MOTIX™ TLE986x
MOTIX™ TLE987x
BF, A, UH, and UI step

BootROM User manual

Rev. 1.9, 2023-10-23

1 Introduction . . . 5
1.1 Purpose . . . 5
1.2 Scope . . . 5
1.3 Abbreviations and special terms . . . 5

2 Overview . . . 7
2.1 Firmware architecture . . . 7
2.2 Program structure . . . 8

3 Startup procedure . . . 9
3.1 Program structure . . . 9
3.1.1 Test and initialization of RAM . . . 9
3.1.2 NVM initialisation routine . . . 10
3.1.3 NVM MapRAM initialisation . . . 10
3.1.3.1 NVM MapRAM initialization at high prio reset . . . 10
3.1.3.2 NVM MapRAM initialization at low prio reset . . . 11
3.1.4 Oscillator trimming and system clock selection . . . 11
3.1.5 Analog module trimming . . . 12
3.1.6 User configuration data initialization . . . 12
3.1.7 Debug Support mode entry . . . 13
3.1.8 User mode and BSL mode entry . . . 13
3.1.8.1 NAC definition . . . 13
3.1.9 Node Address for Diagnostic (NAD) . . . 16

4 FastLIN and UART BSL mode . . . 18
4.1 FastLIN and UART BSL protocol . . . 18
4.2 Phase I for UART BSL: Automatic serial synchronization to the host . . . 19
4.2.1 General description . . . 19
4.2.2 Calculation of BR_VALUE and PRE values . . . 21
4.3 Phase I for FastLIN BSL: FastLIN BSL entry sequence . . . 22
4.4 Phase II: Serial communication protocol and the working modes . . . 22
4.4.1 Serial communication protocol . . . 22
4.4.1.1 Transfer block structure . . . 23
4.4.1.2 Transfer block type . . . 23
4.4.1.3 Response codes to the host . . . 24
4.4.1.4 Block response delay . . . 27
4.4.2 UART BSL modes . . . 29
4.4.2.1 Header block . . . 29
4.4.2.2 Mode 0 - Code/Data download to RAM/100TP . . . 29
4.4.2.3 Mode 1 - Code execution inside RAM . . . 32
4.4.2.4 Mode 2 - Code/Data download to NVM . . . 32
4.4.2.5 Mode 3 - Code execution inside NVM . . . 37
4.4.2.6 Mode 4 - NVM erase . . . 37
4.4.2.7 Mode 6 - NVM protection . . . 39

4.4.2.8 Mode A - NVM readout, Chip ID, checksum, FastLIN BSL entry command 40
4.4.3 16 bits inverted XOR checksum . . . 46
4.5 WDT1 refreshing . . . 46

5 NVM . . . 47
5.1 NVM overview . . . 47
5.1.1 NVM organisation . . . 47
5.2 NVM configuration sectors organisation . . . 50
5.2.1 Chip ID definition . . . 50
5.2.2 100 Time Programmable data . . . 53
5.3 NVM user routines organisation . . . 61
5.3.1 Opening assembly buffer routine . . . 63
5.3.2 NVM programming routine . . . 66
5.3.3 Page verify routine . . . 70
5.3.4 NVM page erasing routine . . . 72
5.3.5 Erase page verify routine . . . 74
5.3.6 Sector erasing routine . . . 75
5.3.7 Erase sector verify routine . . . 76
5.3.8 Abort NVM programming routine . . . 78
5.3.9 MapRAM initialization . . . 78
5.3.10 Read NVM status routine . . . 80
5.3.11 Read 100 Time Programmable parameter data routine . . . 81
5.3.12 Program 100 Time Programmable routine . . . 82
5.3.13 NVM ECC check routines . . . 84
5.3.14 Read NVM ECC2 address routine . . . 86
5.3.15 RAM MBIST starting routine . . . 86
5.3.16 NVM protection status change routines . . . 88
5.3.17 Read NVM config status routine . . . 92
5.3.18 Read user calibration data . . . 92
5.3.19 NVMCLKFAC setting routine . . . 93
5.4 NVM user applications . . . 94
5.4.1 NVM Data sector handling . . . 94
5.4.2 Supporting background NVM operation . . . 101
5.4.3 Emergency operation handling . . . 104
5.4.3.1 Emergency operation handling - Type 1 routines . . . 104
5.4.3.2 Emergency operation handling - Type 2 routines . . . 105
5.4.3.3 Emergency operation handling timing . . . 105
5.4.4 NVM user routines operation . . . 107
5.4.4.1 NVM user programming operation . . . 107
5.4.4.2 Tearing-safe programming . . . 109
5.4.4.3 NVM user erase operation . . . 110
5.4.4.4 NVM user programming abort operation . . . 111

5.4.5 NVM protection mechanism 111
6 Revision history 112

1 Introduction

This document specifies the BootROM firmware behavior for the TLE986x/TLE987x family. The specification contains the following major sections:

- BootROM overview
- Startup procedure
- BSL features
- NVM structure and user routines description

1.1 Purpose

The document describes the functionality of the BootROM firmware.

1.2 Scope

The BootROM firmware for the TLE986x/TLE987x family provides the following features:

- Startup procedure for stable operation of TLE986x/TLE987x chip
- Debugger connection for code debug
- BSL mode for users to download and run code from NVM and RAM
- NVM operation handling, e.g. program and erase

1.3 Abbreviations and special terms

Table 1-1 Abbreviations and terms

BSL	BootStrap Loader
CS	Configuration Sector
EOT	End of Transmission
EVR	Embedded Voltage Regulator
NAC	No Activity Count
NAD	Node address for diagnostic
NEA	NVM End Address
NLS	NVM Linear Size
NSA	NVM Starting Address
NVM	Non Volatile Memory
OCDS	On-Chip Debug Support
OSC	Oscillator
PEM	Program Execution Mode

Table 1-1 Abbreviations and terms (cont'd)

PLL	Phase-Locked Loop
SA	Service Algorithm
SCU	System Control Unit
SWD	Serial Wire Debug
VTOR	Vector Table Offset Register
WDT	WatchDog Timer

2 Overview

This specification describes all firmware features including the operations and tasks defined to support the general startup behaviour and various boot options.

2.1 Firmware architecture

TLE986x/TLE987x on-chip BootROM consists of:

- Startup procedure, see **Chapter 3**
- Bootstrap loader via UART, see **Chapter 4**
- NVM user routines and NVM integrity handling routines, see **Chapter 5**

The BootROM in TLE986x/TLE987x is located at 00000000_H and so represents the standard reset handler routine.

The startup procedure includes:

- EVR calibration
- MapRAM initialisation
- On-chip oscillator configurations
- NVM protection enabling
- Branching to different modes

The latched values of TMS, P0.0 and P0.2 at the rising edge of RESET determine the mode of operation to be entered.

BootROM operation modes:

- User / BSL mode
- Debug Support mode

In User mode BootROM performs the following steps: execute the startup procedure, set the vector table position at the beginning of the NVM in user accessible space (by proper setting of the VTOR register) and jump to the user defined reset handler routine (jump to the location pointed by the address 11000004_H - 11000007_H) to execute the user program.

Note: The firmware will only set the VTOR to point at the beginning of the user accessible NVM region, but it will not write any vector table. It is a task of the user to download a correct vector table.

Table 2-1 lists the boot options available in the TLE986x/TLE987x.

Table 2-1 TLE986x/TLE987x boot options

TMS/SWD	P0.0	P0.2	Mode
0	x	x	User mode / BSL mode [1)2)]
1	1	0	Debug Support mode with Serial Wire (SW) port

1) On-chip OSC is selected as PLL input. System is running on LP_CLK until firmware switches to PLL output before jumping to user code. Exception is with hardware reset where user settings are retained.

2) Boot in User mode or BSL mode depends on the NAC word in user memory (NVM).

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 that is executed after any reset.

The startup procedure will perform configuration sector verification, EVR calibration, on-chip oscillator trimming, MapRAM initialisation, BootROM protection, NVM protection and decode the pin-latched values of the TMS, P0.0 and P0.2 to determine which mode it will jump to.

User mode

User mode supports user code execution in the NVM address space. However, if NVM is not protected and the bytes at address 11000004_H - 11000007_H are erased (FF_H), then device enters Sleep mode. If a valid user reset vector is found at 11000004_H (values at 11000004_H - 11000007_H not equal to $FFFFFFFF_H$) and a proper NAC value is found, then the BootROM proceeds into User mode. In case an invalid NAC value is found, the device waits forever for a FastLIN BSL communication.

FastLIN and UART BSL mode

It is used to support BSL via UART protocol. Downloading of code/data to RAM and NVM related programming is supported in this mode.

3 Startup procedure

This chapter describes the BootROM startup procedure in TLE986x/TLE987x.

The startup procedure is the first software-controlled operation in the BootROM that is automatically started after every reset. Certain operations are skipped depending on the type of reset. Refer to next section for further details.

3.1 Program structure

The first task executed by the startup firmware is the check of the reset source.

For power-on, brown-out reset or wake-up from Sleep mode reset, RAM test and initialization are executed according to user settings, while they are skipped for other reset types.

Firmware code uses part of the RAM for variable storage, literal pools and stack pointer. The startup code only uses a specific RAM region (the first 1 Kbyte mapped from address 18000000_H *to* $180003FF_H$*), subset of the total available RAM address range. In the remaining region, which is not used by the firmware, the user can store values to be valid across reset for all reset sources different from* power-on reset, brown-out reset and wake-up reset. *For these three reset sources, either a RAM test or a RAM clear might be executed thus destroying the previously stored values.*

After that, depending on the reset source, the firmware will do NVM protection, NVM MapRAM initialisation, on-chip oscillator trimming, PLL setting and analog module trimming. It will decode the pin-latched values of the TMS, P0.0 and P0.2 to determine which mode it will jump to.

If bootup mode is Debug Support mode, the WDT1 is disabled. For entry to User mode, the WDT1 remains active. Next, the firmware will wait for NVM module to be ready.

For software, or internal watchdog reset (triggered by the WDT in the SCU), the following steps are skipped:

- RAM test and initialisation
- NVM MapRAM initialisation and service algorithm
- Setting of oscillator and PLL and switching system clock input to PLL output
- Loading of analog modules trimming parameters from first 100TP page
- Loading of user configuration data from 100TP page into the RAM
- Clearing of NMI status before exit to User mode or Debug Support mode

3.1.1 Test and initialization of RAM

A functional test sequence is executed on the entire RAM after power-on reset and brown-out reset. This can be executed optionally after a wake-up reset. The test consists of a linear write/read algorithm using alternating data. Once it is started, the firmware waits until the test is completed, before checking the result and continuing accordingly the startup sequence.

The execution of the RAM test after a wake-up reset is controlled by the MBIST_EN bit in the PMU->SystemStartConfig register. The user can freely set the value of this bit and its value is kept over wake-up reset. If the bit is set to 0, the RAM test is not performed on wake-up. If the bit is set to 1 then the RAM test is performed even for wake-up resets.

If an error is detected the device is set to loop endlessly with WDT1 enabled.

In case of power-on reset, brown-out reset or wake-up reset from Sleep mode the startup procedure will continue with a complete RAM initialization by writing all the RAM to zero with proper ECC status.This is needed to prevent an ECC error during user code execution due to a write operation to an un-initialised location (with invalid ECC code). Afterwards the Firmware proceeds checking the NVM status.

Notes

1. *Via MBIST EN bit user can only disable the RAM test sequence while the RAM initialization to 00_H is still executed.*
2. *The test sequence on the entire RAM takes 500 µs while the initialization of the complete RAM takes 150 µs.*

3.1.2 NVM initialisation routine

This routine will set the NVM protection according to the password in the configuration sector (refer to the **Section 5.4.5** for further details on NVM protection and protection password).

3.1.3 NVM MapRAM initialisation

The MapRAM is being used for the EEPROM emulation which is described in the **Section 5.4.1**. After every reset the system performs the MapRAM initialisation. This operation is triggered to restore the MapRAM content.

The operation is executed with different flows depending on the type of reset. These two flows are described in the following chapters.

3.1.3.1 NVM MapRAM initialization at high prio reset

During power-on reset, brown-out reset, pin reset or wake-up reset, the MapRAM content is cleared. For this reason, during the following startup sequence the system performs a complete MapRAM initialization. In case during the initialisation at least one error is detected, the service algorithm routine is called to do the repair.

In case of mapping errors, the repair mechanism consists of erasing the wrong pages (either faulty or double mapped pages). The repair step then requires the right of modifying the NVM Data sector content, which can be in contrast to the NVM protection

settings user has provided. To avoid any risk of data loss, the user can control via dedicated 100TP page parameter whether the SA is allowed to proceed to the repair step in case NVM password protection for NVM Data sector is installed.

Detailed description of the MapRAM initialization and repair step can be found in the **Section 5.4.1**.

3.1.3.2 NVM MapRAM initialization at low prio reset

During low prio reset (soft reset, internal WDT reset and lockup reset) the content of the MapRAM is not cleared and so a MapRAM initialization is not mandatory. Any of these reset types might occur during an NVM operation on a non-linearly mapped data sector and might result in an inconsistent state of the MapRAM. In order to check MapRAM for consistency, MapRAM initialization is performed for these reset types too. In case of mapping errors no repair step is triggered, so that worst case startup time is not increased.

The result of the NVM Data sector initialization executed during the startup flow is reported to the user via the bit 1 of the SYS_STRTUP_STS register (MRMINITSTS). If this bit is set to 1 then the last initialization failed and the mapping info might be corrupted. In this case, a reset (power-on reset, brown-out reset, pin reset, WDT1 reset or wake-up reset) can be issued in order to start the Service Algorithm to try to fix the integrity issue inside the Data NVM. If the MRAMINITSTS is still flagged afterwards, the Data NVM sector has to be re-initialized by performing a sector erase.

3.1.4 Oscillator trimming and system clock selection

After every power-on reset, brown-out reset, pin reset or wake-up from sleep reset the system runs with an internal low precision clock (nominally 18 MHz). During the startup procedure, the internal oscillator is trimmed and the PLL is programmed to f_{SYS} max of the specific device. In order to reduce the boot time, the startup procedure continues to run with the low precision clock while the PLL is locking. System clock will be switched to PLL output before jumping to user or BSL mode in case of successful lock. In case the PLL does not lock the startup sequence proceeds further using the low precision clock as system clock.

Once User mode is entered, the user is allowed to set the final desired frequency by proper register setting.

Note: After every power-on reset, brown-out reset, pin reset or wake-up reset, the user shall check whether the system is running on the low precision clock or on the PLL output reading the SYSCON0 register.

3.1.5 Analog module trimming

In this routine, the trimming values of voltage regulators, LIN module, temperature sensor, bridge driver and other analog modules are read from the configuration sector and written into the respective SFR. For User mode or Debug Support mode, checksum on 100TP page is evaluated. In case of error, default values are used. Refer to **Table 5-11** for a list of user parameters in 100TP page.

3.1.6 User configuration data initialization

The firmware provides a routine to download data stored in user accessible configuration sector pages (100TP) during the startup flow. In particular, the routine copies a specified number of bytes from a selected 100TP page (starting always from first byte in the page) into the RAM (starting at a given address). The routine is by default disabled and can be enabled and controlled by proper programming of the bytes stored in first 100TP page as described in the **Table 5-11**. This routine is not performed after a software or watchdog reset.

Relevant routine control parameters stored in the first 100TP page are:

- CS_USER_CAL_STARTUP_EN (offset=79_H): When set to $C3_H$ it enables the user data download from a 100TP page into the RAM during startup flow. All other values will be ignored and the routine will not be executed at startup.
- CS_USER_CAL_XADDH: (offset=$7A_H$): It defines the high byte of the RAM starting address where to copy data downloaded from 100TP page. This byte is ignored if the routine is not enabled.
- CS_USER_CAL_XADDL: (offset=$7B_H$): It defines the low byte of the RAM starting address where to copy data downloaded from 100TP page. This byte is ignored if the routine is not enabled.
- CS_USER_CAL_100TP_PAGE: (offset=$7C_H$): It defines the 100TP page where data has to be downloaded from (refer to **Figure 5-8**). This byte is ignored if the routine is not enabled.
- CS_USER_CAL_NUM: (offset=$7D_H$): It defines the number of bytes to be downloaded starting from the first byte of the selected 100TP page. This byte is ignored if the routine is not enabled.

The RAM address where the user configuration data has to be copied to is stored as a 16-bit offset to the RAM start address (18000000_H). This offset is defined by the CS_USER_CAL_XADDL and CS_USER_CAL_XADDH parameters.

The routine has been developed to support downloading of the Customer_ID and the ADC calibration parameters stored at the beginning of the first 100TP page (see **Table 5-11**) into the RAM for an easy access but can be more generally used for all other CS user parameters. If the routine is enabled, firmware will copy the data from the selected 100TP page into the RAM. Moreover, independent of startup setting, a similar routine is provided as NVM user routine (refer to the **Section 5.3.18**).

3.1.7 Debug Support mode entry

Entry to Debug Support mode is determined by pin setting at power-up (see **Table 2-1**).

In case NVM address 11000004_H - 11000007_H is not $FFFFFFFF_H$, the firmware code clears the RAM, waits for debugger to be connected, moves the VTOR to 11000000_H and jumps to user code.

3.1.8 User mode and BSL mode entry

Entry to User mode is determined by the No Activity Count (NAC) value which is defined in the user code (refer to the **Section 3.1.8.1**). 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 jumps to the reset handler.

If NVM double bit error occurs when reading the NAC value, the system goes into endless loop.

Before entering User mode, the system clock frequency is switched to PLL output previously set to the max f_{SYS} of the device. In case the PLL has not locked within 1 ms, the CPU clock source LP_CLK (low precision clock running nominally at 18 MHz) will be used.

Note: User mode is entered jumping to the reset handler. This can happen directly from startup routine, after a waiting time for possible BSL communication, or as a result of BSL commands. In all these cases, jump to User mode will only occur either (1) when NVM is not protected and NVM content at 11000004_H - 11000007_H *is not* $FFFFFFFF_H$ *or (2) when NVM is protected. In all other cases, firmware will put the device in Sleep mode.*

3.1.8.1 NAC definition

The NAC value defines the time window after reset release in which the firmware is able to receive BSL connection messages. The bits 5 to 0 define the duration of the time window while bit 7 of the NAC defines, which BSL interface is selected. Bit 6 is reserved and not used. If no BSL messages are received on the selected BSL interface during the NAC window and NAC time has expired, the firmware code proceeds to User mode.

There are 2 different BSL interfaces supported, FastLIN and UART.

The FastLIN BSL is an enhanced feature in TLE986x/TLE987x device, supporting a fixed baud-rate of 115.2 kBaud. In order to support the faster baud-rate, the protocol is the same of UART BSL, implemented in the integrated LIN transceiver (refer to **Chapter 4** for protocol description).

After ending the startup procedure, the program will detect any activities on the LIN/UART for a period of time, determined by (((NAC & $3F_H$) -1_H) * 5) ms reduced by the time already spent to perform the startup procedure. When nothing is detected on the LIN/UART and (((NAC & $3F_H$) -1_H) * 5) ms is passed from reset going high, the microcontroller will jump to User mode. If NAC(5:0) is 1_H, the BSL window is closed, no BSL connection is possible and User mode is entered without delay.

The maximum NAC value is restricted to C_H as the first open WDT1 window is worst case 65 ms. In case a valid BSL command is detected during the BSL window the firmware suspends 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 the WDT1 before jumping to user code. If NAC is not valid, BootROM code will switch off the WDT1 and wait for a FastLIN entry sequence infinitely.

Table 3-1 gives an overview of the action of the microcontroller with respect to No Activity Count (NAC) values and the **Table 3-2** shows the selection of the BSL interface depending on the NAC bit 7.

Table 3-1 Type of action w.r.t. No Activity Count (NAC) values

NAC value (5:0)	Action
01_H	0 ms delay. Jump to User mode immediately
02_H	5 ms delay before jumping to User mode[1)]
03_H	10 ms delay before jumping to User mode[1)]
04_H	15 ms delay before jumping to User mode[1)]
05_H	20 ms delay before jumping to User mode[1)]
06_H	25 ms delay before jumping to User mode[1)]
07_H	30 ms delay before jumping to User mode[1)]
08_H	35 ms delay before jumping to User mode[1)]
09_H	40 ms delay before jumping to User mode[1)]
$0A_H$	45 ms delay before jumping to User mode[1)]
$0B_H$	50 ms delay before jumping to User mode[1)]
$0C_H$	55 ms delay before jumping to User mode[1)]
$0D_H$ - $3F_H$, 00_H, Invalid	Wait forever for the first frame

[1)] If a FastLIN frame/UART frame is received within the delay period, the following actions occur; (1) the remaining delay is ignored, (2) it will not enter User mode anymore (3) it will process the FastLIN / UART frame accordingly.

Table 3-2 BSL Interface selection via NAC

NAC(7)	Action
0	FastLIN BSL
1	UART BSL

For each variant, the NAC value is stored, together with the NAD value, in the last 4 bytes of the linearly mapped NVM region. To ensure the parameter validity, the actual and inverted values of the 2 parameters are checked. In case the stored value and inverted value are not consistent (value + inverted value + 1 not equal to 0) the parameter is considered to be invalid and the default value is used. The BSL window will be open indefinitely and FastLIN is selected as BSL interface.

The **Table 3-3** shows the addresses for all the available family devices. In the table NSA stands for NVM Starting Address (the value is equal to 11000000_H for all variants) and NLS stands for NVM Linear Size, in bytes, whose value is variant dependent.

Table 3-3 NAC and NAD parameters details

Address	User defined value	Criteria / Range	Default
NSA+(NLS-4)$_H$	NAC	01_H - $0C_H$ for FastLIN BSL 81_H - $8C_H$ for UART BSL	$7F_H$
NSA+(NLS-3)$_H$	$\overline{\text{NAC}}$	1's complement	-
NSA+(NLS-2)$_H$	NAD (for FastLIN BSL only)	01_H - FF_H (00_H is reserved)	$7F_H$
NSA+(NLS-1)$_H$	$\overline{\text{NAD}}$ (for FastLIN BSL only)	1's complement	-

For NSA and NLS values refer to **Table 5-2**.

3.1.9 Node Address for Diagnostic (NAD)

The NAD value is stored similarly to the NAC value in NVM. This field specifies the address of the active slave node. Only slave nodes have an address. The NAD address range supported in TLE986x/TLE987x is listed in **Table 3-4**.

Table 3-4 NAD address range

NAD Value	Description
00_H	Invalid Slave Address
01_H to FE_H	Valid Slave Address

Table 3-4 NAD address range (cont'd)

NAD Value	Description
$7F_H$	Default Address (NAD value is invalid or it is not programmed in NVM linear area)
FF_H	Broadcast Address (The actual NAD value stored in the NVM is not checked. Communication is granted)

If the NAD value is not programmed in the NVM linear region or in case its value is invalid (value and inverted value not consistent), the NAD is assumed to be $7F_H$.

4 FastLIN and UART BSL mode

This chapter describes the protocol used for the FastLIN and UART BSL.

Both FastLIN and UART BSL share the same protocol. FastLIN BSL communication is performed via the integrated LIN transceiver while UART BSL is performed via the full duplex UART interface (UART1, UART send P0.1, UART receive P1.4).

Note: UART BSL expects a full duplex communication. A connection of an external LIN transceiver to P0.1/P1.4 is not supported by UART BSL.

Although FastLIN BSL uses the same protocol as UART BSL, the connection sequence is different. To protect the FastLIN BSL from unwanted entries, a special entry sequence must be executed before FastLIN BSL is fully enabled. In addition, the FastLIN BSL is always executed at a fixed baud-rate of 115.2 kBaud.

4.1 FastLIN and UART BSL protocol

The FastLIN and UART BSL protocol is based on the following two phases:

- **Phase I**: Establish a serial connection
  - For FastLIN BSL the device automatically sets a baud-rate of 115.2 kBaud and waits for a specific command entry sequence before enabling all FastLIN BSL supported features (refer to the **Section 4.3**).
  - For UART BSL the device automatically synchronizes transfer speed (baud-rate) with the serial communication partner (host) for UART. (refer to the **Section 4.2**).
- **Phase II**: Perform the serial communication with the host. The host controls communication by sending header information which selects one of the working modes (refer to the **Section 4.4**) These modes are:
  - **Mode 0 (00H)**: Transfer a user program from the host to RAM or write 100TP pages[1)]
  - **Mode 1 (01H)**: Execute a user program in the RAM[2)]
  - **Mode 2 (02H)**: Transfer a user program from the host to NVM[1)]
  - **Mode 3 (03H)**: Execute a user program in the NVM[2)]
  - **Mode 4 (04H)**: Erase NVM[1)]
  - **Mode 6 (06H)**: Permanent NVM protection mode enabling/disabling scheme[2)]
  - **Mode A (0AH)**: Get Info (based on option byte)[1)]

1) The microcontroller returns to the beginning of phase II and waits for the next command from the host.
2) UART BSL and serial communication are terminated.

Except **Mode 1**, **Mode 3** and **Mode 6**, the microcontroller returns to the beginning of Phase II and waits for the next command from the host after executing all other modes.

All serial communication is performed via the integrated LIN transceiver for FastLIN and via the full duplex serial interface (UART1) of the TLE986x/TLE987x for UART BSL.

The serial transfer works in asynchronous mode with the serial parameters 8N1 (eight data bits, no parity and one stop bit).

The following section provides detailed information on these two phases.

4.2 Phase I for UART BSL: Automatic serial synchronization to the host

Upon entering UART BSL mode, a serial connection is established and the transfer speed (baud-rate) of the serial communication partner (host) is automatically synchronized in the following steps, a simplified entry flow is depicted in **Figure 4-1**:

- STEP 1: Initialize serial interface for reception and timer for baud-rate measurement
- STEP 2: Wait for test byte (80_H) from host
- STEP 3: Synchronize the baud-rate to the host
- STEP 4: Send acknowledge byte (55_H) to the host
- STEP 5: Enter Phase II

4.2.1 General description

The microcontroller will set the serial port of the UART1 to mode 1 (8-bit UART, variable baud-rate) for communication. Timer 2 will be set in Auto-Reload mode (16-bit timer) for baud-rate measurement. In the process of waiting for the test byte (80_H), microcontroller will start the timer on reception of the start bit (0) and stop it on reception of the last bit of the test byte (1). Then the UART BSL routine calculates the actual baud-rate, sets the PRE and BR_VALUE values and activates baud-rate generator. When the synchronization is done, the microcontroller sends back the acknowledge byte (55_H) to the host. If the synchronization fails, the baud-rates for the microcontroller and the host are different, and the acknowledge code from the microcontroller cannot be received properly by the host. In this case, on the host side, the host software may give a message to the user, e.g. asking the user to repeat the synchronization procedure. On the microcontroller side, the UART BSL routine cannot judge whether the synchronization is correct or not. It always enters phase II after sending the acknowledge byte. Therefore, if synchronization fails, a reset of the microcontroller has to be invoked, in order to start a new synchronization attempt.

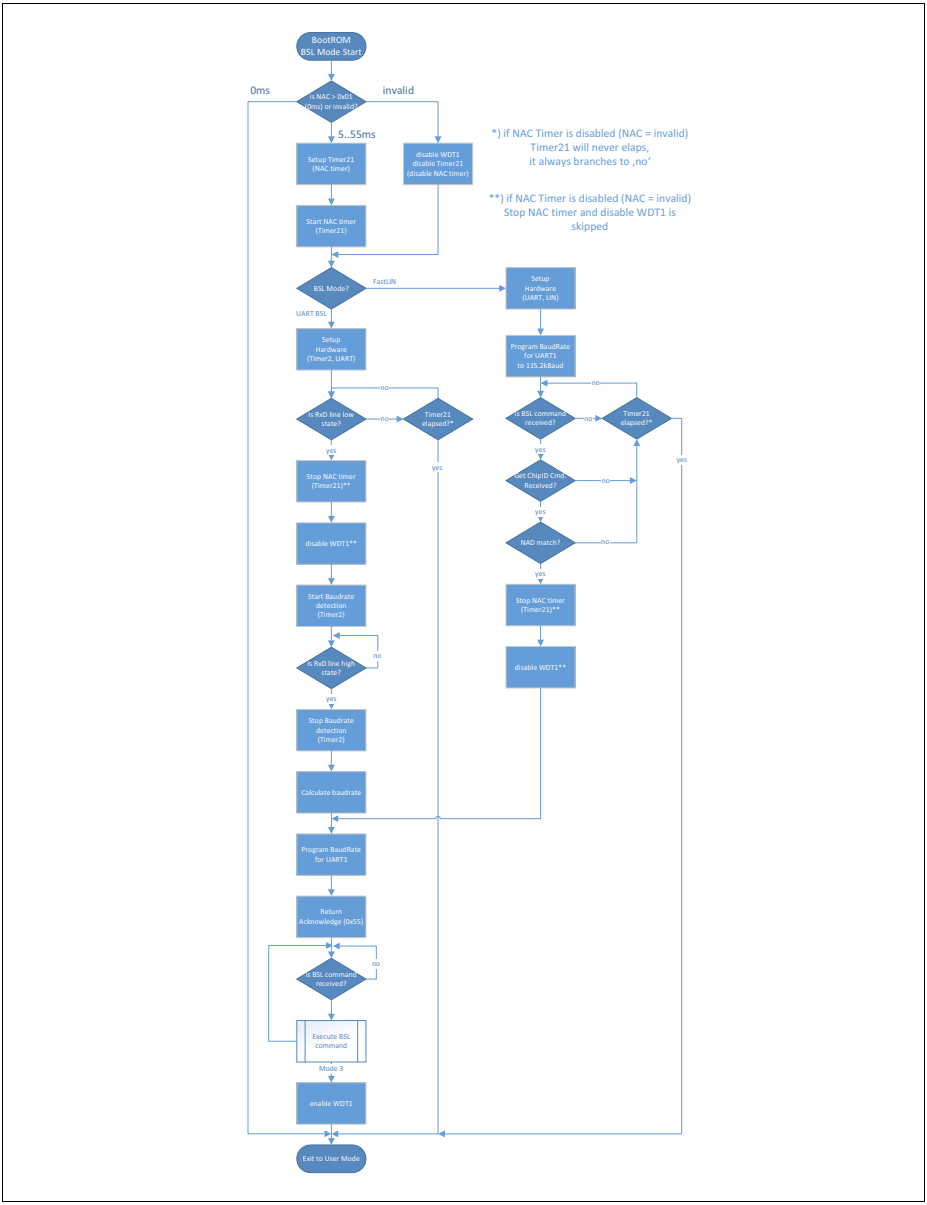


Figure 4-1 BSL entry flow (simplified)

4.2.2 Calculation of BR_VALUE and PRE values

For the baud-rate synchronization of the microcontroller to the fixed baud-rate of the host, the UART BSL routine waits for a test byte (80_H), which has to be sent by the host. By polling the receive port of the serial interface (P1_DATA.4/RxD Pin), the Timer2 (T2) is started on the reception of the start bit (0) and stopped on the reception of the last bit of the test byte (1). Hence the time recorded is the receiving time of 8 bits (1 start bit plus 7 least significant bits of the test byte). The resulting timer value is 16-bit (T2). This value is used to calculate the 11-bit auto-reload value (BR_VALUE stored in the BGH and BGL SFRs), the fractional divider FDSEL and PRE, with T2PRE predefined as 011. This calculation needs two formulas.

First, the correlation between the baud-rate (baud) and the reload value (BG) depends on the internal peripheral frequency (f_{PCLK})

$$\text{baud} = \frac{f_{PCLK}}{16 \times PRE \times \left(BR\_VALUE + \frac{FDSEL}{32}\right)} \quad [4.1]$$

Second, the relation between the baud-rate (baud) and the recording value of Timer2 (T2) depends on the T2 peripheral frequency (f_{T2}) and the number of received bits ($f_{T2}N_b$)

$$\text{baud} = \frac{f_{T2} \times N_b}{T2} \quad [4.2]$$

Combining **Equation [4.1]** and **Equation [4.2]** with N_b=8, f_{T2}=f_{PCLK}/ 8 (T2PRE=011),

$$\frac{f_{PCLK}}{16 \times PRE \times \left(BR\_VALUE + \frac{FDSEL}{32}\right)} = \frac{\frac{f_{PCLK}}{8} \times 8}{T2} \quad [4.3]$$

Simplifying **Equation [4.3]**, we get

$$PRE \times \left(BR\_VALUE + \frac{FDSEL}{32}\right) = \frac{T2}{16} \quad [4.4]$$

After setting BR_VALUE, FDSEL and PRE, the baud-rate generator will then be enabled, and the UART BSL routine sends an acknowledge byte (55_H) to the host. If this byte is received correctly, it will be guaranteed that both serial interfaces are working with the same baud-rate.

4.3 Phase I for FastLIN BSL: FastLIN BSL entry sequence

Upon entering FastLIN BSL mode, there is no automatic synchronization to the host transfer speed. The device sets the baud-rate to 115.2 kBaud. Please also refer to the simplified entry flow in **Figure 4-1**.

In addition, the FastLIN mode is protected against unwanted entries, i.e. because of noise on the communication line. In order to establish a FastLIN connection, the following sequence must be sent to the device during the active BSL connection window (NAC).

- STEP 1: Host to send the “Get Chip ID for FastLIN BSL entry” command
- STEP 2: Device to answer with acknowledge (55_H)
- STEP 3: Device to answer with the Chip ID

Get Chip ID for FastLIN BSL entry is described in the **Section 4.4.2.8**.

If the sequence has been passed to the device during the BSL active window and the device has acknowledged the commands and answered with the Chip ID then the FastLIN BSL connection is established, the NAC and the WDT1 will be disabled and the device will wait for further FastLIN BSL commands.

4.4 Phase II: Serial communication protocol and the working modes

Once the BSL communication is established, the FastLIN or UART BSL enters Phase II, during which it communicates with the host to select the desired working modes. The detailed communication protocol is explained as follows: from now on, both FastLIN and UART BSL modes share the same UART BSL protocol.

4.4.1 Serial communication protocol

The communication between the host and the UART BSL routine is done by a simple transfer protocol. The information is sent from the host to the microcontroller in blocks. All the blocks follow the specified block structure. The host is sending several transfer blocks and the UART BSL routine is just confirming them by sending back single acknowledge or error bytes. The microcontroller itself does not send any transfer blocks.

However, the above rule does not apply to some modes where the microcontroller might need to send the required data to the host besides the acknowledge or error byte (e.g. Mode A).

4.4.1.1 Transfer block structure

A transfer block consists of three parts:

Block Type (1 byte)	Data Area (X bytes)	Checksum (1 byte)

- **Block Type**: The type of block, which determines how the bytes in the data area are interpreted. Implemented block types are:
  - 00_H type “Header”
  - 01_H type “Data”
  - 02_H type “End of Transmission” (EOT)
- **Data area**: A list of bytes, which represents the data of the block. The length of data area cannot exceed 128 bytes for Mode 0 and 2. For Mode 2, the length of data area must always be 128 bytes. This is due to the fact that NVM is written page-wise.
- **Checksum**: The XOR checksum of the block type and data area.

The host will decide the number of transfer blocks and their respective lengths during one serial communication process. For safety purpose, the last byte of each transfer block is a simple checksum of the block type and data area. The host generates the checksum by XOR-ing all the bytes of the block type and data area. Every time the UART BSL routine receives a transfer block, it recalculates the checksum of the received bytes (block type and data area) and compares it with the attached checksum.

Note: If there is less than one page to be programmed to NVM, the PC host will have to fill up the vacancies with 00_H, and transfer data in the length of 128 bytes.

4.4.1.2 Transfer block type

There are three types of transfer blocks depending on the value of the block type. **Table 4-1** provides the general information on these block types. More details will be described in the corresponding sections later.

Table 4-1 Type of transfer block

Block name	Block type	Description
Header block	00_H	This block has a fixed length of 8 bytes. Special information is contained in the data area of the block, which is used to select different working modes.
Data block	01_H	This block length depends on the special information given in the previous header block. This block is used in working Mode 0 and 2 to transfer a portion of program code. The program code is contained in the data area of the block.
EOT block	02_H	This block length depends on the special information given in the previous header block. This block is the last block in data transmission in working Mode 0 and 2. The last program code to be transferred is in the data area of the block.

4.4.1.3 Response codes to the host

The microcontroller communicates to the host whether a block has been successfully received by sending out a response code. If a block is received correctly, an acknowledge code (55_H) is sent. In case of failure, an error code is returned. There are two possible error codes, FF_H or FE_H, reflecting the two possible types of fail, block type or Checksum Error. A Block Type Error occurs when either a not implemented block type or transfer blocks in wrong sequence are received. For example, if in working mode 0 two consecutive header blocks are received a Block Type Error is detected and a Block Type Error (FF_H) indication is returned. A Checksum Error occurs when the checksum comparison on a received block fails. In such a case, the transfer is rejected and a Checksum Error (FE_H) indication is returned. In both error cases the UART BSL routine awaits the actual block from the host again.

When program and erase operation of NVM is restricted due to enabled NVM protection, only Mode 1 and 3 and some options of Mode A are allowed. All other modes are blocked and a Protection Error code (FD_H) will be sent to host. This will indicate that NVM is protected and no programming and erasing are allowed. In this error case, the UART BSL routine will wait for the next header block from the host again.

Table 4-2 gives a summary of the response codes to be sent back to the host by the microcontroller after it receives a transfer block.

Table 4-2 Type of response codes

Communication status	Response code to the host
Acknowledge (Success)	55_H
Block Type Error	FF_H
Checksum Error	FE_H
Protection Error	FD_H
Combined Offset Error (COMBOFFSETFAULT)	$0FB_H$ only valid for Mode 0 option $F0_H$
ID Offset Error (IDOFFSETFAULT)	$0FA_H$ only valid for Mode 0 option $F0_H$
In Page Offset Error (INPAGEOFFSETFAULT)	$0F9_H$ only valid for Mode 0 option $F0_H$

Table 4-3 shows a tabulated summary of the possible responses the device may transmit following the reception of a header, data or EOT block.

Table 4-3 Possible responses for various block types

Mode	Header block	Data block	EOT block
0	Acknowledge, Block Type Error, Checksum Error, Protection Error	Acknowledge, Block Type Error, Checksum Error	Acknowledge, Block Type Error, Checksum Error, Combined/ID/InPage offset error
1	Acknowledge, Block Type Error, Checksum Error		
2	Acknowledge, Block Type Error, Checksum Error, Protection Error	Acknowledge, Block Type Error, Checksum Error	Acknowledge, Block Type Error, Checksum Error
3	Acknowledge, Block Type Error, Checksum Error		
4	Acknowledge, Block Type Error, Checksum Error, Protection Error		

Table 4-3 Possible responses for various block types (cont'd)

Mode	Header block	Data block	EOT block
6	Acknowledge, Block Type Error, Checksum Error, Protection Error		
A	Acknowledge, Block Type Error, Checksum Error, Protection Error		

The responses are defined in **Table 4-4**, which lists the possible reasons and/or implications for error and suggests the possible corrective actions that the host can take upon notification of the error.

Table 4-4 Definitions of responses

Response	Value	Description			
		Block type	BSL mode	Reasons / Implications	Corrective action
Acknow-ledge	55_H	Header	1, 3	The requested operation will be performed once the response is sent.	
			A	The requested operation has been performed and was successful. Requested data transmission follows.	
			6	The requested operation has been performed and was successful.	
		EOT	0, 2, 4		
		All other combinations		Reception of the block was successful. Ready to receive the next block.	
Block Type Error	FF_H	Header	2, 4, A	Start address in mode data is not within NVM address range or invalid 100TP page.	Retransmit a valid header block
		All other combinations		Either the block type is undefined or option is invalid or the flow is invalid.	Retransmit a valid block

Table 4-4 Definitions of responses (cont'd)

Response	Value	Description			
		Block type	BSL mode	Reasons / Implications	Corrective action
Checksum Error	FE_H	All combinations		There is a mismatch between the calculated and the received checksum.	Retransmit a valid block
Protection Error	FD_H	Header	0, 2, 4, 6, A	Protection against external access enabled, i.e. user-password is valid.	Disable protection
Combined Offset Error Code	FB_H	EOT	0	The operation is targeting 100TP page 1 and there is at least 1 byte with a not in page offset and 1 byte pointing to the Customer_ID reserved region.	Check the byte offset
ID Offset Error Code	FA_H	EOT	0	The operation is targeting 100TP page 1 and there is at least 1 byte pointing to the Customer_ID reserved region.	Check the byte offset
Combined Offset Error Code	$F9_H$	EOT	0	There is at least 1 byte with a not in page offset.	Check the byte offset

4.4.1.4 Block response delay

As described in the **Section 4.4.1.3**, after receiving any block the microcontroller communicates to the host whether the block was successfully received by sending out a response code. If a block is received correctly, an acknowledge code (55H) is sent. In case of failure, an error code is returned.

The response is transmitted with a delay that depends on the selected mode and on the type of the block received.

The following **Table 4-5** reports the maximum response delay for each mode and block type.

Table 4-5 Maximum response delay

Mode	Option	Description	Block type		
			Header	Data	EoT
Mode 0	0x00	Download Code/ Data to RAM	250 us	1 µs per byte	1 µs per byte
	0xF0	Download data to 100TP pages	250 µs	1 µs per byte	10 ms [1]
Mode 1	--	RAM code execution	250 µs	--	--
Mode 2	--	Download Code/ Data to NVM	250 µs	10 ms [1]	10 ms [1]
Mode 3	--	NVM code execution	250 µs	--	--
Mode 4	0x00	NVM page erase	4.5 ms	--	--
	0x40	NVM sector erase	4.5 ms	--	--
	0xC0	NVM mass erase	4.5 ms per sector	--	--
Mode 6	--	NVM protection set	10 ms [1]	--	--
	--	NVM protection reset	4.5 ms + 4.5 ms per sector	--	--
Mode A	0x00	Get Chip ID	250 µs	--	--
	0x10	NVM page checksum check	250 µs	--	--
	0x18	NVM mass checksum check	100 ms	--	--
	0x50	100TP page checksum check	250 µs	--	--
	0xC0	NVM page	250 µs	--	--
	0xF0	100TP page	250 µs	--	--

[1] Time needed for data collection, OpenAB, erasing old data (if required) and programming the data given.

4.4.2 UART BSL modes

When the UART BSL routine enters Phase II, it first waits for an 8-byte long header block from the host. The header block contains the information for the selection of the working modes. Depending on this information, the UART BSL routine selects and activates the desired working mode. If the microcontroller receives an incorrect header block, the UART BSL routine sends, instead of an acknowledge code, a checksum or block type Error code to the host and awaits the header block again. In this case the host may react by re-sending the header block or by releasing a message to the user.

4.4.2.1 Header block

The header block is always the first transfer block to be sent by the host during one data communication process. It contains the working mode number and special information on the related mode (referred to as “Mode Data”). The general structure of a header block is shown below.

Block Type $\mathbf{00_H}$ (Header Block)	**Data Area**		**Checksum** (1 byte)
	Mode (1 byte)	**Mode Data** (5 bytes)	

Description:

- **Block Type** $\mathbf{00_H}$: The block type, which marks the block as a **header block**.
- **Mode**: The mode to be selected. The implemented modes are covered in **Chapter 4**.
- **Mode Data**: Five bytes of special information, which are necessary to activate corresponding working mode.
- **Checksum**: The checksum of the header block.

4.4.2.2 Mode 0 - Code/Data download to RAM/100TP

Mode 0 is used to transfer a user program or data from the host to the RAM of the microcontroller via serial interface. Selecting the proper mode option, this mode can be used to transfer data into the user configuration sector pages. In this case, user has to transfer data to the RAM in accordance with the format reported in the **Table 5-12** and after EOT block has been received, data is automatically copied with proper offset in the target page.

Different options supported are:

- Option 00_H: RAM download
- Option $F0_H$: RAM download and configuration sector page programming

The header block for this working mode has the following structure:

The header block for RAM download (option = 00_H)

00_H (Header Block)	00_H (Mode 0)	Mode Data (5 bytes)					Checksum (1 byte)
		StartAddr High (1 byte)	StartAddr Low (1 byte)	Block Length (1 byte)	Not Used (1 byte)	Option = 00_H (1 byte)	

Mode Data Description:

Start Addr High, Low: 16-bit start address, which determines where to copy the received program codes into the RAM.

Block Length: The length (number of bytes) of the following data blocks or EOT block.

Not Used: This byte is not used and will be ignored.

Option: Set to 00_H for RAM download.

Note: RAM address provided as input in Mode 0 has to be considered as an offset to be added to the standard RAM starting address of the TLE986x/TLE987x.

In option 00H start address can be each valid RAM offset address. Data sent in the following data/ EOT blocks will be copied into the RAM at the specified address (18000000_H + StartAddr).

The header block for RAM download and 100TP page programming (option = $F0_H$)

00_H (Header Block)	00_H (Mode 0)	Mode Data (5 bytes)					Checksum (1 byte)
		StartAddr High (1 byte)	StartAddr Low (1 byte)	Block Length (1 byte)	100TP Page (1 byte)	Option (1 Byte)	

Mode Data Description:

Start Addr High, Low: 16-bit start address, which determines where to copy the received data in the RAM.

Block Length: The length of the following data blocks or EOT block.

100TP Page: This byte is used to select the desired 100TP page to be programmed. This byte is relevant only in case option $F0_H$ is used. The 100TP page is selected according to the addressing scheme reported in **Figure 5-8**.

Option: Set to $F0_H$ for RAM download and 100TP page programming.

Using this option, user can write data into the 100TP pages. In this case, data has to be sent to the RAM according to the **Table 5-12** and therefore start address has to be equal to 18000400_H. In case a different starting address is provided, the operation will result in a Block Type Error indication. When this option is selected a proper 100TP page has to be provided.

Note: RAM address provided as input in Mode 0 has to be considered as an offset to be added to the standard RAM starting address of the TLE986x/TLE987x. So, for option $F0_H$, the Start Addr parameter has to be set to 0400_H.

All other options will be treated as option 00_H.

*Note: The **Block Length** refers to the whole length (block type, data area and checksum) of the following transfer block (data block or EOT block).*

After successfully receiving the header block, the microcontroller enters Mode 0, during which the program codes are transmitted from the host to the microcontroller by data block and EOT block, which are described as below.

The data block

01_H (Data Block)	**Program Code** (((Block Length) - 2) bytes)	**Checksum** (1 byte)

Description:

Program Code: The program code has a length of ((**Block Length**) - 2) byte, where the **Block Length** is provided in the previous header block.

The EOT block

02_H (EOT Block)	**Last Codelength** (1 byte)	**Program Code** (Last Codelength bytes)	**Not Used** (((Block Length) – 3 – (Last Codelength)) bytes)	**Checksum** (1 byte)

Description:

Last Codelength: This byte indicates the length of the program code in this EOT block.

Program Code: The last program code to be sent to the microcontroller.

Not used: The length is ((**Block Length**) - 3 - (**Last Codelength**)) bytes.

When trying to program 100TP page, some special error handling is provided.

In particular, in addition to the generic error code, the UART BSL Mode 0 option $F0_H$ may return:

- BLOCKFAULT indication (FF_H) in case of wrong config sector page selection.
- INPAGEOFFSETFAULT indication ($F9_H$) in case at least one byte has an offset > $7E_H$, i.e. has a not in page offset or is targeting the page counter (refer to **Table 5-12**). In this case, the program for the valid bytes is still performed.
- IDOFFSETFAULT indication (FA_H) in case at least one byte is targeting the Customer_ID reserved region when programming 100TP page 1. In this case, the program for the valid bytes is still performed.
- COMBOFFSETFAULT indication (FB_H) in case at least one byte is targeting the Customer_ID reserved region when programming 100TP page 1 and at least 1 byte has a not in page offset or is targeting the page counter. In this case, the program for the valid bytes is still performed.

4.4.2.3 Mode 1 - Code execution inside RAM

Mode 1 is used to execute a user program in the RAM of the microcontroller at the address pointed by the RAM location 18000404_H. The header block for this working mode has the following structure:

The header block

00_H (Header Block)	**01_H** (Mode 1)	**Mode Data (5 bytes)**	**Checksum** (1 byte)
		Not Used	

Mode Data Description:

Not used: The five bytes are not used and will be ignored in Mode 1.

In working mode 1, the header block is the only transfer block to be sent by the host, no further serial communication is necessary. The microcontroller will exit the UART BSL mode, set the vector table in RAM at address 18000400_H and branch to the address pointed by the standard reset handler (18000404_H).

4.4.2.4 Mode 2 - Code/Data download to NVM

Mode 2 is used to transfer a user program from the host to the NVM of the microcontroller via serial interface. This mode is not accessible if NVM protection is installed.

The header block for this working mode has the following structure:

The header block

00_H (Header Block)	02_H (Mode 2)	Mode Data (5 bytes)					Checksum (1 byte)
		StartAddr 4 (MSB)	StartAddr 3	StartAddr 2	StartAddr 1 (LSB)	Block Length (1 byte)	

Mode Data Description:

Start Addr 4, 3, 2, and 1: 32-bit start address, which determines where to copy the received program codes in the NVM. This address must be aligned to the page address (bit[6:0] = 0).

Block Length: The length of the following data blocks or EOT block. If data blocks are to be sent, the block length has to be 130 (128+2) bytes. If only EOT block is sent, the block length has to be 131 (128+3) bytes. Other block length values than 130 (data block) or 131 (EOT block) are not allowed.

*Note: If the data starts in a non-page address, PC host must fill up the beginning vacancies with 00_H and provide the start address of that page. For e.g., if data starts in $11000F82_H$, the PC host will fill up the addresses $11000F80_H$ and $11000F81_H$ with 00_H and provide the **Start Address** $11000F80_H$ to microcontroller. Moreover, if data is only 8 bytes, the PC host will also fill up the remaining addresses with 00_H and transfer 128 data bytes.The **Block Length** refers to the whole length (block type, data area and checksum) of the following transfer block (data block or EOT block).*

After successfully receiving the header block, the microcontroller enters Mode 2, during which the program codes are transmitted from the host to the microcontroller by data block and EOT block, which are described as below.

The data block

01_H (Data Block)	Program Codes (((Block Length) - 2) bytes)	Checksum (1 byte)

Description:

Program Codes: The program codes have a length of ((**Block Length**) - 2) bytes, where **Block Length** is provided in the previous header block.

The EOT block

02_H (EOT Block)	**Last Codelength** (1 byte)	**Program Code** (Last Codelength bytes)	**Not Used** (((Block Length) – 3 – (Last Codelength)) bytes)	**Checksum** (1 byte)

Description:

Last Codelength: This byte indicates the number of program code bytes in this EOT block.

Program Code: The last program code bytes to be sent to the microcontroller.

Not used: The length is ((**Block Length**) - 3 - (**Last Codelength**)) bytes.

The following Figures show examples of how to program one or several NVM pages using working mode 2.

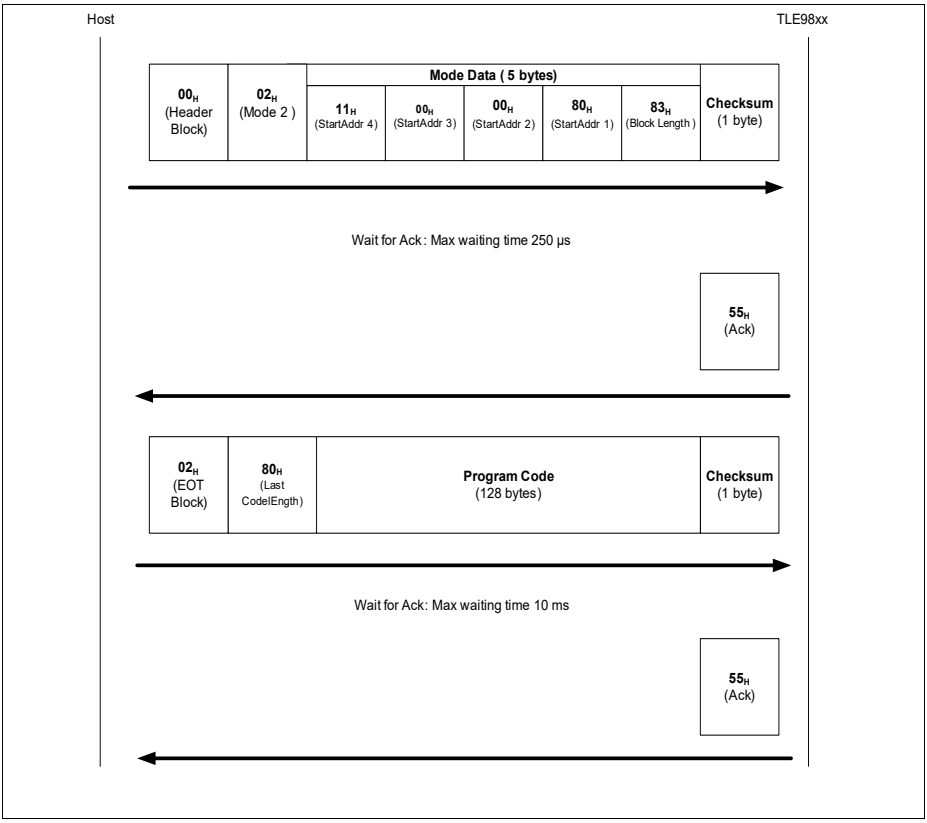


Figure 4-2 Single NVM page program via working mode 2

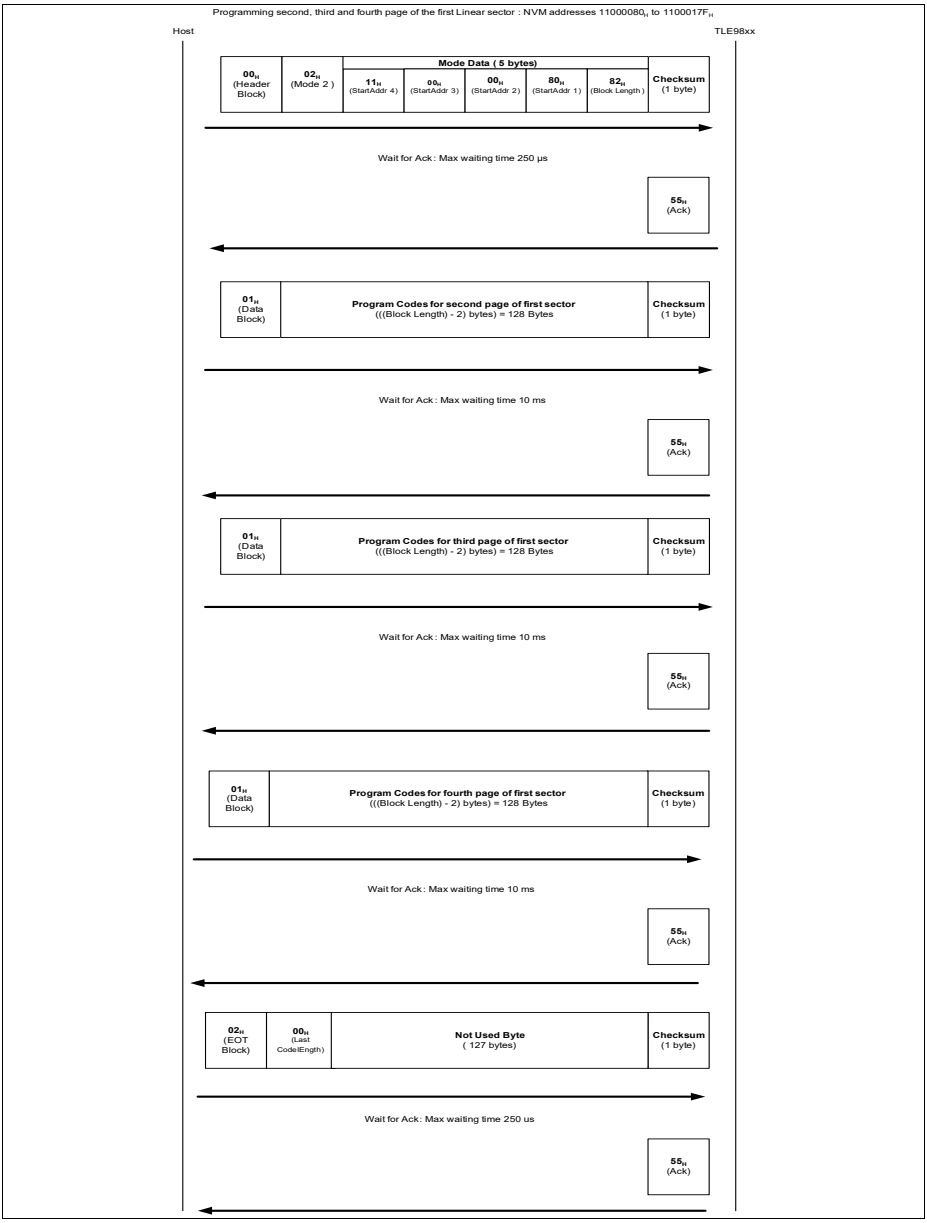


Figure 4-3 Multiple NVM page program via working mode 2

4.4.2.5 Mode 3 - Code execution inside NVM

Mode 3 is used to execute a user program in the NVM of the microcontroller at the address pointed by the NVM location 11000004_H. The header block for this working mode has the following structure:

The header block

00_H (Header Block)	**03_H** (Mode 3)	**Mode Data (5 bytes)**	**Checksum** (1 byte)
		Not Used	

Mode Data Description:

Not used: The five bytes are not used and will be ignored in Mode 3.

In working mode 3, the header block is the only transfer block to be sent by the host, no further serial communication is necessary. The microcontroller will exit the UART BSL mode, set the vector table in NVM at the address 11000000_H and jump to the address pointed by the NVM location 11000004_H.

Note: Jump to NVM will only occur when either (1) NVM is not protected and NVM content at 11000004_H is not FF_H or (2) when NVM is protected. In all other cases, firmware will put the device in Sleep mode.

4.4.2.6 Mode 4 - NVM erase

Mode 4 is used to erase different areas of the NVM. It supports mass erase of all the NVM sectors, individual erase of the sectors for linear area or for non-linear area and single page erase. This is determined by the option byte. This mode is not accessible if the NVM protection is enabled.

Different options supported are:

- Option 00_H : NVM page erase
- Option 40_H : NVM sector erase
- Option $C0_H$: NVM Mass erase

The header block for NVM page erase (with option = 00_H)

00_H (Header Block)	04_H (Mode 4)	Mode Data (5 bytes)					Checksum (1 byte)
		StartAddr 4 (MSB)	StartAddr 3	StartAddr 2	StartAddr 1 (LSB	Option =00_H (1 byte)	

Mode Data Description:

Start Addr High, Low: 32-bit start address, which determines which NVM page to be erased. Address should be page aligned (bit[6:0]=0).

Option: Set to 00_H for page erase.

When the option byte = 00_H, this mode performs an erase of the NVM page specified by the provided address.

The header block for NVM sector erase: (with option = 40_H)

00_H (Header Block)	04_H (Mode 4)	Mode Data (5 bytes)					Checksum (1 byte)
		StartAddr 4 (MSB)	StartAddr 3	StartAddr 2	StartAddr 1 (LSB	Option = 40_H (1 byte)	

Mode Data Description:

Start Addr High, Low: 32-bit start address, which determines which NVM sector to be erased. Address should be sector aligned (bit[11:0]=0).

Option: Set to 40_H for sector erase.

When the option byte = 40_H, this mode performs an erase of the NVM sector specified by the provided address. The time taken to erase a sector is max 4.5 ms.

The header block for NVM mass erase: (with option = $C0_H$)

00_H (Header Block)	04_H (Mode 4)	Mode Data (5 bytes)		Checksum (1 byte)
		Not Used (4 bytes)	Option =$C0_H$ (1 byte)	

Mode Data Description:

Not used: The four bytes are not used and will be ignored in option $C0_H$.

Option: Set to $C0_H$ for mass erase.

When the option byte = $C0_H$, this mode performs a mass erase of all the NVM sectors. The time taken will be max. 4.5 ms * number of sectors, as the erase operation is done sequentially.

Notes

1. *In Mode 4, a Block Type Error will be sent, if an invalid option byte is received. Once password is set, no access to Mode 4 is allowed and Protection Error will be sent.*
2. *NAC and NAD values will also be erased and the device will no longer be accessible in UART BSL, because NAC is invalid and default NAC will be used.*

4.4.2.7 Mode 6 - NVM protection

Mode 6 is used to enable or disable the NVM protection mode by the given user-password. The header block for this working mode has the following structure:

The header block

00_H (Header Block)	**06_H** (Mode 6)	**Mode Data (5 bytes)**		**Checksum** (1 byte)
		User-password (1 byte)	**Not Used** (4 bytes)	

Mode Data Description

User-password: This byte is given by user to enable or disable NVM protection mode.

Not used: The four bytes are not used and will be ignored in Mode 6.

In Mode 6, the header block is the only transfer block to be sent by the host. If the device is unprotected, the provided user-password will be set as NVM_PASSWORD and internally stored. If MSB of the NVM_PASSWORD is 0, only Code Flash mapped sectors are protected. If the bit is 1, both Code Flash and Data Flash are protected. No further commands will be accepted until a power-up or hardware reset. Afterwards, protection mode will be enabled.

However, if the NVM is already protected, the microcontroller will deactivate the protection and erase the NVM if the user-password byte matches the stored NVM_PASSWORD byte. The time between the command and the device response will be max. 4.5 ms + 4.5 ms * number of NVM sectors, as shown in **Table 4-5**. During this time the device should remain powered and should not be reset to prevent interruption of the erase process.

If MSB of the NVM_PASSWORD is 0, only Code Flash mapped sectors are erased. If the bit is 1, both Code Flash and Data Flash are erased. No further commands will be accepted until a power-up or hardware reset. Afterwards, protection mode will be disabled.

In case NVM is protected and the given user-password does not match the stored NVM_PASSWORD, no actions will be triggered and a Protection Error (FD_H) will be returned instead of acknowledge.

Notes

1. *Password value has to be different from 00_H and FF_H. If NVM_PASSWORD is set to either 00_H or FF_H on an unprotected device, the protection will not be set and a Protection Error (FD_H) will be returned.*
2. *When disabling NVM protection, together with NVM, the NAC and NAD values are erased too. As a result, after next reset, invalid NAC and default NAD will be used and chip waits forever for the first FastLIN BSL frame. The device will no longer be accessible in UART BSL mode, because the NAC is invalid and default NAC will be used.*

Table 4-6 Erase NVM during unprotection

NVM_PASSWORD bit 7	Description
0	Only linearly mapped NVM is erased
1	Both linearly and non-linearly mapped NVM are erased

4.4.2.8 Mode A - NVM readout, Chip ID, checksum, FastLIN BSL entry command

Mode A is used to get 4 bytes Chip ID data, NVM or 100TP page read, NVM or 100TP page or NVM mass checksum check depending on the option byte value in the header block.

In addition, the get Chip ID command is used as entry command for the FastLIN BSL mode.

Different options are supported:

- Option 00_H: Get 4 bytes Chip ID
- Option 10_H: NVM page checksum check
- Option 18_H: Mass NVM checksum check
- Option 50_H: 100TP page checksum check
- Option $C0_H$: NVM page read
- Option $F0_H$: 100TP page read

The header block for Get 4 byte Chip ID (option = 00_H)

00_H (Header Block)	$0A_H$ (Mode A)	Mode Data (5 bytes)		Checksum (1 byte)
		Not Used (4 bytes)	Option $=00_H$ (1 byte)	

Mode Data Description:

Not Used: These bytes are not used and will be ignored for option 00_H.

Option: Set to 00_H for Get 4 byte Chip ID.

If this command is successfully received, microcontroller will return an acknowledge followed by 4 data bytes and a single byte checksum. The order of the 4 bytes of data are SFR ID, CHIP_ID2, CHIP_ID1 and CHIP_ID0. Refer to the **Section 5.2.1** for CHIP_ID definition.

Note: The checksum is calculated on the acknowledge and the 4 data bytes.

The header block for Get 4 byte Chip ID as FastLIN entry (option = 00_H)

In order to avoid unwanted entries, the FastLIN connection is established only if this command is successfully received during the active BSL connection window defined by the NAC. This command must then be the first FastLIN BSL command to be sent by the Host.

00_H (Header Block)	$0A_H$ (Mode A)	Mode Data (5 bytes)					Checksum (1 byte)
		NAD (1 byte)	BSL Entry Key			Option $=00_H$ (1 byte)	
			´B` $=42_H$ (1 byte)	`S´ $=53_H$ (1 byte)	`L´ $=4C_H$ (1 byte)		

Mode Data Description:

NAD: Node address for diagnostic, specifies the address of the active slave node. See the **Section 3.1.9**.

BSL Entry Key: “BSL” in ASCII.

Option: Set to 00_H for Get 4 byte Chip ID.

If this command is successfully received, the microcontroller returns an acknowledge followed by 4 data bytes and a single byte checksum. The order of the 4 bytes of data are SFR ID, CHIP_ID2, CHIP_ID1 and CHIP_ID0. Refer to the **Section 5.2.1** for CHIP_ID definition.

On successful completion of the sequence, FastLIN BSL is fully entered and all other commands can be executed.

Note: The checksum is calculated on the acknowledge and the 4 data bytes.

The header block for NVM page checksum check (option = 10_H)

00_H (Header Block)	$0A_H$ (Mode A)	Data Area					Checksum (1 byte)
		StartAddr High (1 byte)	**StartAddr Low** (1 byte)	**Expected CHKSum High** (1 byte)	**Expected CHKSum Low** (1 byte)	**Option =10_H** (1 byte)	

Mode Data Description:

Start Addr High, Low: Address of the NVM page for checksum check. (Address should be page aligned, bit[6:0]=0.)

Expected CHKSum High, Low: Expected checksum High/Low byte.

Option: Set to 10_H to enable NVM page checksum check.

Note: The start address provided with the header block must be shifted by 7 bits to the left and then added to the NVM start address to build the actual address, i.e. it is calculated as follows in Mode A option $C0_H$:
Actual address = 11000000_H + (StartAddrHigh << 15) + (StartAddrLow << 7).

This option will trigger a checksum calculation (16 bits inverted XOR, refer to the **Section 4.4.3**) over the whole page pointed by the address given in the header block and the result will then be compared with the expected checksum (provided as well by the user in the header frame). If the given address is a valid NVM address, the microcontroller will return an acknowledge followed by four data bytes and a single byte checksum. The bytes are, in sequential order, pass/fail indication (00_H if the calculated and expected checksum match, 80_H if they differ), calculated checksum High byte, calculated checksum Low byte, and a final byte equal to 00_H.

Note: The checksum is calculated on the acknowledge and the 4 data bytes.

The input address should always be page aligned. In case it is not aligned, the address will be internally changed to point to the beginning of the addressed page so that checksum is always evaluated on a complete page.

In case the provided address is not a valid NVM address, the microcontroller will return a Block Type Error (FF_H) instead of an acknowledge (55_H) followed by no further bytes.

Note: In case the address is pointing to an erased non linearly mapped page, the address is considered invalid and a Block Type Error (FF_H) is returned.

The header block for Mass checksum check (option = 18_H)

00_H (Header Block)	$0A_H$ (Mode A)	Mode Data (5 bytes)					Checksum (1 byte)
		Not Used (1 byte)	Not Used (1 byte)	Expected CHKSum High (1 byte)	Expected CHKSum Low (1 byte)	Option $=18_H$ (1 byte)	

Mode Data Description:

Not Used: These bytes are not used and will be ignored for option 18_H.

Expected CHKSum High, Low: Expected checksum High/Low byte.

Option: Set to 18_H to enable mass checksum check.

This option will trigger a checksum calculation (16 bits inverted XOR, refer to the **Section 4.4.3**) over all the linearly mapped sectors (including erased pages and sectors). The not linearly mapped sectors and 100TP pages are not included. The result will then be compared with the expected checksum (provided by the user in the header frame). The microcontroller will return an acknowledge followed by four data bytes and a single byte checksum. The bytes are, in sequential order, pass/fail indication (00_H if the calculated and expected checksum match, 80_H if they differ), calculated checksum High byte, calculated checksum Low byte, and a final byte equal to 00_H.

Note: The checksum is calculated on the acknowledge and the 4 data bytes.

The header block for 100TP page checksum check (option = 50_H)

00_H (Header Block)	$0A_H$ (Mode A)	Data Area					Checksum (1 byte)
		100TP Page (1 byte)	Not Used (1 byte)	Expected CHKSum High (1 byte)	Expected CHKSum Low (1 byte)	Option $=50_H$ (1 byte)	

Mode Data Description:

100TP Page: Selection of the 100TP page to be checked (refer to **Figure 5-8**).

Not Used: This byte is not used and will be ignored for option 50_H.

Expected CHKSum High, Low: Expected checksum High/Low byte.

Option: Set to 50_H to enable 100TP page checksum check.

This option will trigger a checksum calculation (16 bits inverted XOR, refer to the **Section 4.4.3**) over the whole 100TP page pointed by the address given in the header block and the result will then be compared with the expected checksum (provided as well by the user in the header frame). The 100TP page address has to be in accordance with the configuration sector address scheme described in the **Figure 5-8**. If the given address is valid, the microcontroller will return an acknowledge followed by four data bytes and a single byte checksum. The bytes are, in sequential order, pass/fail indication (00_H if the calculated and expected checksum match, 80_H if they differ), calculated checksum High byte, calculated checksum Low byte, and a final byte equal to 00_H.

In case the provided address is not valid, the microcontroller will return a Block Type Error (FF_H) instead of an acknowledge (55_H) followed by no further bytes.

Note: The checksum is calculated on the acknowledge and the 4 data bytes.

The header block for NVM page read (option $C0_H$)

00_H (Header Block)	$0A_H$ (Mode A)	Mode Data (5 bytes)					Checksum (1 byte)
		StartAddr High (1 byte)	**StartAddr Low** (1 byte)	**Not Used** (1 byte)	**Not Used** (1 byte)	**Option =$C0_H$** (1 byte)	

Mode Data Description:

Start Addr High, Low: Address of the NVM page to be read (Address should be page aligned, bit[6:0]=0.)

Not Used: These bytes are not used and will be ignored for option $C0_H$.

Option: Set to $C0_H$ to enable NVM page read.

Note: The start address provided with the header block has to be shifted by 7 bits to the left and then added to the NVM start address to build the actual address, i.e. it is calculated as follows in Mode A option $C0_H$:
Actual address = 11000000_H + (StartAddrHigh << 15) + (StartAddrLow << 7).

This option will trigger a read of the addressed NVM page. Microcontroller will return an acknowledge (55_H) followed by the 128 NVM page data bytes (starting from the least significant byte of the page).

The input address should always be aligned with a page. In case it is not aligned, the address will be internally changed to point to the beginning of the addressed page so that the page bytes are always returned ordered from the least to the most significant byte.

In case the provided address is not a valid NVM address, the microcontroller will return a Block Type Error (FF_H) instead of an acknowledge (55_H) followed by no further bytes.

To prevent user code to be read, this option is disabled if NVM is protected and only a Protection Error byte (FD_H) will be returned.

Note: In case the address is pointing to an erased non linearly mapped page, the address is considered invalid and a Block Type Error (FF_H) is returned.

The header block for 100TP page read (option = $F0_H$)

00_H (Header Block)	$0A_H$ (Mode A)	Mode Data (5 bytes)					Checksum (1 byte)
		Not Used (1 byte)	Not Used (1 byte)	Not Used (1 byte)	100TP Page (1 byte)	Option =$F0_H$ (1 byte)	

Mode Data Description:

Not Used: These bytes are not used and will be ignored for option $F0_H$.

100TP Page: Selection of the 100TP page to be checked (refer to **Figure 5-8**).

Option: Set to $F0_H$ to enable 100TP page read.

This option will trigger a read of the addressed 100TP page. Microcontroller will return an acknowledge (55_H) followed by the 128 100TP page data bytes (starting from the least significant byte of the page).

The 100TP page is selected by the CS page byte according to the scheme shown in **Figure 5-8**.

In case an invalid 100TP page is selected the microcontroller will return a Block Type Error (FF_H) instead of an acknowledge (55_H) followed by no further bytes.

To prevent user code to be read, this option is disabled if NVM is protected (NVM password installed) and only a Protection Error byte (FD_H) will be returned.

All other values for option byte

Block Type Error indication (FF_H) is sent back.

In Mode A, the header block is the only transfer block to be sent by the host. The microcontroller will return an acknowledge followed by data bytes if the header block is received successfully. If an invalid option is received, the microcontroller will return a Block Type Error indication (FF_H) and no further bytes.

4.4.3 16 bits inverted XOR checksum

This checksum structure is used in BSL Mode A options 10_H, 18_H, 50_H as a fast data integrity check. These modes will read the specified NVM range, calculate the checksum and compare it against the expected one provided as command parameter.

To calculate this checksum, all Half-Words (16 bits) of the selected NVM region are xored. The resulting value is then logically complemented (1´s complement).

The following figure shows the calculation algorithm.

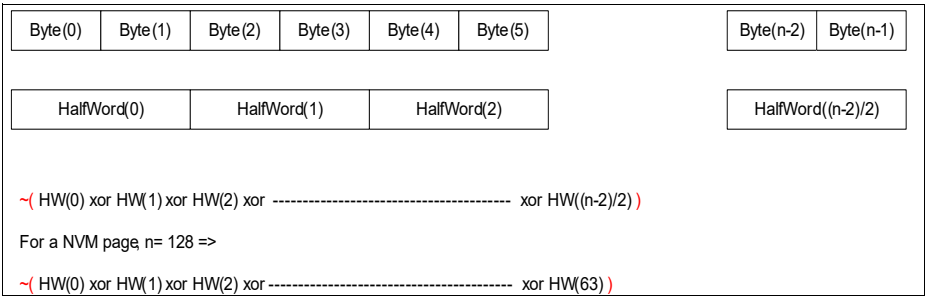


Figure 4-4 16 bits inverted XOR checksum calculation

4.5 WDT1 refreshing

After a reset the WDT1 is starting with a long open window. WDT1 keeps on running while waiting for first UART frame. In case during the UART BSL waiting time, defined by NAC, a UART communication is detected, WTD1 is disabled and its status frozen.

Subsequently, before exiting to RAM or NVM in UART BSL modes 1 and 3 the watchdog is re-enabled and starts from the previously frozen state. The WDT1 is then still in long open window and the remaining valid time is equal to long open window minus the time between reset release and first UART communication. User program needs to trigger the WDT1 refresh accordingly.

5 NVM

Non Volatile Memory (NVM) is the flash module of the TLE986x/TLE987x which partly supports EEPROM emulation.

5.1 NVM overview

The NVM is a single block of NVM memory of up to 256 Kbytes separated into Code and Data space. The following table shows the NVM address range.

Table 5-1 NVM address range

Address	Address range
NSA to NEA	NVM memory

NSA and NEA values are shown in **Table 5-2**:

Table 5-2 NVM size and address range

NVM size (Kbyte)	NVM starting address (NSA)	NVM linear size, NVM CFLASH size (NLS)	NVM DFLASH starting address	NVM DFLASH size	NVM DFLASH end address, NVM FLASH end address (NEA)
36	11000000_H	8000_H	11008000_H	1000_H	$11008FFF_H$
64	11000000_H	$F000_H$	$1100F000_H$	1000_H	$1100FFFF_H$
128	11000000_H	$1F000_H$	$1101F000_H$	1000_H	$1101FFFF_H$
256	11000000_H	$3F000_H$	$1103F000_H$	1000_H	$1103FFFF_H$

Notes

1. *An erased page is ECC-Clean and will not generate an ECC error.*
2. *Reading an erased page in the Code space will return FF_H and will not trigger any error.*
3. *Reading an erased page in the Data space will return 00_H and will also create an NVM Map Error NMI, if enabled in NMICON. As a consequence, an erased page in the Data space has to be written before it can be read without triggering an error.*

5.1.1 NVM organisation

The NVM has 2 types of memory configuration, Code and Data. It is organised in sectors. Each NVM Sector is a block of 4 Kbytes organised into blocks of 128 bytes called page. The page is the minimum data granularity for NVM (Code and Data) write and erase so, with this NVM structure, any NVM update, even when targeting only one byte, actually

involves 128 bytes. **Table 5-3** shows the sector address organisation of the first and last 4 sectors of the 256 Kbytes NVM. The other sector organization can be simply derived per extension of the reported scheme. **Table 5-4** shows the page address organisation of NVM Sector 1 and it can be used as a reference for page organization of any NVM Sector.

Table 5-3 NVM memory sector organisation

Address	NVM Sector number
11000000H to 11000FFFH	1
11001000H to 11001FFFH	2
11002000H to 11002FFFH	3
11003000H to 11003FFFH	4
1103C000H to 1103CFFFH	61
1103D000H to 1103DFFFH	62
1103E000H to 1103EFFFH	63
1103F000H to 1103FFFFH	64

Table 5-4 NVM memory sector 1 page organisation

Address	Page number of NVM Sector
11000000H to 1100007FH	0
11000080H to 110000FFH	1
11000100H to 1100017FH	2
11000180H to 110001FFH	3
11000200H to 1100027FH	4

Table 5-4 NVM memory sector 1 page organisation (cont'd)

Address	Page number of NVM Sector
11000280H to 110002FFH	5
11000300H to 1100037FH	6
11000380H to 110003FFH	7
11000400H to 1100047FH	8
11000480H to 110004FFH	9
11000500H to 1100057FH	10
11000580H to 110005FFH	11
11000600H to 1100067FH	12
11000680H to 110006FFH	13
11000700H to 1100077FH	14
11000780H to 110007FFH	15
11000800H to 1100087FH	16
11000880H to 110008FFH	17
11000900H to 1100097FH	18
11000980H to 110009FFH	19
11000A00H to 11000A7FH	20
11000A80H to 11000AFFH	21

Table 5-4 NVM memory sector 1 page organisation (cont'd)

Address	Page number of NVM Sector
11000B00H to 11000B7FH	22
11000B80H to 11000BFFH	23
11000C00H to 11000C7FH	24
11000C80H to 11000CFFH	25
11000D00H to 11000D7FH	26
11000D80H to 11000DFFH	27
11000E00H to 11000E7FH	28
11000E80H to 11000EFFH	29
11000F00H to 11000F7FH	30
11000F80H to 11000FFFH	31

5.2 NVM configuration sectors organisation

The configuration sector contains important user data needed for proper system initialization.

5.2.1 Chip ID definition

The specific characteristics of the different variants of the product family are captured in the definition of the CHIP_ID bytes.

The Chip_ID bytes can be read via BSL mode A. When triggered, this mode replies providing the 3 CHIP_ID bytes plus the content of the Identification register (ID).

Please refer to the following tables for CHIP_ID details. This is a variant specific identification number. The unique device specific identification number is described in **Table 5-11**.

Table 5-5 Chip ID byte 0

Res	MAX_FREQ	OP_AMP	Phases	DMA	PKG Type

Table 5-6 Chip ID byte 0 bits description

Field	Bits	Description
PKG_Type	[1:0]	Package type 00 VQFN-48 01 TQFP-48 10 Reserved 11 Reserved
DMA	2	DMA 0 With DMA 1 Without DMA
Phases	3	Bridge driver number of phases 0 2 phases 1 3 phases
OP_AMP	4	Op Amp 0 With Op Amp 1 Without Op Amp
Max Freq	[6:5]	Maximum frequency 00 Reserved 01 20 MHz 10 24 MHz 11 40 MHz
Res	7	Reserved

Table 5-7 Chip ID byte 1

NVM_SIZE	EEPROM_SIZE

Table 5-8 Chip ID byte 1 bits description

Field	Bits	Description
EEPROM_SIZE	[3:0]	EEPROM (non-linearly mapped NVM) size 0000 0 Kbyte 0001 4 Kbyte 0010 8 Kbyte 0011 12 Kbyte 0100 16 Kbyte 0101 20 Kbyte 0110 24 Kbyte 0111 28 Kbyte 1000 32 Kbyte 1001 36 Kbyte 1010 40 Kbyte 1011 44 Kbyte 1100 48 Kbyte 1101 52 Kbyte 1110 56 Kbyte 1111 60 Kbyte
NVM_SIZE	[7:4]	Total NVM size 0000 Res 0001 256 Kbyte 0010 Res 0011 36 Kbyte 0100 Res 0101 Res 0110 Res 0111 64 Kbyte 1000 Res 1001 Res 1010 Res 1011 Res 1100 Res 1101 Res 1110 Res 1111 128 Kbyte

Table 5-9 Chip ID byte 2

Res	VARIANT_ID

Table 5-10 Chip ID byte 2 bits Description

Field	Bits	Description
VARIANT_ID	[3:0]	Variant ID
Res	[7:4]	Reserved

5.2.2 100 Time Programmable data

User has eight 100 time programmable pages (100TP). The first one is used to store user configuration parameters for measurement interface and sense amplifier as well as ADC1 calibration parameters. These parameters are usually determined in the user application and might require several iterations before the best fit is found.

The values of the first page, from offset 10_H to 63_H, are automatically copied into the dedicated SFR registers after every power-on reset, brown-out reset or wake-up reset from Sleep mode 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.

The first 4 bytes of the first 100TP page are used to store a device ID that can be read by the user. The content of these 4 bytes are preloaded prior to shipment and cannot be modified by the user. In case the user tries to write these values via the 100TP page writing features offered in BSL or via NVM user routine, an error is reported and the original content of the bytes is preserved. The Customer_ID definition is described in **Figure 5-1**.

The data stored in this first 100TP page can be found in **Table 5-11**.

To read data stored in the 100TP pages, refer to the **Section 5.3.11**.

To perform the programming of these pages, the user is required to preload the contents to be programmed into the RAM as listed in **Table 5-12**. The offset entered for the programming does not need to be in sequential order. Once a page has been programmed 100 times, no further programming on that page is allowed. In the last byte of each 100TP page a program counter is stored (not changeable by user).

Since part of the data stored in the first 100TP page are used as trim or configuration data at reset, the content is protected by an in-page checksum (XOR of the first 126 bytes of the page) stored at one before last byte of the page (in-page offset $7E_H$, refer to **Table 5-11**). User is required to properly calculate and update the checksum when re-programming the 100TP page 1.

Note: At power-on reset, brown-out reset or wake-up reset from Sleep mode, the firmware checks the checksum of the first 100TP page. In case the checksum is not correct, The data stored from offset 10_H to 63_H are not downloaded into the SFRs and backup values are used instead.

For further information regarding 100TP page program, refer to the **Section 5.3.12**.

Table 5-11 100TP page 1

Data offset	SFR / Variable name	Description	Default value	Back-up value[1)]
00_H to 03_H	CUSTOMER_ID	Device ID for user	Device ID dependent	N.A.
04_H	GAIN_VS_10B	Calibration gain for supply voltage measurement	Chip Individual	N.A.
05_H	OFFSET_VS_10B	Calibration offset for supply voltage measurement	Chip Individual	N.A.
06_H	GAIN_VBAT_ SENSE_10B	Calibration gain for battery voltage measurement	Chip Individual	N.A.
07_H	OFFSET_VBAT_ SENSE_10B	Calibration offset for battery voltage measurement	Chip Individual	N.A.
08_H	GAIN_VMON_ ATT_1_5	Calibration gain for high voltage monitoring input voltage measurement	Chip Individual	N.A.
09_H	OFFSET_VMON_ ATT_1_5	Calibration offset for high voltage monitoring input voltage measurement	Chip Individual	N.A.
$0A_H$	CONFIG_VERS	Configuration Sector version	02_H	N.A.
$0B_H$	Reserved	Reserved	00_H	N.A.
$0C_H$ to $0D_H$	CFLASH_PW	Code Flash protection removal password	0000_H	N.A.
$0E_H$ to $0F_H$	DFLASH_PW	Data Flash protection removal password	0000_H	N.A.
10_H to 13_H	MEAS_ADC2_ CTRL1	Measurement unit: Control register 1	00000000_H	00000000_H

Table 5-11 100TP page 1 (cont'd)

Data offset	SFR / Variable name	Description	Default value	Back-up value[1)]
14_H to 17_H	MEAS_ADC2_ CTRL2	Measurement unit: Control register 2	00000703_H	00000703_H
18_H to $1B_H$	MEAS_ADC2_ SQ1_4	Channel controller: Measurement channel enable bits of cycle 1 to 4	29362837_H	29362837_H
$1C_H$ to $1F_H$	MEAS_ADC2_ SQ5_8	Channel controller: Measurement channel enable bits of cycle 5 to 8	28372836_H	28372836_H
20_H to 23_H	MEAS_ADC2_ SQ9_10	Channel controller: Measurement channel enable bits of cycle 9 to 10	00002936_H	00002936_H
24_H to 27_H	ADC2_CAL_CH0_1	Calibration unit: Calibration of channel 0 and 1	Chip Individual	Chip Individual
28_H to $2B_H$	ADC2_CAL_CH2_3	Calibration unit: Calibration of channel 2 and 3	Chip Individual	Chip Individual
$2C_H$ to $2F_H$	ADC2_CAL_CH4_5	Calibration unit: Calibration of channel 4 and 5	Chip Individual	Chip Individual
30_H to 33_H	ADC2_ FILTCOEFF0_5	IIR filter: Filter coefficients of ADC channels 0 to 5	$00000AAA_H$	$00000AAA_H$
34_H to 37_H	ADC2_FILT_UP_ CTRL	Postprocessing: Upper threshold filter enable	$00000F3F_H$	$00000F3F_H$
38_H to $3B_H$	ADC2_FILT_LOW_ CTRL	Postprocessing: Lower threshold filter enable	$00000F3F_H$	$00000F3F_H$
$3C_H$ to $3F_H$	ADC2_TH0_3_ LOWER	Postprocessing: Lower comparator trigger level of channels 0 to 3	$182F423A_H$	$182F423A_H$

Table 5-11 100TP page 1 (cont'd)

Data offset	SFR / Variable name	Description	Default value	Back-up value[1)]
40_H to 43_H	ADC2_TH4_5_ LOWER	Postprocessing: Lower comparator trigger level of channels 4 to 5	$00009A00_H$	$00009A00_H$
44_H to 47_H	ADC2_TH6_9_ LOWER	Postprocessing: Lower comparator trigger level of channels 6 to 9	$C6D39ECD_H$	$C6D39ECD_H$
48_H to $4B_H$	ADC2_TH0_3_ UPPER	Postprocessing: Upper comparator trigger level of channels 0 to 3	$ABBDC5C0_H$	$ABBDC5C0_H$
$4C_H$ to $4F_H$	ADC2_TH4_5_ UPPER	Postprocessing: Upper comparator trigger level of channels 4 to 5	$0000BC00_H$	$0000BC00_H$
50_H to 53_H	ADC2_CNT0_3_ LOWER	Postprocessing: Lower counter trigger level of channels 0 to 3	12131312_H	12131312_H
54_H to 57_H	ADC2_CNT4_5_ LOWER	Postprocessing: Lower counter trigger level of channels 4 to 5	$00000A0A_H$	$00000A0A_H$
58_H to $5B_H$	ADC2_CNT0_3_ UPPER	Postprocessing: Upper counter trigger level of channels 0 to 3	$12131B1A_H$	$12131B1A_H$
$5C_H$ to $5F_H$	ADC2_CNT4_5_ UPPER	Postprocessing: Upper counter trigger level of channels 4 to 5	00001212_H	00001212_H
60_H to 63_H	ADC2_MMODE0_5	Postprocessing: Overvoltage measurement mode of channels 0 to 5	00000000_H	00000000_H
64_H to $6B_H$	Reserved	Reserved	00_H	N.A.
$6C_H$	CHIP_ID_BYTE_00	Chip ID byte 00[2)]	Chip Individual	N.A.
$6D_H$	CHIP_ID_BYTE_01	Chip ID byte 01[2)]	Chip Individual	N.A.

Table 5-11 100TP page 1 (cont'd)

Data offset	SFR / Variable name	Description	Default value	Back-up value[1)]
$6E_H$	CHIP_ID_BYTE_02	Chip ID byte 02[2)]	Chip Individual	N.A.
$6F_H$	CHIP_ID_BYTE_03	Chip ID byte 03[2)]	Chip Individual	N.A.
70_H	CHIP_ID_BYTE_04	Chip ID byte 04[2)]	Chip Individual	N.A.
71_H	CHIP_ID_BYTE_05	Chip ID byte 05[2)]	Chip Individual	N.A.
72_H	CHIP_ID_BYTE_06	Chip ID byte 06[2)]	Chip Individual	N.A.
73_H	CHIP_ID_BYTE_07	Chip ID byte 07[2)]	Chip Individual	N.A.
74_H	CHIP_ID_BYTE_08	Chip ID byte 08[2)]	Chip Individual	N.A.
75_H	CHIP_ID_BYTE_09	Chip ID byte 09[2)]	Chip Individual	N.A.
76_H	CHIP_ID_BYTE_10	Chip ID byte 10[2)]	Chip Individual	N.A.
77_H	CHIP_ID_BYTE_11	Chip ID byte 11[2)]	Chip Individual	N.A.
78_H	CS_SA_WITH_PROT_EN	When set to $A5_H$, enables Service Algorithm even on protected NVM Data Sector	00_H	N.A.
79_H	CS_USER_CAL_STARTUP_EN	Enable byte for user calibration data download during startup. If value=0xC3 then the download is enabled	00_H	N.A.
$7A_H$	CS_USER_CAL_XADDH	High byte of the RAM starting address where downloaded data has to be stored (0xF0 for RAM initial address)	00_H	N.A.

Table 5-11 100TP page 1 (cont'd)

Data offset	SFR / Variable name	Description	Default value	Back-up value[1)]
$7B_H$	CS_USER_CAL_XADDL	LOW byte of the RAM starting address where downloaded data has to be stored (0x00 for RAM initial address)	00_H	N.A.
$7C_H$	CS_USER_CAL_100TP_PAGE	100TP page where calibration data has to be downloaded from. By default 100TP page1 should be used (Value=0x11)	00_H	N.A.
$7D_H$	CS_USER_CAL_NUM	Number of bytes to be downloaded starting from the first byte of the selected 100TP page	00_H	N.A.
$7E_H$	CHECKSUM_100TP_P1	XOR of the first 126 bytes of 100TP page 1	Chip Individual	N.A.
$7F_H$	PROG_TIMES_100TP_P1	This reflects the number of times that this page has been programmed (up to a maximum of 100 times)	00_H	N.A.

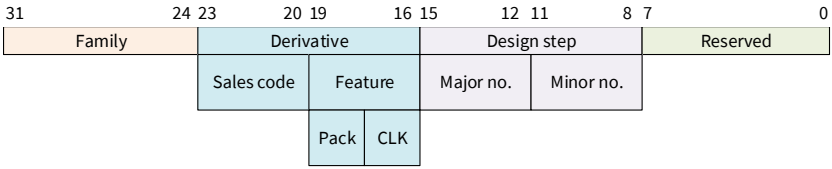
1) Values used during startup for analog module trimming in case a checksum error on 100TP page 1 is detected. Refer to the **Section 3.1.5**.

2) This is a unique device specific identification number. The variant specific identification number is described in the **Section 5.2.1**.

Table 5-12 RAM preloading for 100TP page programming

RAM address	Function
18000400_H	Number of bytes to be programmed (i.e. **N**, up to a maximum of 127 bytes[1)])
18000401_H	100TP offset 1
18000402_H	100TP data 1 to be programmed
18000403_H	100TP offset 2
18000404_H	100TP data 2 to be programmed
...	...
18000401_H + ((N-1) x 2)	100TP offset N
18000402_H + ((N-1) x 2)	100TP data N to be programmed

1) The maximum number of bytes that the user can load into the 100TP pages is limited to 127 since last byte is used as a program operation counter. To ensure that the page are not programmed more than 100 times, even not by accident, the counter byte (last byte in the page) can be read but not overwritten by the user.



Family [Decimal values]	Design step (major) [Hex values]	Design step (minor) [Hex values]
0: Reserved 1: 2-phase (TLE986x) 2: 3-phase (TLE987x) 7: 2-phase (TLE986x, UH-Step, UI-Step) 8: 3-phase (TLE987x, UH-Step, UI-Step) Others: Reserved	A: Design step A, Design step U* B: Design step B Others: Reserved	A: Design step A, Design step H* B: Design step B, Design step I* C: Design step C D: Design step D E: Design step E F: Design step F Others: Reserved
	* valid for TLE986x/7x variants, except the TLE98x2	

Derivative – Sales code [Decimal values]	Derivative – Feature – Pack. [Decimal values]	Derivative – Feature – CLK [Decimal values]
0: TLE98x0 1: TLE98x1 2: TLE98x2 3: TLE98x3 4: TLE98x4 5: TLE98x5 6: TLE98x6 7: TLE98x7 8: TLE98x8 9: TLE98x9 Others: Reserved	0: VQFN48QX 1: TQFP48 Others: Reserved	0: 20 MHz 1: 24 MHz 2: 40 MHz 3: Reserved

Figure 5-1 Customer_ID definition

5.3 NVM user routines organisation

The NVM user routines are BootROM routines called by user and placed from the address $0000383D_H$ to 00003925_H. The complete list of NVM user routines can be found in **Table 5-13**.

Table 5-13 NVM user routines list

Address	Routine	Description
00003925_H	USER_CFLASH_WR_PROT_EN	To enable write protection on the linearly mapped NVM sectors
$0000391D_H$	USER_CFLASH_WR_PROT_DIS	To disable write protection on the linearly mapped NVM sectors
00003915_H	USER_CFLASH_RD_PROT_EN	To enable read protection on the linearly mapped NVM sectors
$0000390D_H$	USER_CFLASH_RD_PROT_DIS	To disable read protection on the linearly mapped NVM sectors
00003905_H	USER_DFLASH_WR_PROT_EN	To enable write protection on the non linearly mapped NVM sectors
$000038FD_H$	USER_DFLASH_WR_PROT_DIS	To disable write protection on the non linearly mapped NVM sectors
$000038F5_H$	USER_DFLASH_RD_PROT_EN	To enable read protection on the non linearly mapped NVM sectors
$000038ED_H$	USER_DFLASH_RD_PROT_DIS	To disable read protection on the non linearly mapped NVM sectors
$000038E5_H$	USER_OPENAB	To open the assembly buffer for writing
$000038DD_H$	USER_PROG	To program the NVM
$000038D5_H$	USER_ERASEPG	To erase an NVM page
$000038CD_H$	USER_ABORTPROG	To abort the NVM programming by closing the assembly buffer
$000038C5_H$	USER_NVMRDY	To access if the NVM is in ready to read status
$000038BD_H$	USER_READ_CAL	To read the NVM calibration data
$000038B5_H$	USER_NVM_CONFIG	To read the NVM configuration status
$000038AD_H$	USER_NVM_ECC2ADDR	To read the NVM ECC2 address
$0000389D_H$	USER_MAPRAM_INIT	To initialize MapRAM

Table 5-13 NVM user routines list (cont'd)

Address	Routine	Description
00003895_H	USER_VERIFY_PAGE 1)	To perform a page verify
$0000388D_H$	USER_ERASE_SECTOR_VERIFY 1)	To perform a sector erase verify
00003885_H	USER_ERASEPG_VERIFY 1)	To perform a page erase verify
00003875_H	USER_READ_100TP	To read the NVM 100TP parameter data
$0000386D_H$	USER_100TP_PROG	To perform the 100TP program (this can be used 100 times per 100TP page)
00003865_H	USER_ERASE_SECTOR	To erase an NVM sector
00003855_H	USER_NVMCLKFAC_SET	To set NVMCLKFAC bit in SYSCON0
$0000384D_H$	USER_RAM_MBIST_START	To perform a sequential checkerboard and inverted checkerboard test on the RAM
00003845_H	USER_NVM_ECC_CHECK	To trigger a complete NVM read and provide cumulated ECC single bit error indication
$0000383D_H$	USER_ECC_CHECK	To provide cumulated ECC single bit error indication since last call of the function

1) This function is not available for variants with 256 Kbyte flash.

Table 5-14 NVM user routines maximum stack usage

Routine	Maximum stack usage BF, and A step [bytes]	Maximum stack usage UH step [bytes]	Maximum stack usage UI step [bytes]
USER_CFLASH_WR_PROT_EN	0004_H	0004_H	0004_H
USER_CFLASH_WR_PROT_DIS	0004_H	0004_H	0004_H
USER_CFLASH_RD_PROT_EN	0004_H	0004_H	0004_H
USER_CFLASH_RD_PROT_DIS	0004_H	0004_H	0004_H

Table 5-14 NVM user routines maximum stack usage (cont'd)

Routine	Maximum stack usage BF, and A step [bytes]	Maximum stack usage UH step [bytes]	Maximum stack usage UI step [bytes]
USER_DFLASH_WR_PROT_EN	0004_H	0004_H	0004_H
USER_DFLASH_WR_PROT_DIS	0004_H	0004_H	0004_H
USER_DFLASH_RD_PROT_EN	0004_H	0004_H	0004_H
USER_DFLASH_RD_PROT_DIS	0004_H	0004_H	0004_H
USER_OPENAB	$003C_H$	$003C_H$	$003C_H$
USER_PROG	$00BC_H$	0100_H	$00FC_H$
USER_ERASEPG	$005C_H$	$00C0_H$	$00BC_H$
USER_ABORTPROG	0014_H	0014_H	0014_H
USER_NVMRDY	0004_H	0004_H	0004_H
USER_READ_CAL	0034_H	0034_H	0034_H
USER_NVM_CONFIG	0010_H	0010_H	0010_H
USER_NVM_ECC2ADDR	0010_H	0010_H	0010_H
USER_MAPRAM_INIT	0024_H	0024_H	0024_H
USER_VERIFY_PAGE[1)]	$003C_H$	$003C_H$	$003C_H$
USER_ERASE_SECTOR_VERIFY[1)]	$005C_H$	$005C_H$	$005C_H$
USER_ERASEPG_VERIFY[1)]	0034_H	0034_H	0034_H
USER_READ_100TP	0034_H	0034_H	0034_H
USER_100TP_PROG	0088_H	$00C0_H$	$00BC_H$
USER_ERASE_SECTOR	0038_H	$00A0_H$	$009C_H$
USER_NVMCLKFAC_SET	$000C_H$	$000C_H$	$000C_H$
USER_RAM_MBIST_START	$01E4_H$	$01E4_H$	$01E4_H$
USER_NVM_ECC_CHECK	0024_H	0024_H	0024_H
USER_ECC_CHECK	0024_H	0024_H	0024_H

[1)] This function is not available for variants with 256 Kbyte flash.

5.3.1 Opening assembly buffer routine

The NVM programming routine consists of two parts: the assembly buffer opening routine, and the programming and verification routine.

The Open Assembly buffer routine reads the content of the physical page into a NVM internal RAM memory block (Assembly Buffer). The address of the page to be read is provided with the OpenAB function call. Once the OpenAB call has been executed successfully the user can update the content of the Assembly Buffer (128 bytes) by (over)writing the data starting from the address handed over to the OpenAB function.

In case the provided address targets the NVM non-linearly mapped Data region , before copying the data, the OpenAB routine will check to which physical page the provided address is linked and make the data of this physical page available in the Assembly Buffer.

In order to prepare the data for the next program operation, the open assembly buffer routine then accesses the data stored in the MapRAM and NVM array (data block and MapBlock) related to the address provided as input. While accessing this data, the routine performs a consistency check of the read information and reports a proper warning or fail to the user by means of the bit 0, 1, 2 and 3 of the return values (**Table 5-16**). The performed check and related warnings/errors depend on the region to which the addressed page belongs, as described in **Table 5-15**:

Table 5-15 Open Assembly Buffer subroutine return value for ECC check

NVM region	ECC fail in MapRAM	ECC fail in MapBlock	ECC fail in data block
Code region	Not applicable	Not applicable	Bit 1 set, only warning, AB is opened
Data region	Bits 0 and 3 set. Error, AB not opened	Bit 2 set, only warning, AB is opened	Bit 1 set, only warning, AB is opened

Note: The assembly buffer opening routine needs to be executed successfully before the NVM programming routine can be called.

Table 5-16 Opening assembly buffer subroutine

Subroutine	**$000038E5_H$: USER_OPENAB** Prototype: unsigned char **USER_OPENAB**(int *Address)

Table 5-16 Opening assembly buffer subroutine (cont'd)

Input	***Address(int pointer)**: Pointer to the NVM address to be programmed
Output	**Return value (unsigned char):** Bit 0: Pass or fail 0 = Assembly Buffer is successfully opened 1 = Assembly Buffer cannot be opened Bit 1: Data Block ECC Pass or fail 0 = Data Block has no ECC2 fail 1 = Data Block has at least one ECC2 fail Bit 2: Map Block ECC Pass or fail 0 = Map Block has no ECC2 fail 1 = Map Block has at least one ECC2 fail Bit 2 is used only for pages in Data sector. Bit 3: MapRAM ECC Pass or fail 0 = MapRAM has no ECC2 fail 1 = MapRAM has at least one ECC2 fail Bit 3 is used only for pages in Data sector. Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reasons of failure: • Corrupted NVM data sector • The range of the address is protected • The range of the address is incorrect Possible reason for execution fail: • Routine called as nested call during the execution of another NVM routine (e.g. via RAM branching) • Assembly Buffer is already opened

The input address will be page aligned internally, bit[6:0] = don't care. Once assembly buffer is opened, user must either proceed with the standard program flow (refer to **Figure 0-4**) or close the assembly buffer using the dedicated abort programming user routine (refer to the **Section 5.3.8**). All other sequences are not allowed and might lead to loss of data. Please check the flow diagram in **Figure 5-15**, it depicts the usage of the USER_OPENAB function.

5.3.2 NVM programming routine

There are 2 types of programming available, Type 1 or Type 2 (Type 1 without or Type 2 with RAM background activity during NVM operation).

For Type 1 programming, the flow control is always kept by the BootROM NVM programming routine. Consequently, no other operations can be run in parallel thus avoiding making use of the NVM operation waiting time. In Type 2 programming, the BootROM routine starts the write operation and then gives back control to the user software by branching to the RAM address 18000400_H. In this scenario, the user software needs to reside in RAM because no access to the NVM is possible while internal program sequence is on-going. The user software needs to hand back the control to the NVM programming routine, which continues with polling the busy bit.

A description of the BootROM programming routine is provided in the following **Table 5-17**. More information on the support for background activity during NVM operation can be found in the **Section 5.4.2**.

The program operation is executed on the page selected by the previously called USER_OPENAB. If the addressed page is already programmed, then an erase will be applied as wel, see timing diagrams below. In case the target page belongs to the NVM Data region, at the end of a successful program operation, the USER_PROG routine properly updates the MapRAM information mapping the page just written and randomly selects a proper spare page between the available (not written and not faulty) pages. In case, for any reason, a valid spare page cannot be found, the routine returns a proper error indication. In such case all data previously written, including the page just written is still accessible (no data loss). Please check the flow diagram in **Figure 5-15**, it depicts the usage of the USER_PROG function.

Timing diagrams

In the following the timing diagrams for various USER_PROG usecases are displayed. The colored areas in the beginning and the end of a block represent the bootrom preamble and postamble code (these are not timing accurate).

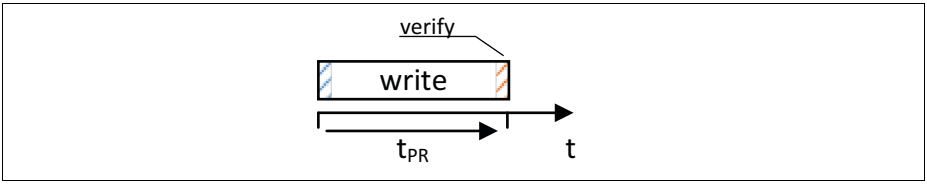


Figure 5-2 Programming an erased paged, applies to code flash as well as to data flash

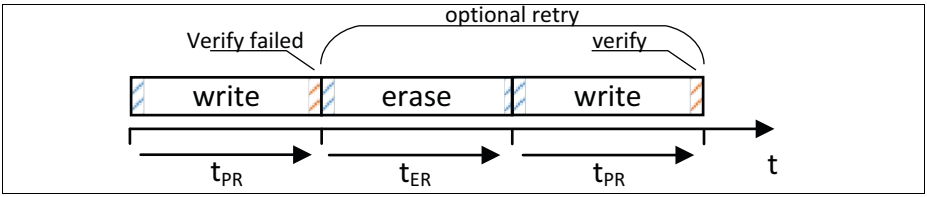


Figure 5-3 Programming an erased paged with enabled retry (corrective action = 1), applies to code flash as well as to data flash

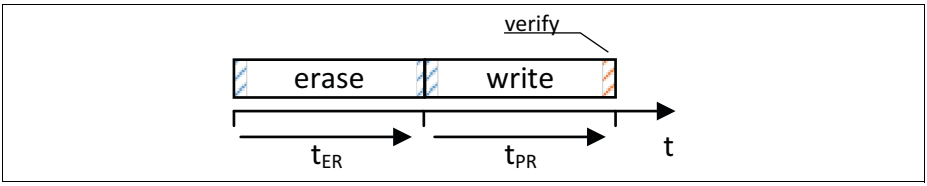


Figure 5-4 Programming an used paged, applies to code flash only

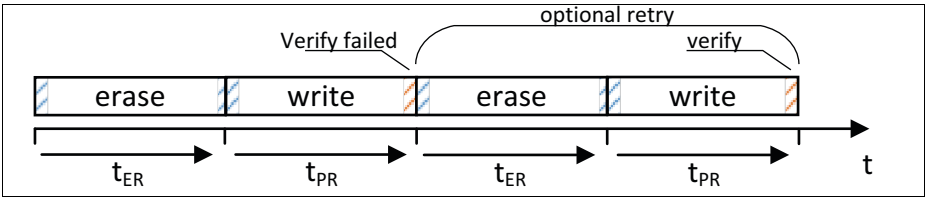


Figure 5-5 Programming an used paged with enabled retry (corrective action = 1), applies to code flash only

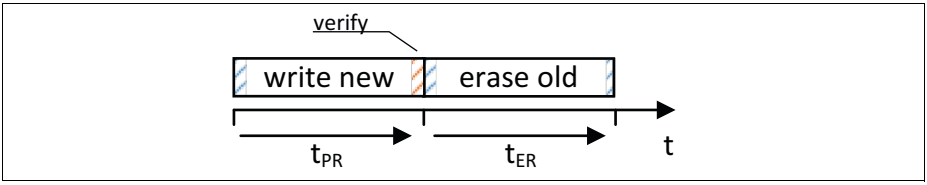


Figure 5-6 Programming an used paged, applies to data flash only

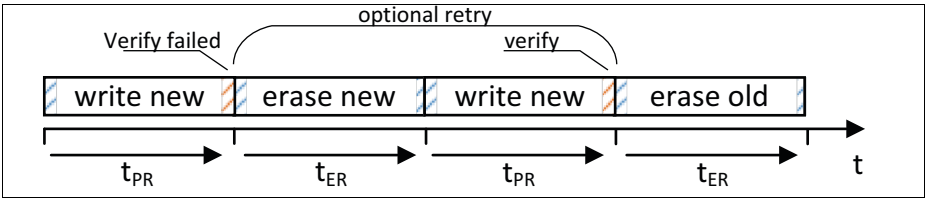


Figure 5-7 Programming an used paged with enabled retry (corrective action = 1), applies to data flash only

Table 5-17 Programming subroutine

Subroutine	**000038DD$_H$**: **USER_PROG** Prototype: unsigned char **USER_PROG**(char PROG_FLAG)

Table 5-17 Programming subroutine (cont'd)

Input	**PROG_FLAG (char)**: Byte for controlling the programing routine. Bit 0: RAM branching control bit 0 = RAM branching disabled 1 = RAM branching enabled Bit 1: Corrective action (retry and disturb handling) control bit 0 = Corrective actions disabled 1 = Corrective actions enabled Bit 2: Failing page erase control bit when addressing non linearly mapped sector (refer to the **Section 5.4.4.2** for more details) 0 = Failing page erase enabled. The programmed data are erased in case of fail. If the page was already used, old data are kept. 1 = Failing page erase disabled. Programmed data are not erased in case of fail. If page was already used, old data are not kept and the new failing data are accessible by reading the target page.

Table 5-17 Programming subroutine (cont'd)

Output	**Return value (unsigned char):** Bit 0: Pass or Fail.This bit is the OR of the bits 4, 5, 6 and 7 0 = Programming completed successfully. No errors occurred 1 = Programming failed. At least one error occurred Bit 1-3: Reserved Bit 4: Verify Pass/Fail 0 = Pass: The verification of the programmed data passed 1 = Fail: The verification of the programmed data failed Bit 5: Emergency Operation Pass/Fail 0 = Pass: The normal flow of the program operation has not been interrupted by an emergency operation request. 1 = Fail: The normal flow of the program operation has not been completed due to a request of an emergency operation Bit 6: Spare page selection Pass/Fail (Valid only for operation run on NVM Data pages) 0 = Pass: A new random spare page has been properly selected 1 = Fail: The random spare page selection failed. No random spare page selected Bit 7: Execution Pass/Fail status 0 = Pass: Routine execution could be properly started 1 = Fail: Routine execution could not be properly started due to missing required setting (Assembly Buffer not opened, target region write protected, nested call execution) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

5.3.3 Page verify routine

Note: This function is not available for variants with 256 Kbyte flash.

This user routine performs a verify on the target page using the Hard Read Level Erased and Hard Read Level Programmed (refer to **Figure 5-10** for read level details). The address of the target page is provided as input parameter.

In case the target page belongs to the linear code flash region, the routine performs the check on the target page and returns the page status. The routine first reads the page at standard read level and then compares the data read with the content read with Hard Read levels. In case an ECC fail is found or data are not identical at all the three used read levels, a fail is returned (03_H, 05_H or 07_H depending on which read level the verify is failing).

In case the page belongs to the non-linearly mapped data flash, the routine first accesses the MapRAM to check that the target page is mapped. In case no valid mapping for the page is found (invalid physical page or ECC fail) a proper error is reported (41_H). If a valid mapping is found, the routine performs the page verify on the related physical page and its status is returned.

The routine is not performed in case write protection is set on the target region. In this case, an error (81_H) is returned and the verify is not executed. In such case, the user is required to (temporarily) disable write protection on the target region before calling the routine.

The routine cannot be executed during any other NVM operation (it cannot be called as nested call inside other NVM related user routines).

Note: A fail returned by this function does not mean the page is no longer readable, it means that at least some cells have reached the Hard Read levels, a refresh of the page is recommended.

Table 5-18 Page verify subroutine

Subroutine	**00003895_H: USER_VERIFY_PAGE** Prototype: unsigned char **USER_VERIFY_PAGE**(int NVMPageAddr)

Table 5-18 Page verify subroutine (cont'd)

Input	**NVMPageAddr (int)**: NVM page address to be checked, page aligned address
Output	**Return value (char):** Bit 0: Pass or Fail. It is the OR of the bits 1, 2, 6 and 7 0 = Overall pass 1 = Overall fail Bit 1: Page verify on Hard Read Level Erased Pass or Fail 0 = Page verify on Hard Read Level Erased pass 1 = Page verify on Hard Read Level Erased fail Bit 2: Page verify on Hard Read Level Programmed Pass or Fail 0 = Page verify on Hard Read Level Programmed pass 1 = Page verify on Hard Read Level Programmed fail Bit 3 to 5: Reserved Bit 6: MapRAM check Pass or Fail (valid only when addressing the DFLASH) 0 = Pass: No MapRAM fail found 1 = Fail: MapRAM fail found (Invalid page) Bit 7: Execution Pass/Fail status 0 = Pass: Routine execution could be properly started 1 = Fail: Routine execution could not be properly started due to missing required setting (e.g.: Opened Assembly Buffer, nested call execution, invalid address, write protection set) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

5.3.4 NVM page erasing routine

Similarly, there are 2 types of erasing available, Type 1 or Type 2 (Type 1 without or Type 2 with RAM background activity during NVM operation). Details in the following table. Please check the flow diagram in **Figure 5-15**, it depicts the usage of the USER_ERASEPG function.

Table 5-19 Page erasing subroutine

Subroutine	**000038D5$_H$: USER_ERASEPG** Prototype: unsigned char **USER_ERASEPG**(int *NVMPageAddr, char RAM_RTNE_BRNCHNG)
Input	***NVMPageAddr (int pointer)**: Pointer to the NVM address (page aligned) to be erased RAM_RTNE_BRNCHNG **(char)**: To enable or disable background execution from RAM. Bit 0: RAM branching control bit 0 = RAM branching disabled 1 = RAM branching enabled
Output	**Return value (unsigned char):** Bit 0: Pass or Fail 0 = Erasing completed successfully 1 = Erasing failed Bit 3: MapRAM ECC Pass or fail 0 = MapRAM has no ECC2 fail 1 = MapRAM has at least one ECC2 fail Bit 3 is used only for pages in Data sector. Bit 3 is set together with bit 0 and erase is not performed. Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reasons of failure: • The range of the address is incorrect • This is a protected range Possible reason for execution fail: • Routine called as nested call during the execution of another NVM routine (e.g. via RAM branching) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

The input NVMPageAddr should be page aligned, with bits[6:0] = 0.

5.3.5 Erase page verify routine

Note: This function is not available for variants with 256 Kbyte flash.

This user routine performs an erase verify on the target page using the Hard Read Level Erased (refer to **Figure 5-10** for read level details). The address of the target page is provided as input parameter.

In case the target page belongs to the linear code flash region, the routine performs the check on the target page and returns the page status.

In case the page belongs to the non-linearly mapped data flash, the check is performed on the spare page regardless of the page address provided as input. In particular, the routine first accesses the MapRAM to check that a valid spare page link is present in the MapRAM (no ECC fail and valid link to an existing physical page). In case no valid spare page link is found, a proper error is reported (41_H). In that case the user shall try to rebuild valid mapping info by calling the USER_MAPRAM_INIT routine. If a valid spare page selection is found, the routine performs an erase page verify on the physical page selected as spare page and its status is returned.

The routine is not performed in case write protection is set on the target region. In this case, an error (81_H) is returned and the verify is not executed. In such case, the user is required to (temporarily) disable the write protection on the target region before calling the routine.

The routine cannot be executed during any other NVM operation (it cannot be called as nested call inside other NVM related user routines.)

Note: A fail returned by this function does not mean the page is no longer usable, it means that at least some cells have reached the Hard Read levels. In case of lineare code flash retry the page erase, in case of mapped data flash rerun USER_MAPRAM_INIT[1]).

Table 5-20 Erase page verify subroutine

Subroutine	**00003885_H: USER_ERASEPG_VERIFY** Prototype: unsigned char **USER_ERASEPG_VERIFY**(int NVMPageAddr)

1) USER_MAPRAM_INIT selects a new erased spare page. With the next restart of the device the Service Algorithm will attempt to erase the affected page again.

Table 5-20 Erase page verify subroutine (cont'd)

Input	**NVMPageAddr (int)**: NVM page address to be checked, page aligned address
Output	**Return value (char):** Bit 0: Pass or Fail. It is the OR of the bits 1, 6 and 7 0 = Overall pass 1 = Overall fail Bit 1: Erase verify on Hard Read Level Erased Pass or Fail 0 = Erase page verify pass 1 = Erase page verify fail Bit 2 to 5: Reserved Bit 6: MapRAM check Pass or Fail (valid only when addressing the DFLASH) 0 = Pass: No MapRAM fail found 1 = Fail: MapRAM fail found (Invalid spare page) Bit 7: Execution Pass/Fail status 0 = Pass: Routine execution could be properly started 1 = Fail: Routine execution could not be properly started due to missing required setting (e.g.: Opened Assembly Buffer, nested call execution, invalid address, write protection set) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

5.3.6 Sector erasing routine

This routine is used to perform an erase of a NVM data sector.

Table 5-21 Sector erasing subroutine

Subroutine	**00003865_H: USER_ERASE_SECTOR** Prototype: unsigned char **USER_ERASE_SECTOR**(unsigned int sectorAddress)

Table 5-21 Sector erasing subroutine

Input	**SectorAddress (unsigned int)**: NVM Sector address
Output	**Returned value (unsigned char):** Bit 0: Pass or Fail 0 = Erasing completed successfully 1 = Erasing failed. Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reason for execution fail: • Routine called as nested call during the execution of another NVM routine (e.g. via RAM branching) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

The input value sectorAddress should be sector aligned, with the bits[11:0] = 0.

5.3.7 Erase sector verify routine

Note: This function is not available for variants with 256 Kbyte flash.

This user routine performs an erase verify on the target sector using the Hard Read Level Erased (refer to **Figure 5-10** for read level details). The address of the target sector is provided as input parameter.

The routine accepts a 32 bit address as input parameter. Since the routine works on a sector base, the last 12 bits of the provided address are not relevant and ignored.

In case the target sector belongs to the linear code flash region, the routine performs the check on the target sector and returns the sector status. The check is performed sequentially on all the pages of the sector one by one.

In case the target sector is the non-linearly mapped data flash, the routine performs a dedicated consistency check of the MapRAM. In case a invalid spare page mapping is found (not linked to any physical page) or any logical page is mapped or in case any MapRAM entry has a ECC fail a proper error is reported (41_H). In that case the user shall try to rebuild valid mapping info by calling the USER_MAPRAM_INIT routine. If no issue in the MapRAM is found, the routine performs an erase sector verify on all the physical pages of the sectors and the sector status is returned.

The routine is not performed in case write protection is set on the target region. In this case, an error (81_H) is returned and the verify is not executed. In such case, the user is

required to (temporarily) disable write protection on the target region before calling the routine.

The routine cannot be executed during any other NVM operation (it cannot be called as nested call inside other NVM related user routines.)

Note: A fail returned by this function does not mean the sector is no longer usable, it means that at least some cells in at least one page have reached the Hard Read levels, it is recommended to perform a sector erase again.

Table 5-22 Erase sector verify subroutine

Subroutine	**0000388D$_H$: USER_ERASE_SECTOR_VERIFY** Prototype: unsigned char **USER_ERASE_SECTOR_VERIFY**(int NVMSectorAddr)
Input	**NVMSectorAddr (int)**: NVM Sector address to be checked, sector aligned address
Output	**Return value (unsigned char):** Bit 0: Pass or Fail. It is the OR of the bits 1, 6 and 7 0 = Overall pass 1 = Overall fail Bit 1: Erase verify on Hard Read Level Erased Pass or Fail 0 = Erase verify for all pages of the target sector passes 1 = Erase page verify for at least one page of the sector fails Bit 2 to 5: Reserved Bit 6: MapRAM check Pass or Fail (valid only when addressing the DFLASH) 0 = Pass: No MapRAM fail found 1 = Fail: MapRAM fail found (Invalid spare page, mapped pages, ECC mapRAM fail) Bit 7: Execution Pass/Fail status 0 = Pass: Routine execution could be properly started 1 = Fail: Routine execution could not be properly started due to missing required setting (e.g.: Opened Assembly Buffer, nested call execution, invalid address, write protection set) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

5.3.8 Abort NVM programming routine

This user routine aborts the NVM programming by closing an opened assembly buffer. Please check the flow diagram in **Figure 5-15**, it depicts the usage of the USER_ABORTPROG function.

Table 5-23 Abort NVM programming subroutine

Subroutine	**$000038CD_H$: USER_ABORTPROG** Prototype: bool **USER_ABORTPROG**(void)
Input	--
Output	**Return value (bool): Pass or Fail** 0 = Abort successfully, assembly buffer closed 1 = Abort failed as programming already started Possible reason of failure: • Programming already started • Routine called as nested call during the execution of another NVM routine (e.g. via RAM branching)

5.3.9 MapRAM initialization

This routine is meant to be used to re-initialize the MapRAM of the DFLASH sector.

The routine performs a complete MapRAM initialization by triggering a dedicated function of the NVM internal Finite State Machine. When triggered, the state machine resets the whole MapRAM and rebuilds information by reading the current logical to physical address information stored directly in the NVM data sector. In case of mapping errors (double or multiple mapping or faulty pages) the initialization of the MapRAM is stopped on the first error found and the routine is exited reporting a proper error indication. In case of fail, the content of the MapRAM might be only partial and the mapping information might be corrupted.

The routine can be used to try to restore a clean MapRAM status in case a MapRAM error has been reported by the startup or by the program routine or in case some data sector pages have been lost. In addition, this routine can be used to check whether the mapped sector has a consistent status. Please check the flow diagram in **Figure 5-15**, it depicts the usage of the USER_MAPRAM_INIT function.

Note: In case an NVM operation on the Data region is interrupted (e.g. due to reset events), the mapped sector might have an inconsistent status depending on the moment in which the interruption occurred. In case of power-on reset, brown-out reset, pin reset or wake-up reset the system performs the MapRAM initialisation during the following startup and triggers the Service Algorithm to try to repair mapping inconsistency, if required. In case of

software reset (e.g. issued during a RAM branching) or internal watchdog reset, the following startup sequence performs a MapRAM initialization without triggering the repair step in case of errors. The user shall then check after every reset the status of the mapped region evaluating the information reported on the MEMSTAT and SYS_STRTUP_STS registers. Refer to the Section 3.1.3 for MapRAM initialization flow for different reset types. If a mapping error is found the user shall not try to program or erase any page in the Data sector but shall try to trigger a new repair step by issuing a proper reset or erasing the complete sector.

Table 5-24 MapRAM initialization subroutine

Subroutine	**0000389D$_H$: USER_MAPRAM_INIT** Prototype: unsigned char **USER_MAPRAM_INIT**(void)
Input	--
Output	**Return value (unsigned char):** Bit 0: Pass or Fail. It is the OR of the bits 5, 6 and 7 0 = MapRAM initialization pass 1 = MapRAM initialization fail Bit 1 to 4: Reserved Bit 5: Double mapping 0 = Pass: No double mapping found 1 = Fail: The initialization failed due to double mapping Bit 6: Faulty page 0 = Pass: No faulty pages found 1 = Fail: The initialization failed due to faulty page Bit 7: Execution Pass/Fail status 0 = Pass: Routine execution could be properly started 1 = Fail: Routine execution could not be properly started due to missing required setting (e.g.: Opened Assembly Buffer, nested call execution) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

5.3.10 Read NVM status routine

This user routine checks for the NVM status.

Table 5-25 Read NVM status subroutine

Subroutine	**$000038C5_H$: USER_NVMRDY** Prototype: bool **USER_NVMRDY**(void)
Input	--
Output	**Return value (bool): Pass or Fail** 0 = NVM is not busy 1 = NVM is busy now

5.3.11 Read 100 Time Programmable parameter data routine

This routine reads the 100TP page content. For the 100TP page 1, the data offset range is listed in **Table 5-11**. Details in the following table.

Table 5-26 Read 100TP subroutine

Subroutine	**00003875_H: USER_READ_100TP** Prototype: bool **USER_READ_100TP**(char 100TP_Page_Sel, unsigned char DataOffset, int *HundredTPData)
Input	**100TP_Page_Sel (char)**: 100TP page selection byte (CS_Byte, refer to **Figure 5-8**) **DataOffset (unsigned char)**: Data Offset in page (00_H to $7F_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Read is successful 1 = Read is not successful due to invalid range selected **HundredTPData (int pointer)** = Pointer to the RAM location where 100TP Data is saved

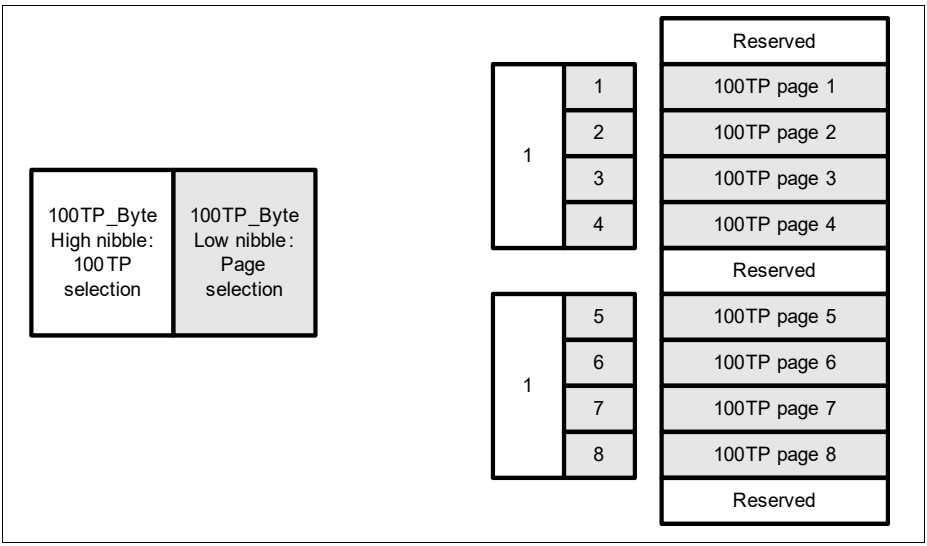


Figure 5-8 User configuration sector pages address byte description

5.3.12 Program 100 Time Programmable routine

This routine programs data into the 100TP pages. The 100TP content to be programmed has to be preloaded into the RAM. The details can be found in the **Section 5.2.2**.

Table 5-27 Program 100TP subroutine

Subroutine	**0000386D$_H$: USER_100TP_PROG** Prototype: unsigned char **USER_100TP_PROG**(char 100TP_Page_Sel)

Table 5-27 Program 100TP subroutine (cont'd)

Input	**100TP_Page_Sel (char):** 100TP page selection byte (CS_Byte, refer to **Figure 5-8**) RAM preloaded with the 100TP data to be programmed.
Output	**Returned value (unsigned char):** Bit 0: Program operation pass or fail flag 0 = Program completed successfully 1 = Program failed Bit 1: In page offset error flag 0 = All bytes have in page offset 1 = At least one byte has a not in page offset *Note: Not in page bytes are not programmed and do not result in a program error on bit 0. Counter position is already considered out of range.* Bit 2: ID protected region fail flag 0 = All bytes do not target the reserved Customer_ID region 1 = At least 1 byte targets the reserved Customer_ID region *Note: Bytes targeting the Customer_ID region are not programmed and do not result in a program fail error on bit 0.* Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reasons of failure: • The NVM code area is protected against programming • The 100TP page is already programmed to a maximum of 100 times Possible reason for execution fail: • Routine called as nested call during the execution of another NVM routine (e.g. via RAM branching) *Note: No NVM prog, erase or verify routine can be called until this NVM operation is completed.*

5.3.13 NVM ECC check routines

The firmware provides 2 different routines to enable the user to check and monitor the quality of the NVM cells upon shipment and/or over the lifetime of the device.

The first routine, USER_NVM_ECC_CHECK, provides an easy way for the user to perform a quick check of the status of the whole NVM array. The routine performs a read of the complete NVM returning the single and double bit ECC flags. This is meant to be used as a quick check of the programming quality of the NVM Code region and the mapped pages of the NVM Data region.

Table 5-28 NVM ECC check subroutine

Subroutine	**00003845_H: USER_NVM_ECC_CHECK** Prototype: unsigned char **USER_NVM_ECC_CHECK**(void)
Input	--
Output	**Returned value (unsigned char): ECC error indication** Bit 0: ECC1READ 0 = No single bit ECC error on the whole NVM read 1 = At least one single bit ECC error on the whole NVM read Bit 1: ECC2READ 0 = No double bit ECC error on the whole NVM read 1 = At least one double bit ECC error on the whole NVM read Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reason for execution fail: • Routine called as nested call during the execution of another NVM routine (e.g. via test is running, no RAM access should be attempted on the whole RAM. branching)

Note: The USER_NVM_ECC_CHECK routine performs a read of the entire NVM Code region and of all the non-erased (mapped) pages of the Data region. All logical pages of the Data NVM region not yet programmed and consequently not mapped are not checked since there is no link to a physical address. In case the user needs to completely check the NVM Data region, a program of all the logical pages of the sector has to be performed before calling the USER_NVM_ECC_CHECK.

Note: The USER_NVM_ECC_CHECK makes use of the RAM byte at 0x18000015. Any data being stored there is overwritten.

The second routine, USER_ECC_CHECK, provides a way to check whether during code execution any ECC error occurred. With its return value the routine indicates if a single or a double bit error ECC error flag was set since last power-off (incl. Sleep mode) of the device, last call of this routine or since last call of a user routine for NVM operation, whatever happened last. This routine is meant to be used over device life time to monitor the occurrence of ECC errors. In addition, in case of EEC2 error, the routine will provide as an output the address of the last ECC2 error occurred.

The address is reported as an output in the RAM location passed as a pointer. The returned value always provides the starting address of the 8 byte section where the ECC error happened.

Table 5-29 ECC check subroutine

Subroutine	**0000383D$_H$: USER_ECC_CHECK** Prototype: unsigned char **USER_ECC_CHECK**(unsigned int *ECC2Addr)
Input	**ECC2Addr (unsigned int pointer):** Pointer to the RAM location where the last NVM address with ECC2 error shall be stored
Output	**Returned value (unsigned char): ECC error indication** Bit 0: ECC1READ 0 = No single bit ECC error on the whole NVM read 1 = At least one single bit ECC error on the whole NVM read Bit 1: ECC2READ 0 = No double bit ECC error on the whole NVM read 1 = At least one double bit ECC error on the whole NVM read Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reason for execution fail: • Routine called as nested call during the execution of another NVM routine (e.g. via test is running, no RAM access should be attempted on the whole RAM. branching)

Note: The ECC error flags, provided as output of the NVM ECC check routines, are a copy of the ECC internal error flags registers. These registers are set when a read access to the NVM results in a single and/or double bit error and are cleared only in case of power-off (incl. Sleep mode) or in the following cases:

1. *When programming or erasing a NVM page.*
2. *When calling the USER_NVM_ECC_CHECK routine before performing the NVM complete read.*
3. *When calling the USER ECC check routine before returning to user code.*

5.3.14 Read NVM ECC2 address routine

This routine returns the result of the last NVM address accessed resulting in a double ECC error. Details in the following table.

Table 5-30 Read NVM ECC2 address subroutine

Subroutine	**$000038AD_H$: USER_NVM_ECC2ADDR** Prototype: unsigned char **USER_NVM_ECC2ADDR**(unsigned int *ECC2Addr)
Input	**ECC2Addr (unsigned int pointer):** Pointer to the RAM location where the last NVM address with ECC2 error shall be stored
Output	**Returned Value (unsigned char):** Bit 0: Pass or Fail 0 = ECC2Addr read out successful 1 = ECC2Addr read out failed Bit 4: ECC2 error detection 0 = No NVM ECC2 detected 1 = NVM ECC2 address detected Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed. 1 = Fail: Routine was not executed Possible reason for execution fail: • Null pointer passed to the routine, or invalid pointer, 0x81 is returned • Routine called as nested call during the execution of another NVM routine (e.g. via RAM branching)

The address reported as an output in the RAM location is passed as a pointer. The returned value always provides the starting address of the 8 byte section where the ECC error happened.

5.3.15 RAM MBIST starting routine

This routine is used to perform a RAM test. A linear write/read algorithm using alternating data is executed on a RAM range specified by the start and stop addresses given as input parameters. When starting the MBIST test, standard RAM interface is disabled. Therefore data stored into it will not be accessible and data stored in the memory range under test will be cleared to zero. The standard interface will be re-enabled after completion before the end of the routine execution.

Note: The start and stop address passed as parameter are offsets to be added to the RAM start address (18000000_H).

Table 5-31 RAM MBIST start subroutine

Subroutine	**$0000384D_H$: USER_RAM_MBIST_START** Prototype: unsigned char **USER_RAM_MBIST_START**(short RAM_MBIST_Stop_Addr, short RAM_MBIST_Start_addr)
Input	**RAM_MBIST_Stop_Addr (short):** RAM offset of the stop address of RAM range to be tested **RAM_MBIST_Start_addr (short):** RAM offset of the start address of RAM range to be tested
Output	**Returned value (unsigned char): Pass or Fail** Bit 0: MBIST result, pass or fail 0 = MBIST test pass 1 = MBIST test fail Bit 1: Address range fail 0 = test routine pass (address range valid) 1 = test routine fail (address range invalid) Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reason for execution fail: • Routine called as nested during the execution of another NVM routine (e.g. via RAM branching)

Note: The range of memory to be tested by this function is limited to the first 4 Kbyte of the RAM. For the UH and UI step, the range of memory to be tested is based on the actual RAM size, which is dependent on the product variant.

Note: While test is running, no RAM access should be attempted on the whole RAM.

Note: In case the RAM MBIST test fails, the RAM location 18000002_H is written.

5.3.16 NVM protection status change routines

These routines allow to enable or disable the read or write protection individually on the NVM Code sectors (linearly mapped NVM sectors) and on the NVM Data sectors (not linearly mapped NVM sectors).

These routines control the protection status updating the value of the lower nibble of the NVM_PROT_STS register. The status of the register will be anyhow restored according to the NVM PASSWORD stored in the Configuration Sector at next reset. Please, refer to User Manual for NVM_PROT_STS bits description.

*Note: Each routine requires a password (16 bit) that shall be provided as an input to the user routine call. The BootROM code will compare this password with the one stored into the configuration sector 100TP page 1 (offset $0C_H$ for the routines addressing the linearly mapped region protection and offset $0E_H$ for the routines addressing the non linearly mapped region protection). Only in case the password read out of the 100TP page 1 matches the password provided as input, the requested protection status change is performed (refer to **Table 5-11**).*

Table 5-32 NVM Code sectors (linearly mapped NVM sectors) write protection enable subroutine

Subroutine	**00003925_H: USER_CFLASH_WR_PROT_EN** Prototype: bool **USER_CFLASH_WR_PROT_EN**(unsigned short CFLASH_PW)
Input	**CFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset $0C_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(1) to 0.

Table 5-33 NVM Code sectors (linearly mapped NVM sectors) write protection disable subroutine

Subroutine	**$0000391D_H$: USER_CFLASH_WR_PROT_DIS** Prototype: bool **USER_CFLASH_WR_PROT_DIS**(unsigned short CFLASH_PW)

Table 5-33 NVM Code sectors (linearly mapped NVM sectors) write protection disable subroutine (cont'd)

Input	**CFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset $0C_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(1) to 1.

Table 5-34 NVM Code sectors (linearly mapped NVM sectors) read protection enable subroutine

Subroutine	**00003915_H: USER_CFLASH_RD_PROT_EN** Prototype: bool **USER_CFLASH_RD_PROT_EN**(unsigned short CFLASH_PW)
Input	**CFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset $0C_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(3) to 0.

Table 5-35 NVM Code sectors (linearly mapped NVM sectors) read protection disable subroutine

Subroutine	**$0000390D_H$: USER_CFLASH_RD_PROT_DIS** Prototype: bool **USER_CFLASH_RD_PROT_DIS**(unsigned short CFLASH_PW)
Input	**CFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset $0C_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(3) to 1.

Table 5-36 NVM Data sectors (not linearly mapped NVM sectors) write protection enable subroutine

Subroutine	**00003905_H: USER_DFLASH_WR_PROT_EN** Prototype: bool **USER_DFLASH_WR_PROT_EN**(unsigned short DFLASH_PW)
Input	**DFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset $0E_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(0) to 0.

Table 5-37 NVM Data sectors (not linearly mapped NVM sectors) write protection disable subroutine

Subroutine	**$000038FD_H$: USER_DFLASH_WR_PROT_DIS** Prototype: bool **USER_DFLASH_WR_PROT_DIS**(unsigned short DFLASH_PW)
Input	**DFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset $0E_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully. 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(0) to 1.

Table 5-38 NVM Data sectors (not linearly mapped NVM sectors) read protection enable subroutine

Subroutine	**000038F5$_H$: USER_DFLASH_RD_PROT_EN** Prototype: bool **USER_DFLASH_RD_PROT_EN**(unsigned short DFLASH_PW)
Input	**DFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset 0E$_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(2) to 0.

Table 5-39 NVM Data sectors (not linearly mapped NVM sectors) read protection disable subroutine

Subroutine	**000038ED$_H$: USER_DFLASH_RD_PROT_DIS** Prototype: bool **USER_DFLASH_RD_PROT_DIS**(unsigned short DFLASH_PW)
Input	**DFLASH_PW(unsigned short)**: Password to be compared to the one stored in the 100TP page 1 (offset 0E$_H$)
Output	**Returned value (bool): Pass or Fail** 0 = Operation completed successfully 1 = Operation failed (Password does not match)

This routine sets the bit NVM_PROT_STS(2) to 1.

User configuration sector pages (100TP) are part of the NVM Code region considering the write protection, enabled by the routine USER_CFLASH_WR_PROT_EN.

Code fetching is not influenced by the read protection.

Note: Copying code from NVM to RAM requires a normal NVM read execution and so is blocked in case NVM read protection is enabled.

The permanent NVM protection (enabled by Mode 6 as described in the **Section 4.4.2.7***) is meant to avoid unwanted writing/reading of the flash during code execution. In case*

the application SW has to write/read Code and/or Data flash, the protection can be temporarily disabled and then enabled again, as described in the **Section 5.3.16***. Even if the read protection is enabled on Code region, Data regions or both, the code executed from Code NVM region can always read both NVM Code and Data regions.*

5.3.17 Read NVM config status routine

This routine reads the NVM Configuration Status. Details in the following table.

Table 5-40 Read NVM config status subroutine

Subroutine	**$000038B5_H$: USER_NVM_CONFIG** Prototype: bool **USER_NVM_CONFIG**(char *NVMSize, char *MapRAMSize)
Input	--
Output	**Return value (bool): Pass or fail** 0 = Configuration read successfully 1 = Configuration read failed **NVMSize (char pointer):** pointer to the RAM location where the number of available sectors of the code area (4 Kbytes each) has to be saved **MapRAMSize (char pointer)**: pointer to the RAM location where to store the number of available sectors of the data area (4 Kbytes each) Possible reason of failure: • NVM Linear sector is set as 00_H

5.3.18 Read user calibration data

All data stored in the 100TP pages can be downloaded into the RAM using this routine. In particular, this routine has been developed to help user in downloading the ADC1 calibration parameters stored at the beginning of 100TP page 1 (see **Table 5-11**) to an easily accessible data space (RAM). To download the data, the user needs to provide the 100TP page where data has to be read from, number of bytes to be copied, and the RAM address where data has to be copied to. The routine will copy the specified number of bytes from the selected page (starting always from first byte in the page) into the RAM (starting at the given address).

Note: The provided RAM address where data have to be copied is just an offset to the device RAM start address (18000000_H).

Table 5-41 Read user calibration data subroutine

Subroutine	**$000038BD_H$: USER_READ_CAL** Prototype: unsigned char **USER_READ_CAL**(unsigned char NumOfBytes, char Sel100TP, short RAMAddr)
Input	**NumOfBytes (unsigned char)**: Number of bytes to be copied from config sector into the RAM (allowed values are form 01_H to 80_H). **Sel100TP (char)**: 100TP page to take data from (refer to **Figure 5-8**). **RAMAddr (short)**: RAM address offset to copy data to ($03FF_H$ < RAMAddr < RAMAddr + NumOfBytes < RAM size) RAM size: 3 kB RAM: $0BFF_H$ 6 kB RAM: $17FF_H$
Output	**Return value (unsigned char):** Bit 0: Pass or Fail 0 = Read is successful 1 = Read is not successful due to invalid input values Bit 7: Execution Pass/Fail status 0 = Pass: Routine was correctly executed 1 = Fail: Routine was not executed Possible reasons of failure: • The input parameters are incorrect. Possible reason for execution fail: • Routine called as nested call during the execution of another NVM routine (e.g. via RAM branching)

5.3.19 NVMCLKFAC setting routine

This routine is used to write the NVMCLKFAC bit in SYSCON0 register.

Table 5-42 NVMCLKFAC setting subroutine

Subroutine	**00003855_H**: **USER_NVMCLKFAC_SET** Prototype: void **USER_NVMCLKFAC_SET**(char Value)
Input	**Value (char)**: SYSCON0.NVMCLKFAC value to be written. b
Output	--

5.4 NVM user applications

The NVM user routines application is described in this section.

5.4.1 NVM Data sector handling

The NVM provides a special sector for data storage. Through a non-linear mapping of the address space, the FW and the NVM module provides a special feature to increase the maximum number of write-erase cycles a logical page can stand and to reduce the risk of data loss in case of interrupted NVM operations (tearing events).

The handling of this special Data sector requires the usage of an NVM internal look-up table (MapRAM) which is used to store and handle the link between logical and physical addresses of the sector's pages.

Since the MapRAM is a volatile memory, the firmware takes care to rebuild the MapRAM content at each power-up based on mapping information stored into a specific field of the Data sectors pages (mapblock). This process is called Data sector initialization (MapRAM initialization).

During this initialization phase, mapping errors induced by tearing events might be found. This would then prevent the firmware from properly restoring the link between the logical and physical addresses thus preventing proper usage of this sector. In this case, the firmware provides a specific algorithm (Service Algorithm) to identify and solve these errors. In particular, the Service Algorithm tries to repair bad pages created unintentionally into the NVM Data region due to, for example, a NVM program or erase operation interrupted by any reset or power loss (tearing events). The Service Algorithm is triggered during the startup by the NVM data sector initialization in case mapping issues are found.

The Service Algorithm provides proper analysis features to try to preserve the integrity of the NVM Data region in case ongoing NVM operation (program or erase) is unintentionally and unexpectedly aborted (e.g. due to power loss). Anyhow, it is not meant to cover all possible scenarios that can be created by an interrupted NVM operation. The user shall put in place proper action to avoid any

possible interruption of NVM operation (e.g. using proper capacitor on the power supply).

The NVM Data sector initialization and Service Algorithm flows are described below.

NVM Data sector initialization

After any reset, as part of the startup, the firmware triggers a NVM initialization of the NVM data sector. This initialization is performed by a hardware state machine which takes care to restore the mapping information into the MapRAM reading specific bytes (called mapblock) of the NVM Data sector pages (see **Figure 5-9**). The state machine accesses these bytes and, page by page, reads out the logical page to which the current physical page has to be linked to, updating accordingly the dedicated MapRAM location.

In case a mapblock is read as erased, the physical page is not mapped. All the logical pages for which no valid mapping is found are marked into the MapRAM as unmapped.

While reading out the info from the mapblock, the hardware state machine might find incorrect mapping info. In particular, following scenarios might appear:

- More physical pages are mapped to the same logical page (double or higher mapping)
- The mapblock information cannot be read correctly due to ECC errors (faulty page)

In this case, the hardware state machine stops the initialization on the first incorrect mapping. In case of power-on reset, brown-out reset, pin reset, WDT1 reset or wake-up from sleep in addition the execution of the Service Algorithm (SA) is triggered.

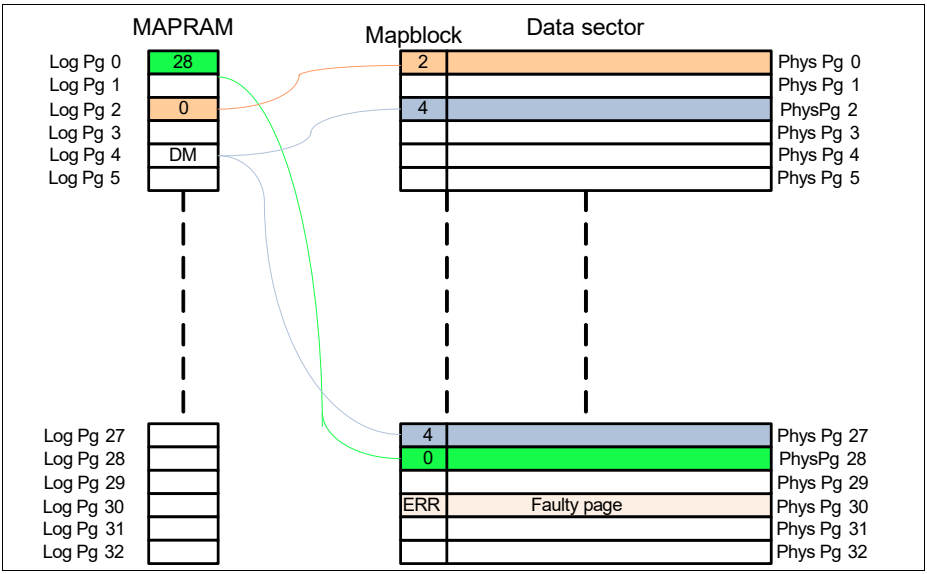


Figure 5-9 MapRAM and mapblocks

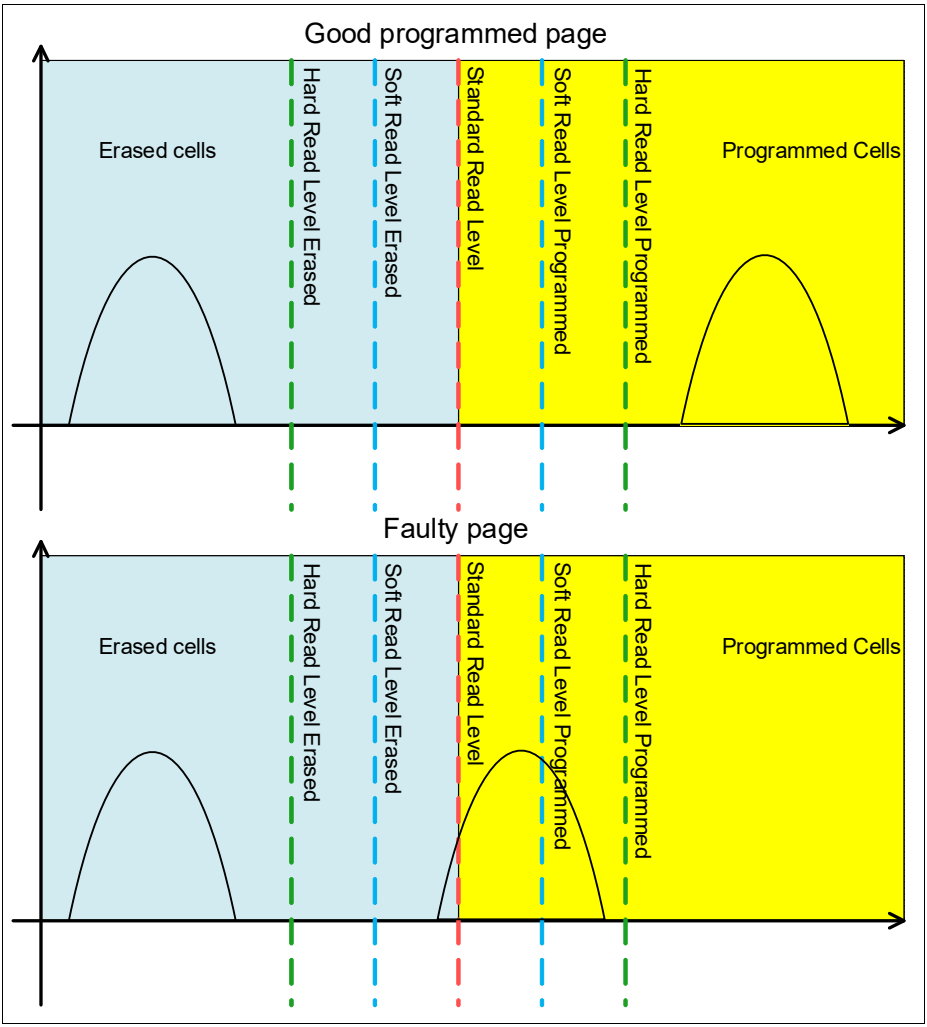


Figure 5-10 Read levels and faulty page

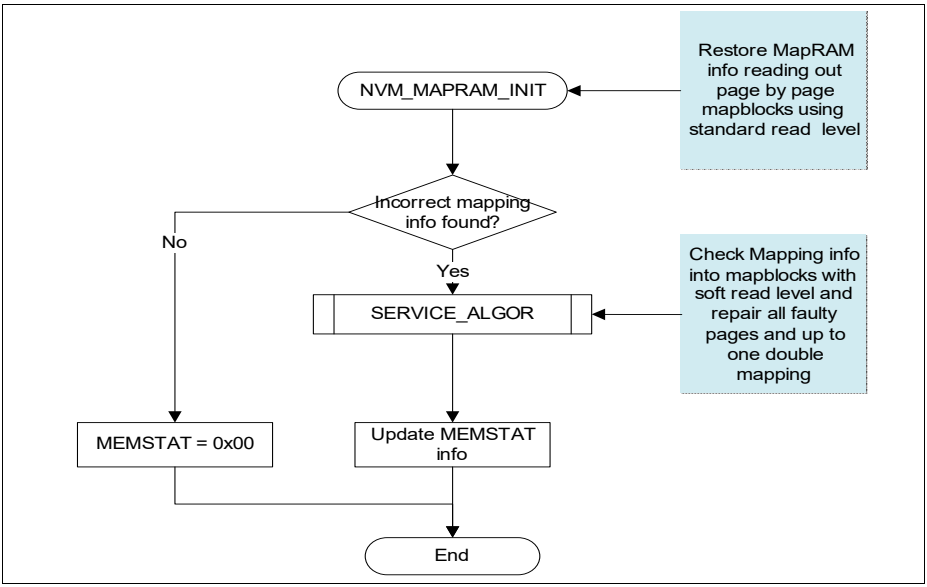


Figure 5-11 NVM data sector initialization flow

In order to detect pages whose mapblock is marginal towards the standard read level, the NVM finite state machine that performs the mapping initialization is triggered three times with three different read levels: standard read margin, soft read level erased and soft read level programmed (refer to **Figure 5-10**). As soon as the first incorrect mapping (faulty or multiple mapping) is detected by any of these three initialization sequences, depending on the reset type, the Service Algorithm is called.

At the end of the Service Algorithm execution, a new initialization of the Data sector is needed to properly initialize the mapping info. This final initialization is again executed by triggering the NVM Finite State Machine and is performed using only the standard read level.

Note: For any reset the result of the last NVM Data sector initialization executed during the startup flow is reported to the user via the bit 1 of the SYS_STRTUP_STS register (MRMINITSTS). If this bit is set to 1 then the last initialization failed and the mapping info might be corrupted. In this case, a reset (power-on reset, brown-out reset, pin reset or wake-up reset) can be issued in order to start the Service Algorithm to try to fix the integrity issue inside the Data NVM. If the MRAMINITSTS is still flagged afterwards, the Data NVM sector has to be re-initialized by performing a sector erase.

Service Algorithm

The Service Algorithm is called by the NVM Data sector initialization in case incorrect mapping issues have been found. The Service Algorithm checks the data sector page by page reading the mapblocks with soft read levels (refer to **Figure 5-10**).

At first, the Service Algorithm looks for faulty pages and tries to repair them by erasing these pages. Following, the algorithm proceeds looking for double or higher mappings.

In case two or more double mappings or at least one triple or higher mapping were found the SA stops execution and reports an error on the MEMSTAT register (MEMSTAT set to $B0_H$). In case, instead only one double mapping is found, the algorithm selects which page has to be erased according to the following steps:

1. The SA checks the 2 pages linked to the double mapping with standard, soft and hard read levels to detect which one has better quality (more margin towards the standard read level, refer to **Figure 5-10**). The page with smaller margin is then erased.
2. In case both pages have same quality, the algorithm checks some specific bits of the mapblock (called map counter) to check which of the pages has been programmed last. In this case, the older one is erased.
3. In case both pages have same map counter value, the SA cannot decide which page has to be erased and ends the flow reporting an error on the MEMSTAT register (e.g. MEMSTAT set to $B0_H$ for a 256 Kbyte variant).

Whenever the SA is triggered, information regarding the addressed data sector number will be stored in SECTORINFO (this is an indication that the SA was executed during the startup phase). In addition, in case the SA cannot recover all incorrect mapped pages, the SA reports a fail into the SASTATUS field of the MEMSTAT writing the value 10_B. In such a case, the user shall properly handle the reported mapping issue by either triggering a reset (power-on reset, pin reset, WDT1 reset, brown-out reset or wake-up from sleep reset) in order to trigger a new NVM initialisation or to erase the whole NVM data sector to reset the mapping info.

Detailed description of the MEMSTAT register can be found in the following table **Table 5-43**.

Table 5-43 MEMSTAT Register status for NVM integrity handling

Field	Bits	Description
SASTATUS	7:6	**Service Algorithm Status** 00 Depending on SECTORINFO, 2 possible outcomes. For SECTORINFO = 00_H: NVM initialisation successful, no SA is executed. For SECTORINFO = Otherwise: SA execution successful. Only 1 mapping error fixed. 01 SA execution successful. At least 1 mapping error fixed. 10 SA execution failed. Map error in data sector. 11 Reserved
SECTORINFO	5:0	**Sector Information** At the startup, the value of tthis field is set to 000000_B and it is written anytime the SA is executed. This field is internally divided into two parts: Bits 5:0 : NVM Class identifier 30_H: 256 Kbyte variants 20_H: 128 Kbyte variants 10_H: 64 Kbyte variants 09_H: 36 Kbyte variants Others: Reserved Once the SA has been executed, regardless of the execution status, the last access sector information will be stored here.

Note: The MEMSTAT register has a dual function. It is used to store the return value of the SA as well as input value for the NVM operations to indicate the Emergency Operation. For this reason, the user shall reset the MEMSTAT register after every power-on reset, brown-out reset, pin reset or wake-up reset before the execution of any NVM operation.

During the repair phase, pages with incorrect mapping are erased. Each page erase operation takes up to 4.5 ms.

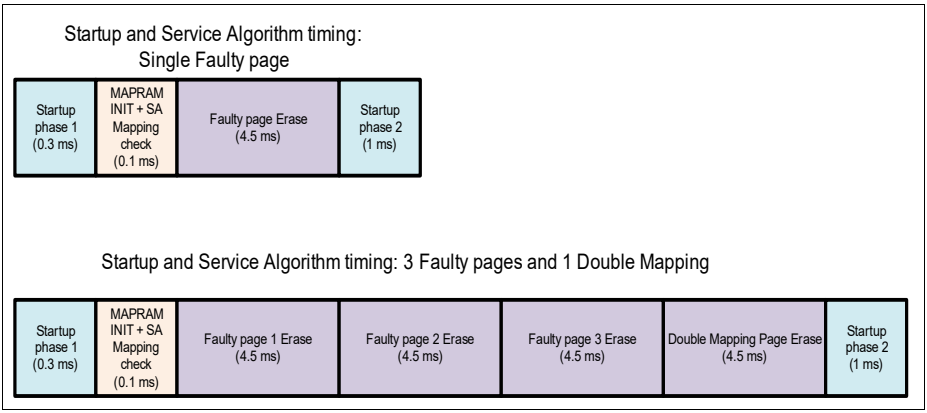


Figure 5-12 Service Algorithm: Timing examples

Due to the duration of the first WDT1 open window after reset (long open window), the maximum number of pages that can be repaired in one Service Algorithm execution is 13.

The result of the Service Algorithm repair phase is reported in the MEMSTAT register. At the end of the startup procedure, user shall evaluate the content of this register to properly handle fails and clear the register before performing any NVM operation.

The value is only available after reset before any NVM operation (Program, Erase, OpenAB) is started. The corresponding NVM address to the Sector Information read is listed in **Table 5-3**.

Service Algorithm and NVM protection

In case the Service Algorithm detects mapping issues, it tries to repair mapping by erasing the wrong pages (either faulty or double mapped pages). Consequently, the repair step can modify the NVM Data sector content. To avoid data loss, the SA checks the NVM Data sector protection and proceeds towards the repair step only if the protection is not enabled.

In case protection is enabled, instead, the repair actions are not performed and a warning is provided to the user by writing the value FE_H in the MEMSTAT register.

Via a dedicated 100TP sector parameter the user can always allow Service Algorithm to perform the repair step even in case the Data sector is protected. The control byte for this feature, CS_SA_WITH_PROT_EN, is stored into the first 100TP page (refer to **Table 5-11**). When this parameter is set to the value $A5_H$ the repair step is executed even in case protection is set. The repair flow saves the protection setting, removes temporarily the protection on the data sector, performs the needed repair operation and then restores the original protection settings. The temporary protection disabling is

performed via NVM protection register setting, no access or changes to the user defined NVM protection password is performed. By default the CS_SA_WITH_PROT_EN parameter is set to 00_H (i.e. protection status is considered).

5.4.2 Supporting background NVM operation

There is only one NVM module present in TLE986x/TLE987x. When NVM is busy executing internal operations (e.g. cells programming or erasing, data verify), no other activities within NVM can be executed. Although the NVM programming or erasing is handled by the NVM module, the user code cannot be read or executed as the NVM module is busy.

For this reason interrupts can only be serviced when the NVM is free if the interrupt vector table or interrupt service routines are located in the NVM. A NVM program operation can take from 3.5 ms to 13.5 ms to be completed. Therefore there is a need to support the user for critical activities.

To support other user activities while NVM is busy, the BootROM can redirect code execution to RAM after triggering time consuming NVM operations like program and erase. This type of background code execution is known as Type 2 NVM operations or RAM branching. When RAM branching is active, the BootROM routines jump to the RAM address 18000400_H every time it has to wait for NVM internal operation to be completed. In this way, the user can execute code from RAM while NVM is busy.

While executing user code from RAM due to RAM branching, if the ongoing internal NVM operation is completed, the BootROM code execution is not automatically restarted and the previously triggered BootROM user routine is suspended. The user needs to explicitly re-trigger the user routine code execution by giving back control to the BootROM via a return instruction (BX LR). In this way the suspended user routine execution is resumed.

The USER_NVMRDY user routine (refer to the **Section 5.3.10**) is provided to check whether the internal on going NVM operation is finished. User can use this routine to poll the busy status of the NVM to decide when to return control to the suspended user routine.

In case the user RAM code returns control to BootROM user routine while NMV is still busy, the BootROM code waits till the internal operation is completed before continuing with the normal user routine execution.

Table 5-44 shows RAM branching address and provides an example for the RAM code exit point. **Figure 5-13** shows how background programming can be supported during calls to a NVM programming routine.

Note: The context switch between BootROM user routine and user RAM code in NVM operation Type 2 is user responsibility. To avoid that RAM code execution interferes with BootROM user code completion, the user must save the content of the used resources (e.g. core registers) upon starting

the RAM code execution and restore them before jumping back to the BootROM code. Under no circumstances shall the user return with modified core registers, as proper resuming the BootROM function cannot be ensured.

Note: During user RAM code execution in Type 2 NVM operations, no calls to NVM user routine are allowed. Calling other NVM user routines can change internal NVM registers content thus affecting the completion of the suspended operation.

Table 5-44 RAM branch code structure

RAM Address	RAM content
18000400H	Start of user defined code. It can be directly code or jump to some other RAM location
End of user defined code location	BX LR (Return instruction)

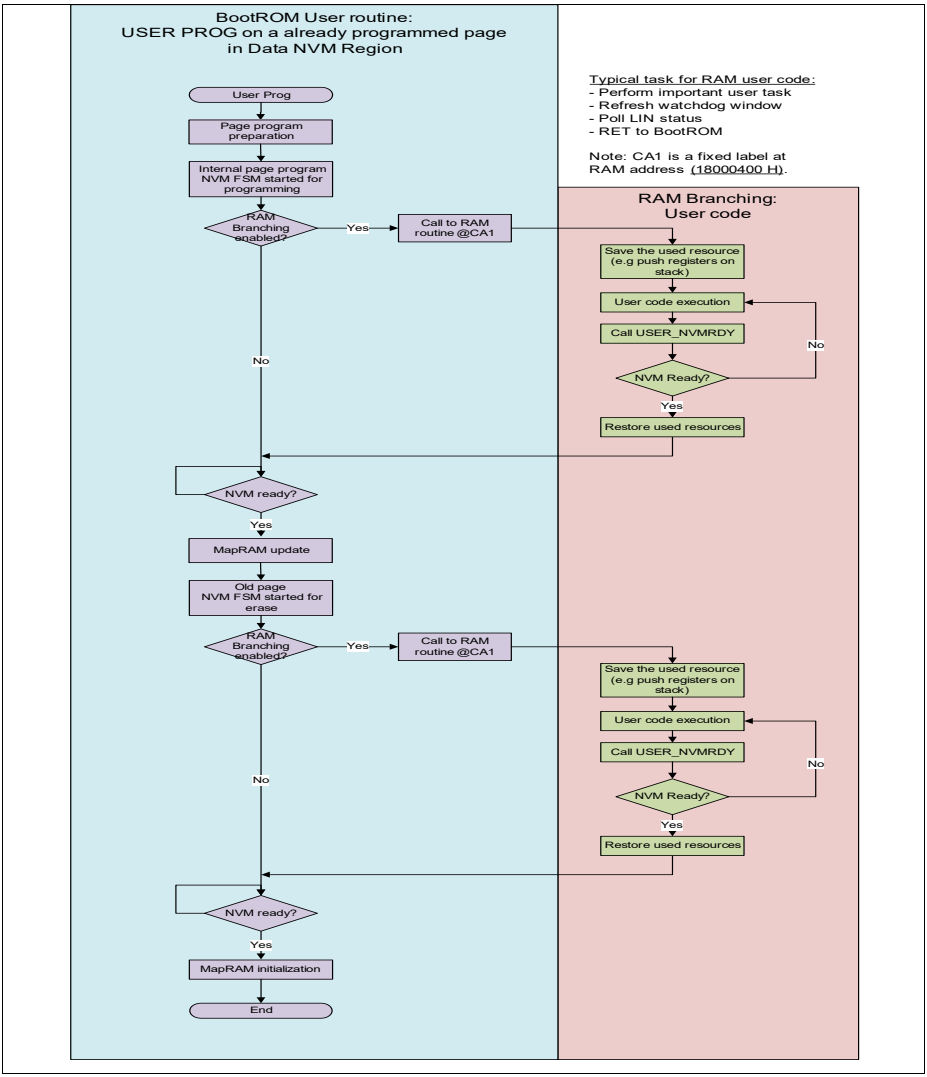


Figure 5-13 Background NVM programming operation with jumps to RAM code (example for non-linearly mapped sector)

5.4.3 Emergency operation handling

Note: Emergency operation provides the possibility to exit an on-going NVM operation in a faster way skipping some internal time consuming steps in case high priority tasks are required. For this reason, leaving an NVM operation via an emergency operation request might leave some inconsistent data into the sector targeted by the interrupted operation. When using this feature on Data NVM sector, the user is recommended, soon after the completion of the execution of the high priority tasks, to issue a pin reset or sleep entry-exit sequence to let the firmware to properly clean the sector.

To ensure that NVM is functioning correctly, all NVM operations (i.e. program or erase) are to be completed before a new NVM operation is started. In addition, corrective activities such as retries and disturb handling are added in an NVM program routine and could add additional time. In an emergency situation, where the system needs to save important user data in the shortest time possible, this becomes critical. Therefore, a mechanism to bypass these corrective activities as well as to inform that a new NVM sequence will not be started, is needed. To support an emergency situation, the following steps are recommended in the code whenever the NVM programming is called.

5.4.3.1 Emergency operation handling - Type 1 routines

For Type 1 routines (including both program and erase), an emergency programming may only be handled with the interrupt enabled shown in **Table 5-45**.

Table 5-45 Emergency operation handling in Type 1 routines

Step	Description
1	User code enables interrupt and sets MEMSTAT.NVMPROP before calling NVM (Program/Erase) routines.
2	While the NVM operation is on-going, an event occurs triggering an interrupt.
3	Interrupt subroutine (ISR) is serviced immediately when the NVM is free.
4	ISR has to check for the MEMSTAT.NVMPROP status. If this bit is set, MEMSTAT.EMPROP has to be set and ISR has to be exited.

Table 5-45 Emergency operation handling in Type 1 routines (cont'd)

Step	Description
5	With control returned to the BootROM, the NVM routines will be executed bypassing the corrective activities. This ensures that the routines are completed in the shortest time possible.
6	Exiting the NVM routines, the user code checks the MEMSTAT.EMPROP. Since it is set, the code can branch to execute a user defined emergency sequence and clear the bits MEMSTAT.NVMPROP and MEMSTAT.EMPROP. These activities can include the programming of the critical data.

5.4.3.2 Emergency operation handling - Type 2 routines

For Type 2 routines (including both program and erase), an emergency programming may be handled with or without the interrupt enabled. In the case with interrupt enabled, it is similar to Type 1 Routines shown in **Table 5-45**. For the case without interrupt enabled, it is shown in **Table 5-46**.

Table 5-46 Emergency operation handling in Type 2 routines (No interrupt)

Step	Description
1	User code sets MEMSTAT.NVMPROP before calling NVM (Program/Erase) routines.
2	While the NVM operation is started, the BootROM jumps to execute a user defined code in the RAM. Within this code, the user checks periodically for critical events.
3	During the checking, an emergency event occurs. The code has to set MEMSTAT.EMPROP and give back control to BootROM.
4	With control returned to the BootROM, the NVM routines will be executed bypassing the corrective activities. This ensures that the routines are completed in the shortest time possible.
5	Exiting the NVM routines, the user code checks the MEMSTAT.EMPROP. Since it is set, the code can branch to execute a user defined emergency sequence and clear the bits MEMSTAT.NVMPROP and MEMSTAT.EMPROP. These activities can include the programming of the critical data.

5.4.3.3 Emergency operation handling timing

In this chapter some information about overall emergency operation worst case timing is provided.
Table 5-47 describes the case in which user data has to be saved into the linear sector due to an emergency event. Flow for programming the critical information in the not lin-

early mapped region of the NVM is similar (step 6 and 7 are inverted and a few µs have to be added for MapRAM update) and overall worst case time is the same.

Table 5-47 Emergency operation handling in Type 1 routines

Phase	Description
1	User code enables interrupt and sets MEMSTAT.NVMPROP before calling NVM (Program/Erase) routines.
2	While the NVM operation is on-going, an event occurs triggering an interrupt. In the worst case interrupt comes soon after a new erase was started.
3	Interrupt subroutine (ISR) is serviced immediately when the NVM is free.
4	With control returned to the BootROM, the NVM routines will be executed bypassing the corrective activities. This ensures that the routines will end in the shortest time possible even if a successful execution of the on going NVM operation is not ensured.
5	Exiting the NVM routines, the user code checks the MEMSTAT.EMPROP. Since it is set, the code can branch to execute a user defined emergency sequence. First step is open AB and load user relevant data.
6	Before programming new data, if target page is already used, a preliminary erase is performed.
7	User critical data are programmed in the target page.

The **Table 5-47** refers to the type 1 routines but data are similar for type 2 routines as well.

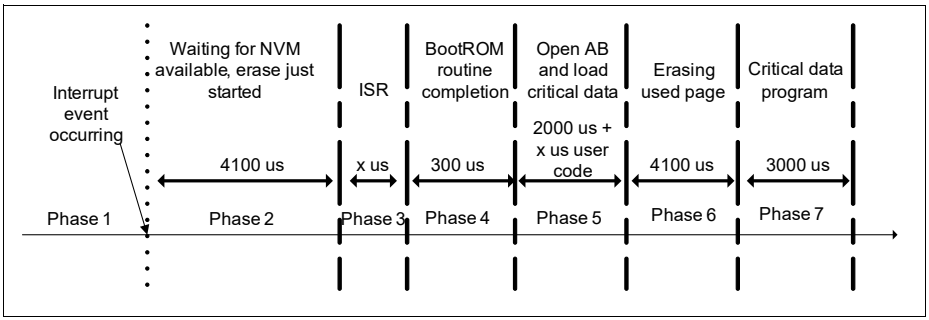


Figure 5-14 Worst case emergency handling timing when linear sector is used

Worst case time, shown in **Figure 5-14**, is then 13.5 ms. This does not include time for user code execution. It can be reduced by about 4.1 ms if the user ensures that the page used for critical data saving is erased.

5.4.4 NVM user routines operation

This section describes the application of some NVM user routines.

5.4.4.1 NVM user programming operation

In TLE986x/TLE987x, the NVM supports programming of up to 128 bytes of data at once. The user can execute the following sequence illustrated in **Figure 5-15** for NVM user programming. Once the assembly buffer has been successfully opened, the user can load the assembly buffer with the user defined contents. This can be achieved by a store instruction targeting the selected byte in the NVM page opened with the OPEN_AB user routine.

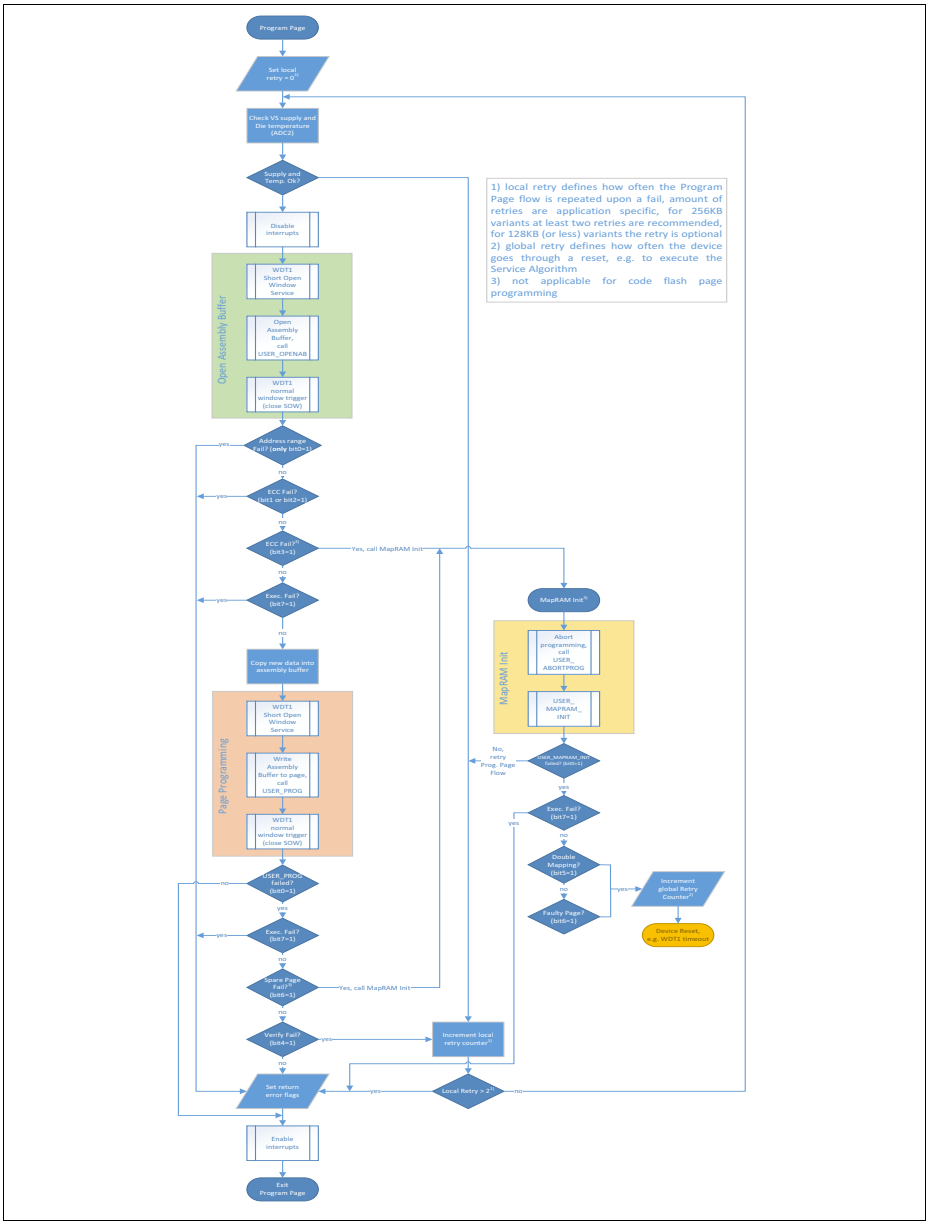


Figure 5-15 NVM user program

5.4.4.2 Tearing-safe programming

In TLE986x/TLE987x, the mapping mechanism of the non-linearly mapped sector is used like a log-structured file system. When a page is programmed in this sector, the old values are not physically overwritten, but a different physical page (spare page) in the same sector is programmed. If the programming fails, the old values are still present in the sector and user can decide, by means of a specific input parameter of the user programming routine (refer to **Table 5-17**), whether the old values or the new failing values should be physically kept in the sector.

When an erase or write procedure is interrupted by a power down, this is identified during the reconstruction of the MapRAM content after the next reset. In this case, the service algorithm routine is automatically started and repairs the NVM state exploiting the fact that either the old or the new data (or both) are fully valid.

5.4.4.3 NVM user erase operation

The user can execute the following sequence illustrated in **Figure 5-16** for NVM user erase.

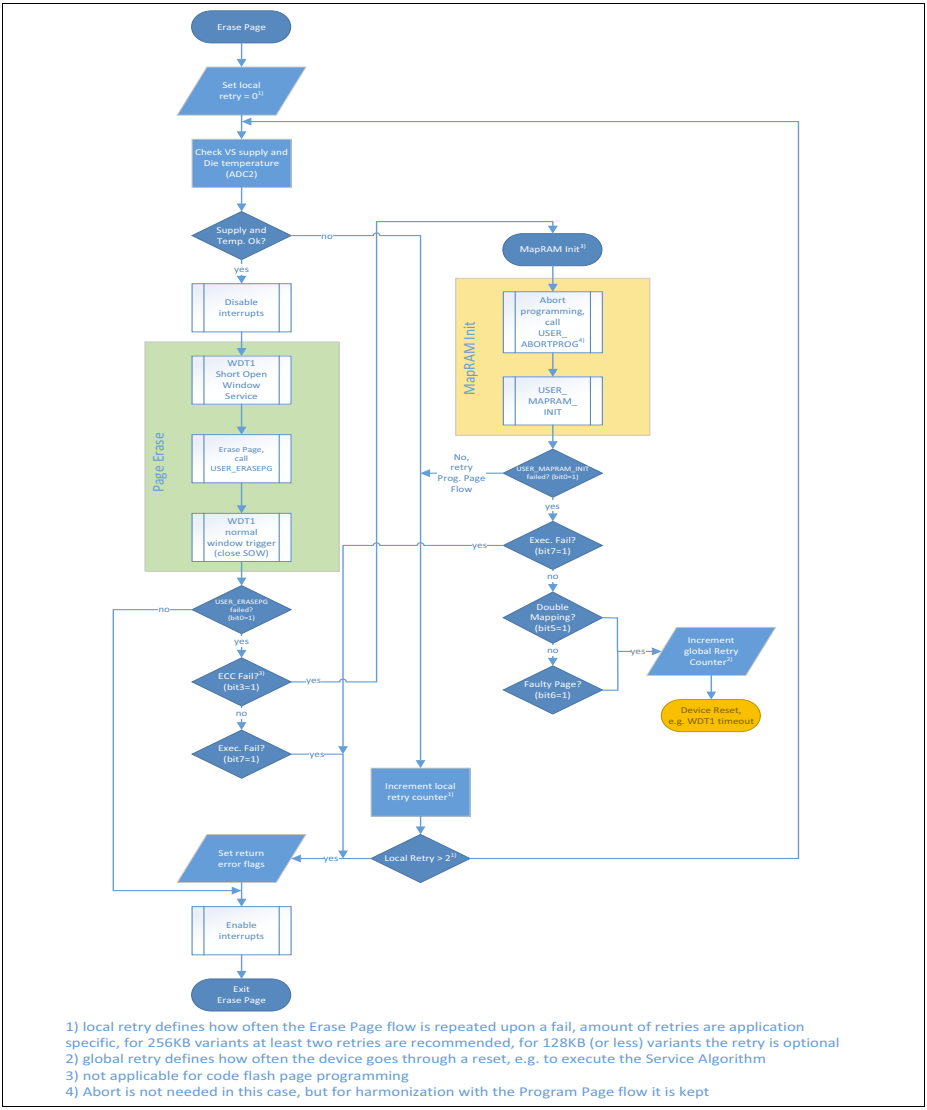


Figure 5-16 NVM user page erase

5.4.4.4 NVM user programming abort operation

The user can execute the following sequence illustrated in **Figure 5-17** for NVM user programming abort.

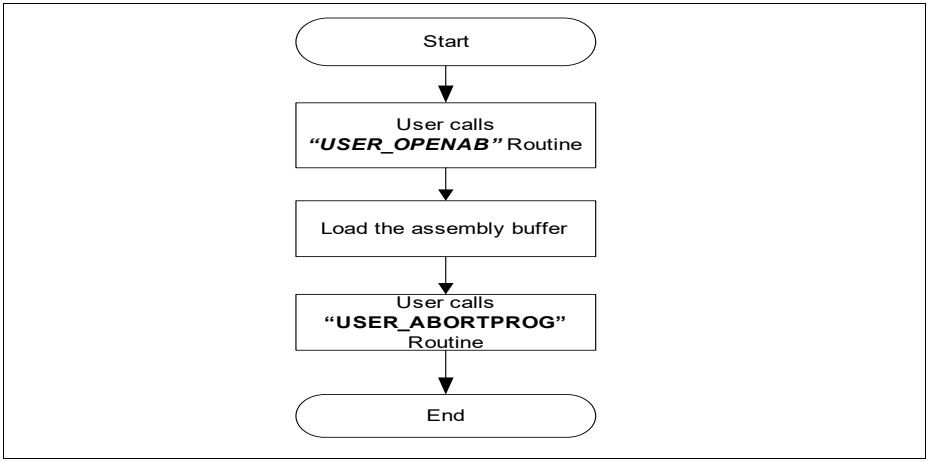


Figure 5-17 NVM user abort program

5.4.5 NVM protection mechanism

The BSL Mode 6 of FastLIN or UART controls the NVM protection, providing or deleting a dedicated password (please refer to the **Section 4.4.2.7**).

Once a valid password (different from 00_H and FF_H) is programmed, write and read from FastLIN or UART is not allowed on both Code and Data NVM regions upon startup, regardless of the reset source.

Moreover, the debugging capability through the SWD interface is reduced as follows: the NVM read access is blocked and there is no possibility to halt the core from any memory location.

During normal operation, if the user wishes to program or read the NVM, the NVM protection can be temporarily be disabled (refer to the **Section 5.3.16**).

Note: The following statement is valid for all the BF-Step devices in VQFN package, produced till December 2022: when the NVM protection is set, the NVM read access from SWD interface is blocked, the core can be halted from memory locations that are not protected. For additional information about the tracking of the devices, please contact Infineon.

6 Revision history

Revision	Date	Changes
1.9	2023-10-23	• Added MOTIX™ • Added UI step • Added “If MSB of the NVM_PASSWORD is 0, only Code Flash mapped sectors are protected. If the bit is 1, both Code Flash and Data Flash are protected.” • **Figure 5-1** updated • **Table 5-14** updated • **Figure 5-15** updated
1.8	2022-09-23	• **Section 4.4.2.2** “If the NVM protection is enabled, programming to RAM is not allowed.” removed • **Section 4.4.2.7** description updated • **Figure 5-1** updated • **Section 5.3.15** third note added • **Section 5.4.5** description updated; note added • **Table 5-14** updated
1.7	2022-05-10	UH-Step added: • Title page and header updated • **Figure 5-1** updated • **Section 5.3.15** note below **Table 5-31** updated • **Section 5.4.5** third paragraph updated
1.6	2021-12-10	Test mode removed (**Section 2.1**, **Table 2-1**)
		Chapter 4 last two paragraphs removed
		Section 4.1, **Section 4.4.2.7** Mode 6 updated
		Table 5-11 CFLASH_PW and DFLASH_PW updated
		Section 5.3.16 last paragraphs updated
		Section 5.4.5 desciption updated
		All chapters editorial changes
1.5	2020-09-25	TLE9862/TLE9872 (256KB) variants added
		Chapter 3.1.1 MBIST timings updated
		Chapter 3.1.4 target PLL frequency updated
		Chapter 3.1.8 target PLL frequency updated
		Table 3-4 order updated

Revision	Date	Changes
		Chapter 4 note regarding the UART interface added
		Table 5-2 256KB NVM size added
		Table 5-3 extended to support 256KB variants
		Table 5-5 PKG_Type extended by TQFP-48
		Table 5-8 NVM_SIZE listed updated to remove irrelevant entries and adding 256KB size
		Chapter 5.2.2 note updated, to use backup values instead of default values
		Figure 5-1 TQFP48 added
		Table 5-13 footnote added to exclude certain function for 256KB variants
		Table 5-14 footnote added to exclude certain function for 256KB variants
		Chapter 5.3 user APIs reordered
		Chapter 5.3.2 timing diagrams of USER_PROG function added
		Chapter 5.3.3 editorial changes
		Chapter 5.3.5 editorial changes
		Chapter 5.3.7 editorial changes
		Chapter 5.3.15 editorial changes
		Table 5-43 SECTORINFO updated
		Chapter 5.4.2 NVM programming timing values updated
		Figure 5-15 NVM user program flow added
		Figure 5-16 NVM user page erase flow added

Edition 2023-10-23

Published by
Infineon Technologies AG
81726 München, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”). 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 your 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 your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems 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.

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG