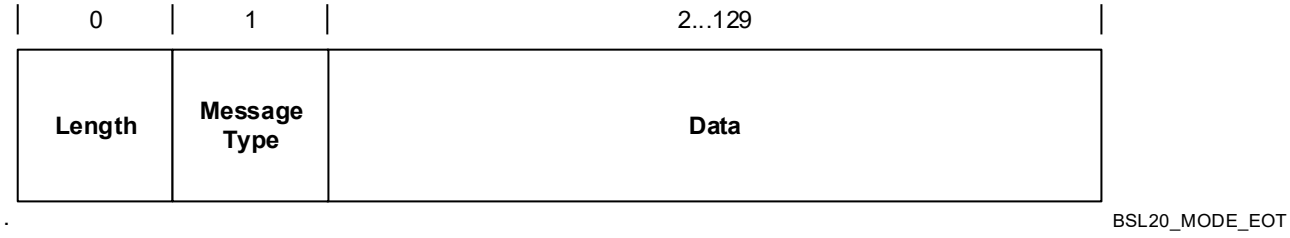


Table 4-10 “Command 05_H – NVM: Write Data/Program” EOT Block Field Description

Field	Description
Length	Number of bytes to follow (Message Type- and Data field)
Message Type	EOT block. Always set to 80_H
Data	Data to be written, minimum size 1 byte, maximum size 128 bytes

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_NVM_IS_READ_PROTECTED

The remaining error codes returned are the same as for the corresponding user API call,

see **“user_nvm_write” on Page 100**

4.3.5 Command 86_H – NVM: Execute

Firmware triggers execution of a NVM user program by the Host via command 86_H. This code could be previously downloaded by the BSL **Command 05_H – NVM: Write Data/Program**.

The host initiates the NVM code execution by sending the header block message. This messages contains the NVM address offset (offset address based on NVM start address) where to jump for code execution.

This command does not check if any valid code is placed in NVM before jumping to the given code location.

Before jumping to NVM the following steps are done:

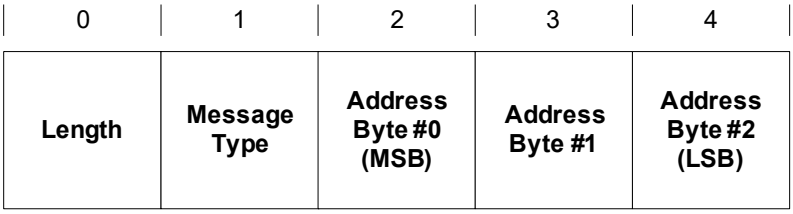
- The BootROM configures the stack pointer to 1800.0400_H. It is recommended that the NVMcode adapts the stack pointer on demand.
- The system clock is switched to PLL at the device default or user defined frequency from NVM CS settings.
- All interrupts are cleared.
- re-enable watchdog

It is not allowed for the NVM code to make a return call. ARM LR register has been set to zero when jumping to NVM. If BSL should be re-entered a system reset must be performed.

This command does not send any response back to the host, unless an error is detected. It performs the NVM code execution right after receiving the header block.

For potential command execution constraints see **“NVM Protection Check for BSL Commands” on Page 31**

Header Block



BSL20_MODE04_HEADER

Table 4-11 “Command 86_H – NVM: Execute” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 04_H
Message Type	NVM execute command. Always set to 86_H
Address Byte #0 (MSB)	24-bit NVM address offset where to jump for code execution. The offset starts counting from the NVM start address 1100.0000_H Maximum supported offset: NVMSIZE-4
Address Byte #1	
Address Byte #2 (LSB)	

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_NVM_IS_READ_PROTECTED
- ERR_LOG_CODE_NVM_ADDR_RANGE_INVALID
- ERR_LOG_CODE_NVM_RAM_EXEC_PARAMS_INVALID

4.3.6 Command 87_H – NVM: Read Data

Firmware supports reading of data and code from the device's internal NVM via command 87_H.

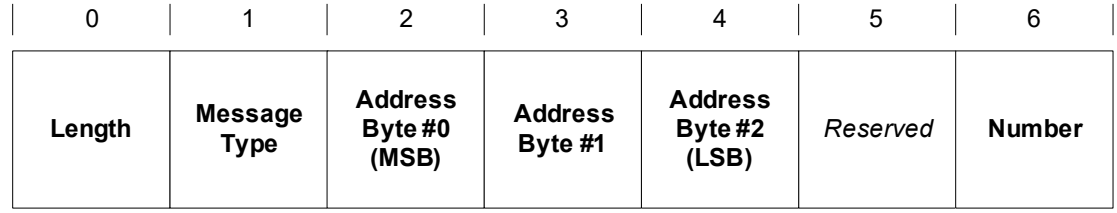
The host initiates the NVM read by sending a header block message. This message contains information about the NVM location (offset address based on NVM start address) and the number of data bytes to read. The BootROM sends back the requested data within the terminating EOT block message. If reading from an address which belongs to the non-linear NVM region and the page is not mapped (previously programmed), the read is rejected. The command protects data read out with ECC2 check.

This command does not support NVM cross page boundary read. It rejects the read operation if the offset is out of range, or offset plus count is out of range. It returns an error code in the response message.

This message supports reading of a maximum of 128 bytes from the NVM. Larger memory blocks need to be split into multiple Command 87H – NVM: Read Data messages.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

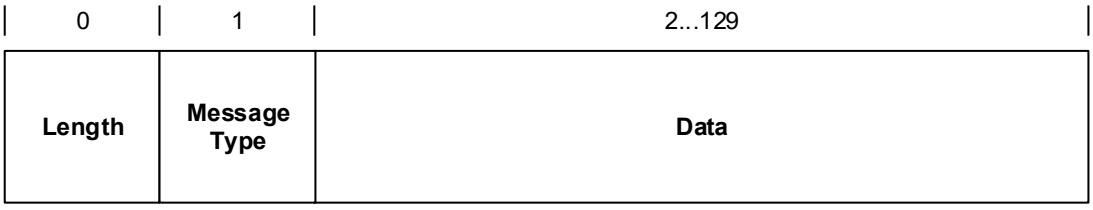


BSL20_MODE05_HEADER

Table 4-12 "Command 87_H – NVM: Read Data" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM read data command. Always set to 87_H
Address Byte #0(MSB)	24-bit NVM address offset where to read the data. The offset starts counting from the NVM start address 1100.0000_H
Address Byte #1	
Address Byte #2(LSB)	
Number	8-bit number of data bytes to read. The BootROM will send these bytes in the EOT block. Maximum supported length: 128 bytes.

EOT block



BSL20_MODE_EOT

Table 4-13 “Command 87_H – NVM: Read Data” EOT Block Field Description

Field	Description
Length	Number of bytes to follow (Message Type- and Data field)
Message Type	EOT block. Always set to 80_H
Data	Data to be read, minimum size 1 byte, maximum size 128 bytes

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_NVM_SEGMENT_READ_PROTECTED
- ERR_LOG_CODE_NVM_ADDR_RANGE_INVALID
- ERR_LOG_CODE_ADDRESS_RANGE_CROSSING_PAGE_BOUNDARY
- ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID
- ERR_LOG_CODE_NVM_PAGE_NOT_MAPPED
- ERR_LOG_CODE_ECC2READ_ERROR

4.3.7 Command 88_H – NVM: Erase

Firmware supports the erasure of NVM pages, NVM sectors and the overall NVM (mass erase) via command 88_H.

The host initiates the NVM erase operation by sending a header block message. This message contains information about the NVM location (an offset address based on the NVM start address) and selects the erase granularity.

The command does not erase any NVM CS (Configuration Sector) pages.

The host waiting time required before a response is sent back:

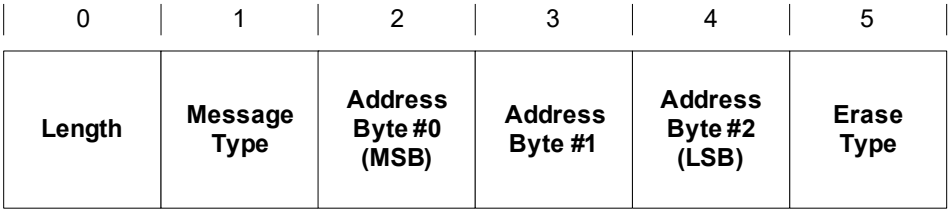
- for page erase (128 bytes) / sector erase (4 KB): 5 ms
- for mass erase of all sectors (except NVM CS): 20 ms

Note: For command execution constraints see **"NVM Protection Check for BSL Commands" on Page HIDDEN**

Note: The mass erase operation is rejected if any NVM region is write protected

Attention: The mass erase operation erases multiple sectors in parallel. However, its effectiveness is affected by aging of memory cells. It is only allowed for engineering purposes or till end of line programming. It is not allowed to use it in the field.

Header Block



BSL20_MODE06_HEADER

Table 4-14 "Command 88_H – NVM: Erase" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 05_H
Message Type	NVM erase command. Always set to 88_H
Address Byte #0(MSB)	24-bit NVM address offset for page, sector or mass erase selection. The offset is based on the NVM start address (1100.0000_H)
Address Byte #1	
Address Byte #2(LSB)	
Erase Type	Supported erase type field values: • 0 - NVM page erase • 1 - NVM sector erase • 2 - NVM mass erase

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED

- ERR_LOG_CODE_ACCESS_AB_MODE_ERROR
- ERR_LOG_CODE_BSL_RECV_BYTES_MISMATCH
- ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED
- ERR_LOG_CODE_NVM_ERASE_PARAMS_INVALID
- ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID
- ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID
- ERR_LOG_CODE_ADDRESS_RANGE_CROSSING_PAGE_BOUNDARY

The remaining error codes returned are the same as for the corresponding user API calls, see:

- **“user_nvm_page_erase” on Page 97**
- **“user_nvm_sector_erase” on Page 99**

4.3.8 Command 89_H – NVM: Protection Password Set

The command supports setting individual NVM region protection password. The protection becomes active at the following reset. The regions protected are the customer bootloader NVM region, code region, data linear region and data mapped region.

NVM region protection includes access protection for read and/or write/erase.

A valid password must not be equal to all ‘1’ or all ‘0’, the password is checked during startup. Only if the password is valid, the given protection gets applied to the HW. This command only updates the specified NVM CS region password and does not apply it to the HW. This is done at the next device reset.

Direct password clearing or modification is not supported. It requires a preparatory step (see **Command 98_H – NVM: Reflash Prepare**) that will erase the complete NVM before erasing all NVM region passwords.

The host waiting time before a response is sent back is 8 ms.

For potential command execution constraints see **“NVM Protection Check for BSL Commands” on Page 31**

Header Block

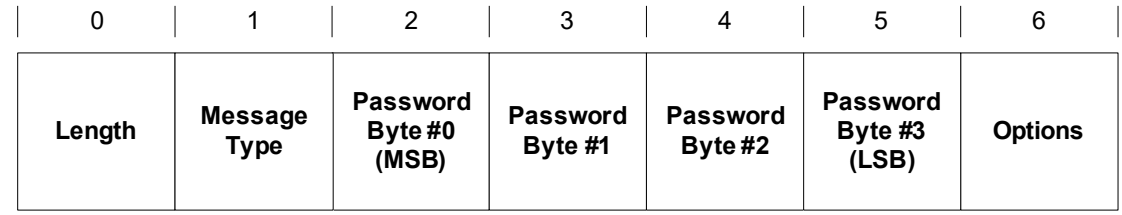


Table 4-15 “Command 89_H – NVM: Protection Set” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM protection set command. Always set to 89_H
Password Byte #0(MSB)	32-bit password parameter.
Password Byte #1	
Password Byte #2	
Password Byte #3(LSB)	
Options	The options field is described in **Table 4-16**.

Table 4-16 “Command 89_H – NVM: Protection Set” Header Block Options Field Description

Field	Bits	Description
Res	7:3	**Reserved**
Password Selector	2:1	**Password Selector** Password assignment to NVM region: 00_B **Customer Bootloader Password**, 01_B **Code Segment Password**, 10_B **Data Mapped Segment Password**, 11_B **Data Linear Segment Password**,
Res	0	**Reserved**

Returned error codes

See .“user_nvm_password_set” on Page 96

4.3.9 Command $8A_H$ – NVM: Switch Keys Set

This command allows to change the RAM test mode (see **"RAM MBIST and RAM Initialization" on Page 17**) or NVM data mode (see **"RAM Mode Key and NVM Data Mode Key" on Page 17**) or both by programming switching keys into the NVM configuration sector. It only allows to write the key value, any other value will be rejected The switching keys are checked upon device boot-up and dependent on the key settings the device configuration gets changed.

Also the command itself is protected through the key value, illegal key values will be rejected and the command will not get executed.

The Key options field is used to:

- select a new configuration setting for RAM test behavior and/or NVM configuration (Switch key ID)
- select in which of 3 possible NVM CS pages the new configuration keys are to be stored (Switch key select)

It is recommended to repeat the command using all switch key selectors available for increased device robustness, as upon device boot, all available keys will be evaluated. As long as at least one valid key is installed, the bootROM configures the device in corresponding RAM and/or NVM mode.

The host waiting time before a response is sent back is 8 ms.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

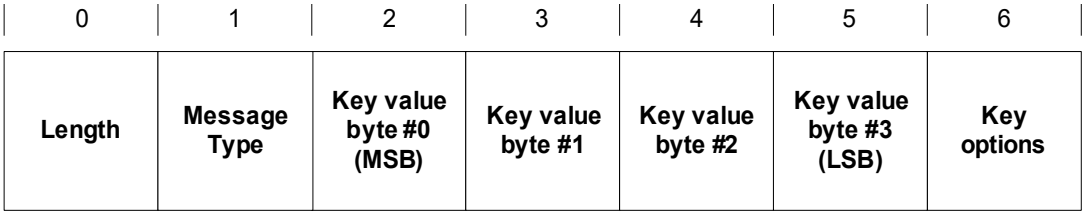


Table 4-17 "Command $8A_H$ – NVM: Switch Keys Set" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM switch keys set command. Always set to $8A_H$
Key value byte #0(MSB)	0xF1 (Byte 0 of 32-bit switch key value 0xF155E01E)
Key value byte #1	0x55 (Byte 1 of 32-bit switch key value 0xF155E01E)
Key value byte #2	0xE0 (Byte 2 of 32-bit switch key value 0xF155E01E)
Key value byte#3(LSB)	0x1E (Byte 3 of 32-bit switch key value 0xF155E01E)
Key options	The key options field is described in **Table 4-18**.

Table 4-18 "Command 89_H – NVM: Switch Keys Set" Header Block Key Options Field Description

Field	Bits	Description
Reserved	7:4	Reserved
Switch key select	3:2	**Switch key selector** Select which of the keys to write to 00_B location **SWITCH_KEY_1**, is selected 01_B location **SWITCH_KEY_2**, is selected 10_B location **SWITCH_KEY_3**, is selected 11_B **Reserved**, for future use
Switch key ID	1:0	**Switch key ID selector** 00_B **RAM_MBIST_RANGE_ID**, the selected key ID for RAM test is written to the location selected by Switch key select 01_B **NVM_DATA_LINEAR_ID**, the selected key ID for NVM data linear is written to the location selected by Switch key select 10_B **RAM_MBIST_RANGE_ID and NVM_DATA_LINEAR_ID**, the selected key ID for both RAM test and NVM data linear is written to the location selected by Switch key select 11_B **Reserved**, For future use

Returned error codes

The message can return the following error codes:

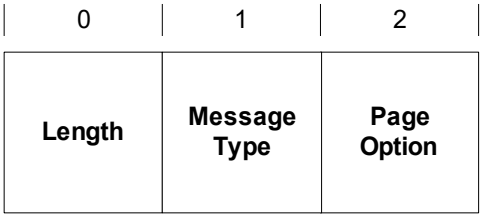
- ERR_LOG_SUCCESS
- ERR_LOG_CODE_NVM_IS_READ_PROTECTED
- ERR_LOG_CODE_USERAPI_CONFIG_SECTOR_WRITE_PROTECTED
- ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED
- ERR_LOG_CODE_SWITCH_KEY_INVALID
- ERR_LOG_CODE_USERAPI_SWITCH_KEY_VALUE_OR_USAGE_INVALID
- ERR_LOG_CODE_ACCESS_AB_MODE_ERROR

4.3.10 Command $8B_H$ – NVM: Page Checksum Check

This command reads a complete NVM page, calculates the overall checksum and compare it with the stored checksum for that page. Command returns success if checksum matches. Pages supported are all NVM 100TP pages and BSL startup page.

For potential command execution constraints see **“NVM Protection Check for BSL Commands” on Page 31**

Header Block



BSL20_MODE8B_HEADER

Table 4-19 “Command 8BH – NVM: Page Checksum Check” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 02_H
Message Type	NVM calculate and check BSL checksum command. Always set to $8B_H$
Page Option	8-bit option value with the following meaning: • 0-7: check one of the corresponding 100TP pages (page 0 to 7) • 8: check BSL page • 9-255: reserved

Returned error codes

The message can return the following error codes:

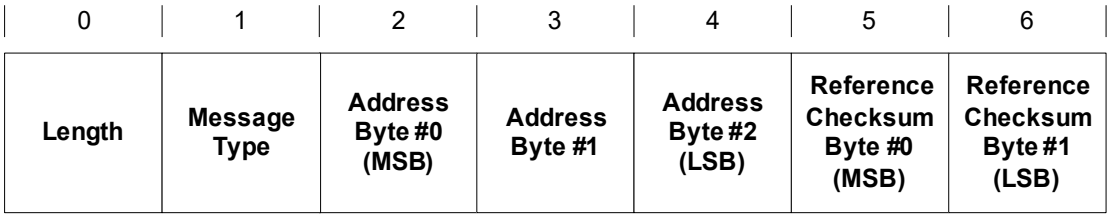
- ERR_LOG_SUCCESS
- ERR_LOG_CODE_USER_INVALID_NVM_PAGE_NUMBER
- ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED
- ERR_LOG_CODE_CS_PAGE_CHECKSUM
- ERR_LOG_CODE_CS_PAGE_ECC2READ

4.3.11 Command $0C_H$ – NVM: NVM Checksum Calculation

This command calculates a target NVM range checksum using a 16 bit XOR algorithm and compares the result with a reference checksum provided as a command parameter. The command returns with a pass indication when the calculated checksum matches the provided reference checksum, otherwise it returns with a fail indication. The command accepts offset address and number of pages to be checked as input. The command rejects and reports error if the target range exceed overall NVM linear size. Check on data mapped sector is not supported.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

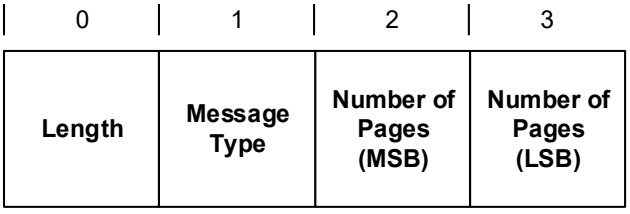


BSL20_MODE0C_HEADER

Table 4-20 "Command $0C_H$ – NVM: NVM Checksum Calculation" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM calculate and check BSL checksum command. Always set to $0C_H$
Address Byte #0 (MSB)	24-bit NVM page aligned address offset indicating where to start the checksum calculation. *Note: The lower 7 bits are ignored to force a page aligned start address*
Address Byte #1	
Address Byte #2 (LSB)	
Reference Checksum Byte #0 (MSB)	MSB of the user provided 16-bit reference checksum LSB of the user provided 16-bit reference checksum
Reference Checksum Byte #1 (LSB)	

EOT Block



BSL20_MODE0C_EOT

Table 4-21 "Command $0C_H$ – NVM: NVM Checksum Calculation" EOT Block Field Description

Field	Description
Length	Number of following bytes in the EOT block. Always set to 03_H
Message Type	EOT block. Always set to 80_H

Table 4-21 "Command $0C_H$ – NVM: NVM Checksum Calculation" EOT Block Field Description (cont'd)

Field	Description
Number of Pages (MSB)	16-bit number indicating the number of pages to be checked. The number must not exceed the number of NVM pages available in linear regions. *Note: In data mapped mode, NVM page availability ends at the end of the code region* *Note: The actual number to be entered has to be decremented by 1* *e.g. 0x0000 means 1 page (128 bytes) is checked, 0x00FF means 256 pages (32 kbyte) get checked*
Number of Pages (LSB)	

Returned error codes

The message can return the following error codes:

- ERR_LOG_SUCCESS
- ERR_LOG_CODE_BSL_NVM_CALC_CHECKSUM_MISMATCH
- ERR_LOG_CODE_NVM_ADDR_RANGE_INVALID

4.3.12 Command $0D_H$ – NVM: 100TP Write

Firmware supports programming of data in the customer-specific 100TP pages via command $0D_H$.

The header block message contains parameter about the 100TP page index, the offset inside that page. The data bytes are sent within the EOT block message.

This command rejects the page write operation if the offset is out of range, or offset plus number is out of range. It returns an error code in the response message. 100TP write supports partial non-page-aligned programming, preserving the page data not passed as an input.

Any page programming is rejected if the page specific programming limit (100 times) is exceeded. It supports downloading flexible number of bytes, maximum of 126 bytes, into the 100TP page. Two bytes stored at the end of the page are reserved for page write counter and checksum. After successful write operation, the page write counter and checksum parameter are updated .

The host waiting time before a response is sent back is 8 ms.

For potential command execution constraints see **“NVM Protection Check for BSL Commands” on Page 31**

Note: Even if only a partial page gets programmed, always a complete NVM page is required to get programmed in the background in a read-modify-write sequence. Therefore, it is still possible to get an error message ERR_LOG_CODE_NVM_VER_ERROR, as also non-modified bytes can cause verification errors.

Header Block

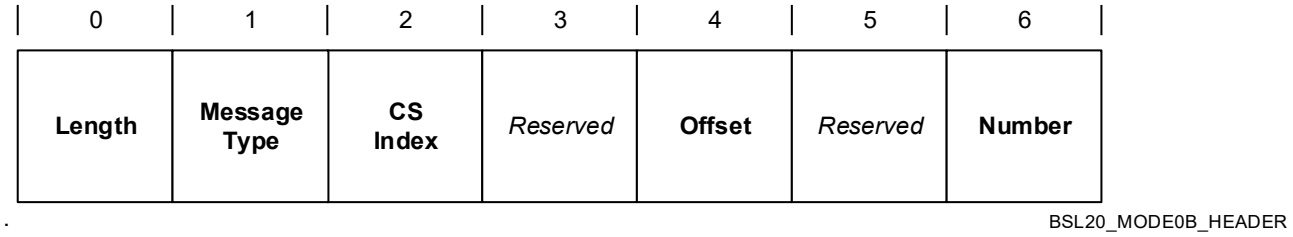
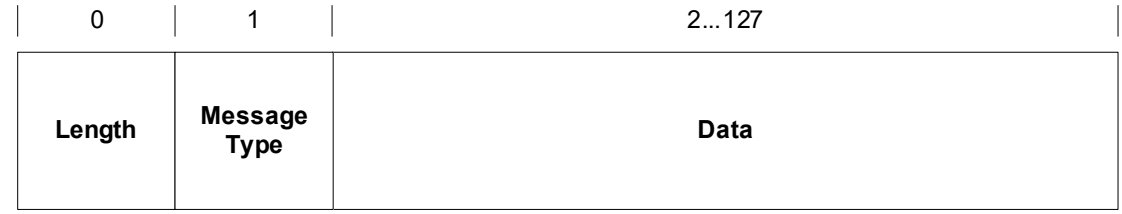


Table 4-22 “Command $0D_H$ – NVM: 100TP Write” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM 100TP write command. Always set to $0D_H$
100TP Index	100TP Selector, supported range: 0...7
Offset	Byte offset within page, valid range 0...125
Number	8-bit number of data bytes to write with the whole message. The BootROM expects to receive these bytes in the EOT block. Maximum supported length: 126 bytes.

EOT Block



BSL20_BSL_NVM_100TP_WRITE_EOT

Table 4-23 "Command $0D_H$ – NVM: 100TP Write" EOT Block Field Description

Field	Description
Length	Number of bytes to follow (Message Type- and Data field)
Message Type	EOT block. Always set to 80_H
Data	Data to be written, minimum size 1 byte, maximum size 126 bytes

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_NVM_IS_READ_PROTECTED

The remaining error codes returned are the same as for the corresponding user API call, see **"user_nvm_100tp_write" on Page 90**

4.3.13 Command $8E_H$ – NVM: 100TP Read

Firmware supports reading of data from the customer-specific 100TP page via command $8E_H$.

The header block message contains parameter about the 100TP page index, the offset inside that page and the number of data bytes to read. The BootROM sends the data bytes within the EOT block message.

This command doesn't support cross page boundary read. It rejects the read operation if the offset is out of range, or offset plus number is out of range.

The read command allows reading the internal used page programming counter. Those parameters are set during the write operation. Details can be found in **Command $0D_H$ – NVM: 100TP Write**.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

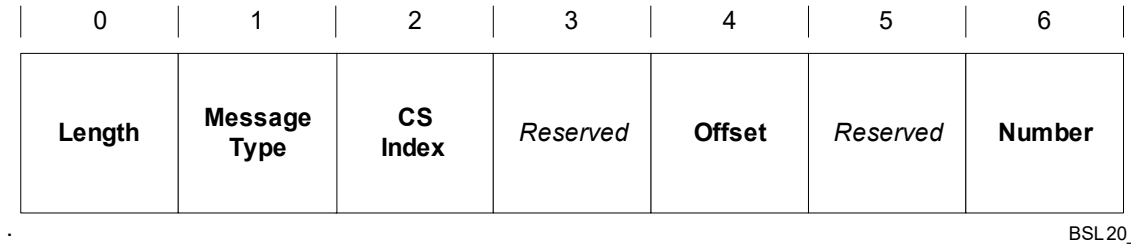


Table 4-24 "Command $8E_H$ – NVM: 100TP Read" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM 100TP read command. Always set to $8E_H$
100TP Index	100TP Selector, supported range: 0...7
Offset	Byte offset within page, valid range 0...126.
Number	Number of data bytes to read. The BootROM will send these bytes in the EOT block. Maximum supported data length: 127 bytes.

EOT Block

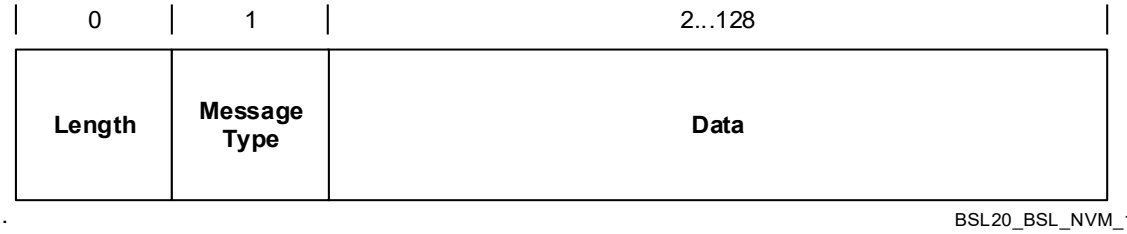


Table 4-25 "Command $8E_H$ – NVM: 100TP Read" EOT Block Field Description

Field	Description
Length	Number of bytes to follow (Message Type- and Data field)
Message Type	EOT block. Always set to 80_H
Data	Data to be read, minimum size 1 byte, maximum size 127 bytes.

Returned error codes

See **“user_nvm_100tp_read” on Page 89**

4.3.14 Command $8F_H$ – BSL: NAC Set

The Firmware supports NAC data setting with the command $8F_H$ - BSL: NAC Set.

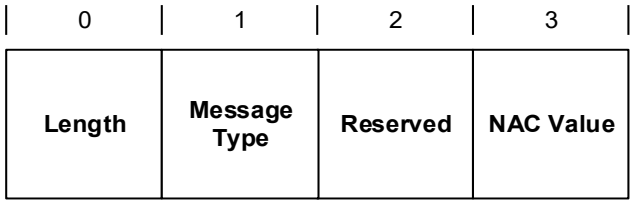
The header block message contains the NAC value, it gets stored in the device NVM CS and is used at the next startup. The command rejects operation if the given NAC value is out of valid range [0, 2-28]. See also **"NAC Definition" on Page 12**.

The BSL NAC value currently in use can be read by the **Command 90_H – BSL: NAC Get** command.

The host waiting time before a response is sent back is 8 ms.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block



BSL20_BSL_NAC_SET_HEADER

Table 4-26 "Command $8F_H$ – BSL: NAC Set" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 03_H
Message Type	BSL NAC set. Always set to $8F_H$
Reserved	Reserved
NAC Value	BSL Timeout before jumping to the User Mode Code execution. The timeout starts counting from device reset release. The possible values to set can be seen from **"NAC Definition" on Page 12**.

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_NVM_IS_READ_PROTECTED

The remaining error codes returned are the same as for the corresponding user API call, see **"user_nac_set" on Page 253**

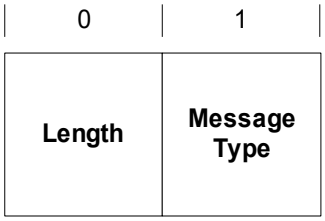
4.3.15 Command 90_H – BSL: NAC Get

Firmware supports reading the current configured BSL option data from the NVM CS, currently only the BSL timeout (NAC) value, via command 90_H.

The header block message contains the information request. The BootROM sends the information by an EOT block message. The BSL NAC value can be changed with the **Command $8F_H$ – BSL: NAC Set**.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

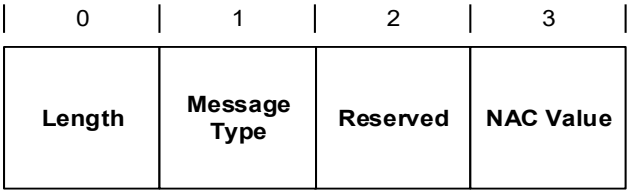


BSL20_BSL_NAC_GET_HEADER

Table 4-27 "Command 90_H – BSL: NAC Get" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 01_H
Message Type	BSL NAC set. Always set to 90_H

EOT block



BSL20_BSL_NAC_GET_EOT

Table 4-28 "Command 90_H – BSL: NAC Get" EOT Block Description

Field	Description
Length	Number of following bytes in the EOT block. Always set to 03_H
Message Type	EOT block. Always set to 80_H
Reserved	Reserved
NAC Value	BSL Timeout before jumping to the User Mode Code execution. The timeout starts counting from device reset release. The possible values to set can be seen from **"NAC Definition" on Page 12**.

Returned error codes

See **"user_nac_get" on Page 86**

4.3.16 Command 91_H – FastLIN: NAD Set

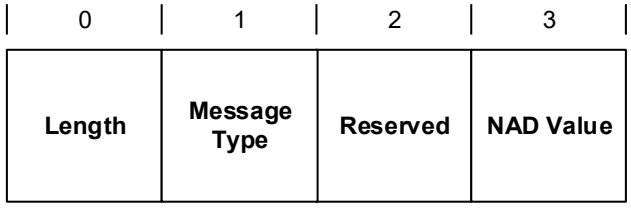
Firmware supports setting of the FastLIN NAD via command 91_H.

The header block message contains as a parameter the FastLIN NAD value. The given NAD address is stored in the device NVM CS and is used for the next startup. The current NAD value can be read by the **Command 92_H – FastLIN: NAD Get** command.

The host waiting time before a response is sent back is 8 ms.

For potential command execution constraints see **“NVM Protection Check for BSL Commands” on Page 31**

Header Block



BSL20_BSL_NAD_SET_HEADER

Table 4-29 “Command 91_H – FastLIN: NAD Set” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 03_H
Message Type	BSL NAD set. Always set to 91_H
Reserved	Reserved
NAD Value	New NAD value to be stored in the NVM CS

Returned error codes

The message can return the following error codes:

ERR_LOG_CODE_NVM_IS_READ_PROTECTED

The remaining error codes returned are the same as for the corresponding user API call, see **“user_nad_set” on Page 255**

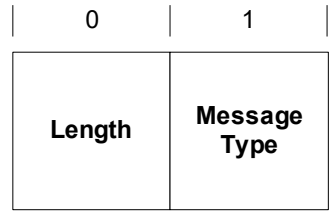
4.3.17 Command 92_H – FastLIN: NAD Get

Firmware supports reading the currently configured FastLIN NAD value via command 92_H.

The header block message contains the information request. The BootROM sends the current FastLIN NAD value within the EOT block message. The given NAD address is read from the NVM CS.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block

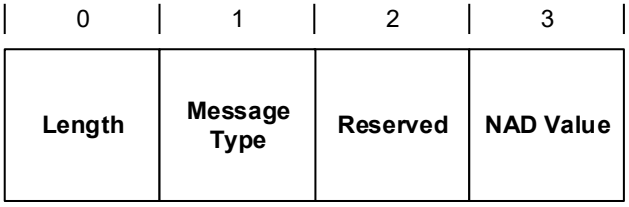


BSL20_BSL_NAD_GET_HEADER

Table 4-30 "Command 92_H – FastLIN: NAD Get" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 01_H
Message Type	BSL NAD get. Always set to 92_H

EOT Block



BSL20_BSL_NAD_GET_EOT

Table 4-31 "Command 92_H – FastLIN: NAD Get" EOT Block Field Description

Field	Description
Length	Number of following bytes in the EOT block. Always set to 03_H
Message Type	EOT block. Always set to 80_H
Reserved	Reserved
NAD value	The configured FastLIN NAD value

Returned error codes

See **"user_nad_get" on Page 87**

4.3.18 Command 93_H – FastLIN: Set Session Baudrate

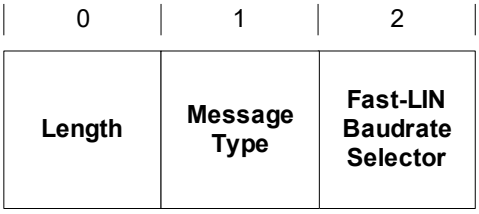
Firmware supports changing the FastLIN baudrate for the current FastLIN BSL session via command 93_H.

The header block message contains the new FastLIN baud rate selection, which will be in effect with the next BSL command. The given parameter is not stored inside the NVM CS, and the FastLIN baud rate prior to the change is still used for the response sent back to the host.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Note: When sending this command, the response to the command will use the old baudrate. The new baudrate will be set only after the response has been transmitted.

Header Block



BSL20_BSL_FAST_LIN_BAUDRATE_SET_HEADER

Table 4-32 "Command 93_H – FastLIN: Set Session Baudrate" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 02_H
Message Type	BSL Set session baud rate. Always set to 93_H
FastLIN Baudrate Selector	Baud rate to be used, starting with the next BSL command: • **0 -** 38.4 kBd • **1 -** 115.2 kBd • **2 -** 230.4 kBd

Returned error codes

The message can return the following error codes:

- ERR_LOG_SUCCESS
- ERR_LOG_CODE_NVM_IS_READ_PROTECTED
- ERR_LOG_CODE_FASTLIN_BAUDRATE_SET_FAIL

4.3.19 Command 97_H – NVM 100TP Erase

NVM 100TP Erase function provides a corrective action in case an ECC2 error is found in 100TP page. The difference to a regular NVM page erase is that the 100TP page counter is preserved. If the page contains an ECC2 error, the write counter is reconfigured to allow a maximum of 5 more operations.

The operation is rejected if the page write counter limit (100 times) is exceeded.

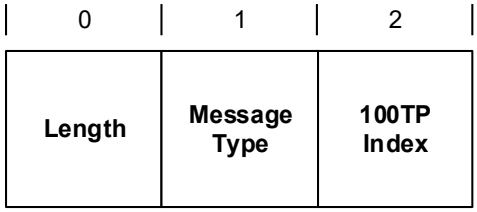
The header block message contains parameter about the 100TP page index.

After a successful erase operation, the checksum byte is invalidated to avoid copying of invalid data into customer analog trimming registers.

The host waiting time before a response is sent back is 5ms.

For potential command execution constraints see **“NVM Protection Check for BSL Commands” on Page 31**

Header Block



BSL20_MODE96_HEADER

Table 4-33 “Command 97_H – NVM: 100TP Erase” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 02_H
Message Type	NVM 100TP erase command. Always set to 97_H
100TP Index	100TP Selector, supported range: 0...7

Returned error codes

The message can return the following error codes:

- ERR_LOG_CODE_NVM_IS_READ_PROTECTED

The remaining error codes returned are the same as for the corresponding user API call, see **“user_nvm_100tp_erase” on Page 258**

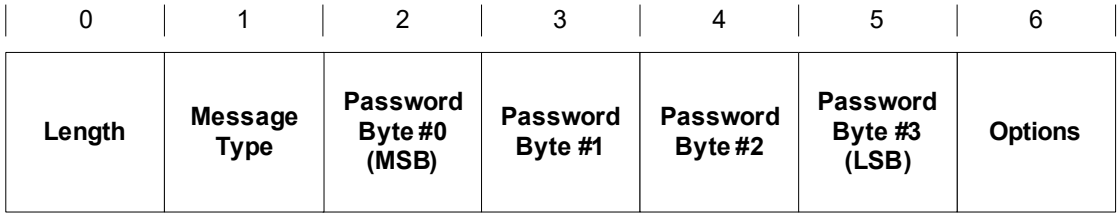
4.3.20 Command 98_H – NVM: Reflash Prepare

The command prepares device for the subsequent reflash steps. If the given password matches the installed password of the given region, it triggers erasure for all user NVM regions and region passwords.

The command is the only way to remove an already set read protection on any NVM region.

For potential command execution constraints see **"NVM Protection Check for BSL Commands" on Page 31**

Header Block



BSL20_MODE98_HEADER

Table 4-34 "Command 98H – NVM: Reflash Prepare" Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 06_H
Message Type	NVM protection set command. Always set to 98_H
Password Byte #0(MSB)	32-bit password parameter.
Password Byte #1	
Password Byte #2	
Password Byte #3(LSB)	
Options	The options field is described in **Table 4-35**.

Table 4-35 "Command 98_H – NVM: Reflash Prepare" Header Block Options Field Description

Field	Bits	Description
Res	7:3	**Reserved**
Password Selector	2:1	**Password Selector** Password assignment to NVM region: 00_B **Customer Bootloader Password**, 01_B **Code Segment Password**, 10_B **Data Mapped Segment Password**, 11_B **Data Linear Segment Password**,
Res	0	**Reserved**

Returned error codes

The message can return the following error codes:

- ERR_LOG_SUCCESS
- ERR_LOG_CODE_USER_NVM_SEGMENT_INVALID
- ERR_LOG_CODE_USER_PROTECT_PWD_INVALID

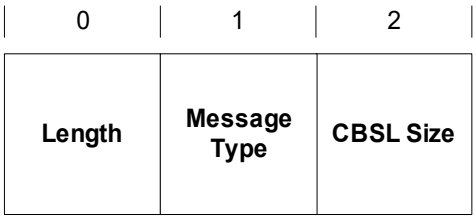
- ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED
- ERR_LOG_CODE_ACCESS_AB_MODE_ERROR
- ERR_LOG_CODE_NVM_ECC2_DATA_ERROR
- ERR_LOG_CODE_ECC2READ_ERROR
- ERR_LOG_CODE_NVM_VER_ERROR
- ERR_LOG_CODE_USER_PROTECT_PWD_EXISTS
- ERR_LOG_CODE_USER_NVM_SEGMENT_CONFIG_MISMATCH

4.3.21 Command 99_H – NVM: Set CBSL Size

With the BSL command “NVM: Set CBSL Size” it is possible to configure the CBSL region size from 0 to 16 kbyte. Setting the CBSL size to 0 supports the option to have a unique code region in case a CBSL region is not needed.

Note: this command allows for a one time configuration only, once set, the setting cannot be changed anymore!

Header Block



BSL20_MODE99_HEADER

Table 4-36 “Command 99H – NVM: Set CBSL Size” Header Block Field Description

Field	Description
Length	Number of following bytes in the header block. Always set to 02_H
Message Type	NVM Set CBSL Size command. Always set to 99_H
CBSL Size	The options field is described in **Table 4-37**.

Table 4-37 “Command 99H – NVM: Set CBSL Size” Header Block CBSL Size Field Description

Field	Description
CBSL Size	CBSL Size: 00_H **0 kB**, 01_H **4 kB**, 02_H **8 kB**, 03_H **16 kB**,

Returned error codes

The message can return the following error codes:

- ERR_LOG_SUCCESS
- ERR_LOG_CODE_INVALID_CUSTOMER_CONFIG_CBSL_SIZE
- ERR_LOG_CODE_CUSTOMER_CONFIG_CBSL_PROGRAMMED_OR_READ_ERR
- ERR_LOG_CODE_NVM_IS_READ_PROTECTED
- ERR_LOG_CODE_NVM_CONFIG_SECTOR_WRITE_PROTECTED
- ERR_LOG_CODE_NVM_CODE_PROGRAMMED
- ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED
- ERR_LOG_CODE_ACCESS_AB_MODE_ERROR
- ERR_LOG_CODE_NVM_ECC2_DATA_ERROR

- ERR_LOG_CODE_NVM_VER_ERROR
- ERR_LOG_CODE_BSL_RECV_BYTES_MISMATCH
- ERR_LOG_CODE_MSG_VALIDITY_FAIL

4.3.22 End of Transmission Message (80_H)

Firmware supports sending an end of transmission (EOT) message (80_H), which is used to send back data requested by a BSL command. The data field in the message can range from 1 to 128 bytes

EOT Block

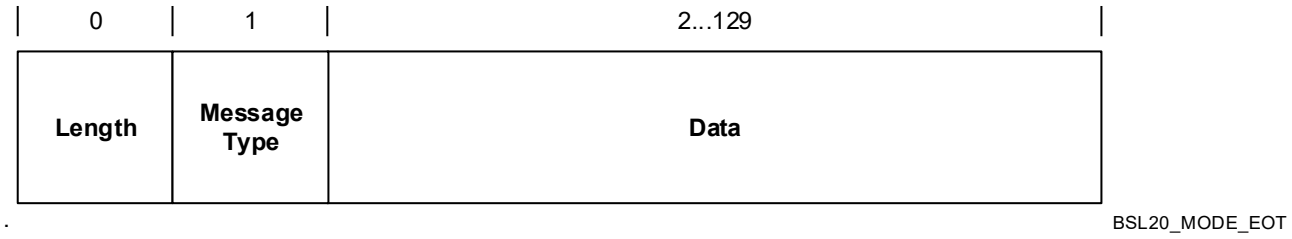


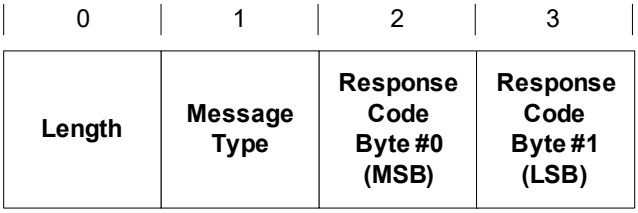
Table 4-38 EOT Block Field Description

Field	Description
Length	Number of bytes to follow (Message Type- and Data field)
Message Type	EOT block. Always set to 80_H
Data	Data to be read, minimum size 1 byte, maximum size 128 bytes

4.3.23 Acknowledge Response Message (81_H)

The firmware supports sending back an acknowledge response message (81_H) if the requested BSL command does not retrieve any data or the requested data cannot be provided. It is also sent if a problem occurred during processing the requested command data.

Response Block



BSL20_MODE_RESPONSE

Table 4-39 Acknowledge Response Block Field Description

Field	Description
Length	Number of following bytes in the Response Block. Always set to 03_H
Message Type	Response Block. Always set to 81_H
Response Code Byte #0 (MSB)	Signed 16-bit command response code. The value is set to zero if the requested command could be executed without any problems.
Response Code Byte #1 (LSB)	

5 NVM

5.1 NVM Overview

The NVM module consists of three regions, the Config Sector, the user code region and the user data region.

5.1.1 Config Sector Region

The Config Sector holds device specific information as well the eight 100TP pages. The Config Sector is not directly addressable by the user. To access the 100TP pages, dedicated user API functions are provided.

5.1.2 USER CODE Region

The user code region is intended to store the user application and/or constant user configurations. The user code area is divided into two parts. The first 4KB are called customer BSL region. It is a user code area which can be protected separately from the remaining user code. The customer BSL region is provided to store a user defined boot up code. The remaining user code is used to store the user application code. The entire user code area is directly addressable for read accesses. For writing/erasing data to the user code area dedicated user API functions are provided.

5.1.3 USER DATA Region

The last 4KB of the NVM module are the user data flash region. It is intended to store dynamical application data inside this NVM region. Constant data is recommended to be stored inside the user code area. For this sector an EEPROM emulation is implemented offering two modes of operation, data linear mode and data mapped mode, which are mutually exclusive to each other and get determined through the Configuration Sector (CS) switching key evaluation during device boot-up. Using a special BSL command (**“Command 8A$_H$ – NVM: Switch Keys Set” on Page 46**), the data region mode can get changed with the next reset (applies to all reset types). The default data region type is data mapped.

5.1.3.1 Data Mapped Mode

Data mapped mode allows to automatically mitigate NVM aging effects as with every page write a page previously marked as spare page gets mapped in and programmed. In data mapped mode, one NVM sector (4KByte) is available for data storage.

The EEPROM emulation is being achieved by the implementation of an address translation table, the so called MapRAM. At all accesses to the data flash, read or write, the user given address (logical) is being translated to the physical address by using the MapRAM. The data flash is directly addressable for read accessed (through the MapRAM), but only mapped pages return data. The read access to an unmapped page causes a NMI, to signal to the user application the attempt of reading not existing data. For writing/erasing data to the user data area dedicated user API functions are provided.

5.1.3.2 Data Linear Mode

Data linear mode allows direct access to all data words in the NVM data region without any mapping. In data linear mode, no spare pages are available, therefore the available data region encompasses two NVM sectors (8 Kbytes).

5.1.4 NVM Password Protection

Firmware supports setting and clearing of NVM protection for different NVM regions. These regions are the customer Bootloader NVM sectors, code and data NVM sectors. NVM region protection includes access protection for read and/or write/erase. NVM protection passwords are 32-bit in lenght, the two MSBs are reserved for read/write protection handling.

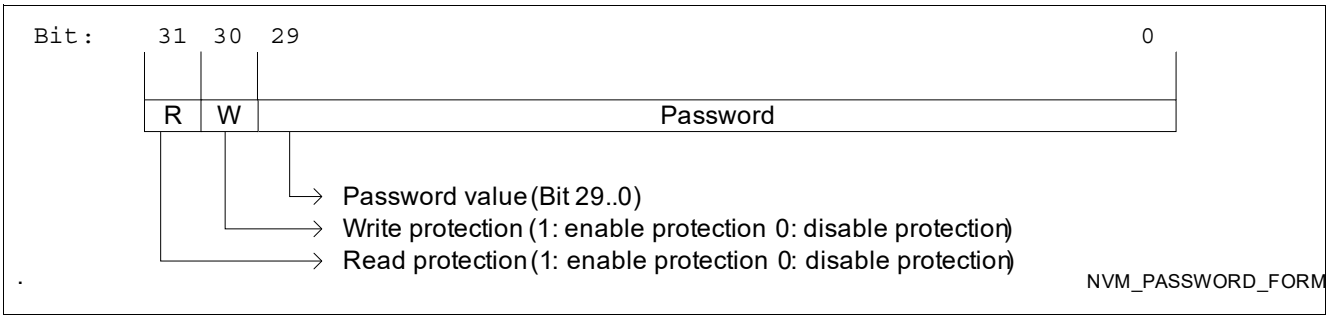


Figure 5-1 NVM Password Format

See also **"Command 89_H – NVM: Protection Password Set" on Page 44** for details on how to set or clear the NVM protection password

5.2 NVM Write

In order to write/modify data to a NVM page inside the user code or user data area, several user API functions are provided. From a user point of view there is no need to differentiate between the two major user NVM areas by using the API functions. The called user API functions are identifying by the given address the target user NVM region.

For writing a NVM page two scenarios are considered:

1.Writing a new page (erased or unmapped)

2.Writing a used page

The handling of these two scenarios, differ slightly between user code area and user data area. All the following described actions are performed by the user API functions, it does not describe user activities.

Note: It is strongly recommended to the user that no flash operations which modify the content of the flash, like write and erase, get interrupted at any time.

NVM Code Region and Data Linear Region

For writing a new code or data flash page the assembly buffer (AB) is opened. The AB is a small portion of SRAM inside the NVM hardware module to buffer the content of a NVM page for write activities. The AB is filled with 0xFF, the content of an erased page. The AB is updated with the data provided by the user along with the calling of the user API function. Then the content of the AB is written to the erased page addressed by the address provided by the user as parameter for the user API function.

For writing a used code or data flash area the AB is opened and the data inside the used page is copied into the assembly buffer. The AB is updated with the data provided by the user along with the calling of the user API function. The addressed NVM page is being erased afterwards the content of the AB is written to the erased page.

NVM Data Mapped Region

For writing a new data flash page, the user API function checks the content of the MapRAM for the given address. Since the page is not used, the MapRAM entry is marked unused. An internally maintained spare page points to a randomly selected erased physical data flash page. The assembly buffer (AB) is opened for the selected physical data flash page. The AB is updated with the data provided by the user along with the calling of the user API function. Then the content of the AB is written to the page addresses by the spare page. The link to the physical page is entered into the MapRAM and a new spare page is randomly selected.

For writing a used data flash page, the user API function checks the content of the MapRAM for the given address. The AB is opened and the data of the used (still mapped) page is copied into the AB. The AB is updated with the data provided by the user along with the calling of the user API function. Then the content of the AB is written to the page addresses by the spare page. The link to the new physical page is entered into the MapRAM. Now the old data flash page is being erased and a new spare page is randomly selected.

NVM Data Verification

For all NVM data write scenarios the data just written is verified against the content of the AB. The user can select whether a retry (erase-write) is performed in case the verify failed.

For the data flash along with the enabled retry feature also the Disturb Handling (DH) feature gets enabled. The goal of the Disturb Handling is to compensate for retention losses of pages long time not updated by the user. Retry and Disturb Handling are executed exclusively, either of the two can be executed with one user API call but not both. In case no retry is performed and based on a pseudo-random number generator the DH is called (in average on every 1000th write operation), a copying of a used page (not the one which was just written) is triggered. The actions performed by copying a used page inside the data flash sector are the same as for writing a used data flash page.

5.3 NVM Fast Write

A NVM fast write operation is provided to support timing critical NVM programming operations. It allows to program a page with reduced retention time by giving the user control over the number of programming pulses applied.

A complete fast write operation comprises a sequence of 4 mandatory sub-operations. The execution time per operation is guaranteed to not exceed 400us. The operations are required to be executed in the following order:

- NVM Fast Write Start: initiates a fast write operation on the target page.
- NVM Fast Write Continue: performs one write pulse on the target page (repeatable).
- NVM Fast Write Verify: performs a verifiy operation on the target page.
- NVM Fast Write End: finalizes the fast write operation

User APIs are provided for the above steps, refer to **“User API Routines” on Page 74**

The fast write flow provides only a limited margin towards NVM cell readout levels, resulting in a warranted data retention time of 2 days. If a longer data retention time is required, the user should rewrite the target page with full margins using the normal NVM write API during non time critical application states.

5.4 Data Flash Initialization

After a reset of the volatile memory, the MapRAM has to be recovered in order to be able to perform the address translation for data flash accesses. The content of the MapRAM is being recovered out of the MapBlock, control information stored along with each data flash page. If during the initialization of the

MapRAM an error occurred, i.e. a MapBlock of a data flash page contains ECC failures or if two pages are pointing to the same MapRAM entry (double mapping) then a repair function is called, the Service Algorithm (SA).

The Service Algorithm (SA) is executed only during startup as part of the MapRAM initialization function and only if failures occurred during the MapRAM initialization. The Service Algorithm scans the entire data flash sector and tries to repair as much as possible faulty pages and pages with ECC failures in the MapBlock. The SA further scans for double mappings, pages which point to the same MapRAM entry. Up to one double mapping can be resolved by deleting one of the two pages. If more than one double mapping is existent the SA is not able to repair.

The result of the Service Algorithm is being reflected in the register MEMSTAT. The user shall read this register upon user code entry. If unrecoverable failures in the data flash are signaled, appropriate data flash recovery has to be performed by the user, such as erase of the entire data flash sector.

Note: It is not recommended to perform any write/erase action to the data flash memory of the data memory mapping integrity provided through the MapRAM is not assured.

6 User Routines

The BootROM exports some library functions to the user mode software. These library functions allow to configure the device boot parameter and access the NVM.

6.1 List of Supported Features

- Read and write the various 100TP pages inside the NVM.
- Read, write and erase the NVM pages and sectors.
- Configure the BSL parameter (e.g. timeout configuration, FastLIN NAD address).
- Retrieve the customer identification number.
- Perform a RAM MBIST test.
- Check for ECC single- and double-errors on NVM and RAM.

All library functions are accessible over a branch table, where the user mode software can directly jump to. The branch table is stored at a fix location and in return branches to the function implementation.

An API reference to the user routines is given in **“User API Routines” on Page 74**.

6.2 Reentrance Capability and Interrupts

With the exception of a few functions, **no** support is provided for reentrance of user API routines – user calls must be atomic (i.e., they must not be interrupted and reentered before completion). The customer application must not call these routines from different multitasking levels (e.g. different thread/interrupt levels). As user API routines are potentially timing dependent, it is recommended to disable interrupts prior to calling API routines.

6.3 Address Parameters Range Checks

Some of the user API implementations use pointers to exchange data with the API. All pointers must point to a valid RAM address range. If the address points outside the valid RAM area, an error code is returned.

Other routines support branching or callbacks. If the callback is different from zero, it must point to a valid NVM or RAM range, otherwise an error is returned. If the callback is equal to zero, it simply behaves like the one without callback.

The user API functions reading or writing the NVM or RAM check the address range and report an error if the memory region is exceeded. The number of bytes to be read or written must be greater than zero.

6.4 NVM Region Write Protection Check

The user API functions writing or erasing a page in NVM or NVM CS check for NVM region write protection, and return an error code if the protection is set for that page.

6.5 Watchdog Handling When Using NVM Functions

The execution of all user API function is given in **Appendix E, "Execution time of BootROM User API Functions" on page 136**. The execution time of user API functions has to be observed for watchdog timeout calculations, especially when NVM operations are involved (programming or erase). Prior to invoking NVM write routines, it is recommended to do the following:

- Perform a short-open-window trigger to WDT1 before a NVM operation is called
- Invoke NVM write routine

- Reconfigure watchdog for normal operation

Doing this ensures that watchdog will not expire and cause reset in the middle of a longer NVM operation.

6.6 Interrupts

System interrupts are not used by any BootROM functions during startup or when any user APIs are executed. Interrupts are disabled by default. Customer software must enable interrupts and service system interrupts in a normal fashion, which means installing interrupt vectors at the correct locations for the system CPU.

6.7 Resources used by user API functions

Listed below is a list of user API functions with what kind of HW resources they use.

The stack usage of the functions is listed in **Appendix B, "Stack usage of user API functions" on page 128**.

Table 6-1 Resources used by User API functions

User API function	Non Re-entrance	NVM HW	GPT12 timer
user_adc1_offset_calibration			
user_nvm_page_checksum_check	X	X (read)	
user_nvm_service_algorithm	X	X (write)	
user_nvm_mapram_recover	X		
user_nvm_mapram_init	X		
user_nvm_ecc_events_get			
user_nvm_ecc_check	X	X (read)	
user_nac_get		X (read)	
user_nac_set	X	X (write)	
user_nad_get		X (read)	
user_nad_set	X	X (write)	
user_nvm_100tp_read	X	X (read)	
user_nvm_100tp_write / user_nvm_100tp_erase	X	X (write)	
user_nvm_config_get			
user_nvm_protect_get			
user_nvm_protect_set / user_nvm_protect_clear	X	X (read)	
user_nvm_password_set	X	X (write)	
user_nvm_ready_poll			
user_nvm_page_erase / user_nvm_page_erase_branch / user_nvm_sector_erase	X	X (write)	
user_nvm_page_verify / user_nvm_page_erase_verify / user_nvm_sector_erase_verify	X	X (write)	

Table 6-1 Resources used by User API functions

User API function	Non Re-entrance	NVM HW	GPT12 timer
user_nvm_write / user_nvm_write_branch / user_nvm_write_fast / user_nvm_write_fast_retry / user_nvm_write_fast_abort /	X	X (write)	
user_ram_mbist			X
user_nvm_clk_factor_set			
user_vbg_temperature_get		X (read)	
user_dflash_mode			

6.8 User API Routines

These routines are exported by the BootROM to the customer user mode software.

User API Routines support features like accessing memory resources like NVM and 100TP pages. They also support to configure some protection mechanism and BSL parameters. The API functions check the valid parameter range, which is depending on the device.

Table 6-2 User API Routines Function Overview

Name	Description
user_adc1_offset_calibration	This user API function will perform ADC1 core calibration in software mode. It updates ADC1_OFFSETCALIB register if the calibration algorithm is executed successfully. It returns ERR_LOG_ERROR in case a proper calibration offset value can not be found.
user_dflash_mode	This function returns the current configured NVM data region mode.
user_nac_get	This user API function reads out the NAC value that is currently configured in the non volatile device configuration memory.
user_nac_set	This user API function configures the NAC value in the non volatile device configuration memory.
user_nad_get	This user API function reads out the FASTLIN NAD value that is currently configured in the non volatile device configuration memory.
user_nad_set	This user API function configures the FASTLIN NAD value in the non volatile device configuration memory.
user_nvm_100tp_erase	This user API function erases all data in one of the 100TP NVM pages preserving the write counter. The erase operation is not executed in case the NVM code segment write protection is set and a dedicated protection error is returned. In case the erase operation is executed, the page is initialized with a wrong checksum.
user_nvm_100tp_read	This user API function reads data from the customer accessible configuration pages (100TP). The read address is relative inside the configuration NVM area (8x one page, 1024 bytes). Invalid parameters (page number out of range, offset plus count larger than page boundary, count is 0) returns an error, and no read operation is performed.

Table 6-2 User API Routines Function Overview (cont'd)

Name	Description
user_nvm_100tp_write	This user API function writes data to the configuration NVM, the write address is relative inside the configuration NVM area (8x one page, 1024 bytes). The function supports partial page programming, preserving the page data not passed as an input. The function performs an implicit update of the page checksum and write counter. The write counter is increased by 1 at each write operation, and when 99 is reached an error is reported. The function does not allow the customer to change the page checksum or write counter. Any invalid parameters (page number out of range, offset plus count larger than page boundary, count is 0) returns an error, and no write operation is performed. The function also returns an error in case the NVM code segment is write protected. The write counter and the page checksum are located in the last two bytes of the page.
user_nvm_clk_factor_set	This user API function sets the SCU_SYSCON0.NVMCLKFAC divider
user_nvm_config_get	This user API function allows to gather the NVM configuration, this is the number of sectors for customer bsl region, code region and data region.
user_nvm_ecc_check	This user API function checks for single and double ECC checking over the entire NVM on all NVM linear and NVM non-linear sectors. The NVM data sector configuration is taken into account. Any existing ECC errors are cleared before the read starts. Upon exit, the function will clear the current ECC status in the NVM module. The API rejects operation in case any read protection is set on the user NVM.
user_nvm_ecc_events_get	This user API function checks if any single or double ECC events have occurred during runtime. It reports any single or double ECC event coming from NVM. The corresponding last double ECC failure address is returned via modified pointer.
user_nvm_mapram_init	This user API function triggers NVM FSM mapRAM update sequence from mapped sector.
user_nvm_mapram_recover	This user API function will manually do mapRAM initialization by extracting mapping info on good mapped pages. This can be called if NVM SA fails repairing a corrupted mapped data sector. A request to initialize the mapRAM for a not available sector as well as for a linearly mapped sector is ignored. Double mapped pages or more are counted as faulty pages.
user_nvm_page_checksum_check	This user API function will perform a checksum check on the specified NVM page

Table 6-2 User API Routines Function Overview (cont'd)

Name	Description
user_nvm_page_erase	This user API function erases a given NVM page (address). In case of an unsused (new) page in non-linear sector, the function does nothing and returns success. In case of erasing a page in linear sector, the function should always perform the erase.
user_nvm_page_erase_branch	This user API function erases a given NVM page (address) and branches to an address (branch_address) for code execution during the NVM operation.
user_nvm_page_erase_verify	This function verifies with HardRead Erased margin on a page to check that all bits are erased. For linear region the check is done on the page pointed by the address provided as input. In case the routine target a mapped page, the check is performed on the current spare page.
user_nvm_page_verify	This function reads the physical page content into the NVM assembly buffer using Normal Read Margins. The content of the assembly buffer is then used to check the physical page content by using the hardread margins erase and written.
user_nvm_password_set	This user API function sets password for NVM region individually. The API does not change the protection state for a region where password protection is currently installed.
user_nvm_protect_clear	This user API function clears write protection for any NVM region individually, except CBSL. Read protection changes is ignored. The API changes the protection state for a region, but does not update the installed password in config sector.
user_nvm_protect_get	This user API function checks for the hardware current applied NVM protection status.
user_nvm_protect_set	This user API function sets write protection for any NVM region individually, except CBSL. The API changes the protection state for a region, but does not update the installed password in configuration sector. It is not possible to change read protection for the segments. It will be silently ignored.
user_nvm_ready_poll	This user API function checks for the readiness of the NVM module. The API is called within the NVM programming or erase branch callback operation. It checks if the NVM operation has finished and the callback could return to the NVM routine.
user_nvm_sector_erase	This user API function erases the NVM sector-wise. It operates on user code and NVM data region.
user_nvm_sector_erase_verify	This function performs a page-by-page erase check for a full sector. Each page is checked against Hardread-Margin-Erased.
user_nvm_service_algorithm	This user API function will run service algorithm on a mapped NVM sector. Only if a mapped sector is configured, NVM SA will be executed.

Table 6-2 User API Routines Function Overview (cont'd)

Name	Description
user_nvm_write	This user API function programs the NVM. It operates on the user NVM, as well as on the user data NVM. The API shall write a number of bytes (count) from the source (data) to the NVM location (address) with the programming options (options). The options provide parameters like disturb handling and fail scenario handling.
user_nvm_write_branch	This user API function programs the NVM. It operates on the user NVM, as well as on the user data NVM. The API shall write a number of bytes (count) from the source (data) to the NVM location (address) with the programming options (options). During the NVM operation the program execution branches to a given SRAM location (branch_address) and continues code execution from there. The options provide parameters like disturb handling and fail scenario handling.
user_nvm_write_fast_continue	This user API function executes the second step of fast write sequence.
user_nvm_write_fast_end	This user API function executes the last step of fast write sequence.
user_nvm_write_fast_start	This user API function starts the NVM faster than normal programming sequence (see also user_nvm_write_fast_continue(), user_nvm_write_fast_verify() and user_nvm_write_fast_end()). It operates on the NVM code region, as well as on the NVM data region. The API shall write a number of bytes (count) from the source (data) to the NVM location (address). It supports partial non-page-aligned programming, preserving the page data not passed as an input. Crossing page boundary is not supported. This function rejects with an error in case the accessed NVM page is write protected.
user_nvm_write_fast_verify	This user API function executes the third step of fast write sequence.
user_ram_mbist	This user API function performs a MBIST on the integrated SRAM. The range to check is provided as parameter. The function rejects the call in case the parameter exceeds the RAM address range.
user_vbg_temperature_get	This user API function returns the V bandgap temperature hot or cold. The customer can do temperature compensation in software based on these values, as the VBG is the reference for the ADC.

6.8.1 user_nvm_write_fast_start

Description

This user API function starts the NVM faster than normal programming sequence (see also **user_nvm_write_fast_continue**, **user_nvm_write_fast_verify** and **user_nvm_write_fast_end**). It

operates on the NVM code region, as well as on the NVM data region. The API shall write a number of bytes (count) from the source (data) to the NVM location (address). It supports partial non-page-aligned programming, preserving the page data not passed as an input. Crossing page boundary is not supported. This function rejects with an error in case the accessed NVM page is write protected.

Failing to start the fast write sequence with user_nvm_write_fast_start will result in the error code ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE when calling the subsequent fast write functions.

Prototype

```
int32_t user_nvm_write_fast_start (
  uint32_t address
  const void * data
  uint32_t count
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	NVM address where to program the data. Range is 0x11000000 + device NVM size.	-
const void *	data	Pointer to the data where to read the programming data. Pointer must be within valid RAM range (0x18000000 + device RAM size) or an error code is returned.	-
uint32_t	count	Amount of bytes to program. Range from 1-128 bytes.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful fast write operation start, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_PARAM_INVALID, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_ADDRESS_RANGE_CROSSING_PAGE_BOUNDARY, ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPBLOCK_ERROR, ERR_LOG_CODE_NVM_MAPRAM_UNKNOWN_TYPE_USAGE, ERR_LOG_CODE_NVM_FAST_PROG_NOT_ALLOWED

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a reentrant context.

6.8.2 user_nvm_write_fast_continue

Description

This user API function executes the second step of fast write sequence.

It can only be called after successful sequence start by prior call to **user_nvm_write_fast_start**, otherwise the operation is rejected returning error ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE.

Calling the API in reentrant context is not allowed. Only when the current step is completed, can the user execute the next fast write step. Otherwise, the operation is rejected returning ERR_LOG_CODE_NVM_WRITE_FAST_SEMAPHORE_RESERVED.

Calls to this function can be performed multiple times although upon reaching a certain number it will return ERR_LOG_CODE_NVM_WRITE_FAST_REACH_MAX_RETRIES, meaning no further NVM write attempts are being made.

Prototype

```
int32_t user_nvm_write_fast_continue (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS if fast write continue operation was OK, otherwise a negative value. Returned status code can be one of the following: ERR_LOG_SUCCESS ERR_LOG_CODE_NVM_WRITE_FAST_SEMAPHORE_RESERVED, ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE, ERR_LOG_CODE_NVM_WRITE_FAST_REACH_MAX_RETRIES

6.8.3 user_nvm_write_fast_verify

Description

This user API function executes the third step of fast write sequence.

It can only be called after successful sequence start by prior call to **user_nvm_write_fast_continue**, otherwise the operation is rejected returning error ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE.

Calling the API in reentrant context is not allowed. Only when the current step is completed, can the user execute the next fast write step. Otherwise, the operation is rejected returning ERR_LOG_CODE_NVM_WRITE_FAST_SEMAPHORE_RESERVED.

Calls to this function should be placed after the scheduled calls to nvm_write_fast_continue() have been completed in order to determine current data quality.

To finish the entire process the user must then call **user_nvm_write_fast_end**.

Prototype

```
int32_t user_nvm_write_fast_verify (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS if fast write continue operation was OK, otherwise a negative value. Returned status code can be one of the following: ERR_LOG_SUCCESS ERR_LOG_CODE_NVM_WRITE_FAST_SEMAPHORE_RESERVED, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE

6.8.4 user_nvm_write_fast_end

Description

This user API function executes the last step of fast write sequence.

Calling the API in reentrant context is not allowed, the operation would be rejected returning error ERR_LOG_CODE_NVM_WRITE_FAST_SEMAPHORE_RESERVED.

It can only be called after successful sequence initially start by prior call to **user_nvm_write_fast_verify**, otherwise the operation is rejected returning error ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE.

If called without prior call to **user_nvm_write_fast_verify**, no data would be written.

Prototype

```
int32_t user_nvm_write_fast_end (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS if the fast write end was OK, otherwise a negative value. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_NVM_WRITE_FAST_SEMAPHORE_RESERVED, ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE, ERR_LOG_CODE_NVM_MAPRAM_MANUAL_SPARE_PAGE_FAILED

6.8.5 user_adc1_offset_calibration

Description

This user API function will perform ADC1 core calibration in software mode. It updates ADC1_OFFSETCALIB register if the calibration algorithm is executed successfully. It returns ERR_LOG_ERROR in case a proper calibration offset value can not be found.

Prototype

```
int32_t user_adc1_offset_calibration (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been executed successfully, Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_ERROR

6.8.6 user_nvm_page_checksum_check

Description

This user API function will perform a checksum check on the specified NVM page

This function rejects with an error in case the NVM page is out of range.

Prototype

```
int32_t user_nvm_page_checksum_check (
  uint32_t page_no
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	page_no	Page to check, which is one of the 100TP pages (0..7) or NVM CS page containing BSL (startup) settings (8).	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been called successfully, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_INVALID_NVM_PAGE_NUMBER, ERR_LOG_CODE_CS_PAGE_CHECKSUM, ERR_LOG_CODE_CS_PAGE_ECC2READ, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context. Upon exit, the function will clear the current ECC status.

6.8.7 user_nvm_service_algorithm

Description

This user API function will run service algorithm on a mapped NVM sector. Only if a mapped sector is configured, NVM SA will be executed.

This function rejects with an error in case the NVM data mapped region is write protected.

Prototype

```
int32_t user_nvm_service_algorithm (
  uint32_t * sa_result
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t *	sa_result	Pointer where to store the result of the NVM SA run. Format follows the same layout as lower 8 bits of register SCU_MEMSTAT, but it doesn't update SCU_MEMSTAT. Since the API executes SA explicitly, for a completely good mapped sector, SASTATUS bit field of the returned result will be 11B, which is not soft error in this context. And SECTORINFO bit field of the returned result will always be the mapped sector. Pointer must be located in RAM.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been called successfully, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_USER_NO_NVM_MAPPED_SECTOR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.8 user_nvm_mapram_recover

Description

This user API function will manually do mapRAM initialization by extracting mapping info on good mapped pages. This can be called if NVM SA fails repairing a corrupted mapped data sector. A request to initialize the mapRAM for a not available sector as well as for a linearly mapped sector is ignored. Double mapped pages or more are counted as faulty pages.

Prototype

```
int32_t user_nvm_mapram_recover (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
int32_t	A positive number (including 0) tells the amount of good mapped pages. In case of other errors, a negative number is returned as an error code. Returned status code can be one of the following: ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_USER_NO_NVM_MAPPED_SECTOR, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.9 user_nvm_mapram_init

Description

This user API function triggers NVM FSM mapRAM update sequence from mapped sector.

Prototype

```
int32_t user_nvm_mapram_init (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been called successfully, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_NVM_INIT_MAPRAM_SECTOR, ERR_LOG_CODE_USER_NO_NVM_MAPPED_SECTOR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.10 user_nvm_ecc_events_get

Description

This user API function checks if any single or double ECC events have occurred during runtime. It reports any single or double ECC event coming from NVM. The corresponding last double ECC failure address is returned via modified pointer.

Pointer must be within valid RAM range (0x18000000 + device RAM size) or an error code is returned.

Prototype

```
int32_t user_nvm_ecc_events_get (
  uint32_t * pNVM_Addr
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t *	pNVM_Addr	Pointer where to store the ECC2 failing NVM address. This pointer stays untouched in case no NVM ECC2 errors was detected. The address format is: NVM area: 00010001000000XXXXXXXXXXXXXXXX000b, where X is the NVM offset NVM 100TP area: 0x100000XY, where X = 100TP page number, Y = block offset inside the page Internal NVM CS area: 0x01000000	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case no single or double ECC event have occurred, A negative error code for single, double or single and double ECC errors A negative error code if the NVM semaphore is not free. This function will also fail if NVM is still busy with another operation. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_SINGLE_ECC_EVENT_OCCURRED, ERR_LOG_CODE_DOUBLE_ECC_EVENT_OCCURRED, ERR_LOG_CODE_SINGLE_AND_DOUBLE_ECC_EVENT_OCCURRED,

Remarks

Upon exit, the function will clear the current ECC status. Any NVM write/erase/verify operation will also clear the ECC status.

6.8.11 user_nvm_ecc_check

Description

This user API function checks for single and double ECC checking over the entire NVM on all NVM linear and NVM non-linear sectors. The NVM data sector configuration is taken into account. Any existing ECC errors are cleared before the read starts. Upon exit, the function will clear the current ECC status in the NVM module. The API rejects operation in case any read protection is set on the user NVM.

Prototype

```
int32_t user_nvm_ecc_check (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case no single or double ECC event have occurred, otherwise a negative error code for single, double or single and double ECC errors. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_SINGLE_ECC_EVENT_OCCURRED, ERR_LOG_CODE_DOUBLE_ECC_EVENT_OCCURRED, ERR_LOG_CODE_SINGLE_AND_DOUBLE_ECC_EVENT_OCCURRED, ERR_LOG_CODE_NVM_SEGMENT_READ_PROTECTED

Remarks

This routine does not provide the ECC error address. Please use the **user_nvm_ecc_events_get** routines to retrieve the addresses. It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

Upon exit, the function will clear the current ECC status

6.8.12 user_nac_get

Description

This user API function reads out the NAC value that is currently configured in the non volatile device configuration memory.

Prototype

```
int32_t user_nac_get (
  uint8_t * nac_value
)
```

Parameters

Data Type	Name	Description	Dir
uint8_t *	nac_value	Pointer where to store the BSL NAC value read from the device configuration sector. Pointer must be located in RAM. NAC value is the one found in the configuration memory.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been called successfully, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID,

6.8.13 user_nac_set

Description

This user API function configures the NAC value in the non volatile device configuration memory.

This function rejects with an error in case the NVM code segment is write protected or NAC value is out of valid range [0, 2-28, 255].

Prototype

```
int32_t user_nac_set (
  uint8_t nac
)
```

Parameters

Data Type	Name	Description	Dir
uint8_t	nac	NAC value to be stored in the device configuration memory.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful write operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_NVM_CONFIG_SECTOR_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NAC_VALUE_INVALID, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_NVM_VER_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.14 user_nad_get

Description

This user API function reads out the FASTLIN NAD value that is currently configured in the non volatile device configuration memory.

Prototype

```
int32_t user_nad_get (
  uint8_t * nad_value
```

```
)
```

Parameters

Data Type	Name	Description	Dir
uint8_t *	nad_value	Pointer where to store the BSL nad value read from the device configuration sector. Pointer must be located in RAM.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been called successfully, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID,

6.8.15 user_nad_set

Description

This user API function configures the FASTLIN NAD value in the non volatile device configuration memory. This function rejects with an error in case the NVM code segment is write protected.

Prototype

```
int32_t user_nad_set (
  uint8_t nad
)
```

Parameters

Data Type	Name	Description	Dir
uint8_t	nad	FASTLIN NAD value to be stored in the device configuration memory. Valid range is from 0x00-0xFF.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful write operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_NVM_CONFIG_SECTOR_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_NVM_VER_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.16 user_nvm_100tp_read

Description

This user API function reads data from the customer accessible configuration pages (100TP). The read address is relative inside the configuration NVM area (8x one page, 1024 bytes). Invalid parameters (page number out of range, offset plus count larger than page boundary, count is 0) returns an error, and no read operation is performed.

A maximum number of 127 bytes can be read by this function (including the page counter).

Prototype

```
int32_t user_nvm_100tp_read (
  uint32_t page_num
  uint32_t offset
  void * data
  uint32_t count
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	page_num	Page number where to read from. Valid range: 0 to 7	-
uint32_t	offset	Byte offset inside the selected page address, where to start reading from. Maximum is 127 bytes. If count plus offset is larger than 127, an error code is returned.	-
void *	data	Data pointer where to store the data read. Pointer plus valid count must be within valid RAM range or an error code is returned	-
uint32_t	count	Amount of data bytes to read. If count is zero, there is no operation performed and an error code is returned. Maximum is 127 bytes.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful read operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_100TP_PAGE_INVALID, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.17 user_nvm_100tp_write

Description

This user API function writes data to the configuration NVM, the write address is relative inside the configuration NVM area (8x one page, 1024 bytes). The function supports partial page programming, preserving the page data not passed as an input. The function performs an implicit update of the page checksum and write counter. The write counter is increased by 1 at each write operation, and when 99 is reached an error is reported. The function does not allow the customer to change the page checksum or write counter. Any invalid parameters (page number out of range, offset plus count larger than page boundary, count is 0) returns an error, and no write operation is performed. The function also returns an error in case the NVM code segment is write protected. The write counter and the page checksum are located in the last two bytes of the page.

The maximum value for writing is 126 bytes.

Prototype

```
int32_t user_nvm_100tp_write (
  uint32_t page_num
  uint32_t offset
  const void * data
  uint32_t count
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	page_num	Page number where to write to. Valid range: 0 to 7	-
uint32_t	offset	Byte offset inside the selected page address, where to start writing. Maximum is 126 bytes.	-
const void *	data	Data pointer where to read the data to write. Pointer plus valid count must be within valid RAM range or an error code is returned	-
uint32_t	count	Amount of data bytes to write. If count is zero, there is no write operation done and an error code is returned. Maximum is 126 bytes.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful write operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_NVM_CONFIG_SECTOR_WRITE_PROTECTED, ERR_LOG_CODE_100TP_PAGE_INVALID, ERR_LOG_CODE_100TP_WRITE_COUNT_EXCEEDED, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_ECC2READ_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.18 user_nvm_100tp_erase

Description

This user API function erases all data in one of the 100TP NVM pages preserving the write counter. The erase operation is not executed in case the NVM code segment write protection is set and a dedicated protection error is returned. In case the erase operation is executed, the page is initialized with a wrong checksum.

In order to restore the write counter, the routine needs to access the existing data stored into the page. If page contains ECC2 error, write counter is reconfigured to allow a maximum of 5 more write operations.

To avoid triggering an ECC2 NMI, the write counter read is performed using an special internal read flow

Prototype

```
int32_t user_nvm_100tp_erase (
  uint32_t page_num
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	page_num	100TP page number to erase. Valid range: 0 to 7	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful write operation, otherwise a negative error code. Exception is ERR_LOG_CODE_NVM_100TP_PAGE_CNT_ERROR, which means page is erased OK but ECC2 error was detected while reading write counter. Write counter is set so it allows 5 more writes to the page. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_NVM_100TP_PAGE_CNT_ERROR, ERR_LOG_CODE_100TP_PAGE_INVALID, ERR_LOG_CODE_NVM_CONFIG_SECTOR_WRITE_PROTECTED ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_100TP_WRITE_COUNT_EXCEEDED ERR_LOG_CODE_NVM_ECC2_DATA_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.19 user_nvm_config_get

Description

This user API function allows to gather the NVM configuration, this is the number of sectors for customer bsl region, code region and data region.

Pointer must be within valid RAM range or an error code is returned.

Prototype

```
int32_t user_nvm_config_get (
  uint32_t * cbsl_nvm_size
  uint32_t * code_nvm_size
  uint32_t * data_nvm_size
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t *	cbsl_nvm_size	Pointer where to store the retrieved NVM cbsl size. Valid RAM range is 0x18000000 + device RAM size.	-

Data Type	Name	Description	Dir
uint32_t *	code_nvm_size	Pointer where to store the retrieved NVM code size. Valid RAM range is 0x18000000 + device RAM size.	-
uint32_t *	data_nvm_size	Pointer where to store the retrieved NVM data size. Valid RAM range is 0x18000000 + device RAM size. The value returned can reflect either number of mapped or linear data sectors. NVM configuration is checked inside the routine.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful configuration retrieve operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID

6.8.20 user_nvm_protect_get

Description

This user API function checks for the hardware current applied NVM protection status.

Prototype

```
uint32_t user_nvm_protect_get (
  NVM_PASSWORD_SEGMENT_t segment
)
```

Parameters

Data Type	Name	Description	Dir
NVM_PASSWORD_SEGMENT_t	segment	Which NVM segment to retrieve the current password protection status	-

Return Values

Data Type	Description
uint32_t	Current protection status of the NVM segment selected: Protection disabled: 0x00000000 read protection enabled: 0x80000000 Write protection enabled: 0x40000000 Read and write protection enabled: 0xC0000000 Segment not recognized: 0xFFFFFFFF

6.8.21 user_nvm_protect_set

Description

This user API function sets write protection for any NVM region individually, except CBSL. The API changes the protection state for a region, but does not update the installed password in configuration sector. It is not possible to change read protection for the segments. It will be silently ignored.

A valid password must be provided in case any valid NVM protection password is installed for this region.

Set bit 30 of the password parameter to enable write protection. The bits (0...29) of the password parameter shall match the password installed before. In case no valid protection password is currently installed, bits (0...29) are ignored. Bit 31 (read protection) is ignored.

If selected data segment doesn't match the NVM data segment configuration a mismatch error is returned.

Prototype

```
int32_t user_nvm_protect_set (
  uint32_t password
  NVM_PASSWORD_SEGMENT_t segment
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	password	Protection password to apply on the given segment	-
NVM_PASSWORD_SEGMENT_t	segment	Segment which should be password protected	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the password could be successfully applied, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_NVM_SEGMENT_INVALID, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_USER_PROTECT_WRONG_PASSWORD, ERR_LOG_CODE_USER_NVM_SEGMENT_CONFIG_MISMATCH, ERR_LOG_CODE_ECC2READ_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.22 user_nvm_protect_clear

Description

This user API function clears write protection for any NVM region individually, except CBSL. Read protection changes is ignored. The API changes the protection state for a region, but does not update the installed password in config sector.

A valid password must be provided in case any valid NVM protection password is installed for this region. Set bit 30 of the password parameter to disable write protection. The bits (0...29) of the password parameter shall match the password installed. Bit 31 (read protection) is ignored/not supported.

The password parameter is ignored in case no valid protection password is currently installed.

If selected data segment doesn't match the NVM data segment configuration a mismatch error is returned.

Prototype

```
int32_t user_nvm_protect_clear (
  uint32_t password
  NVM_PASSWORD_SEGMENT_t segment
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	password	Protection password to apply on the given segment	-
NVM_PASSWORD_SEGMENT_t	segment	Segment which should be password protected	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the password could be successfully applied, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_NVM_SEGMENT_INVALID, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_USER_PROTECT_WRONG_PASSWORD, ERR_LOG_CODE_USER_NVM_SEGMENT_CONFIG_MISMATCH, ERR_LOG_CODE_ECC2READ_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.23 user_nvm_password_set

Description

This user API function sets password for NVM region individually. The API does not change the protection state for a region where password protection is currently installed.

The password parameter consists of a 30-bit password (bit 0...29) and two additional protection bits (bit 30 + bit 31).

A valid password must be different from 0x3FFFFFFF and 0x00000000 (bit 0...29). The two MS bits in the password contain the protection type, where setting bit 31 activates the read protection and setting bit 30 activates the write protection. A non-compliant password is rejected.

A password can only be applied in case no valid password is currently set for the requested region.

Before updating starts, all interrupts including NMI are temporarily disabled and any NVM and NVM CS protection is disabled.

If selected data segment doesn't match the NVM data segment configuration a mismatch error is returned.

Prototype

```
int32_t user_nvm_password_set (
  uint32_t password
  NVM_PASSWORD_SEGMENT_t segment
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	password	Protection password to apply on the given segment	-
NVM_PASSWORD_SEGMENT_t	segment	Segment which should be password protected	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the password could be successfully applied, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_NVM_SEGMENT_INVALID, ERR_LOG_CODE_USER_PROTECT_PWD_INVALID, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_USER_PROTECT_PWD_EXISTS, ERR_LOG_CODE_USER_NVM_SEGMENT_CONFIG_MISMATCH, ERR_LOG_CODE_ECC2READ_ERROR, ERR_LOG_CODE_NVM_IS_BUSY

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.24 user_nvm_ready_poll

Description

This user API function checks for the readiness of the NVM module. The API is called within the NVM programming or erase branch callback operation. It checks if the NVM operation has finished and the callback could return to the NVM routine.

Prototype

```
bool user_nvm_ready_poll (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
bool	True in case the requested NVM operation is already finished, otherwise false.

6.8.25 user_nvm_page_erase

Description

This user API function erases a given NVM page (address). In case of an unsused (new) page in non-linear sector, the function does nothing and returns success. In case of erasing a page in linear sector, the function should always perform the erase.

This function rejects with an error in case the accessed NVM page is write protected.

Prototype

```
int32_t user_nvm_page_erase (
  uint32_t address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	Address of the NVM page to erase. Non-aligned address is accepted. Range is 0x11000000 + device NVM size.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful erase operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR

Remarks

This function does not support erasing any 100TP pages.

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.26 user_nvm_page_erase_branch

Description

This user API function erases a given NVM page (address) and branches to an address (branch_address) for code execution during the NVM operation.

This function rejects with an error in case the accessed NVM page is write protected.

Prototype

```
int32_t user_nvm_page_erase_branch (
  uint32_t address
  user_callback_t branch_address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	Address of the NVM page to erase. Non-aligned address is accepted. Range is 0x11000000 + device NVM size.	-
user_callback_t	branch_address	Function callback address where to jump while waiting for the NVM module to finish the erase operation. Address must be within valid RAM range (0x18000000 + device RAM size). RAM end address - 4 is the upper limit.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful erase operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR

Remarks

This function does not support to erase any 100TP pages.

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.27 user_nvm_sector_erase

Description

This user API function erases the NVM sector-wise. It operates on user code and NVM data region.

This function rejects with an error in case the NVM region the address belongs to is write protected.

Prototype

```
int32_t user_nvm_sector_erase (
  uint32_t address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	Address of the NVM sector to erase. Non-aligned address is accepted. Range is 0x11000000 + device NVM size.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful erase operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_NVM_INIT_MAPRAM_SECTOR ERR_LOG_CODE_ACCESS_AB_MODE_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

In case of non linear sector , the sector erase function has to run mapram init starting from a random position to get a random spare page (for the next programming).

6.8.28 user_nvm_write

Description

This user API function programs the NVM. It operates on the user NVM, as well as on the user data NVM. The API shall write a number of bytes (count) from the source (data) to the NVM location (address) with the programming options (options). The options provide parameters like disturb handling and fail scenario handling.

Supported option parameters:

- NVM_PROG_CORR_ACT (for linear sector: it enables retry. for mapped sector: it enables retry and disturb handling)
- NVM_PROG_NO_FAILPAGE_ERASE (only support mapped sector: when program new page verify fails, without the option, the newly programmed data is erased; with the option, the faulty page will remain. When program used page verify fails, without the option, the newly programmed data will be erased, the old page remains; With the option, the old page is erased, the newly programmed faulty page remains and MapRAM swapped.)

It supports partial non-page-aligned programming, preserving the page data not passed as an input. Crossing page boundary is not supported.

This function rejects with an error in case the accessed NVM page is write protected.

Prototype

```
int32_t user_nvm_write (
  uint32_t address
  const void * data
  uint32_t count
  uint32_t options
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	NVM address where to program the data. Range is 0x11000000 + device NVM size.	-
const void *	data	Pointer to the data where to read the programming data. Pointer must be within valid RAM range (0x18000000 + device RAM size) or an error code is returned.	-
uint32_t	count	Amount of bytes to program. Range from 1-128 bytes.	-
uint32_t	options	NVM programming options (e.g. **NVM_PROG_CORR_ACT** or **NVM_PROG_NO_FAILPAGE_ERASE**)	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful write operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_PARAM_INVALID, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_ADDRESS_RANGE_CROSSING_PAGE_BOUNDARY, ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_MAPRAM_UNKNOWN_TYPE_USAGE, ERR_LOG_CODE_NVM_ECC2_MAPBLOCK_ERROR, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_NVM_PROG_MAPRAM_INIT_FAIL, ERR_LOG_CODE_NVM_PROG_VERIFY_MAPRAM_INIT_FAIL

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.29 user_nvm_write_branch

Description

This user API function programs the NVM. It operates on the user NVM, as well as on the user data NVM. The API shall write a number of bytes (count) from the source (data) to the NVM location (address) with the programming options (options). During the NVM operation the program execution branches to a given SRAM location (branch_address) and continues code execution from there. The options provide parameters like disturb handling and fail scenario handling.

Supported option parameters:

- NVM_PROG_CORR_ACT (for linear sector: it enables retry. for mapped sector: it enables retry and disturb handling)
- NVM_PROG_NO_FAILPAGE_ERASE (only support mapped sector: when program new page verify fails, without the option, the newly programmed data is erased; with the option, the faulty page will remain. When program used page verify fails, without the option, the newly programmed data will be erased, the old page remains; With the option, the old page is erased, the newly programmed faulty page remains and MapRAM swapped.)

It supports partial non-page-aligned programming, preserving the page data not passed as an input. Crossing page boundary is not supported.

This function rejects with an error in case the accessed NVM page is write protected.

Prototype

```
int32_t user_nvm_write_branch (
  uint32_t address
  const void * data
  uint32_t count
  uint32_t options
  user_callback_t branch_address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	NVM address where to program the data. Range is 0x11000000 + device NVM size.	-
const void *	data	Pointer to the data where to read the programming data. Pointer must be within valid RAM range (0x18000000 + device RAM size) or an error code is returned.	-
uint32_t	count	Amount of bytes to program. Range from 1-128 bytes.	-
uint32_t	options	NVM programming options (e.g. **NVM_PROG_CORR_ACT** or **NVM_PROG_NO_FAILPAGE_ERASE**)	-
user_callback_t	branch_address	Function callback address where to jump while waiting for the NVM module to finish the program operation. Address must be within RAM range (0x18000000 + device RAM size). RAM end address - 4 is the upper limit.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful write operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_PARAM_INVALID, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_ADDRESS_RANGE_CROSSING_PAGE_BOUNDARY, ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_MAPRAM_UNKNOWN_TYPE_USAGE, ERR_LOG_CODE_NVM_ECC2_MAPBLOCK_ERROR, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_NVM_PROG_MAPRAM_INIT_FAIL, ERR_LOG_CODE_NVM_PROG_VERIFY_MAPRAM_INIT_FAIL,

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.30 user_ram_mbist

Description

This user API function performs a MBIST on the integrated SRAM. The range to check is provided as parameter. The function rejects the call in case the parameter exceeds the RAM address range.

Prototype

```
int32_t user_ram_mbist (
  uint32_t start_address
  uint32_t end_address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	start_address	RAM memory address where to start the MBIST test. Range is 0x18000000 + device RAM size.	-
uint32_t	end_address	RAM memory address till where to perform the MBIST test. Range is 0x18000000 + device RAM size.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful MBIST execution, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_MBIST_RAM_RANGE_INVALID, ERR_LOG_CODE_MBIST_FAILED, ERR_LOG_CODE_MBIST_TIMEOUT

Remarks

Customer needs to pay attention: the BootROM stack pointer must not get destroyed.

6.8.31 user_nvm_clk_factor_set

Description

This user API function sets the SCU_SYSCON0.NVMCLKFAC divider

Prototype

```
void user_nvm_clk_factor_set (
  uint8_t clk_factor
)
```

Parameters

Data Type	Name	Description	Dir
uint8_t	clk_factor	value is shifted and set to the corresponding bit fields of the register. All the other bit fields are not touched. No checks are done on the value. It is the responsibility of the user to know the range based on device technical data sheet.	-

6.8.32 user_vbg_temperature_get

Description

This user API function returns the V bandgap temperature hot or cold. The customer can do temperature compensation in software based on these values, as the VBG is the reference for the ADC.

Prototype

```
int32_t user_vbg_temperature_get (
  VBG_TEMP_SELECT_t temp_select
  uint32_t * temperature
)
```

Parameters

Data Type	Name	Description	Dir
VBG_TEMP_SELECT_t	temp_select	Selects the temperature to read	-
uint32_t *	temperature	Pointer where to store the temperature value read from the device configuration sector. Pointer must be located in RAM.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been called successfully, otherwise a negative error code. Returned error code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID, ERR_LOG_CODE_USERAPI_INVALID_VBG_TEMP_SELECT

6.8.33 user_nvm_page_verify

Description

This function reads the physical page content into the NVM assembly buffer using Normal Read Margins. The content of the assembly buffer is then used to check the physical page content by using the hardread margins erase and written.

This function rejects with an error in case the accessed NVM page belongs to a write protected region.

Prototype

```
int32_t user_nvm_page_verify (
  uint32_t address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	Address of the NVM page to check. Non-aligned address is accepted. Range is 0x11000000 + device NVM size.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful verify operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_NVM_PAGE_NOT_MAPPED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPBLOCK_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.34 user_nvm_page_erase_verify

Description

This function verifies with HardRead Erased margin on a page to check that all bits are erased. For linear region the check is done on the page pointed by the address provided as input. In case the routine target a mapped page, the check is performed on the current spare page.

This function rejects with an error in case the accessed NVM page belongs to a write protected region.

Prototype

```
int32_t user_nvm_page_erase_verify (
  uint32_t address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	Address of the NVM page to check. Non-aligned address is accepted. Range is 0x11000000 + device NVM size.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful verify operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_NVM_PAGE_IS_MAPPED, ERR_LOG_CODE_NVM_SPARE_PAGE_IS_NOT_MAPPED, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.35 user_nvm_sector_erase_verify

Description

This function performs a page-by-page erase check for a full sector. Each page is checked against Hardread-Margin-Erased.

This function rejects with an error in case the accessed NVM page belongs to a write protected region.

Prototype

```
int32_t user_nvm_sector_erase_verify (
  uint32_t address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	Address of the NVM sector to check. Non-aligned address is accepted, as the used sector range will be the sector where the address belongs. Range is 0x11000000 + device NVM size.	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case of successful verify operation, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID, ERR_LOG_CODE_NVM_PAGE_IS_MAPPED, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.8.36 user_dflash_mode

Description

This function returns the current configured NVM data region mode.

Prototype

```
NVM_DFLASH_SECTOR_MODE_t user_dflash_mode (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
NVM_DFLASH_SECTOR_MODE_t	Any of the data sector types defined in

6.9 User API support routines

These routines are only to be used by the user API as support functions.

Table 6-3 User API support routines Function Overview

Name	Description
get_nac_from_nvm_cs	This user API support function gets the BSL NAC value from NVM CS.
handle_segment_protection_get	This user API support function checks the protection for an address to NVM. Address must point into valid NVM area as address it not checked beforehand..
misc_handle_nvm_segment_data_mode_check	This function checks the current configured NVM data mode configuration against the provided segment. NVM data mode has to match or an error is returned.
misc_nvm_reflash_prepare	This function will try to erase a complete segment and password + its lower priority segments and related passwords. All segments are supported. The NVM data mode is checked and an error is returned in case a wrong NVM data segment is selected. Only if the selected segments are sucessfully erased, then the passwords will also be erased. Basically the function will exit at the first failure.
misc_user_nvm_password_set	This function sets a read and/or write protection for any NVM region individually. The API does not change the protection state for a region where password protection is currently installed.
misc_user_nvm_switch_key_set	This user API function updates the specified switch key in NVM config sector. Note that the setting in CS_SWITCH_KEY_CTRL_EN must be configured correctly to enable the use of the function. The specified key value must match the specified one or it will fail.
misc_user_read_nvm_password_ecc	This user API support function reads the specified segment password and checks for ECC2 errors.
valid_pointer_ram_range_check	This user API support function checks for valid pointer range. It must always point into valid RAM area. Although the valid address/pointer range for 4KB RAM is from 0x18000000 to 0x18000FFF, this routine actually checks (start_addr + length), where length is the number of bytes that can be programmed. So the valid range is 0x18000000 to 0x18001000. Length must be > 0. If the real address/pointer range shall be checked, start_addr and end_addr+1 must be provided.

6.9.1 misc_handle_nvm_segment_data_mode_check

Description

This function checks the current configured NVM data mode configuration against the provided segment. NVM data mode has to match or an error is returned.

Prototype

```
int32_t misc_handle_nvm_segment_data_mode_check (
```

```
  NVM_PASSWORD_SEGMENT_t segment
)
```

Parameters

Data Type	Name	Description	Dir
NVM_PASSWORD_SEGMENT_t	segment	Segment to check the NVM configuration against	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the password could be successfully applied, otherwise a negative error code.

6.9.2 misc_nvm_reflash_prepare

Description

This function will try to erase a complete segment and password + its lower priority segments and related passwords. All segments are supported. The NVM data mode is checked and an error is returned in case a wrong NVM data segment is selected. Only if the selected segments are sucessfully erased, then the passwords will also be erased. Basically the function will exit at the first failure.

The function rejects with an error in case a wrong password is provided vs the one installed for the chosen segment or if no password is installed. Segments and passwords will not be erased for the segments where no password is installed.

Prototype

```
int32_t misc_nvm_reflash_prepare (
  uint32_t password
  NVM_PASSWORD_SEGMENT_t segment
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	password	Current active password for the segment. A valid password parameter consists of a 30-bit password (bits 0...29), bits 30 and 31 are ignored.	-
NVM_PASSWORD_SEGMENT_t	segment	Segment where password should be cleared	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the password could be successfully applied, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_NVM_SEGMENT_INVALID ERR_LOG_CODE_NO_PASSWORD_EXISTS, ERR_LOG_CODE_USER_PROTECT_WRONG_PASSWORD, ERR_LOG_CODE_NVM_INIT_MAPRAM_SECTOR ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_USER_NVM_SEGMENT_CONFIG_MISMATCH

6.9.3 misc_user_nvm_password_set

Description

This function sets a read and/or write protection for any NVM region individually. The API does not change the protection state for a region where password protection is currently installed.

The password parameter consists of a 30-bit password (bit 0...29) and two additional protection bits (bit 30 + bit 31).

A valid password must be different from 0x3FFFFFFF and 0x00000000 (bit 0...29). The two MS bits in the password contain the protection type, where setting bit 31 activates the read protection and setting bit 30 activates the write protection. A non-compliant password is rejected.

A password can only be applied in case no valid password is currently set for the requested region.

Before updating starts, all interrupts including NMI are temporarily disabled and any NVM and NVM CS protection is disabled.

If selected data segment doesn't match the NVM data segment configuration a mismatch error is returned.

Prototype

```
int32_t misc_user_nvm_password_set (
  uint32_t password
  NVM_PASSWORD_SEGMENT_t segment
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	password	Protection password to apply on the given segment	-
NVM_PASSWORD_SEGMENT_t	segment	Segment which should be password protected	-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the password could be successfully applied, otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USER_NVM_SEGMENT_INVALID, ERR_LOG_CODE_USER_PROTECT_PWD_INVALID, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR, ERR_LOG_CODE_NVM_ECC2_DATA_ERROR, ERR_LOG_CODE_NVM_VER_ERROR, ERR_LOG_CODE_USER_PROTECT_PWD_EXISTS, ERR_LOG_CODE_USER_NVM_SEGMENT_CONFIG_MISMATCH

Remarks

It is not allowed to be called by NVM callback routines or any interrupt or multi-threaded environment in a re-entrant context.

6.9.4 misc_user_nvm_switch_key_set

Description

This user API function updates the specified switch key in NVM config sector. Note that the setting in CS_SWITCH_KEY_CTRL_EN must be configured correctly to enable the use of the function. The specified key value must match the specified one or it will fail.

Prototype

```
int32_t misc_user_nvm_switch_key_set (
  NVM_SWITCH_ID_SELECT_t switch_id
  NVM_SWITCH_KEY_SELECT_t key_select
  uint32_t key_value
)
```

Parameters

Data Type	Name	Description	Dir
NVM_SWITCH_ID_SELECT_t	switch_id		-
NVM_SWITCH_KEY_SELECT_t	key_select		-
uint32_t	key_value		-

Return Values

Data Type	Description
int32_t	ERR_LOG_SUCCESS in case the function has been called successfully and written the key value to the selected key. Otherwise a negative error code. Returned status code can be one of the following: ERR_LOG_SUCCESS, ERR_LOG_CODE_USERAPI_CONFIG_SECTOR_WRITE_PROTECTED, ERR_LOG_CODE_SWITCH_KEY_INVALID, ERR_LOG_CODE_USERAPI_SWITCH_KEY_VALUE_OR_USAGE_INVALID, ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED, ERR_LOG_CODE_ACCESS_AB_MODE_ERROR

6.9.5 handle_segment_protection_get

Description

This user API support function checks the protection for an address to NVM. Address must point into valid NVM area as address it not checked beforehand..

Prototype

```
uint32_t handle_segment_protection_get (
  uint32_t address
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	address	Address where in NVM to check which protection that is enabled for the NVM segment.	-

Return Values

Data Type	Description
uint32_t	Protection type for the segment (read, write, none). It follows the normal password format

6.9.6 valid_pointer_ram_range_check

Description

This user API support function checks for valid pointer range. It must always point into valid RAM area. Although the valid address/pointer range for 4KB RAM is from 0x18000000 to 0x18000FFF, this routine actually checks (start_addr + length), where length is the number of bytes that can be programmed. So the valid range is 0x18000000 to 0x18001000. Length must be > 0. If the real address/pointer range shall be checked, start_addr and end_addr+1 must be provided.

Prototype

```
int32_t valid_pointer_ram_range_check (
  uint32_t ptr_address
  uint32_t length
)
```

Parameters

Data Type	Name	Description	Dir
uint32_t	ptr_address	Start address of RAM range to check.	-
uint32_t	length	Function Length of the RAM range to check	-

Return Values

Data Type	Description
int32_t	Negative error if range is not valid or length is zero, otherwise ERR_LOG_SUCCESS

6.9.7 get_nac_from_nvm_cs

Description

This user API support function gets the BSL NAC value from NVM CS.

Prototype

```
uint8_t get_nac_from_nvm_cs (void)
```

Parameters

```
void
```

Return Values

Data Type	Description
uint8_t	The NAC value found in NVM CS

6.9.8 misc_user_read_nvm_password_ecc

Description

This user API support function reads the specified segment password and checks for ECC2 errors.

Prototype

```
int32_t misc_user_read_nvm_password_ecc (
  NVM_PASSWORD_SEGMENT_t segment
  uint32_t * read_nvm_password
)
```

Parameters

Data Type	Name	Description	Dir
NVM_PASSWORD_SEGMENT_t	segment	The segment from where the password should be read	-
uint32_t *	read_nvm_password	Pointer to a location where to store the read password	-

Return Values

Data Type	Description
int32_t	Negative error if ECC2 error happened, otherwise ERR_LOG_SUCCESS.

6.10 NVM Protection API types

6.10.1 user_callback_t

Description

User NVM callback function

Prototype

```
typedef void(* user_callback_t) (void)
```

6.11 Data Types and Structure Reference

This chapter contains the reference of data types and structures of all modules.

6.11.1 Enumerator Reference

This chapter contains the Enumerator reference.

Table 6-4 Enumerator Overview

Name	Description
NVM_SWITCH_ID_SELECT_t	NVM switch ID selection
NVM_SWITCH_KEY_SELECT_t	NVM switch keys selection
NVM_PASSWORD_SEGMENT_t	NVM protection API password segment

Table 6-4 Enumerator Overview (cont'd)

Name	Description
VBG_TEMP_SELECT_t	VBG Temperature selection.
NVM_DFLASH_SECTOR_MODE_t	User API NVM data region mode

6.11.1.1 NVM_SWITCH_ID_SELECT_t

Description

NVM switch ID selection

Prototype

```
typedef enum
{
  STARTUP_RAM_MBIST_RANGE_ID,
  NVM_DATA_LINEAR_ID,
  NVM_DATA_LINEAR_AND_RAM_MBIST_RANGE_ID,
}NVM_SWITCH_ID_SELECT_t;
```

Parameters

Name	Value	Description
STARTUP_RAM_MBIST_RANGE_ID		RAM MBIST reduced range ID selected
NVM_DATA_LINEAR_ID		NVM mapped/linear data sector switch ID selected
NVM_DATA_LINEAR_AND_RAM_MBIST_RANGE_ID		Both NVM data linear and RAM MBIST keys selected

6.11.1.2 NVM_SWITCH_KEY_SELECT_t

Description

NVM switch keys selection

Prototype

```
typedef enum
{
  SWITCH_KEY_1,
  SWITCH_KEY_2,
  SWITCH_KEY_3,
}NVM_SWITCH_KEY_SELECT_t;
```

Parameters

Name	Value	Description
SWITCH_KEY_1		Switch key 1 selected
SWITCH_KEY_2		Switch key 2 selected
SWITCH_KEY_3		Switch key 3 selected

6.11.1.3 NVM_PASSWORD_SEGMENT_t

Description

NVM protection API password segment

Prototype

```
typedef enum
{
  NVM_PASSWORD_SEGMENT_BOOT,
  NVM_PASSWORD_SEGMENT_CODE,
  NVM_PASSWORD_SEGMENT_DATA,
  NVM_PASSWORD_SEGMENT_DATA_LINEAR,
  NVM_PASSWORD_SEGMENT_LAST,
}NVM_PASSWORD_SEGMENT_t;
```

Parameters

Name	Value	Description
NVM_PASSWORD_SEGMENT_BOOT		NVM password for customer segment, used for customer bootloader
NVM_PASSWORD_SEGMENT_CODE		NVM password for customer code segment, which is not used by the customer bootloader.
NVM_PASSWORD_SEGMENT_DATA		NVM password for customer data mapped segment.
NVM_PASSWORD_SEGMENT_DATA_LINEAR		NVM password for customer data linear segment.
NVM_PASSWORD_SEGMENT_LAST		Can be ignored and should not be used

6.11.1.4 VBG_TEMP_SELECT_t

Description

VBG Temperature selection.

Prototype

```
typedef enum
{
  VBG_TEMP_HOT,
```

```
  VBG_TEMP_COLD,
}VBG_TEMP_SELECT_t;
```

Parameters

Name	Value	Description
VBG_TEMP_HOT		VBG Temperature selection Hot
VBG_TEMP_COLD		VBG Temperature selection Cold

6.11.1.5 NVM_DFLASH_SECTOR_MODE_t

Description

User API NVM data region mode

Prototype

```
typedef enum
{
  NVM_MAPPED_DATA_SECTOR_MODE,
  NVM_LINEAR_DATA_SECTOR_MODE,
  NVM_BAD_DATA_SECTOR_MODE,
}NVM_DFLASH_SECTOR_MODE_t;
```

Parameters

Name	Value	Description
NVM_MAPPED_DATA_SECTOR_MODE		NVM is in mapped data sector mode.
NVM_LINEAR_DATA_SECTOR_MODE		NVM is in linear data sector mode.
NVM_BAD_DATA_SECTOR_MODE		Bad NVM data sector configuration.

6.11.2 Constant Reference

This chapter contains the Constant reference.

Table 6-5 Constant Overview

Name	Value	Description
NVM_PASSWORD_PROTECTION_NONE	0x00000000u	NVM protection API password protection status NVM segment no protection enabled
NVM_PASSWORD_PROTECTION_READ	0x80000000u	NVM segment read protection enabled
NVM_PASSWORD_PROTECTION_WRITE	0x40000000u	NVM segment write protection enabled
NVM_PROG_FLAG_NULL	0x00u	NVM programming options No options provided

Table 6-5 Constant Overview (cont'd)

Name	Value	Description
NVM_PROG_CORR_ACT	0x02u	Disturb handling and retry enabled (data mapped mode only)
NVM_PROG_NO_FAILPAGE_ERASE	0x04u	Erasing of programmed data on fail enabled (data linear mode only)

Terminology

#

100-Time Programming (100TP)	The BootROM offers eight 100-time programmable pages to the user mode software. The size of a 100TP page is 128 bytes. The last two bytes of each 100TP page store the programming counter, followed by the page checksum byte.

A

AB	Assembly Buffer
API	Application Programming Interface

B

BootROM	Device-internal ROM code that the CPU executes directly after reset release
BSL	Boot Strap Loader
BSL command message	The BootROM receives these messages via FastLIN. A message of this kind contains commands and related data. A complete command could consist of multiple messages. The BootROM processes and execute these commands.
BSL response message	The BootROM replies to BSL command messages by BSL response messages. A response message contains requested data or an error code. The response message format and content depend on the given command.

C

CS	Configuration sector, see also NVM CS

D

Data block	Part of the BSL command message. This block follows a header block for data download commands.A data block could also be part of a BSL response message if the header block message requests read-out of some data from the device. The last data block is always followed by an EOT block.

E

EOT block	End of Transmission (EOT) block, part of the BSL command message or BSL response message. This block follows a data block to terminate a larger data download message.

F

FastLIN	FastLIN is a LIN enhancement supporting higher baud rates of up 230.4 kBd. This rate is higher than the standard LIN. This mode is especially useful during back-end programming, where faster programming time is desirable.
f_{INTOSC}	internal oscillator (80MHz)

G

H

HAL	Hardware Abstraction Layer. This software layer abstracts all module-specific hardware registers by API functions. It performs all device hardware register (SFR) accesses, which includes timing of critical register accesses and polling mechanisms. This layer exports its functionality to other software modules by means of API functions.

Header block	Part of the BSL command message. The host initiates a command by sending the header block. Some commands require further data transmission, during which the header block is followed by one or multiple data blocks and a terminating EOT block.
Host	The host communicates with the BootROM device over the LIN interface. The host sends BSL command messages and receives BSL response messages.

I

J

K

L

LIN	Local Interconnect Network

M

MBIST	Memory Built-In Self-Test (MBIST writes and reads all locations of the RAM to ensure that its cells are operating correctly)

N

NAC	No Activity Counter (millisecond timeout counter polling BSL LIN before jumping to user mode code execution)
NAD	Node Address for Diagnostics (LIN protocol parameter)
NVM	Non-Volatile Memory (device-internal)
NVM CS	NVM configuration sector. The BootROM uses one NVM sector to store device-specific calibration and trimming values. It configures such values on the device during startup. This sector also contains the One Time programmable and 100 Time programmable sectors, which are offered to the user mode software. The configuration sector is not directly accessible by user mode software.

O

OSC	Oscillator

P

POR	Power-On Reset
PLL	Phase-Locked Loop

Q

R

Response block	Part of the BSL response message, in which the corresponding BSL command message does not request read-out of data. This block reports the command execution status.
ROM	Read Only Memory

S

SA	Service Algorithm
SCU	System Control Unit

SFR	Special Function Register (CPU memory-mapped device hardware registers)
SWD	Serial Wire Debug
T	
Tearing Safe Programming	The mapping mechanism of the NVM module is intended to be used like a log-structured file system: When a page is programmed in the cell array, the old values are not physically overwritten, but a different physical page (the spare page) in the same sector is programmed in fact. If the programming fails (e.g. because of power loss during the erase or write procedure), either the old values are still present in the cell array or the verified new values are present in the cell array. The firmware therefore can program a single page in a tearing safe way.
U	
User mode code	Customer application code for download and execution in NVM.
UART	Universal asynchronous receiver/transmitter
V	
VTOR	Vector Table Offset Register
W	
WDT	WatchDog Timer

Appendix A Error Codes

This chapter provides a table that lists all available error codes.

Table A-1 List of Possible Errors during Startup

Error Name	Error Code	Errors Description
ERR_LOG_ERROR	-1_D	Standard Error
ERR_LOG_CODE_WRONG_BSL_MEDIA_TYPE	-2_D	Mismatch in configured BSL protocol media type and received media type in media driver
ERR_LOG_CODE_NVM_SEGMENT_READ_PROTECTED	-3_D	Trying to read from a NVM segment that is read protected
ERR_LOG_CODE_MEM_READWRITE_PARAMS_INVALID	-4_D	Invalid parameters to RAM/NVM/NVM_CS read/write command
ERR_LOG_CODE_BSL_RECV_BYTES_MISMATCH	-5_D	Mismatch in received number of bytes for the BSL message
ERR_LOG_CODE_NVM_IS_READ_PROTECTED	-6_D	BSL message is not allowed access when NVM is read protected on any region
ERR_LOG_CODE_NVM_ERASE_PARAMS_INVALID	-7_D	Invalid BSL parameters to NVM erase command
ERR_LOG_CODE_INVALID_CUSTOMER_CONFIG_CBSL_SIZE	-8_D	Specified customer configured CBSL size is invalid
ERR_LOG_CODE_CUSTOMER_CONFIG_CBSL_PROGRAMMED_OR_READ_ERR	-9_D	Customer configured CBSL size already programmed or read error
ERR_LOG_CODE_NVM_CODE_PROGRAMMED	-10_D	Customer configured CBSL size can't be changed when code is present in NVM
ERR_LOG_CODE_BSL_NVM_CALC_CHECKSUM_MISMATCH	-11_D	BSL: Calculated NVM checksum does not match
ERR_LOG_CODE_FASTLIN_BAUDRATE_SET_FAIL	-12_D	Invalid FastLIN baudrate parameter or current BSL interface is not FASTLIN
ERR_LOG_CODE_BSL_USER_MODE_PATCH_NOT_ALLOWED	-13_D	Patch ID exceeded. Not allowed to execute bootROM patch in user mode
ERR_LOG_CODE_BSL_PATCH_ID_EXCEEDED	-14_D	Max number of patch IDs exceeded
ERR_LOG_CODE_SFR_WRITE_PARAMS_INVALID	-15_D	Invalid SFR write parameters
ERR_LOG_CODE_BSL_NVM_UNLOCK_PASSWD_INVALID	-16_D	Invalid NVM unlock password received
ERR_LOG_CODE_MSG_VALIDITY_FAIL	-17_D	BSL message validity failed
ERR_LOG_CODE_USER_PATCH_INVALID_ADDRESS	-18_D	Patch execution: Patch address is not within NVM CS, NVM or RAM range or not programmed
ERR_LOG_CODE_TEST_HTOL_FBI_MATH_FAIL	-19_D	Factory test HTOL functional burn-in math test failed

Table A-1 List of Possible Errors during Startup (cont'd)

Error Name	Error Code	Errors Description
ERR_LOG_CODE_TEST_HTOL_FBI_MDU_FAIL	-20_D	Factory test HTOL functional burn-in MDU test failed
ERR_LOG_CODE_TEST_HTOL_FBI_RAM_FAIL	-21_D	Factory test HTOL functional burn-in XRAM verification failed
ERR_LOG_CODE_FTEST_BOOTROM_SIGNATURE_READ	-22_D	Factory test BootROM signature read error
ERR_LOG_CODE_TEST_HTOL_PWR_LOW_TEST_FAIL	-23_D	Factory test power module test, P0.1 was not low
ERR_LOG_CODE_TEST_HTOL_PWR_HIGH_TEST_FAIL	-24_D	Factory test power module test, P0.1 was not high
ERR_LOG_CODE_NVM_WRITE_FAST_WRONG_MODE	-25_D	NVM is not in fast write mode
ERR_LOG_CODE_NVM_WRITE_FAST_SEMAPHORE_RESERVED	-26_D	{
ERR_LOG_CODE_MEM_ADDR_RANGE_INVALID	-27_D	Memory address range is invalid
ERR_LOG_CODE_NVM_SEMAPHORE_RESERVED	-28_D	NVM semaphore already reserved
ERR_LOG_CODE_ECC1_READ_ERROR	-29_D	ECC1READ error happened
ERR_LOG_CODE_NVM_PAGE_IS_MAPPED	-30_D	NVM erase page/sector verify: Page is mapped / not erased
ERR_LOG_CODE_NVM_SPARE_PAGE_IS_NOT_MAPPED	-31_D	NVM erase page verify: Spare page is not mapped
ERR_LOG_CODE_ECC2READ_ERROR	-32_D	ECC2READ error generated when reading NVM data
ERR_LOG_CODE_NVM_FAST_PROG_NOT_ALLOWED	-33_D	Fast programming option not allowed for used NVM page
ERR_LOG_CODE_NVM_VER_ERROR	-34_D	2 or more bit errors detected in NVM page when verifying NVM data
ERR_LOG_CODE_NVM_PROG_MAPRAM_INIT_FAIL	-35_D	NVM mapRAM update failed after mapped page programming or after execution of DH
ERR_LOG_CODE_NVM_PROG_VERIFY_MAPRAM_INIT_FAIL	-36_D	NVM programming and mapRAM init update failed after mapped page programming
ERR_LOG_CODE_NVM_ECC2_MAPBLOCK_ERROR	-37_D	ECC2 mapBlock error generated while reading from a NVM page
ERR_LOG_CODE_NVM_MAPRAM_UNKNOWN_TYPE_USAGE	-38_D	MAPRAM physical page number for a given logical sector/page is larger than the number of physical pages in a sector
ERR_LOG_CODE_NVM_ECC2_MAPRAM_ERROR	-39_D	ECC2 mapRAM error generated while reading mapRAM

Table A-1 List of Possible Errors during Startup (cont'd)

Error Name	Error Code	Errors Description
ERR_LOG_CODE_NVM_PAGE_NOT_MAPPED	-40_D	NVM page is not mapped
ERR_LOG_CODE_NVM_INIT_MAPRAM_SECTOR	-41_D	Mapped page has double mapping or ECC2 error when trying to init mapRAM
ERR_LOG_CODE_NVM_MAPRAM_MANUAL_SPARE_PAGE_FAILED	-42_D	NVM manual spare page selection failed as part of mapRAM update
ERR_LOG_CODE_NVM_WRITE_FAST_REACH_MAX_RETRIES	-43_D	Reach maximum tries
ERR_LOG_CODE_ACCESS_AB_MODE_ERROR	-44_D	Error when setting the assembly buffer to mode 1 or 2
ERR_LOG_CODE_NVM_ECC2_DATA_ERROR	-45_D	ECC2 data error generated while reading from a NVM page
ERR_LOG_CODE_ADDRESS_RANGE_CROSSING_PAGE_BOUNDARY	-46_D	Attempt to access NVM address range which is crossing NVM page boundary
ERR_LOG_CODE_100TP_WRITE_COUNT_EXCEEDED	-47_D	NVM 100TP page write count was exceeded
ERR_LOG_CODE_100TP_PAGE_INVALID	-48_D	Attempt to access NVM 100TP page address outside of the valid range
ERR_LOG_CODE_NVM_100TP_PAGE_CNT_ERROR	-49_D	NVM 100TP page write counter contains ECC2 error. Value hardcoded according to documentation
ERR_LOG_CODE_CS_PAGE_CHECKSUM	-50_D	NVM config sector checksum calculation failed
ERR_LOG_CODE_CS_PAGE_ECC2READ	-51_D	NVM config sector checksum calculation failure based on NVM ECC2 error
ERR_LOG_CODE_ANA_TRIM_ADDRESS	-52_D	Analog trimming address check failed
ERR_LOG_CODE_ANA_TRIM_MAGIC	-53_D	Analog trimming data block contains wrong magic field
ERR_LOG_CODE_ANA_TRIM_NOT_ALIGNED	-54_D	Analog trimming data block is not 32-bit aligned
ERR_LOG_CODE_AM_TRIM_INTERNAL_1	-55_D	First phase internal analog module trimming failed
ERR_LOG_CODE_AM_TRIM_INTERNAL_2	-56_D	Second phase internal analog module trimming failed
ERR_LOG_CODE_AM_TRIM_CUSTOMER	-57_D	Customer analog module trimming phase failed
ERR_LOG_CODE_AM_TRIM_DATA_NOT_VALID	-58_D	Analog module trimming data validity check failed
ERR_LOG_CODE_ARM_VECT_HARDFAULT_EXCEP	-59_D	ARM hard fault exception was triggered

Table A-1 List of Possible Errors during Startup (cont'd)

Error Name	Error Code	Errors Description
ERR_LOG_CODE_NVM_APPLY_PROTECTION_FAIL	-60_D	Applying of NVM protection from NVM CS failed during bootup
ERR_LOG_CODE_VIRGIN	-61_D	Device is detected as virgin during startup
ERR_LOG_CODE_NVM_NOT_AVAILABLE	-62_D	NVM HW is not available during startup
ERR_LOG_CODE_MBIST_FAILED	-63_D	MBIST test detected an error
ERR_LOG_CODE_MBIST_TIMEOUT	-64_D	MBIST test failed due to timeout
ERR_LOG_CODE_USER_INVALID_NVM_PAGE_NUMBER	-65_D	Invalid NVM page number
ERR_LOG_CODE_USER_INVALID_VBG_TEMP_SELECT	-66_D	user_vbg_temperature_get(): Wrong selection of vbg temperature
ERR_LOG_CODE_NAC_VALUE_INVALID	-67_D	NAC value out of range in user_nac_set()
ERR_LOG_CODE_MBIST_RAM_RANGE_INVALID	-68_D	user_ram_mbist() RAM range for MBIST is invalid
ERR_LOG_CODE_PARAM_LENGTH	-69_D	user_version_read() the provided length parameter is to small to store the information.
ERR_LOG_CODE_USER_PATCH_ID_OUT_OF_RANGE	-70_D	user_execute_patch() The patch ID is out of range
ERR_LOG_CODE_USER_NO_NVM_MAPPED_SECTOR	-71_D	NVM has no configured mapped sectors
ERR_LOG_CODE_USER_NVM_SEGMENT_INVALID	-72_D	Provided NVM segment is invalid
ERR_LOG_CODE_SINGLE_ECC_EVENT_OCCURRED	-73_D	user_ecc_events_get()/user_ecc_get() single ECC event has occurred
ERR_LOG_CODE_DOUBLE_ECC_EVENT_OCCURRED	-74_D	user_ecc_events_get()/user_ecc_get() double ECC event has occurred
ERR_LOG_CODE_SINGLE_AND_DOUBLE_ECC_EVENT_OCCURRED	-75_D	user_ecc_events_get()/user_ecc_get() single and double ECC events have occurred
ERR_LOG_CODE_PARAM_INVALID	-76_D	user_nvm_write/branch(): data parameter is invalid
ERR_LOG_CODE_USER_PROTECT_NVM_WRITE_PROTECTED	-77_D	Operation not allowed when NVM is write protected
ERR_LOG_CODE_USER_POINTER_RAM_RANGE_INVALID	-78_D	Provided pointer doesn't point to a valid RAM range
ERR_LOG_CODE_USER_NVM_SEGMENT_CONFIG_MISMATCH	-79_D	NVM segment does not match the current configured NVM region type
ERR_LOG_CODE_NVM_IS_BUSY	-80_D	NVM is busy doing another operation
ERR_LOG_CODE_USER_PROTECT_PWD_INVALID	-81_D	user_protect_password_set() provided password not valid

Table A-1 List of Possible Errors during Startup (cont'd)

Error Name	Error Code	Errors Description
ERR_LOG_CODE_USER_PROTECT_PWD_EXISTS	-82_D	nvm_protect_password_set() segment password already exists when trying to set a new one in
ERR_LOG_CODE_NO_PASSWORD_EXISTS	-83_D	Password clear: No password installed when trying to clear password
ERR_LOG_CODE_USER_PROTECT_WRONG_PASSWORD	-84_D	user_nvm_protect_set(), user_nvm_protect_clear(): wrong password given
ERR_LOG_CODE_NVM_CONFIG_SECTOR_WRITE_PROTECTED	-85_D	Not allowed to change values in write protected config sector
ERR_LOG_CODE_SWITCH_KEY_INVALID	-86_D	User_nvm_switch_key_set() NVM switch key ID is invalid
ERR_LOG_CODE_USERAPI_SWITCH_KEY_VALUE_OR_USAGE_INVALID	-87_D	User_nvm_switch_key_set() NVM switch key value is invalid, API usage not allowed or selected key invalid

Appendix B Stack usage of user API functions

This chapter provides a table that lists maximum used stack for each user API function.

Table B-1 Maximum used stack for user API functions

User API function	Maximum stack usage (bytes)
user_nvm_write_fast_continue	72_D
user_nvm_write_fast_verify	64_D
user_nvm_write_fast_end	88_D
user_adc1_offset_calibration	64_D
user_nvm_page_checksum_check	44_D
user_nvm_mapram_recover	48_D
user_nvm_mapram_init	40_D
user_cid_get	24_D
user_vbg_temperature_get	32_D
user_nvm_ecc_events_get	80_D
user_nvm_ecc_check	80_D
user_nac_get	32_D
user_nac_set	112_D
user_nad_get	24_D
user_nad_set	112_D
user_nvm_100tp_read	64_D
user_nvm_100tp_write	136_D
user_nvm_100tp_erase	88_D
user_nvm_config_get	48_D
user_nvm_protect_get	16_D
user_nvm_protect_set	80_D
user_nvm_protect_clear	80_D
user_nvm_password_set	152_D
user_nvm_page_erase	120_D
user_nvm_page_erase_branch	120_D
user_nvm_sector_erase	120_D
user_nvm_write_fast_start	240_D
user_nvm_write	248_D
user_nvm_write_branch	248_D
user_ram_mbist	40_D
user_nvm_page_verify	96_D
user_nvm_page_erase_verify	96_D
user_nvm_sector_erase_verify	96_D

Table B-1 Maximum used stack for user API functions (cont'd)

User API function	Maximum stack usage (bytes)
user_dflash_mode	8_D
user_nvm_service_algorithm	224_D
user_nvm_ready_poll	0_D
user_nvm_clk_factor_set	0_D

Appendix C Exported bootROM functions

This chapter provides a table that lists all exported bootROM functions and its address that can be called by user code.

Table C-1 Exported user API functions

User API function	BootROM thumb address
user_nvm_protect_clear	0x000000a1
user_nvm_protect_set	0x000000a3
user_nvm_protect_get	0x000000a5
user_cid_get	0x000000a7
user_nvm_ecc_events_get	0x000000a9
user_nvm_ecc_check	0x000000ab
user_nac_get	0x000000ad
user_nac_set	0x000000af
user_nad_get	0x000000b1
user_nad_set	0x000000b3
user_nvm_100tp_read	0x000000b5
user_nvm_100tp_write	0x000000b7
user_nvm_config_get	0x000000b9
user_nvm_page_erase	0x000000bb
user_nvm_page_erase_branch	0x000000bd
user_nvm_ready_poll	0x000000bf
user_nvm_sector_erase	0x000000c1
user_nvm_write	0x000000c3
user_nvm_write_branch	0x000000c5
user_ram_mbist	0x000000c7
user_nvm_mapram_init	0x000000cb
user_nvm_clk_factor_set	0x000000cd
user_vbg_temperature_get	0x000000cf
user_nvm_page_verify	0x000000d3
user_nvm_page_erase_verify	0x000000d5
user_nvm_sector_erase_verify	0x000000d7
user_nvm_mapram_recover	0x000000d9
user_dflash_mode	0x000000db
user_nvm_service_algorithm	0x000000dd
user_nvm_page_checksum_check	0x000000df
user_nvm_100tp_erase	0x000000e1
user_adc1_offset_calibration	0x000000e3

Table C-1 Exported user API functions (cont'd)

User API function	BootROM thumb address
user_nvm_password_set	0x000000e5
user_nvm_write_fast_start	0x000000e7
user_nvm_write_fast_continue	0x000000e9
user_nvm_write_fast_verify	0x000000eb
user_nvm_write_fast_end	0x000000ed

Appendix D Analog Module Trimming (100TP Pages)

The TLE985x contains 8 x 100TP (100 Time Programmable) pages and each page has a size of 128 bytes but only the first 126 Bytes are usable. The last two Bytes of each 100TP page store the programming counter followed by the page checksum Byte.

User could read and write into the 100TP pages using the user API functions :

- user_nvm_100tp_read

- user_nvm_100tp_write

In case the checksum of any page is incorrect, the whole content of the 100TP pages is ignored and considered as unsafe. User could call user_nvm_page_checksum_check before performing 100TP read operation.

Each user_nvm_100tp_write page programming operation leads to a programming counter increase. user_nvm_100tp_write returns with an error in case the user tries to program one page where the counter has reached 100 programming cycles. Page programming counter range is from 0 - 99.

The first and second 100TP pages contain customer specific analog module trimming values.

Table D-1 100TP page 0 and page 1 : Analog Module Trimming registers

Data Offset	**100 TP Page 0** SFR Registers to TRIM	**100 TP Page 1** SFR Register to TRIM
0x00	SCU_ADC1_CLK	ADC1_SQ10_11
0x04	ADC1_CAL_CH12_13	ADC1_SQ8_9
0x08	ADC1_SQ12_13	ADC1_SQ6_7
0x0C	ADC1_PP_MAP4_7	ADC1_SQ4_5
0x10	ADC1_PP_MAP0_3	ADC1_SQ2_3
0x14	ADC1_IRQEN_2	ADC1_SQ0_1
0x18	ADC1_DUIN_SEL	ADC1_CTRL3
0x1C	ADC1_MMODE0_7	ADC1_MAX_TIME
0x20	ADC1_DCHCNT1_4_UPPER	ADC1_CHX_ESM
0x24	ADC1_CNT4_7_UPPER	ADC1_CHX_EIM
0x28	ADC1_CNT0_3_UPPER	
0x2C	ADC1_DCHCNT1_4_LOWER	
0x30	ADC1_CNT4_7_LOWER	
0x34	ADC1_CNT0_3_LOWER	
0x38	ADC1_DCHTH1_4_UPPER	
0x3C	ADC1_TH4_7_UPPER	
0x40	ADC1_TH0_3_UPPER	
0x44	ADC1_DCHTH1_4_LOWER	
0x48	ADC1_FILT_UPLO_CTRL	
0x4C	ADC1_IRQEN_1	

Table D-1 100TP page 0 and page 1 : Analog Module Trimming registers

0x50	ADC1_FILTCOEFF0_13	
0x54	ADC1_CAL_CH10_11	
0x58	ADC1_CAL_CH8_9	
0x5C	ADC1_CAL_CH6_7	
0x60	ADC1_CAL_CH4_5	
0x64	ADC1_CAL_CH2_3	
0x68	ADC1_CAL_CH0_1	
0x6C	ADC1_TH4_7_LOWER	
0x70	ADC1_TH0_3_LOWER	
0x74	ADC1_OFFSETCALIB	
0x78	ADC1_SQ_CH_MAP	

Example

Table D-2 100TP Analog Module Trimming example

Code example

```
#define PAGE0                   (uint32_t 0)
#define ADC1_CNT_UPPER_OFFSET (uint32_t 0x24)
#define ADC1_CNT_UPPER_LEN     (uint32_t sizeof(ADC1_TRIM_Data_table))

uint32_t ADC1_TRIM_Data_table[]= {0x11111111, 0x12121212};

int32_t User_Trimming_100TP(void)
{
 int32_t status=0;
 status = user_nvm_100tp_write(PAGE0,
                               ADC1_CNT_UPPER_OFFSET,
                               ADC1_TRIM_Data_table,
                               ADC1_CNT_UPPER_LEN);

 return status;
}
```

Description

This function write into the 100TP page 0 at the offset 0x0C, which correspond to the address allowed for ADC1_CNT4_7_UPPER. The length of the data to write is 8 bytes, which means 2 words : ADC1_CNT4_7_UPPER, then ADC1_CNT0_3_UPPER.
This function trims the following registers :
ADC1_CNT4_7_UPPER = 0x11111111
ADC1_CNT0_3_UPPER = 0x12121212

Table D-3 Alternative predefined values to trim in case 100TP page 0 and page 1 CRC is incorrect

100 TP Page 0		
Data Offset	**SFR registers**	**Alternative Back up values**

Table D-3 Alternative predefined values to trim in case 100TP page 0 and page 1 CRC is incorrect

0x00	SCU_ADC1_CLK	0x00000000
0x04	ADC1_CAL_CH12_13	0x00000000
0x08	ADC1_SQ12_13	0x00000000
0x0C	ADC1_PP_MAP4_7	0x08070604
0x10	ADC1_PP_MAP0_3	0x03020100
0x14	ADC1_IRQEN_2	0x00000000
0x18	ADC1_DUIN_SEL	0x00000000
0x1C	ADC1_MMODE0_7	0x00000000
0x20	ADC1_DCHCNT1_4_UPPER	0x00000000
0x24	ADC1_CNT4_7_UPPER	0x00000000
0x28	ADC1_CNT0_3_UPPER	0x00001B1A
0x2C	ADC1_DCHCNT1_4_LOWER	0x00000000
0x30	ADC1_CNT4_7_LOWER	0x00000000
0x34	ADC1_CNT0_3_LOWER	0x00001312
0x38	ADC1_DCHTH1_4_UPPER	0x000000FF
0x3C	ADC1_TH4_7_UPPER	0xFFFFFFFF
0x40	ADC1_TH0_3_UPPER	0xFFFFC5C0
0x44	ADC1_DCHTH1_4_LOWER	0x00000000
0x48	ADC1_FILT_UPLO_CTRL	0x000000FF
0x4C	ADC1_IRQEN_1	0x00000000
0x50	ADC1_FILTCOEFF0_13	0x0AAAAAAA
0x54	ADC1_CAL_CH10_11	0x00000000
0x58	ADC1_CAL_CH8_9	0x00000000
0x5C	ADC1_CAL_CH6_7	0x00000000
0x60	ADC1_CAL_CH4_5	0x00000000
0x64	ADC1_CAL_CH2_3	0x00000000
0x68	ADC1_CAL_CH0_1	0x00000000
0x6C	ADC1_TH4_7_LOWER	0x00000000
0x70	ADC1_TH0_3_LOWER	0x0000423A
0x74	ADC1_OFFSETCALIB	0x00000000
0x78	ADC1_SQ_CH_MAP	0x00000000
	100 TP Page 1	
0x00	ADC1_SQ10_11	0x00000000
0x04	ADC1_SQ8_9	0x00000000
0x08	ADC1_SQ6_7	0x00000000
0x0C	ADC1_SQ4_5	0x00000000
0x10	ADC1_SQ2_3	0x00000000

Table D-3 Alternative predefined values to trim in case 100TP page 0 and page 1 CRC is incorrect

0x14	ADC1_SQ0_1	0x00000000
0x18	ADC1_CTRL3	0x00020A01
0x1C	ADC1_MAX_TIME	0x00000000
0x20	ADC1_CHX_ESM	0x00000000
0x24	ADC1_CHX_EIM	0x00000000

Appendix E Execution time of BootROM User API Functions

This appendix provides a table that lists the execution time of the BootROM User API functions.

Table E-1 User API execution time

User API Function	Execution Time [1] [µs]
user_adc1_offset_calibration	28.8
user_dflash_mode	3.2
user_nac_get	4.8
user_nac_set	6816
user_nad_get	3.2
user_nad_set	6814.4
user_nvm_100tp_erase	6817.6
user_nvm_100tp_read	59.2
user_nvm_100tp_write	6884.8
user_nvm_clk_factor_set	1.6
user_nvm_config_get	9.6
user_nvm_ecc_check	3502.4
user_nvm_ecc_events_get	4.8
user_nvm_mapram_init	12.8
user_nvm_mapram_recover	56
user_nvm_page_checksum_check	14.4
user_nvm_page_erase	3660.8
user_nvm_page_erase_verify	120
user_nvm_page_verify	126.4
user_nvm_password_set	6884
user_nvm_protect_clear	12.8
user_nvm_protect_get	3,2
user_nvm_protect_set	14.4
user_nvm_sector_erase	3665.6
user_nvm_sector_erase_verify	1889.6
user_nvm_service_algorithm	1740.8
user_nvm_write, precondition: written_mapped_sector()	6921.6
user_nvm_write, precondition: erased_mapped_sector()	3328
user_nvm_write_fast_continue()	292
user_nvm_write_fast_end()	17
user_nvm_write_fast_start()	172
user_nvm_write_fast_verifiy()	316

Table E-1 User API execution time (cont'd)

User API Function	Execution Time [1] [µs]
user_ram_mbist (4k RAM)	180
user_vbg_temperature_get	3.2

1) Execution time relative to the following conditions : 40MHz, 25°C, 12V

www.infineon.com

Edition 2019-03-05

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office. Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.