TLE983x BC-Step

BootROM User Manual

V 2.60, 2012-05

Automotive Power

Edition 2012-05

Published by
Infineon Technologies AG
81726 München, Germany

Table 0-1 Revision History

Version	Date	Comments	Authors
Previous Version: V 2.12			
V2.60	02 May 12	Initial Release	IFX Technologies

1 Introduction

This document specifies the BootROM firmware behavior for the TLE983x family. The specification is organised into the following major sections:

- BootROM Overview
- Startup Procedure
- LIN and UART BSL features
- NVM structure and user routines description.

1.1 Purpose

The document describes the functionalities of the BootROM firmware.

1.2 Scope

The BootROM firmware for the TLE983x family will provide the following features

- Startup procedure for stable operation of TLE983x chip
- BSL mode for users to download and run code from NVM and XRAM
- NVM organization and operation

1.3 Abbreviations and special terms

List of terms and abbreviations used throughout the document:

• BSL	BootStrap Loader
• CS	Configuration Sector
• DAP	Device Access Port
• EOT	End of Transmission
• EVR	Embedded Voltage Regulator
• NAC	No Activity Count
• NAD	Node address for diagnostic
• NVM	Non Volatile Memory
• OCDS	On-Chip Debug Support
• PEM	Program Execution Mode
• SA	Service Algorithm
• WDT	WatchDog Timer

2 Overview

This specification includes the description of the operations and tasks defined to support the general startup behaviour and various boot options

2.1 Firmware architecture

TLE983x on-chip BootROM consists of the startup procedure, the bootstrap loader via LIN, the bootstrap loader via UART, NVM user routines, NVM integrity handling routines and on-chip debug support.

The BootROM in TLE983x, is located at 0000_H in Bank 0 during active memory map 1 and 9000_H in Bank 2 at active memory map 0. Upon a hardware reset, microcontroller will always be in the active memory map 1 (by default) and execute the BootROM code. The beginning of the startup procedure changes the memory map setting and does a long jump (ljmp) to the subroutine, MAIN. This instruction will then capture the absolute address of subroutine MAIN, do a swap to active memory map 0 and execute the remaining BootROM code at 9000_H in Bank 2 in active memory map 0. Active memory map 1 will be just a stepping stone to active memory map 0, all other instructions will be executed in the active memory map 0.

The startup procedure also includes the EVR calibration, map-RAM initialisation, on-chip oscillator configurations, NVM protection enabling and branching to the different modes. The deciding factor will be on the latch values of TMS, P0.0 and P0.2 upon a reset. During reset, these signals are latched at the rising edge of RESET pin.

There are generally 2 operation modes in the BootROM:

- User / BSL mode
- OCDS mode

For user mode, it will just execute the startup procedure and finally jumping to 0000_H to execute the user program.

Table 2-1 lists the boot options available in the TLE983x.

Table 2-1 TLE983x **Boot options**

TMS[1)]/DAP1	P0.0 /DAP0	P0.2	Mode / Comment
0	X	X	User mode / BSL mode [2)][3)]
1	0	X	Device test mode[4)]
1	1	0	OCDS mode with DAP port
1	1	1	Device test mode[4)]

1) When TMS is latched 1 (i.e. high) upon reset, DAP pins will be enabled by hardware. When TMS = 1, P0.0 must be 1. The hardware enable of DAP pins with these boot configuration must not be changed by port control.

2) 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.

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

4) Power up with special internal settings. At completion, device runs in endless loop. No flash code execution is performed.

2.2 Program structure

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

Startup procedure

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

The startup procedure will perform configuration sector verification, EVR calibration, on-chip oscillator trimming, map-RAM 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

It is used to support user code execution at NVM address 0000_H.

LIN BSL mode

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

UART BSL mode

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

OCDS mode

To support the OCDS, a portion of the BootROM is used to store the OCDS initialization, and a Monitor program which will interact with the external debugger. By default, through the OCDS mode, the DAP interface is enabled.

3 Startup procedure

This chapter describes the BootROM startup procedure in TLE983x.

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 and the error which might occur. Refer to **Section 3.1** for further details.

3.1 Program structure

First task executed by the startup firmware is switching to active memory map 0. Afterwards, the type of reset is checked. For power-on reset or brown-out reset or wake-up reset from sleep mode, XRAM and IRAM memory tests and initialization are started while they are skipped for the other resets. Anyhow, in the startup code monitor IRAM is used to avoid destroying the user IRAM content.

After that, depending on the type of reset, the firmware will do NVM protection, NVM map-RAM 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 OCDS mode, the WDT1 is disabled. For entry to user mode, the WDT1 remains active. Next, it will wait for NVM module to be ready.

For software, or watchdog reset, the following steps are skipped:

- NVM Map-RAM initialisation
- Setting of oscillator and PLL
- Download of analog modules trimming parameters from first 100TP page
- Download of user configuration data from OTP or 100TP page into the XRAM
- Switching system clock input to PLL output
- Clearing of NMI status and presupply warning status before exit to user mode or OCDS mode

3.1.1 Test and initialisation of IRAM and XRAM

A functional test on IRAM (both user IRAM and monitor IRAM) and XRAM (optional) is executed after power on reset, brown out reset or wake-up reset from sleep mode. The test consists of a linear write/read algorithm using alternating data.

In case an error is detected the device is set to loop endlessly. Moreover, in case the test is completed successfully, XRAM and IRAMs are initialised 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).

While IRAM tests are automatically started after every power on reset, brown out reset or wake-up reset from sleep mode, the XRAM test is optional. It can be enabled and controlled by proper programming of the bytes stored in first 100TP page as described in the **Table 6-4**.

In particular the relevant parameters are:

- CS_XRAM_MBIST_STARTUP_EN (offset=74_H): When set to $C3_H$ it enables the XRAM test after a power on reset, brown out reset or wake-up reset from sleep mode. All other values will be ignored and XRAM test at startup will not be enabled.
- CS_XRAM_MBIST_LOW_BOUND_H: (offset=75_H): It defines the high Byte of the starting address of the XRAM range to be tested. This Byte is ignored if XRAM test is not enabled.
- CS_XRAM_MBIST_LOW_BOUND_L: (offset=76_H): It defines the low Byte of the starting address of the XRAM range to be tested. This Byte is ignored if XRAM test is not enabled.
- CS_XRAM_MBIST_HIGH_BOUND_H: (offset=77_H): It defines the high Byte of the ending address of the XRAM range to be tested. This Byte is ignored if XRAM test is not enabled.
- CS_XRAM_MBIST_HIGH_BOUND_L: (offset=78_H): It defines the low Byte of the ending address of the XRAM range to be tested. This Byte is ignored if XRAM test is not enabled.

Once the user IRAM test, IRAM initialisation and the optional XRAM test are done, monitor IRAM is selected and the startup procedure proceeds to check on the NVM status while XRAM initialisation is still running. The firmware will wait for XRAM initialization to be completed before exiting to user code.

3.1.2 NVM initialisation routine

This routine will set the NVM protection according to the password in the configuration sector.

3.1.3 NVM map-RAM initialisation

The map-RAM initialisation operation is triggered to restore the map-RAM contents. If one or more errors are detected in the map-RAM initialisation, the service algorithm routine is called to do the repair (refer to **Section 6.4.1**).

3.1.4 Oscillator trimming and system clock selection

After every power on reset, brown out reset or wake-up reset from sleep mode the system runs with an internal low precision clock (nominally 20 MHz). During the start up procedure, the internal oscillator and PLL are trimmed to a fixed standard value of 24 MHz. In order to reduce the boot time, the start up 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. Once user mode is entered, user is allowed to set the final desired frequency by proper register setting or by means of the dedicated user routine (refer to **Section 6.3.26**).

3.1.5 Analog module trimming

In this routine, the trimming values of voltage regulators, LIN module, temperature sensor, LS switch, HS switch and other analog modules are read from the configuration sector and written into the respective external SFR. Protected Bits of the addressed external SFRs will not be affected by this operation. For user mode or OCDS mode, checksum on 100TP page is evaluated. In case of error, default values are used. Refer to **Table 6-4** for a list of user parameters in 100TP page.

3.1.6 User configuration data download

The firmware provides a routine to download data stored in user accessible configuration sector pages (OTP and 100TP) during the startup flow. In particular, the routine copies a specified number of Bytes from a selected CS page (starting always from first Byte in the page) into the XRAM (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 (refer to **Figure 6-1**) as described in the **Table 6-4**. This routine is anyhow not performed after a software or watchdog reset.

Relevant parameters for routine control are:

- CS_USER_CAL_STARTUP_EN (offset=79_H): When set to $C3_H$ it enables the user data download from an OTP or 100TP page into the XRAM 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 XRAM starting address where to copy data downloaded from CS. This Byte is ignored if the routine is not enabled.
- CS_USER_CAL_XADDL: (offset=$7B_H$): It defines the low Byte of the XRAM starting address where to copy data downloaded from CS. This Byte is ignored if the routine is not enabled.
- CS_USER_CAL_CS_PAGE: (offset=$7C_H$): It defines the CS page where data has to be downloaded from (refer to **Figure 6-1**). 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 CS page. This Byte is ignored if the routine is not enabled.

The routine has been developed to support downloading of the ADC1 calibration parameters stored at the beginning of the first 100TP page (see **Table 6-4**) into the XRAM for an easy access but can be more generally used for all other CS user parameters. If the routine is enabled, firmware will wait for XRAM initialization completion before copying the data. Moreover, independent of startup setting, a similar routine is provided as NVM user routine (refer to **Section 6.3.6**)

3.1.7 User / BSL mode entry

Entry to user mode is determined by the No Activity Count (NAC) value which is defined in the user code.

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

Before exiting to user mode, the system clock frequency is switched to PLL output previously set by default to 24 MHz. In case PLL has not locked within 1 ms, the CPU clock source LP_CLK (low precision clock running nominally at 20 MHz) will be used. All SFRs are reset to default values and the user internal ram is selected.

Note: User mode is entered jumping to NVM starting address. 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 0000_H is not FF_H or (2) when NVM is protected. In all other cases, firmware will put the device in sleep mode.

3.1.7.1 NAC definition

The NAC value specifies the duration of delay before jumping to user mode measured from the reset release. The Bit 7 of the NAC will determine which BSL mode to enter. In particular, the device will enter LIN BSL mode if the NAC Bit 7 is equal to zero and UART BSL mode if it is equal to one.

After ending the start up procedure, the program will detect any activities on the LIN bus / UART for a period of time, determined by $(((NAC \& 7F_H) - 1_H) * 5)$ ms reduced by the time already spent to perform the start up procedure. When nothing is detected on the LIN bus / UART and $(((NAC \& 7F_H) - 1_H) * 5)$ ms is passed from reset going high, the microcontroller will jump to user mode. Anyhow, if NAC is 1_H or 81_H, user mode is entered immediately.

NAC value is restricted to C_H as the first open WDT1 window is worst case 65 ms. The firmware has to either refresh the WDT within the 65 ms or jump to user mode. If NAC is not valid, BootROM code will switch off the WDT and wait for a LIN frame indefinitely.

Table 3-1 gives an overview of the action of the microcontroller with respect to No Activity Count (NAC) values

Note: Timer 0 is initialized to have 5 ms overflow and is used to create the delay.

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

NAC Value	Action
01_H, 81_H	0 ms delay. Jump to user mode immediately
02_H, 82_H	5 ms delay before jumping to user mode[1]
03_H, 83_H	10 ms delay before jumping to user mode [1]
04_H, 84_H	15 ms delay before jumping to user mode [1]
05_H, 85_H	20 ms delay before jumping to user mode [1]
06_H, 86_H	25 ms delay before jumping to user mode [1]
07_H, 87_H	30 ms delay before jumping to user mode [1]
08_H, 88_H	35 ms delay before jumping to user mode [1]
09_H, 89_H	40 ms delay before jumping to user mode[1]
$0A_H$, $8A_H$	45 ms delay before jumping to user mode[1]
$0B_H$, $8B_H$	50 ms delay before jumping to user mode[1]
$0C_H$, $8C_H$	55 ms delay before jumping to user mode[1]
$0D_H$ - $7F_H$, 00_H, Invalid	Wait forever for the first LIN frame
$8D_H$ - FF_H, 80_H	Wait forever for the first UART frame

1) If a LIN 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 LIN / UART frame accordingly

The NAC value is stored, together with the NAD value, in the last 4 Bytes of the linearly mapped NVM region. The **Table 3-2** shows the addresses for the all the available family devices. To ensure the parameter validity, the 2 parameters actual values and their inverted values 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 will be used.

Table 3-2 NAC and NAD parameters details

Address	User Defined Value	Criteria / Range	Default
$YFFC_H$[1]	NAC	01_H - $0C_H$ for LIN BSL 81_H - $8C_H$ for UART BSL	$7F_H$
$YFFD_H$[1]	$\overline{NAC}$	Inverted NAC value	
$YFFE_H$[1]	NAD (for LIN BSL only)	01_H - FF_H (00_H is reserved)	$7F_H$
$YFFF_H$[1]	$\overline{NAD}$ (for LIN BSL only)	Inverted NAD value	-

[1] Y is equal to 4, 7, A or E for the 24, 36, 48 or 64 kByte respectively

4 LIN BSL mode

LIN BSL is a LIN like protocol based on LIN 2.0 but for security reason the checksum is inverted for most of the supported modes. Standard LIN protocol can support a max. baud rate of 20 kBaud. FastLIN BSL protocol is an enhanced feature implemented in TLE983x device. This is introduced to support baud rates of 20 kBaud to 57.6 kBaud and 115.2 kBaud via a single-wire UART using UART BSL protocol (See **Section 4.6**).

4.1 LIN BSL features

Features that are implemented include:

1. Re-synchronization to the transfer speed (baud rate) of the communication partner upon receiving every frame
2. Using Diagnostic Frame (Master Request and Slave Response)
3. Usage of user values (NAD and NAC) stored in uppermost linearly mapped NVM
4. Non standard LIN checksum (Programming checksum, see **Section 4.4.3.2**)
5. Fast LIN BSL using UART protocol on single-wire UART (LIN)

4.2 LIN BSL mode overview

The LIN BSL mode consists of three functional phases described below:

- **Phase I**: To establish a connection with every frame (Master Request or Slave Response frame) received by automatically synchronizing to the transfer speed (baud rate) of the communication partner (host).
- **Phase II**: To execute the host specified command. In order to execute the commands, host needs to send a Master Request Header first, followed by a Command frame. The selected mode information is embedded in the Command frame.
- The possible modes are:
  – **Mode 0 (00_H)**: Transfer a user program from the host to XRAM[1)]
  – **Mode 1 (01_H)**: Execute a user program in the XRAM[2)]
  – **Mode 2 (02_H)**: Transfer a user program from the host to NVM[1)]
  – **Mode 3 (03_H)**: Execute a user program in the NVM[2)]
  – **Mode 4 (04_H)**: Erase NVM[1)]
  – **Mode 6 (06_H)**: NVM Protection mode enabling/disabling Scheme[2)]
  – **Mode 8 (08_H):** Transfer a user program from the host to XRAM using classic LIN checksum[3)]
  – **Mode 9 (09_H)**: Execute a user program in the XRAM using classic LIN checksum[4)]
  – **Mode A ($0A_H$)**: Get info (based on Option Byte)[1)]

1) The microcontroller returns to the beginning of Phase I/II and wait for the next command from the host
2) LIN BSL and serial communication are exited.
3) Similar to mode 0. mode 8 uses classic LIN checksum instead of Programming checksum.
4) Similar to mode 1. mode 9 uses classic LIN checksum instead of Programming checksum.

LIN BSL supports Fast Programming through modes 0, 2 and 8 with the selection of Fast Programming Option. Refer to **Section 4.4.4.2** for more details.

- **Phase III**: To send microcontroller status to host. In order to receive the microcontroller status, host needs to send a Slave Response Header first.

Re-synchronization and setup of baud rate (Phase I) are done at all times (before Phases II and III). Thus, different baud rates can be supported. Phase II is entered when its Master Request Header is received, otherwise Phase III is entered (Slave Response Header). The Master Request Header has a Protected ID of $3C_H$ while the Slave Response Header has a Protected ID of $7D_H$. The Command and Response frames are identified as Diagnostic LIN frame which has a standard 8 data Byte structure (instead of 2 or 4).

Figure 4-1 shows the relationship between the PC host and the microcontroller for the 3 phases, while **Figure 4-2** shows the Master Request Header, Slave Response Header, Command and Response frames.

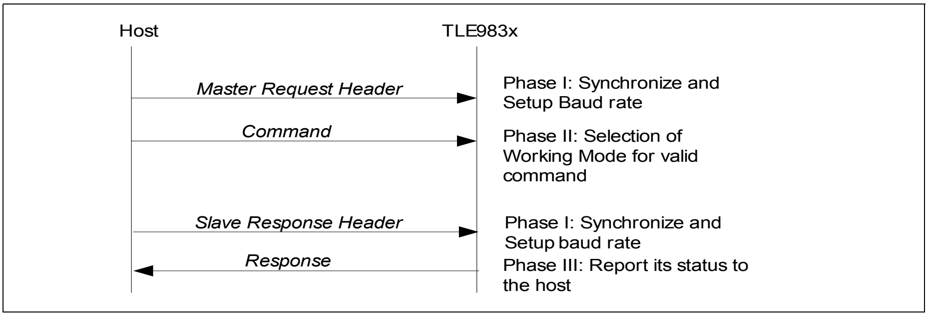

Host
TLE983x
Master Request Header
Phase I: Synchronize and Setup Baud rate
Command
Phase II: Selection of Working Mode for valid command
Slave Response Header
Phase I: Synchronize and Setup baud rate
Response
Phase III: Report its status to the host

Figure 4-1 LIN mode - Phases I, II and III

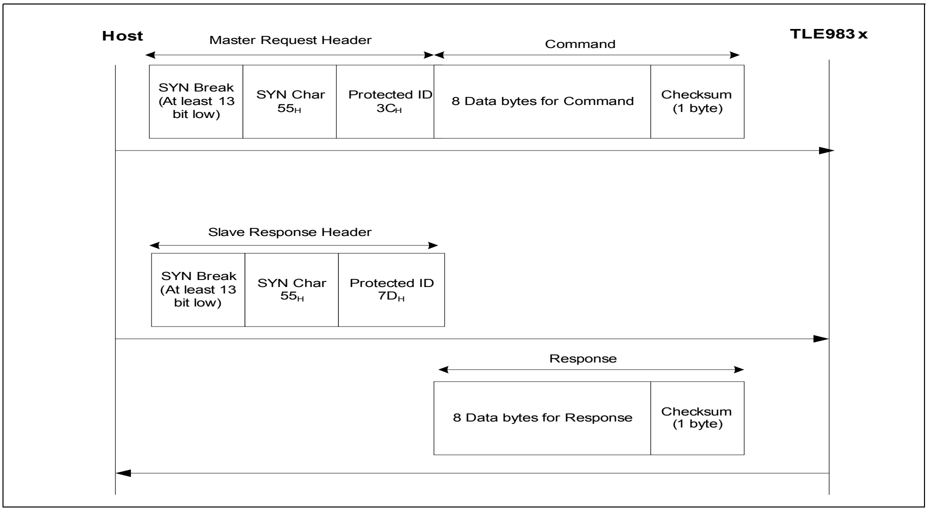
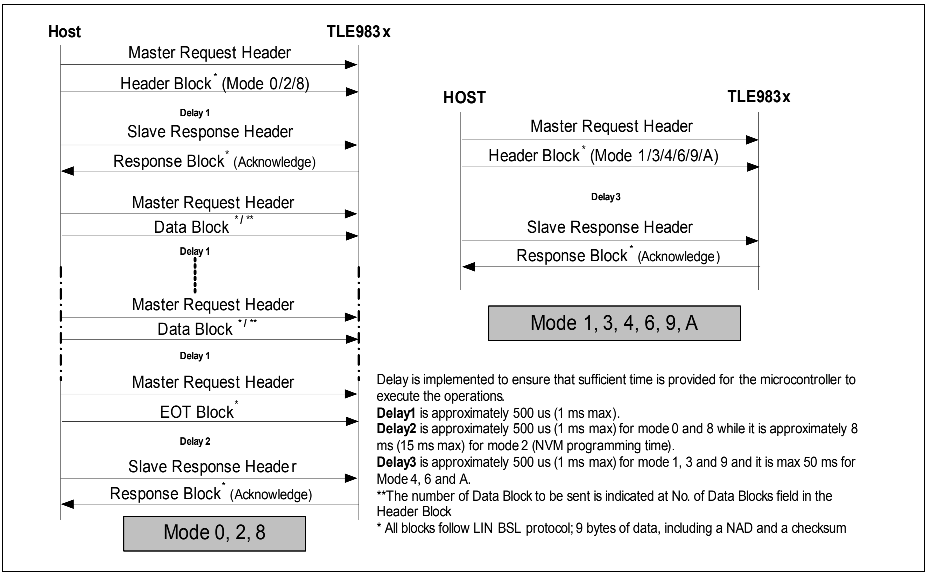


Figure 4-2 LIN mode - Frames

For all modes' entry, the Master Request Header is transmitted from host to microcontroller, followed by the command, which is the header block. The Slave Response Header is transmitted to check the status of the operation. For mode 0, 2 and 8, there is no need to send a Slave Response Header after every data block. The microcontroller supports multiple data block transfers (up to 256 data blocks) without sending a Slave Response Header, which saves overhead. As the commands are sent one after another without waiting for any status indication, a certain delay is required as shown in **Figure 4-3** to ensure sufficient time is provided for the microcontroller to execute the desired operations.



Delay is implemented to ensure that sufficient time is provided for the microcontroller to execute the operations.
Delay1 is approximately 500 us (1 ms max).
Delay2 is approximately 500 us (1 ms max) for mode 0 and 8 while it is approximately 8 ms (15 ms max) for mode 2 (NVM programming time).
Delay3 is approximately 500 us (1 ms max) for mode 1, 3 and 9 and it is max 50 ms for Mode 4, 6 and A.
**The number of Data Block to be sent is indicated at No. of Data Blocks field in the Header Block
* All blocks follow LIN BSL protocol; 9 bytes of data, including a NAD and a checksum

Figure 4-3 Communication structure of the LIN BSL modes

4.3 Phase I: Automatic synchronization to the host

Upon entry to LIN mode, a connection is established. The transfer speed (baud rate) of the device is automatically synchronized to the serial communication partner (host) in the following steps:

STEP 1: Initialize LIN interface for reception and timer 2 for baud rate measurement

STEP 2: Wait for an incoming frame from host

STEP 3: Synchronize the baud rate to the host

STEP 4: Enter Phase II (for Master Request Frame) or Phase III (for Slave Response Frame)

*Note: Re-synchronization and setup of baud rate are always done for **every** Master Request Header or Slave Response Header frame.*

4.3.1 General description

The LIN baud rate detection feature provides the capability to detect the baud rate within the LIN protocol using timer 2. Initialization consists of:

- Serial port of the microcontroller set to mode 1 (8-bit UART, variable baud rate) for communication

- baud rate range for detection, controlled by the field BGSEL of the BCON, set to “5.5 to 166.7 kBaud”.
- Capture Timer 2 data register contents on negative transition at pin T2EX
- Timer 2 external events are enabled (EXF2 flag is set when a negative transition occurs at pin T2EX)
- $f_{T2}=f_{PCLK}$ / 8 (T2PRE=011_B)

As shown in **Figure 4-2**, the LIN Header frame consists of the:

- synch Break (13 Bits time low)
- synch Byte (55_H)
- Protected ID field

The Break is used to signal the beginning of a new frame and must be at least 13 Bits of dominant value. When negative transition is detected at pin T2EX at the beginning of Break, the Timer 2 External Start Enable Bit (T2MOD.T2RHEN) is set. This will automatically start Timer 2 at the next negative transition of pin T2EX. Finally, the end of synch Byte flag (LINST.EOFSYN) is polled. When this flag is set, Timer 2 is stopped. T2 Reload/Capture register (RC2H/L) is the time taken for 8 Bits. Then the LIN routine calculates the actual baud rate, sets the PRE and BG values and activates baud Rate Generator. The baud rate detection for LIN is shown in **Figure 4-4**

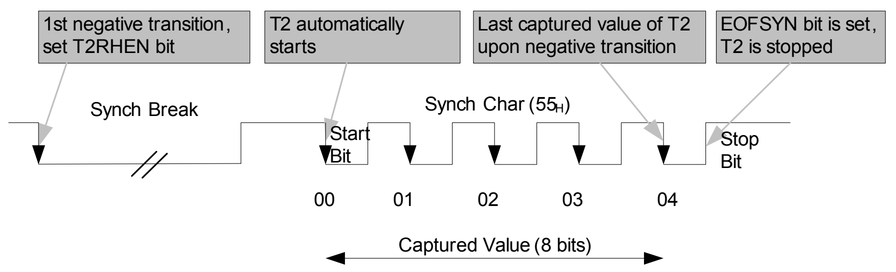

1st negative transition, set T2RHEN bit
T2 automatically starts
Last captured value of T2 upon negative transition
EOFSYN bit is set, T2 is stopped
Synch Break
Synch Char (55H)
Start Bit
Stop Bit
00
01
02
03
04
Captured Value (8 bits)

Figure 4-4 LIN autobaud rate detection

4.3.2 Calculation of BR_VALUE and PRE values

To set up auto baud rate detection, the BG and PRE values must be calculated. As there are two unknown values, two formulas are therefore needed. Firstly, the correlation between the baud rate (baud) and the reload value BR_VALUE (stored in the registers BGL and BGH) depends on the internal peripheral frequency (f_{PCLK}):

$$\mathrm{baud} = \frac{f_{PCLK}}{16 \times PRE \times (BR\_VALUE)} \tag{4.1}$$

The previous reported formula is valid in the hypothesis that the use of fractional divider is not required.

Secondly, the relation between the baud rate (baud) and the captured value of Timer 2 (T2) depends on the T2 peripheral frequency (f_{T2}) and the number of received Bits (N_b):

$$\mathrm{baud} = \frac{f_{T2} \times N_b}{T2} \tag{4.2}$$

Combining **Equation [4.1]** and **Equation [4.2]** with N_b=8, f_{T2}=f_{PCLK} / 8 (T2PRE=011_B) results in the following:

$$\frac{f_{PCLK}}{16 \times PRE \times (BR\_VALUE)} = \frac{\frac{f_{PCLK}}{8} \times 8}{T2} \tag{4.3}$$

By simplifying **Equation [4.3]**, the following is obtained:

$$PRE \times (BR\_VALUE) = \frac{T2}{16} \tag{4.4}$$

After setting BR_VALUE and PRE, the baud rate generator will then be enabled, and the subsequent Command frame or Response frame will follow this baud rate.

115.2 kBaud for FAST LIN

To support FAST LIN with baud rate 115.2 kBaud, fractional divider needs to be enabled. The detection of 115.2 kBaud is determined by the T2 timing. If T2H and T2L is less than 154_H (i.e. baud rate roughly above 70 kBaud), baud rate will be set to 115.2 kBaud.

In such a case, the correlation between the baud rate, the reload value BR_VALUE, the fractional divider setting and the internal peripheral frequency (f_{PCLK}) is:

$$\mathrm{baud} = \frac{f_{PCLK}}{16 \times PRE \times \left(BR\_VALUE + \frac{FDSEL}{32}\right)} \tag{4.5}$$

As a consequence, the settings for 115.2 kBaud are:

- BR_VALUE = 13 (SFR BGL.BR_VALUE = 101_B and SFR BGH = 1_H)
- PRE (SFR BCON.BRPRE) = 000_B
- Read FD_SEL value from configuration sector and store into SFR BGL. FD_SEL

4.4 Phase II: LIN BSL communication protocol and the working modes

Once successful synchronization to the host is completed (with a Master Request Header), the routine enters Phase II. Here, the host communicates to the microcontroller the desired working mode.

A simple transfer protocol is defined for the communication between the host and TLE983x. The protocol data is performed in information blocks. The information block follows a specified block structure and termed transfer block. Each transfer block is 9 Bytes long. A transfer block has the following structure:

NAD (1 byte)	**Block Type** (1 byte)	**Data Area** (6 bytes)	**Checksum** (1 byte)

- **NAD**: Node Address for Diagnostic, specifies the address of the active slave node. See **Section 4.4.1**.
- **Block Type**: This field determines the type of the message (See **Section 4.4.2**).
- **Data Area**: This is the data of the block. The length is fixed at 6 Bytes.
- **Checksum**: This checksum is calculated based on the NAD, Block Type and Data Area. See **Section 4.4.3**.

4.4.1 Node Address for Diagnostic (NAD)

This field specifies the address of the active slave node. Only slave nodes have an address. The NAD address range supported in TLE983x is listed in **Table 4-1**.

Table 4-1 NAD address range

NAD Value	Description
00_H	Invalid Slave Address
$7F_H$	Default Address (NAD value is invalid or it is not programmed in NVM linear area)
01_H to $7E_H$ 80_H to FF_H	Valid Slave Address

Note: LIN block with Broadcast NAD ($7F_H$) is ignored if valid NAD value is programmed in NVM linear area.

*Note: For NAD address and details refer to **Table 3-2**.*

4.4.2 Block type

This field determines the types of transfer blocks. There are 3 transfer block types shown in **Table 4-2**.

Table 4-2 Type of transfer block

Block Name	Block Type	Description
Header block	00_H	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 is used in working modes 0, 2 and 8 to transfer a portion of program code. The program code is in the data area of the block.
End of Transmission (EOT) block	02_H	This block is the last block in data transmission in working modes 0, 2 and 8. The last program code to be transferred is in the data area of the block.

4.4.3 Checksum

Diagnostic LIN frame always uses classic checksum where checksum calculation is over the data Bytes only. The Checksum is the last field of Command and Response LIN frames. For TLE983x, there are 2 types of checksum implemented, Classic (LIN) and Programming checksum. Both Programming and LIN Checksum are supported and are indicated in the respective modes.

4.4.3.1 Classic / LIN checksum

The classic checksum is a standard LIN checksum used for communication with LIN 2.0 slaves. The classic checksum contains the inverted eight Bits sum with carry[1] over all data Bytes.

4.4.3.2 Programming checksum

The programming checksum, or Inverted Classic checksum is a non-LIN standard checksum. This is implemented in TLE983x to allow other slaves (not in TLE983x BSL mode) on the LIN bus to ignore this Programming frame. The inversion of the classic checksum yields the programming checksum.

[1] the checksum is calculated summing all values (8-bit sum with carry) and subtracting 255 every time the sum is greater or equal to 256 (which is not the same as modulo-255 or modulo-256).

An example of the calculation of the Programming checksum is provided in **Table 4-3**. For this example, data of $4A_H$, 55_H, 93_H and $E5_H$ is considered. The calculated programming checksum is 19_H. The classic checksum is an inversion of the programming checksum value (i.e. $E6_H$).

Table 4-3 Programming checksum

Addition of data	HEX	Result	CARRY	Addition with CARRY
$4A_H$	$4A_H$	$4A_H$	0	$4A_H$
($4A_H$) + **55_H**	$9F_H$	$9F_H$	0	$9F_H$
($9F_H$) + **93_H**	0132_H	32_H	1	33_H
(33_H) + **$E5_H$**	0118_H	18_H	1	**19_H**

4.4.4 Mode selection

When Phase II is entered, TLE983x waits for the Command frame and the header block from the host containing indication about the desired mode to be selected.

4.4.4.1 Receiving the header block

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

NAD (1 byte)	**Block Type** **00_H** (Header Block)	Data Area		**Checksum** (1 byte)
		Mode (1 byte)	**Mode Data** (5 bytes)	

Description:

- **NAD:** Node Address for Diagnostic. See **Section 4.4.1**
- **Block Type 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 **Section 4.2**
- **Mode Data:** Five Bytes of special information to activate corresponding mode.
- **Checksum:** The programming or LIN checksum of the header block.

Note: Mode 8 and mode 9 support LIN checksum, while mode 0 - 4, 6, and A support Programming checksum.

4.4.4.2 The activation of working mode 0, 2 and 8

Mode 0, 2 and 8 are used to transfer a user program from host to microcontroller. Mode 0 and 8 allow XRAM transfers, while mode 2 allows NVM transfers.

The header block has the following structure:

The header block

NAD (1 byte)	**00_H** (Header Block)	**$00_H/02_H/08_H$** (Mode 0/2/8)	**Mode Data** (5 bytes)					**Checksum** (1 byte)
			Start Addr High (1 byte)	**Start Addr Low** (1 byte)	**No of Data Blocks Used** (1 byte)	**Not Used** (1 byte)	**Fast_ Prog** (1 byte)	

Mode Data Description:

Start Addr High, Low: 16-bit Start Address, which determines where to copy the received program codes in the XRAM/NVM[1].

No. of data blocks used: Total number of data blocks to be sent, maximum 256 (FF_H) for mode 0 and 8 and maximum 21 (15_H) for mode 2. It is verified when EOT block is received. If number does not match, microcontroller will send a Block Type Error. PC host will then have to re-send the whole series of blocks (header, data and EOT blocks).

Not used: This Byte is not used and will be ignored in mode 0/2/8.

Fast_Prog: Indication Byte to enter Fast LIN BSL[2]

- 01_H: Enter Fast LIN BSL
- Other values: Ignored. Fast LIN BSL is not entered.

Note: The programming of NVM in mode 2 will be started after 128 Bytes or EOT are received. All Bytes sent during the program operation will be lost.

When this Command frame (header block) is used for entering Fast LIN BSL, no other Master Request Header and Command frames (for data block or EOT block) should be sent. Instead, the microcontroller expects a Slave Response Header frame and sends a Response frame to Acknowledge receiving correct header block to enter Fast LIN BSL where UART BSL protocol is used. See **Section 4.6**

On successful receipt of the header block, the microcontroller enters mode 0/2/8, whereby the program code is transmitted from the host to the microcontroller by data block and EOT block, which are described below.

1) NVM address should be aligned to the Page address (low Byte of the start address equal to 00_H or 80_H). If the data starts in a non-page address, PC host should fill up the beginning vacancies with 00_H and provide the start address of that page address.

2) In the case NVM is protected, entry to FastLIN BSL is not possible.

The data block

NAD (1 byte)	01_H (Data Block 1 byte)	Data Area (6 bytes)	Checksum (1 byte)
		Program Code (6 bytes)	

Data area Description:

Program Code: The program code has a fixed length of 6 Bytes per data block.

The EOT block

NAD (1 byte)	02_H (EOT Block 1 byte)	Data Area (6 bytes)			Checksum (1 byte)
		Last_Code length (1 byte)	**Program Code** (Last_Codelength bytes)	**Not Used** (6-1-Last_Codelength bytes)	

Data area Description:

Last_Codelength: This Byte indicates the length of the program code in this EOT block.

Program Code: The last program code (valid data) to be sent to the microcontroller.

Not used: The length is (6 - 1 - Last_Codelength). These Bytes are not used and they can be set to any value.

Note:

1. *NVM programming needs to be performed in multiples of page, 1 page is 128 Bytes. Host is expected to introduce a delay of 15 ms after 128 Bytes of program code are sent. Refer to example given below on mode 2 downloading.*
2. *To prevent external access, once the NVM is protected, modes 0, 2 and 8 are not accessible.*

Table 4-4 Example for 200 Bytes downloading using mode 0/8 and mode 2

Mode 0/8 - (XRAM download)	Mode 2 - (NVM download)
Send Master Request Header	Send Master Request Header
Send header block • No of data blocks used = 33 • Start address (e.g. F000$_H$)	Send header block • No of data blocks used = 21 • Start address (e.g. 0100$_H$)
Delay	Delay
Send Slave Response Header	Send Slave Response Header
Check for Acknowledge	Check for Acknowledge
Send 33 times (Master Request Header + data blocks) (Delay after each data block required)	Send 21 times (Master Request Header + data blocks) (Delay after each data block required)
Send Master Request Header	Send Master Request Header
Send EOT block • Last_Codelength = 2	Send EOT Block • Last_Codelength = 2
Delay	Delay
Send Slave Response Header	Send Slave Response Header
Check for Acknowledge	Check for Acknowledge
	Send Master Request Header
	Send header block • No of data blocks used = 12 • Start address (e.g. 0180$_H$)
	Delay
	Send Slave Response Header
	Check for Acknowledge
	Send 12 times (Master Request Header + data blocks) (Delay after each data block required)
	Send Master Request Header
	Send EOT block • Last_Codelength = 0
	Delay
	Send Slave Response Header
	Check for Acknowledge

33 blocks * 6 Bytes + 2 Bytes = 200 Bytes

4.4.4.3 The activation of working mode 1, 3 and 9

Mode 1, 3 and 9 are used to trigger execution of a user program by the microcontroller. Mode 1 and 9 trigger execution of user program in XRAM at address F000$_H$. Mode 3 triggers execution of user program in NVM at address 0000$_H$.The header block for this working mode has the following structure:

The header block

NAD (1 byte)	**00$_H$** (Header Block)	**01$_H$/03$_H$/09$_H$** (Mode 1/3/9)	**Mode Data**	**Checksum** (1 byte)
			Not Used (5 bytes)	

Mode Data Description:

Not used: The five Bytes are not used and will be ignored in mode 1/3/9.

For modes 1, 3 and 9, the header block is the only transfer block to be sent by the host followed by a Slave Response Header. The microcontroller will send a response block (Acknowledgement code, 55$_H$), exit the LIN BSL and jump to the XRAM address at F000$_H$ (mode 1 and mode 9) or jump to NVM address at 0000$_H$ (mode 3) respectively.

Note: For mode 3, jump to NVM will only occur either (1) when NVM is not protected and NVM content at 0000$_H$ is not FF$_H$; or (2) when NVM is protected. In all other cases, firmware will put the device in sleep mode.

4.4.4.4 The activation of working mode 4

Mode 4 is used to erase the NVM. 3 different options are supported:

- Option 00$_H$: Page Erase
- Option 40$_H$: Sector Erase
- Option C0$_H$: Mass Erase

The header block for Option = 00$_H$ has the following structure:

The header block for page erase (Option = 00$_H$)

NAD (1 byte)	**00$_H$** (Header Block)	**04$_H$** (Mode 4)	**Mode Data** (5 bytes)					**Checksum** (1 byte)
			Addr High byte (1byte)	**Addr Low byte** (1 byte)	**Not Used** (1byte)	**Not Used** (1 byte)	**Option** = 00$_H$ (1 byte)	

Mode Data Description:

Start Addr High, Low: 16-bit address of the NVM page to be erased.

Not used: This Byte is not used and will be ignored in mode 4.

Option: set to 00_H to enable page erase.

When the Option Byte is 00_H, the NVM page selected by the address provided in the Mode Data field is erased. The address should be aligned with the beginning of the chosen page.

The header block for sector erase (Option = 40_H)

The header block for Option = 40_H has the following structure:

NAD (1 byte)	**00_H** (Header Block)	**04_H** (Mode 4)	**Mode Data** (5 bytes)					**Checksum** (1 byte)
			Addr High byte (1 byte)	**Addr Low byte** (1 byte)	**Not Used** (1 byte)	**Not Used** (1 byte)	**Option** = 40_H (1 byte)	

Mode Data Description:

Start Addr High, Low: 16-bit address of the NVM sector to be erased.

Not used: This Byte is not used and will be ignored in mode 4.

Option: set to 40_H to enable sector erase.

When the Option Byte = 40_H, the NVM sector selected by the address provided in the Mode Data field is erased. The address should be aligned with the beginning of the chosen sector.

The Header for mass erase (Option = $C0_H$)

The header block for Option = $C0_H$ has the following structure:

NAD (1 byte)	**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: This Byte is not used and will be ignored in mode 4.

Option: set to $C0_H$ to enable mass erase.

When the Option Byte = $C0_H$, mass erase on all the sectors in the NVM unit is performed.

Note: Not Used Bytes should be set to 0.

Note: When NVM is protected, mode 4 is not accessible and so NVM cannot be erased.

4.4.4.5 The activation of working mode 6

Mode 6 is used to enable or disable the NVM protection mode (read and write protection of the Linearly and Non-Linearly mapped sectors) via the given user-password. The header block for this working mode has the following structure:

The header block

NAD (1 byte)	**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 device is unprotected, the provided user-password will be set as NVM_PASSWORD and internally stored. 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. If MSB of the NVM_PASSWORD is 0, only NVM Linearly mapped sectors are erased. If the Bit is 1, both NVM Linearly and Non-linearly mapped regions 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 Byte will be returned instead of Acknowledge.

Note:

1. *Password value has to be different from 00_H and FF_H.*
2. *When disabling NVM protection, together with NVM, the NAC and NAD values are erased too. As a result, after next reset, default NAD will be used and chip waits for ever for the first BSL LIN frame.*

4.4.4.6 The activation of working mode A

Mode A is used to get 4 Bytes Chip ID data, NVM page or CS page or mass NVM checksum check info depending on the Option Byte value in the header block.

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: Configuration sector page checksum check

The header block - Get 4 Bytes Chip ID (Option = 00_H)

The header block for Option = 00_H has the following structure:

NAD (1 byte)	**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.

Option: set to 00_H to enable get 4 Bytes Chip ID info.

When the Option Byte = 00_H, the 4 Byte Chip ID Number will be returned (see **Chapter 4.5.3**).

The header block - NVM page checksum check (Option = 10_H)

The header block for Option = 10_H has the following structure:

NAD (1 byte)	**00_H** (Header Block)	**$0A_H$** (Mode A)	**Mode Data** (5 bytes)					**Checksum** (1 byte)
			Start Addr High (1 byte)	**Start Addr Low** (1 byte)	**Exp CHKS High** (1 byte)	**Exp CHKS 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).

Exp. CHKS High, Low: Expected checksum High/Low Byte.

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

This option will trigger a checksum calculation (16 bits inverted XOR) 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). The response frame will then return an Acknowledge followed by four data Bytes. These 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.

The input address should always be aligned with a 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.

The header block - Mass NVM checksum check (Option = 18_H)

The header block for Option = 18_H has the following structure:

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

Mode Data Description:

Not used: These Bytes are not used and will be ignored.

Exp. CHKS High, Low: Expected checksum High/Low Byte.

Option: set to 18_H to enable Mass NVM checksum check.

Checksum (16 Bits inverted XOR) on the whole linearly and non-linearly mapped sectors (configuration sector pages not included) is calculated and then compared with the expected values (provided as well as an input). The response frame will then give back a pass or fail indication plus the calculated checksum.

The header block - Configuration sector page checksum check (Option = 50_H)

The header block for Option = 50_H has the following structure:

NAD (1 byte)	**00_H** (Header Block)	**$0A_H$** (Mode A)	**Mode Data** (5 bytes)					**Checksum** (1 byte)
			CS Page (1 byte)	**Not Used** (1 byte)	**Exp CHKS High** (1 byte)	**Exp CKSum Low** (1 byte)	**Option** = 50_H (1 byte)	

Mode Data Description:

CS Page: Selection of the CS Page to be checked (refer to **Figure 6-1**).

Not used: This Byte is not used and will be ignored.

Exp. CHKS High, Low: Expected checksum High/Low Byte.

Option: set to 50_H to enable configuration sector page checksum check.

Checksum (16 Bits inverted XOR) on the selected configuration sector page is calculated and then compared with the expected values (provided as well as an input). The response frame will then give back a pass or fail indication plus the calculated checksum. In case the provided CS address is not valid, the microcontroller will return a Block Type Error (FF_H) followed by no further Bytes.

For mode A, the header block is the only transfer block to be sent by the host followed by a Slave Response Header. In case of valid header block, the microcontroller will send a response block (Acknowledgement code, 55_H) followed by the 4 Bytes data. The response for mode A is described in **Section 4.5.3**.

4.5 Phase III: Response protocol to the host

The microcontroller status is sent to the host only when a Slave Response Header frame is received. The microcontroller status is always sent in a transfer block of 9 Bytes.

A typical transfer block consists of four parts:

NAD (1 byte)	**Response** (1 byte)	**Response Data** (6 bytes)	**Checksum** (1 byte)

- **NAD**: Node Address for Diagnostic, specifies the address of the active slave node.
- **Response**: Response code indicating Acknowledge or Error status. See **Table 4-7**.
- **Response Data**: These 6 Bytes are generally not used and set to 00_H. An exception is mode A response which is described in detail in **Section 4.5.3**.
- **Checksum**: The checksum is calculated based on NAD, Response and Response Data Bytes. All responses sent by microcontroller will adopt classic checksum. See **Section 4.4.3.1**.

4.5.1 Acknowledgement response

The Acknowledge response code (55_H) is sent by microcontroller to host to indicate that a block has been successfully received.

4.5.2 Error response

There are 3 error responses indicated by microcontroller.

4.5.2.1 Block Type Error (FF_H)

This error can occur in the following conditions.

1. A Block Type other than the implemented ones was received. See **Table 4-2**.
2. An incorrect sequence of transfer blocks was received. For example, in mode 0 operation upon receiving a header block, a slave response request is expected. However, if another header block is received, this will result in a Block Type Error.

4.5.2.2 Checksum Error (FE_H)

This error occurs when the checksum comparison fails. Microcontroller will reject the transfer block by sending back a Checksum Error code (FE_H) to the host.

4.5.2.3 Protection Error (FD_H)

This error occurs when selected NVM sectors, for programming or erasing, are protected. As the selected NVM sectors are protected, no programming or erasing is allowed. In this special error case, the LIN routine will abort current command and wait for the next header block from the host again.

4.5.2.4 Response overview

Table 4-5 shows a tabulated summary of the possible responses the device may transmit following the reception of a header, data or EOT block.

Table 4-5 Possible responses for various block types

Mode	Header block	Data block	EOT block
0, 2, 8	Acknowledge, Block Type Error, Checksum Error, Protection Error	Acknowledge, Block Type Error, Checksum Error	Acknowledge, Block Type Error, Checksum Error
1, 3, 9	Acknowledge, Block Type Error, Checksum Error		
4, 6	Acknowledge, Block Type Error, Checksum Error, Protection Error		
A	Acknowledge, Block Type Error, Checksum Error		

The responses are defined in **Table 4-6**, 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 4-6 Definitions of responses

Response	Value	Description			
		Block Type	**BSL Mode**	**Reasons / Implications**	**Corrective Action**
Acknow-ledge	55_H	Header	1, 3, 9	The requested operation will be performed once the response is sent.	
			A	The requested operation has been performed and is successful. 4 Byte data transmission follows.	
			6	The requested operation has been performed and is successful.	
		EOT	0, 2, 4, 8		
		All other combinations		Reception of the Block is successful. Ready to receive the next block.	
Block Type Error	FF_H	Header	2, 4	NVM start address out of range.	Retransmit a valid header block.
		All other combinations		Either the block Type is undefined or the flow is invalid (see **Figure 4-2**).	Retransmit a valid block
Checksum Error	FE_H	All combinations		There is a mismatch between the calculated and the received Checksum (see **Section 4.4.3**).	Retransmit the block
Protection Error	FD_H	Header	0, 2, 4, 6, 8	Protection against external access enabled, i.e. NVM_PASSWORD is valid.	Disable protection

Table 4-7 gives a summary of the response code to be sent back to the host by the microcontroller.

Table 4-7 Type of Response Code

Communication status	Response code to the host
Acknowledge (Success)	55_H
Block Type Error	FF_H
Checksum Error	FE_H
Protection Error	FD_H

4.5.3 Mode A response

This response frame is only applicable for mode A. The response frame depends on the option Byte value used.

Option Byte = 00_H:

NAD (1 byte)	**ACK Response** 55_H	**ID** (1 bytes)	**CHIP_ID_2** (1 bytes)	**CHIP_ID_1** (1 bytes)	**CHIP_ID_0** (1 bytes)	**Not Used** (2 bytes)	**Checksum** (1 byte)

Option Byte = 10_H, 18_H, 50_H:

NAD (1 byte)	**ACK Response** 55_H	**Error indicator** (1 bytes)	**Calculated CHKS High** (1 bytes)	**Calculated CHKS Low** (1 bytes)	**Not Used** (3 bytes)	**Checksum** (1 byte)

Error indicator:

- 00_H: the calculated checksum and the expected one (provided as an input in the header frame) are equal.
- 80_H: the calculated checksum and the expected one (provided as an input in the header frame) differ.

4.6 Fast LIN BSL

Fast LIN BSL is an enhanced feature in TLE983x device, supporting higher baud rate up to 57.6 kBaud or 115.2 kBaud. This is higher than Standard LIN, which supports only a baud rate of up to 20 kBaud. This mode is especially useful during back-end programming, where faster programming time is desirable.

4.6.1 Entering Fast LIN BSL

User can enter Fast LIN BSL by sending TLE983x device a command frame with Fast_Prog set to 1. See **Section 4.4.4.2**. (The Fast_Prog option Byte is supported in LIN BSL modes 0, 2 and 8). In case Fast_Prog option is set, all other information sent with the frame will be ignored. The baud rate, used for the Fast LIN BSL, is calculated based on the LIN frame received. The LIN transceiver slope mode is set to Flash mode, a specific setting of the internal slew rate control that allows transmission up to 115.2 kBaud, and it is set to normal mode before it exits to user code through mode 1 and mode 3. Once entered to Fast LIN BSL, the protocol used will be the same as UART BSL (refer to **Chapter 5** for transfer protocol). The Fast LIN BSL entry is shown in **Figure 4-5**.

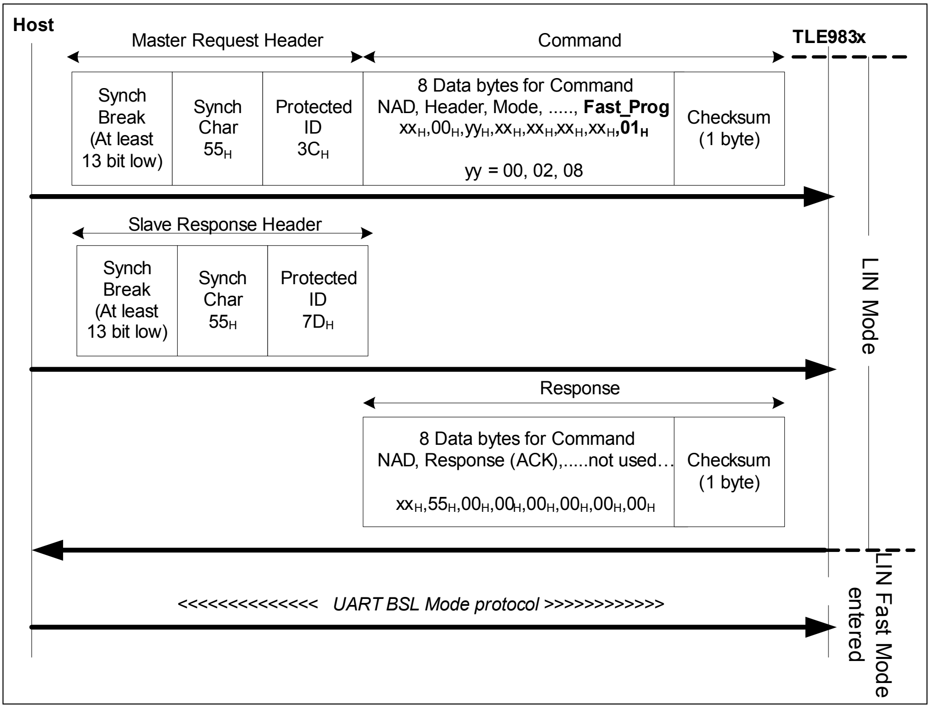

Host
Master Request Header
Command
TLE983x
Synch Break (At least 13 bit low)
Synch Char 55H
Protected ID 3CH
8 Data bytes for Command NAD, Header, Mode, ..., Fast_Prog xxH,00H,yyH,xxH,xxH,xxH,xxH,01H
yy = 00, 02, 08
Checksum (1 byte)
Slave Response Header
Synch Break (At least 13 bit low)
Synch Char 55H
Protected ID 7DH
Response
8 Data bytes for Command NAD, Response (ACK),.....not used...
xxH,55H,00H,00H,00H,00H,00H,00H
Checksum (1 byte)
LIN Mode
<<<<<<<<<<<<<<< UART BSL Mode protocol >>>>>>>>>>>>
LIN Fast Mode entered

Figure 4-5 Fast LIN BSL mode entry

4.7 After-Reset conditions

When more than one parameter in the transfer block is invalid, different actions are performed. The different scenarios, and its consequent actions, are listed in **Table 4-8**.

Table 4-8 LIN BSL After-Reset conditions

First Frame	ID	Check sum	NAD	Block Type (Header only)	Mode	Action
Yes	Invalid	Don't care	Don't care	Don't care	Don't care	Save LIN Message to XRAM and jump to NVM 0000_H[1][2].
No	Invalid	Don't care	Don't care	Don't care	Don't care	Message is ignored. Wait for next frame.
Yes	$7D_H$	N.A.	N.A.	N.A.	N.A.	Save LIN ID to XRAM and jump to NVM 0000_H[1][2]
No	$7D_H$	N.A.	N.A.	N.A.	N.A.	Reply if there is a previous valid Master Request (Command Frame) else wait for next frame
Yes	$3C_H$	LIN	Don't care	Invalid	Valid[3]	Error flag is triggered. Wait for Response frame to reflect error
Yes	$3C_H$	LIN	Don't care	Don't care	Invalid [3]	Save LIN message to XRAM and jump to NVM 0000_H[1][2]
Yes	$3C_H$	LIN	Valid	Valid	Valid[3]	Execute command
Yes	$3C_H$	LIN	Invalid	Valid	Valid[3]	Message is ignored. Wait for next frame.
Yes	$3C_H$	Prog	Invalid	Don't care	Don't care	Message is ignored. Wait for next frame.
Yes	$3C_H$	Prog	Valid	Invalid	Valid[4]	Error flag is triggered. Wait for Response frame to reflect error
Yes	$3C_H$	Prog	Valid	Valid	Invalid [4]	Error flag is triggered. Wait for Response frame to reflect error
Yes	$3C_H$	Prog	Valid	Valid	Valid[4]	Execute command
Yes	$3C_H$	Invalid	Don't care	Don't care	Don't care	Save LIN message to XRAM and jump to NVM 0000_H[1][2]

[1] Jump to user mode will only occur either (1) when NVM is not protected and NVM content at 0000_H is not FF_H; or (2) when NVM is protected.

[2] Up to max 10 Bytes are saved into the XRAM. In case less than 10 Bytes are received, firmware proceeds to user code after a time out of 35 ms.

[3] Valid modes for LIN checksum are mode 8 and mode 9. Other modes are considered invalid.

[4] Valid modes for programming checksum are mode 0-6 and A. Other modes are considered invalid.

4.8 User defined parameters for LIN BSL

There are 2 programmable parameters in the uppermost linearly mapped NVM Bank that are used in LIN BSL. The parameter values are specified by the user:

1. No Activity Count (NAC): Number of delay (in multiple of 5 ms) before jumping to user mode, measured from reset release.
2. Node Address for Diagnostic (NAD): Contains the address of the active slave node.

Note: Timer 0 is initialized to have 5 ms overflow and is used to create the delay.

The program will detect any activities on the LIN bus for a period of time, determined by $(((NAC \& 7F_H) - 01_H) * 5)$ ms. When nothing is detected on the LIN bus during this time, it will jump to user mode.

NAC value is restricted to $0C_H$ as the first open WDT1 window is worst case 65 ms. The firmware has to either refresh the WDT within the 65 ms or jump to user mode. If NAC value is bigger than $0C_H$, BootROM code will refresh the WDT and wait for a LIN frame indefinitely.

Table 4-9 gives an overview of the action of the microcontroller with respect to No Activity Count (NAC) values:

Table 4-9 Type of action w.r.t. No Activity Count values

NAC Value	Action
01_H	0 ms delay. Jump to user mode immediately[1]
02_H	5 ms delay before jumping to user mode[1] [2]
03_H	10 ms delay before jumping to user mode[1] [2]
04_H	15 ms delay before jumping to user mode[1] [2]
05_H	20 ms delay before jumping to user mode[1] [2]
06_H	25 ms delay before jumping to user mode[1] [2]
07_H	30 ms delay before jumping to user mode[1] [2]
08_H	35 ms delay before jumping to user mode[1] [2]
09_H	40 ms delay before jumping to user mode[1] [2]
$0A_H$	45 ms delay before jumping to user mode[1] [2]
$0B_H$	50 ms delay before jumping to user mode[1] [2]
$0C_H$	55 ms delay before jumping to user mode[1] [2]
$0D_H$ - $7F_H$, 00_H	Wait forever for the first LIN frame

[1] Jump to user mode will only occur either (1) when NVM is not protected and NVM content at 0000_H is not FF_H; or (2) when NVM is protected

[2] If a valid LIN 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 LIN frame accordingly

4.8.1 Programming NAC and NAD

User needs to program the NAC and NAD in the format listed in **Table 3-2**. To ensure the parameter validity, the 2 parameters actual values and their inverted values are checked.

If the NAD parameter is not valid nor within the range, the default value is used in the LIN BSL.

4.9 WDT1 refreshing

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

Subsequently, before exiting to XRAM or NVM in LIN BSL modes 1, 3 and 9 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 LIN communication. User program needs to trigger the WDT1 refresh accordingly.

5 UART BSL mode

UART BSL mode consists of two functional parts that present two phases as described below:

- **Phase I**: Establish a serial connection and automatically synchronize to the transfer speed (baud rate) of the serial communication partner (host).
- **Phase II**: Perform the serial communication with the host. The host controls the communication by sending special header information which selects one of the working modes. These modes are:
  - **Mode 0 (00_H)**: Transfer a user program from the host to XRAM or write 100TP and OTP pages[1)]
  - **Mode 1 (01_H)**: Execute a user program in the XRAM[2)]
  - **Mode 2 (02_H)**: Transfer a user program from the host to NVM[1)]
  - **Mode 3 (03_H)**: Execute a user program in the NVM[2)]
  - **Mode 4 (04_H)**: Erase NVM[1)]
  - **Mode 6 (06_H)**: NVM protection mode enabling/disabling Scheme[2)]
  - **Mode A ($0A_H$)**: Get Info (based on Option Byte)[1)]

Except mode 1, mode 3 and mode 6, the microcontroller would return to the beginning of Phase II and wait for the next command from the host after executing all other modes.

The serial communication, which is activated in Phase II, is performed with the full-duplex serial interface (UART) of the TLE983x. The microcontroller is connected to the serial port of the host via a serial cable (RS232).

The serial transfer is working in asynchronous mode with the serial parameters 8N1 (eight data Bits, no parity and one stop Bit). The host can vary the baud rate in a wide range because the microcontroller does an automatic synchronization with the host in Phase I.

The following section provides detailed information on these two UART BSL phases.

5.1 Phase I: 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:

- 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

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 exited.

5.1.1 General description

The microcontroller will set the serial port 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, to restart it for a new synchronization attempt.

5.1.2 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 (P0_DATA.1/RxD Pin), the Timer 2 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) 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})

$$\text{baud} = \frac{f_{PCLK}}{16 \times PRE \times BR\_VALUE} \qquad [5.1]$$

Second, the relation between the baud rate (baud) and the recording value of Timer 2 (T2) depends on the T2 peripheral frequency (f_{T2}) and the number of received Bits ($f_{T2}N_b$)

$$\text{baud} = \frac{f_{T2} \times N_b}{T2} \qquad [5.2]$$

Combining **Equation [5.1]** and **Equation [5.2]** with N_b=8, f_{T2}=f_{PCLK}/ 8 (T2PRE=011),

$$\frac{f_{PCLK}}{16 \times PRE \times BR\_VALUE} = \frac{\frac{f_{PCLK}}{8} \times 8}{T2} \qquad [5.3]$$

Simplifying **Equation [5.3]**, we get

$$PRE \times BR\_VALUE = \frac{T2}{16} \qquad [5.4]$$

After setting BR_VALUE 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.

5.2 Phase II: Serial communication protocol and the working modes

After the successful synchronization to the host, the UART BSL routine enters Phase II, during which it communicates with the host to select the desired working modes. The detailed communication protocol is explained as follows:

5.2.1 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 communication is nearly unidirectional: 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 regulation 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).

5.2.1.1 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.

5.2.1.2 Transfer block type

There are three types of transfer blocks depending on the value of the Block Type. **Table 5-1** provides the general information on these block types. More details will be described in the corresponding sections later.

Table 5-1 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.

5.2.1.3 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 modes 1, 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 5-2 gives a summary of the response codes to be sent back to the host by the microcontroller after it receives a transfer block.

Table 5-2 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

Table 5-3 shows a tabulated summary of the possible responses the device may transmit following the reception of a header, data or EOT block.

Table 5-3 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
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		
6	Acknowledge, Block Type Error, Checksum Error, Protection Error		
A	Acknowledge, Block Type Error, Checksum Error, Protection Error		

The responses are defined in Table 5-4, 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 5-4 Definitions of responses

Response	Value	Description			
		Block Type	**BSL Mode**	**Reasons / Implications**	**Corrective Action**
Acknowledge	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 CS 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
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

5.2.2 The selection of working 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.

5.2.2.1 Receiving the 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 00_H (Header Block)	Data Area		Checksum (1 byte)
	Mode (1 byte)	**Mode Data** (5 bytes)	

Description:

- **Block Type 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 **Section 5**
- **Mode Data**: Five Bytes of special information, which are necessary to activate corresponding working mode.
- **Checksum**: The checksum of the header block.

5.2.2.2 The activation of working mode 0

Mode 0 is used to transfer a user program or data from the host to the XRAM 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 XRAM in accordance with the format reported in the **Table 6-6** and after EOT block has been received, data is automatically copied with proper offset in the target page. If NVM protection is installed, programming to XRAM is not allowed.

Different option supported are:

- Option 00_H: XRAM download
- Option $F0_H$: XRAM download and Configuration sector page programming

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

The header block for XRAM download (Option = 00$_H$)

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

Mode Data Description:

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

Block Length: The length 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 XRAM download.

In option 00$_H$ start address can be each valid XRAM address. Data sent in the following data/ EOT blocks will be copied into the XRAM at the specified address.

The header block for XRAM download and CS page programming (Option = F0$_H$)

		Mode Data (5 bytes)					
00$_H$ (Header Block)	**00$_H$** (Mode 0)	**StartAddr High** (1 byte)	**StartAddr Low** (1 byte)	**Block Length** (1 byte)	**CS Page** (1 byte)	**Option = F0$_H$** (1 byte)	**Checksum** (1 byte)

Mode Data Description:

Start Addr High, Low: 16-bit Start Address, which determines where to copy the received program codes in the XRAM.

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

CS Page: This Byte is used to select the desired user configuration sector page to be programmed. This Byte is relevant only in case option F0$_H$ is used. CS page is selected according to the addressing scheme reported in **Figure 6-1**.

Option: Set to F0$_H$ for XRAM download and CS page programming

Using this option, user can write data into the user CS pages (OTP and 100TP pages). In this case, data has to be sent to the XRAM according to the **Table 6-6** and therefore start address has to be equal to F000$_H$. 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 CS page has to be provided.

Note: 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 and should be filled with zeros.

5.2.2.3 The activation of working mode 1

Mode 1 is used to execute a user program in the XRAM of the microcontroller at $F000_H$. 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 and jump to the XRAM address at $F000_H$.

5.2.2.4 The activation of working mode 2

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 High (1 byte)	StartAddr Low (1 byte)	Block Length (1 byte)	Not Used (2 bytes)	

Mode Data Description:

Start Addr High, Low: 16-bit Start Address, which determines where to copy the received program codes in the NVM. This address must be aligned to the page address.

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.

Not used: 2 Bytes, these Bytes are not used and will be ignored in mode 2.

*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 $0F82_H$, the PC host will fill up the addresses $0F80_H$ and $0F81_H$ with 00_H and provide the **Start Address** $0F80_H$ 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).*

Since the data area is 128 Bytes, the ***Block Length*** *is 130 Bytes for data blocks and 131 Bytes for EOT blocks.*

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) 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 codes in this EOT block.

Note: If data blocks are sent, this Byte should be zero. If this Byte is not zero, additional undesired Bytes will be programmed and this will affect the verification.

Program Codes: The last program codes to be sent to the microcontroller

Not used: The length is ((**Block Length**) - 3 - (**Last Codelength**)) bytes and should be filled with zeros.

5.2.2.5 The activation of working mode 3

Mode 3 is used to execute a user program in the NVM of the microcontroller at 0000_H. The header block for this working mode has the following structure:

The header block

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

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 and jump to the NVM address at 0000_H.

Note: Jump to NVM will only occur either (1) when NVM is not protected and NVM content at 0000_H is not FF_H; or (2) when NVM is protected. In all other cases, firmware will put the device in sleep mode.

5.2.2.6 The activation of working mode 4

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)

		Mode Data (5 bytes)				
00_H (Header Block)	**04_H** (Mode 4)	**StartAddr High** (1 byte)	**StartAddr Low** (1 byte)	**Not Used** (2 bytes)	**Option =00_H** (1 byte)	**Checksum** (1 byte)

Mode Data Description:

Start Addr High, Low: 16-bit Start Address, which determines which NVM page to be erased. Address should be page aligned.

Not used: The two Bytes are not used and will be ignored in option 00_H.

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)

		Mode Data (5 bytes)				
00_H (Header Block)	**04_H** (Mode 4)	**StartAddr High** (1 byte)	**StartAddr Low** (1 byte)	**Not Used** (2 bytes)	**Option = 40_H** (1 byte)	**Checksum** (1 byte)

Mode Data Description:

Start Addr High, Low: 16-bit Start Address, which determines which NVM sector to be erased. Address should be sector aligned.

Not used: The two Bytes are not used and will be ignored in option 40_H.

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 4 ms.

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

		Mode Data (5 bytes)		
00_H (Header Block)	**04_H** (Mode 4)	**Not Used** (4 bytes)	**Option =$C0_H$** (1 byte)	**Checksum** (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, 4ms * number of sectors, as the erase operation is done sequentially.

Note:

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.*

5.2.2.7 The activation of working mode 6

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 device is unprotected, the provided user-password will be set as NVM_PASSWORD and internally stored. 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. If MSB of the NVM_PASSWORD is 0, only NVM Linearly mapped sectors are erased. If the Bit is 1, both NVM Linearly and Non-linearly mapped regions 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.

Note:

1. *Password value has to be different from 00_H and FF_H.*
2. *When disabling NVM protection, together with NVM, the NAC and NAD values are erased too. As a result, after next reset, default NAD will be used and chip waits for ever for the first BSL LIN frame.*

Table 5-5 Erase NVM during unprotection

NVM_PASSWORD Bit7	Description
0	Only linearly mapped NVM is erased.
1	Both linearly and non-linearly mapped NVM are erased.

5.2.2.8 The activation of working mode A

Mode A is used to get 4 Bytes Chip ID data, NVM or CS page read out, NVM or CS page or NVM mass checksum check depending on the Option Byte value in the header block.

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: Configuration sector page checksum check
- Option $C0_H$: NVM page read
- Option $F0_H$: Configuration sector 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. The order of the 4 Bytes of data are SFR ID, CHIP_ID2, CHIP_ID1 and CHIP_ID0.

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

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

Mode Data Description:

Start Addr High, Low: Address of the NVM page for checksum check. (Address should be page aligned).

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

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

This option will trigger a checksum calculation (16 bits inverted XOR) 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. 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.

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.

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) over the whole NVM (both linearly and not linearly mapped regions not including CS pages) and 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. 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.

The header block for CS page checksum check (Option = 50_H)

00_H (Header Block)	**$0A_H$** (Mode A)	**Mode Data (5 bytes)**					**Checksum** (1 byte)
		CS 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:

CS Page: Selection of the CS Page to be checked (refer to **Figure 6-1**).

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 CS page checksum check.

This option will trigger a checksum calculation (16 bits inverted XOR) over the whole CS 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). CS page address has to be in accordance with the configuration sector address scheme described in the **Figure 6-1**. If the given address is valid, the microcontroller will return an Acknowledge followed by four data Bytes. 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.

The header block for NVM page read out (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 out (Address should be page aligned).

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.

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 first Byte in 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 Byte are always returned ordered from the first to the last 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 out, this option is disabled if NVM is protected and only a Protection Error Byte (FD_H) will be returned.

The header block for user configuration sector page read out (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)	**CS Page** (1 byte)	**Option =$F0_H$** (1 byte)	

Mode Data Description:

Not Used: These Bytes are not used and will be ignored for Option $F0_H$.

CS Page: Selection of the CS Page to be checked (refer to **Figure 6-1**).

Option: set to $F0_H$ to enable configuration sector page read.

This option will trigger a read of the addressed configuration sector page. Microcontroller will return an Acknowledge (55_H) followed by the 128 CS page data Bytes (starting from first Byte in the page).

Configuration Sector page to be read out is selected by the CS Page Byte according to the scheme shown in **Figure 6-1**.

In case an invalid CS 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 out, 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.

5.3 User defined parameters for UART mode

There is 1 programmable parameter in the uppermost linearly mapped NVM Bank that is used in UART mode. The parameter value is specified by the user:

- No Activity Count (NAC): Number of delay (in multiple of 5 ms[1]) before jumping to user mode measured from reset release.

1) Delay ranges from 0 ms to 55 ms, derived from equation ((($NAC \& 7F_H$) -01_H) * 5) ms. In case of invalid or bigger values the device waits forever for the first UART frame.

Note: Timer 0 is initialized to have 5 ms overflow and is used to create the delay.

The program will detect any activities on the UART bus for a period of time, determined by (((NAC & $7F_H$) - 01_H) * 5) ms. When nothing is detected on the UART bus during this time, it will jump to user mode.

For UART BSL NAC Bit 7 has to be set to 1 and (NAC & $7F_H$) value is restricted to $0C_H$ as the first open WDT1 window is worst case 65 ms. The firmware has to either refresh the WDT within the 65 ms or jump to user mode. If (NAC & $7F_H$) value is bigger than $0C_H$, BootROM code will refresh the WDT and wait for a UART frame indefinitely.

Table 5-6 gives an overview of the action of the microcontroller with respect to No Activity Count (NAC) values:

Table 5-6 Type of action w.r.t. No Activity Count values

NAC Value	Action
81_H	0 ms delay. Jump to user mode immediately[1)]
82_H	5 ms delay before jumping to user mode[1)] [2)]
83_H	10 ms delay before jumping to user mode[1)] [2)]
84_H	15 ms delay before jumping to user mode[1)] [2)]
85_H	20 ms delay before jumping to user mode[1)] [2)]
86_H	25 ms delay before jumping to user mode[1)] [2)]
87_H	30 ms delay before jumping to user mode[1)] [2)]
88_H	35 ms delay before jumping to user mode[1)] [2)]
89_H	40 ms delay before jumping to user mode[1)] [2)]
$8A_H$	45 ms delay before jumping to user mode[1)] [2)]
$8B_H$	50 ms delay before jumping to user mode[1)] [2)]
$8C_H$	55 ms delay before jumping to user mode[1)] [2)]
$8D_H$ - FF_H, 80_H	Wait forever for the first UART frame

1) Jump to user mode will only occur either (1) when NVM is not protected and NVM content at 0000_H is not FF_H; or (2) when NVM is protected

2) If a valid UART synchronization 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 wait for next UART frame.

5.3.1 Programming NAC

User needs to program the NAC in the format listed in **Table 3-2**. To ensure the parameter validity, the actual value and its inverted value are checked.

5.4 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 XRAM 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.

6 NVM

Non Volatile Memory (NVM) is the flash module of the TLE983x which partly supports EEPROM emulation.

6.1 NVM overview

The NVM is a single block of NVM memory of up to 64 kBytes separated into Code and Data space. The TLE983x device family provides products with different NVM size (24, 36, 48 and 64 kBytes) all sharing the same architecture and features. The following table shows the NVM address range for all the supported sizes.

Table 6-1 NVM address range for different NVM module sizes

Address	Address Range	Size (kBytes)
0’0000$_H$ to 0’5FFF$_H$	NVM memory	24
0’0000$_H$ to 0’8FFF$_H$	NVM memory	36
0’0000$_H$ to 0’BFFF$_H$	NVM memory	48
0’0000$_H$ to 0’FFFF$_H$	NVM memory	64

6.1.1 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. Each update to the NVM, even when targeting 1 Byte only, accesses 1 page (128 Bytes). With this NVM structure, writing to the NVM can be any data size, up to 128 Bytes. **Table 6-2** shows the sector address organisation of 36 kBytes NVM. Sector organization for other NVM sizes can be simply derived per extension of the reported scheme. **Table 6-3** shows the page address organisation of NVM Sector 1 and is can be used as e reference for page organization of any NVM Sector.

Table 6-2 NVM memory sector organisation (example for a 36 kByte module)

Address	NVM sector number
0’0000$_H$ to 0’0FFF$_H$	1
0’1000$_H$ to 0’1FFF$_H$	2
0’2000$_H$ to 0’2FFF$_H$	3
0’3000$_H$ to 0’3FFF$_H$	4
0’4000$_H$ to 0’4FFF$_H$	5
0’5000$_H$ to 0’5FFF$_H$	6
0’6000$_H$ to 0’6FFF$_H$	7
0’7000$_H$ to 0’7FFF$_H$	8
0’8000$_H$ to 0’8FFF$_H$	9

Note: The lower 32 kBytes (Sector 1 to 8) are always located and can be accessed in the lower half (0000$_H$ to 7FFF$_H$) of each bank in the code area (except for Bank A).

Table 6-3 NVM memory sector 1 page organisation

Address	Page number of NVM sector
0’0000$_H$ to 0’007F$_H$	0
0’0080$_H$ to 0’00FF$_H$	1
0’0100$_H$ to 0’017F$_H$	2
0’0180$_H$ to 0’01FF$_H$	3
0’0200$_H$ to 0’027F$_H$	4
0’0280$_H$ to 0’02FF$_H$	5
0’0300$_H$ to 0’037F$_H$	6
0’0380$_H$ to 0’03FF$_H$	7
0’0400$_H$ to 0’047F$_H$	8
0’0480$_H$ to 0’04FF$_H$	9
0’0500$_H$ to 0’057F$_H$	10
0’0580$_H$ to 0’05FF$_H$	11
0’0600$_H$ to 0’067F$_H$	12
0’0680$_H$ to 0’06FF$_H$	13
0’0700$_H$ to 0’077F$_H$	14
0’0780$_H$ to 0’07FF$_H$	15

Table 6-3 NVM memory sector 1 page organisation

0’0800$_H$ to 0’087F$_H$	16
0’0880$_H$ to 0’08FF$_H$	17
0’0900$_H$ to 0’097F$_H$	18
0’0980$_H$ to 0’09FF$_H$	19
0’0A00$_H$ to 0’0A7F$_H$	20
0’0A80$_H$ to 0’0AFF$_H$	21
0’0B00$_H$ to 0’0B7F$_H$	22
0’0B80$_H$ to 0’0BFF$_H$	23
0’0C00$_H$ to 0’0C7F$_H$	24
0’0C80$_H$ to 0’0CFF$_H$	25
0’0D00$_H$ to 0’0D7F$_H$	26
0’0D80$_H$ to 0’0DFF$_H$	27
0’0E00$_H$ to 0’0E7F$_H$	28
0’0E80$_H$ to 0’0EFF$_H$	29
0’0F00$_H$ to 0’0F7F$_H$	30
0’0F80$_H$ to 0’0FFF$_H$	31

6.2 NVM configuration sectors organisation

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

6.2.1 100 Time Programmable data

User has 4 100 time programmable pages. 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 0F$_H$ to 3E$_H$, are automatically copied into the dedicated XSFR registers of e.g. the Measurement Interface after every power on reset, brown out reset or wake-up reset from sleep mode thus replacing the registers default reset values. If the user wants to check them, he could do it by reading the dedicated XSFRs. The data stored in this first 100 time programmable page can be found in **Table 6-4**.

To read data stored in the 100TP pages, refer to **Section 6.3.23**.

To perform the programming of these pages, the user will need to preload the contents to be programmed into the XRAM as listed in **Table 6-5**. 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).

For further information regarding programming data into the 100TP pages, refer to **Section 6.3.24**.

Table 6-4 100 Time Programmable page 1

Data Offset	XSFR/ Variable Name	Description	Default Value
00_H	GAIN_VS_10B	Calibration gain for 10-bit ADC Vs measurement	Chip Individual
01_H	OFFSET_VS_10B	Calibration offset for 10-bit ADC Vs measurement	Chip Individual
02_H	GAIN_VBAT_SENSE_10B	Calibration gain for 10-bit ADC VBAT_SENSE measurement	Chip Individual
03_H	OFFSET_VBAT_SENSE_10B	Calibration offset for 10-bit ADC VBAT_SENSE measurement	Chip Individual
04_H	GAIN_VMON_ATT_1_5	Calibration gain for 10-bit ADC VMON_ATT 1 to 5 measurement	Chip Individual
05_H	OFFSET_VMON_ATT_1_5	Calibration offset for 10-bit ADC VMON_ATT 1 to 5 measurement	Chip Individual
06_H to $0C_H$	Reserved	Reserved	00_H
$0D_H$	ADC2_CNT0_LOWER	Refer to user manual	12_H
$0E_H$	ADC2_CNT4_LOWER	Refer to user manual	$0A_H$
$0F_H$	MEAS_ADC2_SQ1	Refer to TLE983x User´s Manual	37_H
10_H	MEAS_ADC2_SQ2	Refer to TLE983x User´s Manual	28_H
11_H	MEAS_ADC2_SQ3	Refer to TLE983x User´s Manual	36_H
12_H	MEAS_ADC2_SQ4	Refer to TLE983x User´s Manual	29_H

Table 6-4 100 Time Programmable page 1

13_H	MEAS_ADC2_SQ5	Refer to TLE983x User´s Manual	36_H
14_H	MEAS_ADC2_SQ6	Refer to TLE983x User´s Manual	28_H
15_H	MEAS_ADC2_SQ7	Refer to TLE983x User´s Manual	37_H
16_H	MEAS_ADC2_SQ8	Refer to TLE983x User´s Manual	28_H
17_H	MEAS_ADC2_SQ9	Refer to TLE983x User´s Manual	36_H
18_H	MEAS_ADC2_SQ10	Refer to TLE983x User´s Manual	29_H
19_H	ADC2_FILTCOEFF0_3	Refer to TLE983x User´s Manual	AA_H
$1A_H$	ADC2_FILTCOEFF4_5	Refer to TLE983x User´s Manual	AA_H
$1B_H$	ADC2_TH0_UPPER	Refer to TLE983x User´s Manual	EA_H
$1C_H$	ADC2_CNT0_UPPER	Refer to TLE983x User´s Manual	$1A_H$
$1D_H$	ADC2_CNT1_UPPER	Refer to TLE983x User´s Manual	$1B_H$
$1E_H$	ADC2_CNT2_UPPER	Refer to TLE983x User´s Manual	13_H
$1F_H$	ADC2_CNT3_UPPER	Refer to TLE983x User´s Manual	12_H
20_H	ADC2_CNT4_UPPER	Refer to TLE983x User´s Manual	12_H
21_H	ADC2_TH0_LOWER	Refer to TLE983x User´s Manual	$4E_H$
22_H	ADC2_CNT5_UPPER	Refer to TLE983x User´s Manual	12_H
23_H	ADC2_CNT8_UPPER	Refer to TLE983x User´s Manual	$1A_H$
24_H	ADC2_CNT9_UPPER	Refer to TLE983x User´s Manual	$1A_H$

Table 6-4 100 Time Programmable page 1

25_H	ADC2_TH5_LOWER	Refer to TLE983x User´s Manual	32_H
26_H	ADC2_TH6_LOWER	Refer to TLE983x User´s Manual	39_H
27_H	ADC2_TH7_LOWER	Refer to TLE983x User´s Manual	39_H
28_H	ADC2_TH8_LOWER	Refer to TLE983x User´s Manual	BA_H
29_H	ADC2_TH9_LOWER	Refer to TLE983x User´s Manual	$C6_H$
$2A_H$	ADC2_CNT5_LOWER	Refer to TLE983x User´s Manual	$0A_H$
$2B_H$	ADC2_CNT6_LOWER	Refer to TLE983x User´s Manual	$0A_H$
$2C_H$	ADC2_CNT7_LOWER	Refer to TLE983x User´s Manual	$0A_H$
$2D_H$	ADC2_CNT8_LOWER	Refer to TLE983x User´s Manual	$0A_H$
$2E_H$	ADC2_CNT9_LOWER	Refer to TLE983x User´s Manual	$0A_H$
$2F_H$	ADC2_CALOFFS_CH0	Refer to TLE983x User´s Manual	00_H
30_H	ADC2_CALGAIN_CH0	Refer to TLE983x User´s Manual	00_H
31_H	ADC2_CALOFFS_CH1	Refer to TLE983x User´s Manual	00_H
32_H	ADC2_CALGAIN_CH1	Refer to TLE983x User´s Manual	00_H
33_H	ADC2_CALOFFS_CH2	Refer to TLE983x User´s Manual	00_H
34_H	ADC2_CALGAIN_CH2	Refer to TLE983x User´s Manual	00_H
35_H	ADC2_CALOFFS_CH3	Refer to TLE983x User´s Manual	00_H
36_H	ADC2_CALGAIN_CH3	Refer to TLE983x User´s Manual	00_H

Table 6-4 100 Time Programmable page 1

37_H	ADC2_CALOFFS_CH4	Refer to TLE983x User´s Manual	00_H
38_H	ADC2_CALGAIN_CH4	Refer to TLE983x User´s Manual	00_H
39_H	ADC2_CALOFFS_CH5	Refer to TLE983x User´s Manual	00_H
$3A_H$	ADC2_CALGAIN_CH5	Refer to TLE983x User´s Manual	00_H
$3B_H$	ADC2_CALOFFS_CH8	Refer to TLE983x User´s Manual	00_H
$3C_H$	ADC2_CALGAIN_CH8	Refer to TLE983x User´s Manual	00_H
$3D_H$	ADC2_CALOFFS_CH9	Refer to TLE983x User´s Manual	00_H
$3E_H$	ADC2_CALGAIN_CH9	Refer to TLE983x User´s Manual	00_H
$3F_H$ to 73_H	Reserved	Reserved	00_H
74_H	CS_XRAM_MBIST_STARTUP_EN	Enable Byte for XRAM MBIST during startup flow. If Value = $C3_H$ then the MBIST is enabled	00_H
75_H	CS_XRAM_MBIST_LOW_BOUND_H	High Byte of the starting address of the XRAM range to be checked during XRAM MBIST at startup. ($F0_H$ for XRAM initial address)	00_H
76_H	CS_XRAM_MBIST_LOW_BOUND_L	Low Byte of the starting address of the XRAM range to be checked during XRAM MBIST at startup. (00_H for XRAM initial address)	00_H
77_H	CS_XRAM_MBIST_HIGH_BOUND_H	High Byte of the ending address of the XRAM range to be checked during XRAM MBIST at startup. (FB_H for XRAM initial address)	00_H

Table 6-4 100 Time Programmable page 1

78_H	CS_XRAM_MBIST_HIGH_BOUND_L	Low Byte of the ending address of the XRAM range to be checked during XRAM MBIST at startup. (FF_H for XRAM initial address)	00_H
79_H	CS_USER_CAL_STARTUP_EN	Enable Byte for user calibration data download during startup. If value=$C3_H$ then the download is enabled	00_H
$7A_H$	CS_USER_CAL_XADDH	High Byte of the XRAM starting address where downloaded data has to be stored($F0_H$ for XRAM initial address)	00_H
$7B_H$	CS_USER_CAL_XADDL	Low Byte of the XRAM starting address where downloaded data has to be stored(00_H for XRAM initial address)	00_H
$7C_H$	CS_USER_CAL_CS_PAGE	CS page where calibration data has to be downloaded from. By default 100TP page1 should be used (Value=11_H)	00_H
$7D_H$	CS_USER_CAL_NUM	Number of Bytes to be downloaded starting from the first Byte of the selected CS page.	00_H
$7E_H$	Checksum_100TP_P1	Checksum_100TP_P1, XOR first 126 Bytes of 100TP page 1	----
$7F_H$	PROG_TIMES	This reflects the number of times that this page has been programmed. (Up to a maximum of 100 times.)	00_H

Table 6-5 XRAM preloading for 100 Time Programmable page programming

XRAM Address	Function
$F000_H$	Number of Bytes to be programmed (i.e. **N**, up to a maximum of 127 Bytes)
$F001_H$	100TP offset 1

Table 6-5 XRAM preloading for 100 Time Programmable page programming

$F002_H$	100TP data 1 to be programmed
$F003_H$	100TP offset 2
$F004_H$	100TP data 2 to be programmed
.....	....
$F001_H$ + ((N-1) x 2)	100TP offset N
$F002_H$ + ((N-1) x 2)	100TP data N to be programmed

6.2.2 One-Time Programmable (OTP)

The TLE983x contains 4 one-time programmable (OTP) pages. The user can program up to 128 Bytes into each of these pages. However, the programming of these pages can only be done once. Thereafter, no further programming will be possible. To complete the programming, the user will need to preload the contents to be programmed into the XRAM as listed in **Table 6-6**. The offset entered for the programming does not need to be in sequential order.

For further information regarding OTP page program, refer to **Section 6.3.19**.

For further information regarding OTP page read, refer to **Section 6.3.18**.

OTP pages are not used during the startup flow and content definition is completely left to the user.

Table 6-6 XRAM preloading for OTP

XRAM Address	Function
$F000_H$	Number of Bytes to be programmed (i.e. **N**, up to a maximum of 128 Bytes)
$F001_H$	OTP offset 1
$F002_H$	OTP data 1 to be programmed
$F003_H$	OTP offset 2
$F004_H$	OTP data 2 to be programmed
.....	....
$F001_H$ + ((N-1) x 2)	OTP offset N
$F002_H$ + ((N-1) x 2)	OTP data N to be programmed

6.3 NVM user routines organisation

The NVM user routines are BootROM routines called by user and placed from $2'E830_H$ to $2'E88F_H$. The complete list of NVM user routines can be found in **Table 6-7**.

Table 6-7 NVM user routines list

S/N	Address	Routine	Description
1	2’E88D$_H$	USER_OPENAB	To open the assembly buffer for writing
2	2’E88A$_H$	USER_PROG	To program the NVM
3	2’E887$_H$	USER_ERASEPG	To erase an NVM page
4	2’E884$_H$	USER_ABORTPROG	To abort the NVM programming by closing the assembly buffer
5	2’E881$_H$	USER_NVMRDY	To access if the NVM is in ready to read status
6	2’E87E$_H$	USER_READ_CAL	To read the NVM calibration data.
7	2’E87B$_H$	USER_NVM_CONFIG	To read the NVM configuration status
8	2’E878$_H$	USER_NVM_ECC2ADDR	To read the NVM ECC2 address
9	2’E875$_H$	USER_NVMPROT_STATUS	To check for the NVM protection status
10	2’E872$_H$	USER_SET_PRGPROT_CODE	To set the protection against programming for NVM code area
11	2’E86F$_H$	USER_CLR_PRGPROT_CODE	To clear the protection against programming for NVM code area
12	2’E86C$_H$	USER_SET_RDPROT_CODE	To set the protection against reading for NVM code area
13	2’E869$_H$	USER_CLR_RDPROT_CODE	To clear the protection against reading for NVM code area
14	2’E866$_H$	USER_SET_PRGPROT_DATA	To set the protection against programming for NVM data area
15	2’E863$_H$	USER_CLR_PRGPROT_DATA	To clear the protection against programming for NVM data area

Table 6-7 **NVM user routines list**

16	2'E860$_H$	USER_SET_RDPROT_DATA	To set the protection against reading for NVM data area
17	2'E85D$_H$	USER_CLR_RDPROT_DATA	To clear the protection against reading for NVM data area
18	2'E85A$_H$	USER_READ_OTP	To read the NVM OTP data
19	2'E857$_H$	USER_OTP_PROG	To perform the OTP program. (This can be used only once per OTP page)
20	2'E854$_H$	USER_LIN_AUTOBAUD	To perform LIN autobaud
21	2'E851$_H$	USER_UART_AUTOBAUD	To perform UART autobaud
22	2'E84E$_H$	USER_XRAM_DOWNLOAD	To perform an XRAM download
23	2'E84B$_H$	USER_READ_100TP	To read the NVM 100TP parameter data
24	2'E848$_H$	USER_100TP_PROG	To perform the 100TP program. (This can be used used 100 times per 100TP page)
25	2'E845$_H$	USER_ERASE_SECTOR	To erase an NVM Sector
26	2'E83F$_H$	USER_SET_USER_CLK	To set system clock frequency
27	2'E83C$_H$	USER_NVMCLKFAC_SET	To set NVMCLKFAC Bit in SYSCON0
28	2'E833$_H$	USER_XRAM_MBIST_START	To start a sequential checkerboard and inverted checkerboard test on the XRAM
29	2'E830$_H$	USER_XRAM_MBIST_CHECK	To check the result of the XRAM MBIST test

6.3.1 Opening assembly buffer routine

The NVM programming routine consists of two parts: The assembly buffer opening routine, and the programming and verification routine. In between calling of the routines the user software can adapt the page data to be programmed to the memory field. This assembly buffer opening routine needs to be executed successfully before the NVM programming routine can be called. Once the assembly buffer is successfully opened, the user can load the user contents into the assembly buffer.

Table 6-8 Opening assembly buffer subroutine

Subroutine	2'E88D$_H$: USER_OPENAB
Input	**R6(High Byte), R7(Low Byte):** Address of NVM page.
Output	**PSW.CY** 0 = Assembly Buffer is successfully opened 1 = Assembly Buffer cannot be opened. Possible reasons of failure: - Assembly Buffer is already opened. - The range of the address is protected. - The range of the address is incorrect.
Stack size required	6
Resource used/ destroyed	R0,R1,R2,R3,R4

Note: Once assembly buffer is opened, user must either proceed with the standard program flow (refer to **Figure 6-4***) or close the assembly buffer using the dedicated abort programming user routine (refer to* **Chapter 6.3.4***). All other sequences are not allowed and might lead to loss of data.*

6.3.2 NVM programming routine

There are 2 types of programming available, Type 1 or Type 2 (Type 1 without or Type 2 with XRAM 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 jumping (via call) to an address provided by the user. In this scenario, the user software needs to reside in XRAM 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 (via return), which continues with polling the busy Bit.

A description of the BootROM programming routine is provided in the following **Table 6-9**. More information on the support for background activity during NVM operation can be found in **Section 6.4.2**.

Table 6-9 Programming subroutine

Subroutine	2'E88A$_H$: USER_PROG;
Input	**R7[0]**: To control XRAM routine branching. 0: No XRAM routine branching enabled. 1: XRAM routine branching enabled. **R7[1]:** To enable corrective activities (i.e. disturb handling and retries sequences) 0: No corrective activities enabled. 1: Corrective activities are enabled. **R7[2]:** To disable erasing of a failing page when addressing non-linearly mapped sector (refer to **Chapter 6.4.4.2** for more details) 0: Programmed data erased in case of fail. If page was already used, old data are kept. 1: Programmed data are not erased even in case of fail. If page was already used, old data are not kept and the new data are accessible by reading the page.
Output	**PSW.CY** 0 = Programming completed successfully. 1 = Programming failed. Possible reasons of failure: - This is the 2nd time this routine is called. - The range of the address is incorrect. - This is a protected range. - The internal verify, using special read margin, fails Note: No NVM prog or erase routine can be called until this NVM operation is completed.
Stack size required	16
Resource used/ destroyed	R0, R1, R2, R3, R4, R5, DPTR

6.3.3 NVM page erasing routine

Similarly, there are 2 types of erasing available, Type 1 or Type 2 (Type 1 without or Type 2 with XRAM background activity during NVM operation). Details in the following table.

Table 6-10 Page erasing subroutine

Subroutine	2'E887$_H$: USER_ERASEPG;
Input	**R6(High Byte), R7(Low Byte):** Address of NVM page **R5[0]:** Input to control XRAM routine branching. 0: No XRAM routine branching enabled. 1: XRAM routine branching enabled.
Output	**PSW.CY** 0 = Erasing completed successfully. 1 = Erasing failed. Possible reasons of failure: - This is the 2nd time this routine is called. - The range of the address is incorrect. - This is a protected range. Note: No NVM prog or erase routine can be called until this NVM operation is completed.
Stack size required	10
Resource used/ destroyed	R0,R3,R4,R5, DPTR

6.3.4 Abort NVM programming routine

This user routine aborts the NVM programming by closing an opened assembly buffer.

Please, note that every time assembly buffer is opened it should be then closed either by calling the program routine or the abort program routine. Any other sequence might leave the assembly buffer opened thus potentially corrupting data.

Table 6-11 Abort NVM programming subroutine

Subroutine	2'E884$_H$: USER_ABORTPROG
Input	--
Output	**PSW.CY** 0 = Abort successfully, assembly buffer closed. 1 = Abort failed as programming already started. Possible reason of failure: - Programming already started.

Table 6-11 Abort NVM programming subroutine (cont'd)

Stack size required	3
Resource used/ destroyed	R4

6.3.5 Read NVM status routine

This user routine checks for the NVM status.

Table 6-12 Read NVM status subroutine

Subroutine	2'E881$_H$: USER_NVMRDY
Input	--
Output	**PSW.CY** 0 = NVM is not busy. 1 = NVM is busy now.
Stack size required	3
Resource used/ destroyed	--

6.3.6 Read user calibration data

All data stored in user accessible config sector pages (OTP and 100TP) can be downloaded into the XRAM 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 6-4**) to an easily accessible data space (XRAM). To download the data, the user needs to provide the config sector page where data has to be read from, number of Bytes to be copied, and the XRAM 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 XRAM (starting at the given address).

Table 6-13 Read user calibration data subroutine

Subroutine	2'E87E$_H$: USER_READ_CAL
Input	**R7**: Number of Bytes to be copied into the XRAM (01$_H$ to 80$_H$). **R6**: user CS page to take data from (refer to **Figure 6-1**). **R4,R5**: High-Low XRAM address to copy data to (F000$_H$ < R4,R5 < (R4,R5) + R7 < FC00$_H$).

Table 6-13 Read user calibration data subroutine (cont'd)

Output	**PSW.CY** 0 = Read is successful. 1 = Read is not successful due to invalid input values.
Stack size required	4
Resource used/ destroyed	R0, DPTR

6.3.7 Read NVM config status routine

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

Table 6-14 Read NVM config status subroutine

Subroutine	2'E87B$_H$: USER_NVM_CONFIG
Input	--
Output	**R0**: Number of available sectors of the code area (4 kBytes each) **R1**: Number of available sectors of the data area (4 kBytes each) **PSW.CY** 0 = NVM status successfully read 1 = NVM status cannot be read. Possible reason of failure: - NVM Linear sector is set as 00$_H$.
Stack size required	3
Resource used/ destroyed	--

6.3.8 Read NVM ECC2 address routine

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

Table 6-15 Read NVM ECC2 address subroutine

Subroutine	2'E878$_H$: USER_NVM_ECC2ADDR
Input	--

Table 6-15 Read NVM ECC2 address subroutine (cont'd)

Output	**R0**: Last ECC2 address failing.(High Byte) **R1**: Last ECC2 address failing.(Low Byte) **PSW.CY** 0 = No NVM ECC2 detected 1 = NVM ECC2 address detected
Stack size required	3
Resource used/ destroyed	--

6.3.9 Read NVM protection status routine

This user routine returns the NVM protection status. Details in the following table.

Table 6-16 Read NVM protection status subroutine

Subroutine	2'E875$_H$: USER_NVMPROT_STATUS
Input	--
Output	**ACC**: NVM Protection status. Refer to **Table 6-17**.
Stack size required	2
Resource used/ destroyed	--

Table 6-17 NVM protection status

Field	Bits	Function
RD_CODE	3	**Status of code area** This Bit shows the read protection status of the code area. 0 Reading of code area is not permitted. 1 Reading of code area is permitted.
RD_DATA	2	**Status of data area** This Bit shows the read protection status of the data area. 0 Reading of data area is not permitted. 1 Reading of data area is permitted.
PRG_CODE	1	**Status of code area** This Bit shows the program protection status of the code area. 0 Programming of code area is not permitted. 1 Programming of code area is permitted.

Table 6-17 NVM protection status

PRG_DATA	0	**Status of data area** This Bit shows the program protection status of the data area. 0 Programming of data area is not permitted. 1 Programming of data area is permitted.
Reserved	7:4	Reserved, always read ‘0’

For the NVM protection mechanism, user configuration sector pages (OTP and 100TP) are considered being part of the NVM code area.

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

Read protection does not block code fetching.

6.3.10 Set NVM program protection (code) routine

This user routine sets the write protection on the NVM code area. Details in the following table.

Table 6-18 Set NVM program protection (code) subroutine

Subroutine	2’E872$_H$: USER_SET_PRGPROT_CODE
Input	--
Output	--
Stack size required	2
Resource used/ destroyed	--

Note: For the NVM protection mechanism, user configuration sector pages (OTP and 100TP) *are considered being part of the NVM code area.*

Note: This routine can be used to temporarily set write protection on the NVM code area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.11 Clear NVM program protection (code) routine

This user routine removes the write protection on the NVM code area. Details in the following table.

Table 6-19 Clear NVM program protection (code) subroutine

Subroutine	2’E86F$_H$: USER_CLR_PRGPROT_CODE
Input	--
Output	--
Stack size required	2
Resource used/ destroyed	--

Note: For the NVM protection mechanism, user configuration sector pages (OTP and 100TP) *are considered being part of the NVM code area.*

Note: This routine can be used to temporarily remove write protection on the NVM code area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.12 Set NVM read protection (code) routine

This user routine sets the read protection on the NVM code area. Details in the following table.

Table 6-20 Set NVM read protection (code) subroutine

Subroutine	2’E86C$_H$: USER_SET_RDPROT_CODE
Input	--
Output	--
Stack size required	2
Resource used/ destroyed	--

Note: For the NVM protection mechanism, user configuration sector pages (OTP and 100TP) *are considered being part of the NVM code area.*

Note: This routine can be used to temporarily set read protection on the NVM code area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.13 Clear NVM read protection (code) routine

This user routine removes the read protection on the NVM code area. Details in the following table.

Table 6-21 Clear NVM read protection (code) subroutine

Subroutine	2’E869$_H$: USER_CLR_RDPROT_CODE
Input	--
Output	--
Stack size required	2
Resource used/ destroyed	--

Note: For the NVM protection mechanism, user configuration sector pages (OTP and 100TP) *are considered being part of the NVM code area.*

Note: This routine can be used to temporarily remove read protection on the NVM code area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.14 Set NVM program protection (data) routine

This user routine sets the write protection on the NVM data area. Details in the following table.

Table 6-22 Set NVM program protection (data) subroutine

Subroutine	2’E866$_H$: USER_SET_PRGPROT_DATA
Input	--
Output	--
Stack size required	2
Resource used/ destroyed	--

Note: This routine can be used to temporarily set write protection on the NVM data area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.15 Clear NVM program protection (data) routine

This user routine removes the write protection on the NVM data area. Details in the following table.

Table 6-23 Clear NVM program protection (data) subroutine

Subroutine	2'E863$_H$: USER_CLR_PRGPROT_DATA
Input	--
Output	--
Stack size required	2
Resource used/ destroyed	--

Note: This routine can be used to temporarily remove write protection on the NVM data area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.16 Set NVM read protection (data) routine

This user routine sets the read protection on the NVM data area. Details in the following table.

Table 6-24 Set NVM read protection (data) subroutine

Subroutine	2'E860$_H$: USER_SET_RDPROT_DATA
Input	--
Output	--
Stack size required	2
Resource used/ destroyed	--

Note: This routine can be used to temporarily set read protection on the NVM data area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.17 Clear NVM read protection (data) routine

This user routine removes the read protection on the NVM data area. Details in the following table.

Table 6-25 Clear NVM read protection (data) subroutine

Subroutine	2'E85D$_H$: USER_CLR_RDPROT_DATA
Input	--

Table 6-25 Clear NVM read protection (data) subroutine (cont'd)

Output	--
Stack size required	2
Resource used/ destroyed	--

Note: This routine can be used to temporarily remove read protection on the NVM data area. It will overwrite the default setting controlled by NVM_PASSWORD and is only valid till next power on reset, brown-out reset or wake-up from sleep mode occurs.

6.3.18 Read OTP data routine

This routine reads the OTP data. Details in the following table.

Table 6-26 Read OTP subroutine

Subroutine	2'E85A$_H$: USER_READ_OTP
Input	**R7**: Data Offset (00$_H$ to 7F$_H$) **R6**: OTP page selection Byte (CS_Byte, refer to **Figure 6-1**)
Output	**ACC** = OTP Data **PSW.CY** 0 = Read is successful. 1 = Read is not successful due to invalid range selected.
Stack size required	2
Resource used/ destroyed	R0, DPTR

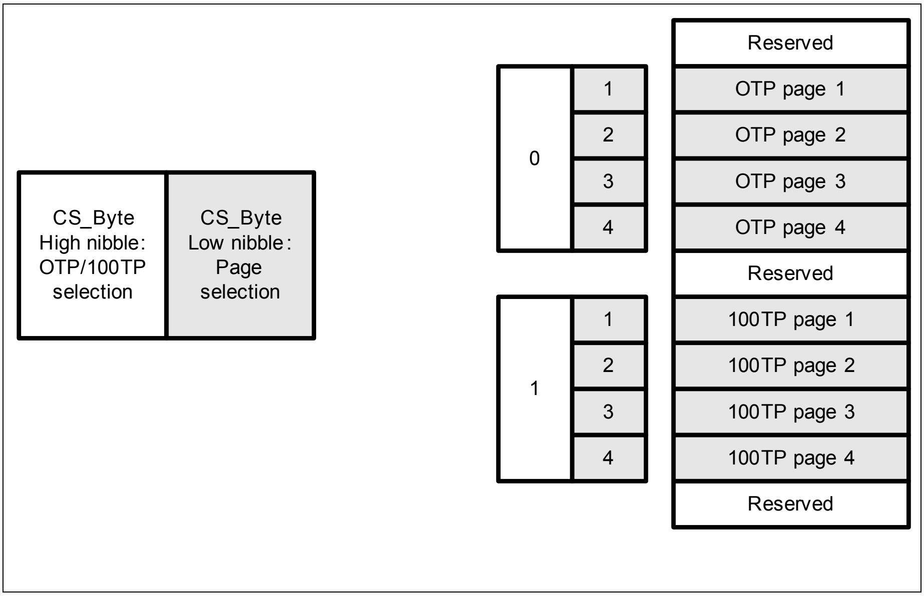


Figure 6-1 User configuration sector pages address Byte description

6.3.19 Program OTP routine

This routine programs data into the OTP pages. The OTP contents to be programmed are preloaded into the XRAM. The details can be found in **Section 6.2.2**.

Table 6-27 Program OTP subroutine

Subroutine	2’E857$_H$: USER_OTP_PROG
Input	**R7**: OTP page selection Byte (CS_Byte, refer to **Figure 6-1**) XRAM preloaded with the OTP data to be programmed.
Output	**PSW.CY** 0 = Program completed successfully 1 = Program failed. Possible reasons of failure: - The NVM code area is protected against programming. - The OTP page is already programmed.
Stack size required	11
Resource used/ destroyed	R0, R5, R6, R7, DPTR

6.3.20 LIN autobaud routine

This user routine configures the device baud rate and is provided to facilitate the LIN activity. The routine calculates the baud rate upon receiving of a valid LIN SYN Break and SYN Char and sets the device baud rate generator accordingly. User can use this routine as part of LIN software driver. The routine does not use the fractional divider. The frame type information can be found in LIN BSL chapter (refer to **Section 4.3** for further details).

Table 6-28 LIN autobaud subroutine

Subroutine	2'E854$_H$: USER_LIN_AUTOBAUD
Input	LIN frame
Output	--
Stack size required	6
Resource used/ destroyed	R0, R1, R2, R3, R4, DPTR, MEX2, MEX3, UART registers Timer 0 and 2 is used for timeout and baud rate calculation.

6.3.21 UART autobaud routine

This user routine configures the UART for data transmission. After calling this routine, the host needs to send a frame containing 80$_H$ to start the autobaud process. Once the operation is completed successfully, a response frame of 55$_H$ will be received.

Table 6-29 UART autobaud subroutine

Subroutine	2’E851$_H$: USER_UART_AUTOBAUD
Input	80$_H$ (Same protocol used in UART BSL)
Output	Response code of 55$_H$ is sent to host
Stack size required	8
Resource used/ destroyed	--

6.3.22 User XRAM download routine

This routine allows user to download code into the XRAM starting from address F000$_H$. To prepare for the download, the user will need to call the USER_UART_AUTOBAUD routine to configure the UART port settings for the download. Thereafter, calling this routine, the host will need to send the code in data frames of 64 Bytes each. To terminate the download and execute the downloaded code, an EOT frame needs to be sent. Information of Data and EOT frame is available in UART BSL chapter.

Table 6-30 User XRAM download subroutine

Subroutine	2’E84E$_H$: USER_XRAM_DOWNLOAD
Input	Data frames of 64 Bytes each (Protocol used same as UART BSL)
Output	Code is downloaded into XRAM and jumps to XRAM code after receiving an EOT frame.
Stack size required	9
Resource used/ destroyed	--

6.3.23 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 6-4**. Details in the following table.

Table 6-31 Read 100 Time Programmable subroutine

Subroutine	2'E84B$_H$: USER_READ_100TP
Input	**R7**: Data Offset (00$_H$ to 7F$_H$) **R6**: 100TP page selection Byte (CS_Byte, refer to **Figure 6-1**)
Output	**ACC** = 100TP Data **PSW.CY** 0 = Read is successful. 1 = Read is not successful due to invalid range selected.
Stack size required	2
Resource used/ destroyed	R0, DPTR

6.3.24 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 XRAM. The details can be found in **Section 6.2.1**.

Table 6-32 Program 100 Time Programmable subroutine

Subroutine	2'E848$_H$: USER_100TP_PROG
Input	**R7**: 100TP page selection Byte (CS_Byte, refer to **Figure 6-1**) XRAM preloaded with the 100TP data to be programmed.
Output	**PSW.CY** 0 = Program completed successfully 1 = Program failed. Possible reasons of failure: - The NVM code area is protected against programming. - The 100TP page is already programmed to a maximum of 100 times.
Stack size required	11
Resource used/ destroyed	R0, R5, R6, R7, DPTR

6.3.25 Sector Erasing Routine

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

Table 6-33 Sector Erasing Subroutine

Subroutine	2'E845$_H$: USER_ERASE_SECTOR
Input	**R6(high Byte), R7(low Byte)**: NVM Sector address
Output	**PSW.CY** 0 = Erasing completed successfully. 1 = Erasing failed.
Stack size required	8
Resource used/ destroyed	R0, R3, R4

6.3.26 User clock setting routine

This routine is used to change the system clock frequency. 3 possible predetermined clock frequencies are supported: 24 MHz, 32 MHz and 40 MHz. The routine uses internal low precision clock as intermediate clock while changing PLL settings and waiting for the lock. In case PLL does not lock within 1 ms, a fail is reported and the system will go on running using the low precision clock. The routine will change analog module clocks frequency according to the new system frequency.

Table 6-34 Clock setting routine

Subroutine	2'E83F$_H$: USER_SET_USER_CLK
Input	**R7**: Clock Frequency Selection Byte: = 03$_H$: 40 MHz = 02$_H$: 32 MHz = All other Values: 24 MHz
Output	**None**
Stack size required	10
Resource used/ destroyed	R0, R7

6.3.27 NVMCLKFAC setting routine

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

Table 6-35 NVMCLKFAC setting subroutine

Subroutine	2’E83C$_H$: USER_NVMCLKFAC_SET
Input	**R7[0]**: NVMCLKFAC value to be written in SYSCON0.
Output	--
Stack size required	1
Resource used/ destroyed	R0, R7

6.3.28 XRAM MBIST starting routine

This routine is used to start a XRAM test. A linear write/read algorithm using alternating data is executed on a XRAM range specified by the start and stop addresses given as input parameters. When starting a MBIST test, standard XRAM interface is disabled. Therefore data stored into it will not be accessible and data stored in the memory range under test will be destroyed. The standard interface will be re-enabled by the XRAM MBIST check routine in case the test is completed.

Table 6-36 XRAM MBIST start subroutine

Subroutine	2’E833$_H$: USER_XRAM_MBIST_START
Input	**R4(High Byte), R5(Low Byte):** Stop address of XRAM range. **R6(High Byte), R7(Low Byte):** Start address of XRAM range.
Output	**PSW.CY** 0 = Pass, address range valid. 1 = Fail, address range invalid
Stack size required	9
Resource used/ destroyed	R4, R5, R6, R7, DPTR

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

No breakpoints between a call to the XRAM MBIST start routine and a call to XRAM MBIST check routine resulting in a completed test are allowed.

6.3.29 XRAM MBIST check routine

This routine checks the result of the MBIST test. It returns a fail in case the test is completed with errors, the test is still running or no test has been started. The routine will re-enable the standard XRAM interface and clear the result of the test in case it is completed.

Table 6-37 XRAM MBIST check subroutine

Subroutine	2'E830$_H$: USER_XRAM_MBIST_CHECK
Input	**R7 [0]**: XRAM range initialization with zeros option. 0 = XRAM range initialization disabled. 1 = XRAM range initialization enabled.
Output	**R7 [0]** 0 = MBIST test pass 1 = MBIST test fail **R7 [1]** 0 = MBIST test started 1 = MBIST test not started **R7 [2]** 0 = MBIST test completed 1 = MBIST test running **PSW.CY:** Overall pass/ fail indicator. OR of Bits R7[0], R7[1] and R7[2] 0 = Test passing 1 = Check fails
Stack size required	9
Resource used/ destroyed	R7, DPTR

6.4 NVM user applications

The NVM user routines application is described in this section.

6.4.1 NVM integrity handling (Service Algorithm)

Upon reset, NVM initialisation of the NVM data space is started in the startup routine. The NVM initialisation includes any repair to the data space by Service Algorithm (SA). The repair by the SA may include erasing of faulty NVM pages or double mapped NVM pages. The outcome of the NVM initialisation is reported to MEMSTAT at the end of the startup routine. The user is recommended to check this SFR.

In the event of an SA failure, the user will need to be informed once the startup is completed, as the failing sector could be a critical sector. In such a case, the user can choose either to reset the system to perform the NVM re-initialisation again or to erase the failing NVM sector to recover the NVM. In both cases of the SA execution, regardless of the execution status, the last access sector information will be stored in SECTORINFO. The user is recommended to check the SECTORINFO for the repair sector information. This is to ensure that recovery procedures can be executed in user code for any critical user information that may be affected by the SA repair.

Detailed description of the MEMSTAT register can be found in the user manual.

6.4.2 Supporting background NVM operation

To support other user activities while NVM is busy, the BootROM can direct code execution to XRAM after triggering time consuming NVM operations like program and erase. This type of background code execution is known as Type 2 NVM operations. When active, BootROM routine will jump to the 3rd last Byte of the XRAM area ($FBFD_H$) every time it has to wait for NVM internal operation to be finished. **Table 6-38** shows XRAM branching address and provides an example for the XRAM code preparing for a branch to the user code at $F000_H$. At the end of the user defined code, a RET instruction needs to be present for the normal NVM operation to be completed. **Figure 6-2** shows how background programming can be supported during calls to an NVM programming routine.

There is only one NVM module present in TLE983x. 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 7.4 ms to 22.2 ms to be completed. Therefore there is a need to support the user for critical activities, e.g watchdog refreshing.

Table 6-38 XRAM contents for branching to user code at $F000_H$

XRAM Address	**XRAM contents to support jump to user code located at $F000_H$**
$F000_H$	Start of user defined code
End of user defined code location	22_H (RET)
$FBFD_H$	02_H (LJMP)
$FBFE_H$	$F0_H$ (user defined code address high Byte)
$FBFF_H$	00_H (user defined code address low Byte)

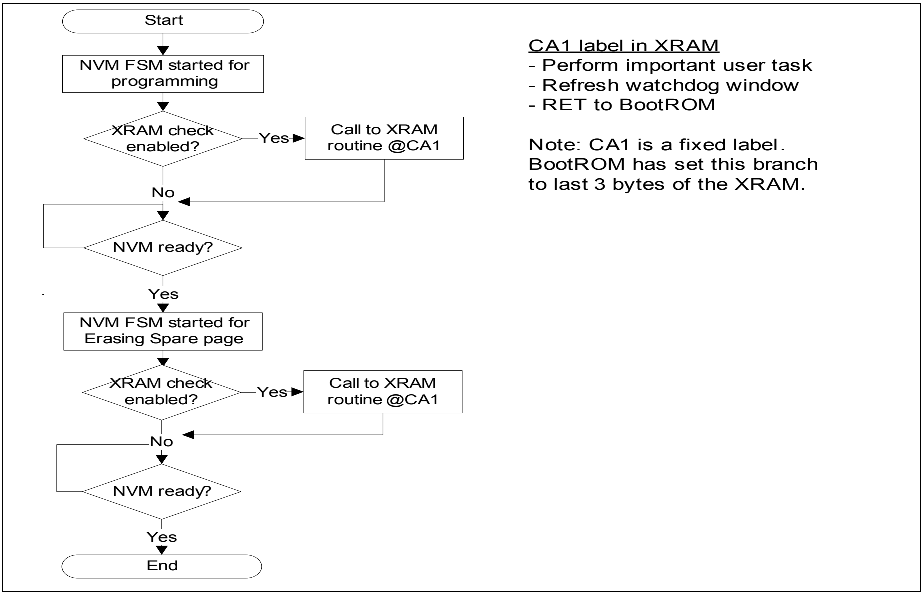

Start
NVM FSM started for programming
XRAM check enabled?
Yes
Call to XRAM routine @CA1
No
NVM ready?
Yes
NVM FSM started for Erasing Spare page
XRAM check enabled?
Yes
Call to XRAM routine @CA1
No
NVM ready?
Yes
End
CA1 label in XRAM
- Perform important user task
- Refresh watchdog window
- RET to BootROM
Note: CA1 is a fixed label. BootROM has set this branch to last 3 bytes of the XRAM.

Figure 6-2 Background NVM programming operation with jumps to XRAM code (example for non-linearly mapped sector)

6.4.3 Emergency operation handling

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.

6.4.3.1 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 6-39**.

Table 6-39 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.
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.

6.4.3.2 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 6-39**. For the case without interrupt enabled, it is shown in **Table 6-40**.

Table 6-40 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 XRAM. 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.

6.4.3.3 Emergency operation handling timing

In this chapter some information about overall emergency operation worst case timing is provided.

Table 6-41 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 linearly mapped region of the NVM is similar (step 6 and 7 are inverted and a few us have to be added for mapram update) and overall worst case time is the same.

Table 6-41 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 performed.
7	User critical data are programmed in the target page.

The Table 6-41 refers to the type 1 routines but data are similar for type 2 routines as well.

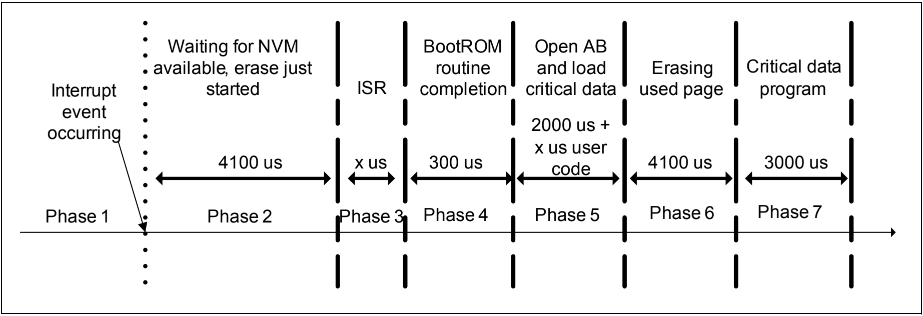


Figure 6-3 Worst case emergency handling timing when linear sector is used

Worst case time, shown in Figure 6-3, 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.

6.4.4 NVM user routines operation

This section describes the application of some NVM user routines.

6.4.4.1 NVM user programming operation

In TLE983x, the NVM supports programming of up to 128 Bytes of data at once. The user can execute the following sequence illustrated in **Figure 6-4** 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 ***"MOVC (@DPTR++), A"*** instruction. The details of this instruction can be found in **Table 6-42**.

Table 6-42 Write code Byte instruction: MOVC (@DPTR++), A

Description	Store the Byte content of accumulator to program memory. The address in program memory is pointed to by the data pointer. The data pointer is incremented by hardware, after the write. No flags are affected.
Example	Store value $E4_H$ to program memory at 1000_H. MOV A, #E4H MOV DPTR, #1000H ANL EO, #11101000B ; select MOVC instruction DB 0A5H ; MOVC (@DPTR++), A ORL EO,#00010000B ; select TRAP instruction

This instruction "MOVC (@DPTR++), A" is XC800-specific, therefore will not be supported by standard 8051 assembler. For this reason, it has to be explicitly invoked in user application code according to the example reported in the **Table 6-42**

Note: This instruction shares the same opcode with another XC800-specific instruction TRAP. MOVC is selected only if EO.TRAP_EN = 0.

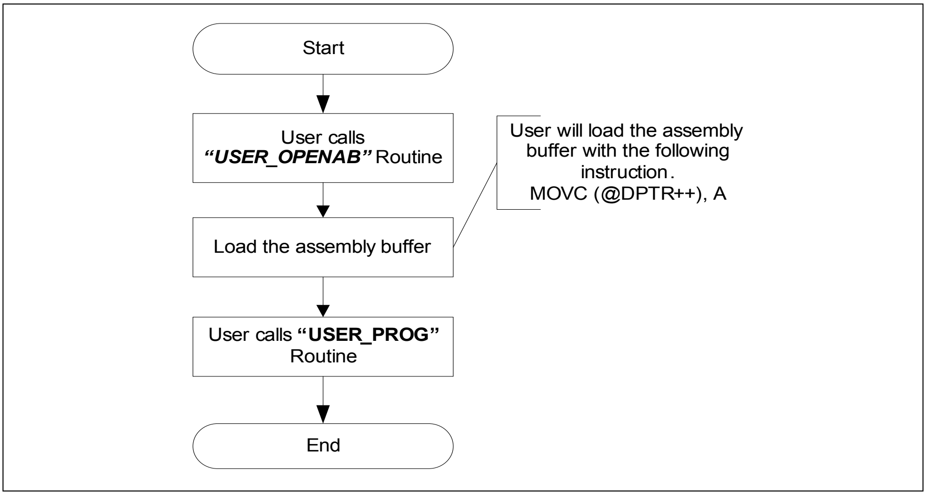

Start
User calls
"USER_OPENAB" Routine
Load the assembly buffer
User calls "USER_PROG" Routine
End
User will load the assembly buffer with the following instruction.
MOVC (@DPTR++), A

Figure 6-4 NVM user program

6.4.4.2 Tearing-safe Programming

In TLE983x, 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 decided, by means of an specific input parameters of the user programming routine (refer to **Table 6-9**), 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 map-RAM 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

6.4.4.3 NVM user erase operation

The user can execute the following sequence illustrated in **Figure 6-5** for NVM user erase.

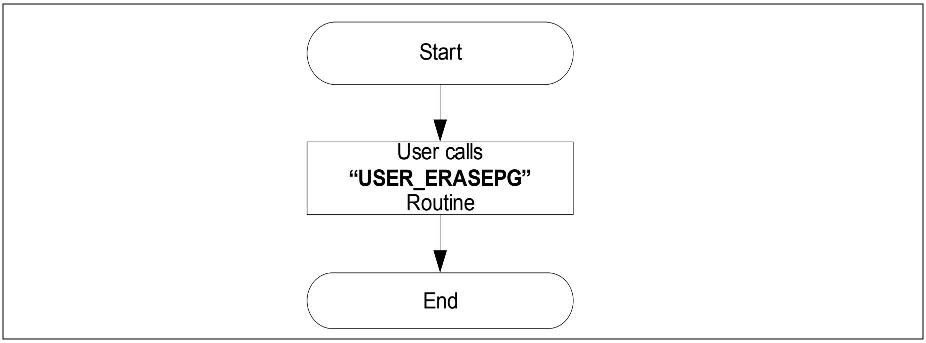


Figure 6-5 NVM user erase

6.4.4.4 NVM user programming abort operation

The user can execute the following sequence illustrated in Figure 6-6 for NVM user programming abort.

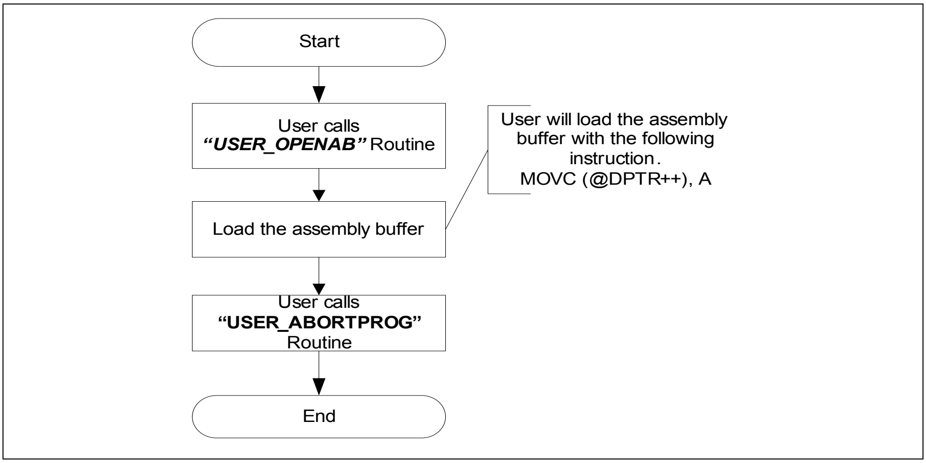


Figure 6-6 NVM user abort program

6.4.5 Protection mechanism on NVM

User can use BSL mode 6 (LIN or UART), to control the NVM protection. Once the password is programmed, program and read protection on the NVM is enabled upon startup. During normal operation, if user wishes to program or read the NVM memories, he can temporarily disable the NVM protection using the user routines provided for their intended operation.