TLE986x/TLE987x Family BF-Step

Application Note for FastLIN BSL Mode

Application Note

Rev. 1.00, 2016-12-05

Automotive Power

Table of Contents

1 Abstract

Note: The following information is given as a hint for the implementation of the device only and shall not be regarded as a description or warranty of a certain functionality, condition or quality of the device.

This Application Note is intended to provide further information about the handling and usage of the boot-strap-loader and the FastLIN mode inparticular.

2 Introduction

The devices of the TLE986x/TLE987x family provide a built-in boot-strap loader (BSL) mode inside the firmware ROM. The BSL mode supports the FastLIN protocoll using the integrated LIN tranceiver. The following chapters do provide a detailed view of the features supported by the FastLIN BSL protocol. For further information please also read the “TLE986xQX-BootROM-User-Manual-14-Infineon.pdf”.

3 FastLIN BSL Mode

The device has a built-in boot-strap-loader mode implemented which can be triggered at the device startup. The protocol is a Infineon Technologies proprietary half-duplex UART protocol, running at a fixed speed of 115.2kBaud with 8N1, it is named FastLIN. The FastLIN protocol is providing the following features:

- transfere code or data to the device internal NVM module
- transfere code or data to the device internal RAM module
- read code or data from the device internal NVM module
- erase of the device internal NVM module
- set/reset read/write protection of the device internal NVM module
- device indentification

Possible usecase for the FastLIN BSL mode are:

- end-of-line code download
- end-of-line parameterization
- customer firmware update in the garage

The following chapters will dive into the details of the FastLIN protocol.

4 BSL Connection Window

A BSL connection can only be established when the device is executing the BootROM. To get the device to execute the BSL in the BootROM the device has to be reset, either by a power-up-reset (POR) or by a PIN reset. After the reset release and some fundamental initializations of the device has been done, the BSL connection acception window starts. The duration of this timing window is defined by the None-Activity-Counter (NAC). A BSL connection attempt has to be started once the BSL connection window has been opened and the BSL connection attempt has o be finished before the BSL connection window closes with the expiration of the NAC. The following chapter provides some more inside view into the NAC.

4.1 None-Activity-Counter - NAC

The NAC timer is started before the BSL mode inside the firmware starts during startup. Once the NAC timer has expired the boot-up process will be continued and the control of the device will be given to the user application. The NAC value is a value which will be defined by the user and stored inside the code flash. Usually the NAC value is part of the user application which was downloaded before.

The NAC value is stored inside one byte, only six bits of the NAC byte are defining the timeout value of the NAC. The NAC value inside the NVM is secured by storeing it as a 1s-complement value in the byte following the NAC value. Only if the true NAC value and the complement NAC do match (means: not (true NAC) == complement NAC) the NAC value is valid and the NAC timer will be activated during start-up. In case of an invalid value, i.e. like for an erased flash where both bytes containing 0xFF, the NAC timer never expires, means the device will stay and wait in BSL mode. This behavior is especially usefull for fresh devices, where no user application has been downloaded to the device yet. Here the firmware will not branch into user mode, but instead it will stay in BSL mode and keeps waiting for any BSL communication to download any user application into the flash.

The NAC value and its complement is stored at the following addresses inside the user accessible flash:

Table 1 Address of the NAC values inside flash

Address	Usage
0x11000000 + (Total_Flash_Size - 0x1004)	true NAC value
0x11000000 + (Total_Flash_Size - 0x1003)	complement NAC value

The meaning of the NAC bits are listed in table 2.

Table 2 NAC bit meaning

NAC Bit	Usage
5..0	NAC expire value n, (n - 1) * 5ms, 0ms..55ms 0x00, 0x0D..0x3F: NAC never expires, device stays in BSL Mode, BSL mode is active 0x01: BSL mode is skipped, BSL mode deactivated 0x02..0x0C: NAC timeouts between 5ms..55ms, BSL mode is active
7..6	0b01: BSL interface selection Fast-LIN 0b1x: BSL interface selection UART

A fresh device, or a completly ereased device always selects Fast-LIN as BSL interface and stays in BSL mode during start-up.

4.2 Node Address - NAD

The NAD is used for the BSL communication to select an individual node. The NAD is a 8 bit value, where only the values 0x01 to 0xFF are valid, 0x00 is an invalid value. The value 0xFF acts as a broadcast, this means all devices connected to the LIN line are addressed no matter which NAD value is programmed inside the device. The broadcast can be used to establish a BSL connection to devices where the programmed NAD value is unknown.

The NAD value is stored inside one byte inside the code flash area. The NAD value inside the NVM is secured by storeing it as a 1s-complement value in the byte following the NAD value. Only if the true NAD value and the complement NAD do match (means: not (true NAD) == complement NAD) the NAD value is valid otherwise the device will only react on the default NAD value, which is 0x7F.

The NAD value and its complement is stored at the following addresses inside the user accessible flash:

Table 3 Address of the NAD values inside flash

Address	Usage
0x11000000 + (Total_Flash_Size - 0x1002)	true NAD value
0x11000000 + (Total_Flash_Size - 0x1001)	complement NAD value

Table 4 NAD meaning

NAD Bit	Usage
0x00	invalid NAD value, device will not react on these value, default NAD = 0x7F is used
0x01..0xFE	valid NAD values
0xFF	broadcast NAD, all devices will react on this value

A fresh device, or a completly ereased device always react on the default NAD (0x7F) only.

5 The Connection Sequence

The FastLIN mode has been protected agains unwanted entries, i.e. because of noise on the communication line. In order to establish a FastLIN connection the following sequence has to be sent to the device during the active BSL connection window (NAC):

- Host: sending the “Get Chip Id” command
- Device: answers Acknowledge (0x55)
- Device: answerd with the ChipID

If this sequence has been passed to the device during the BSL active window and the device has acknowleged the commands and answered with the ChipID then the BSL connection is establed. The “ChipID” command has been extended just for the purpose of the BSL entry. The NAC will be disabled, the WDT1 will be disabled and the device waits for further FastLIN commands. Once the FastLIN mode has been entered successfully there is no timeout which would exit the FastLIN mode and automatically resets the device or jumps to user mode. All the actions of the FastLIN mode has to be triggered by the host.

The **Figure 1** displays a scope shot of an successfull connection establishment. It shows the signal on the LIN line and the decoded UART data in the bottom. Sections highlighted in red color are sent by the host, while the sections highlighted in green color are the responses of the target device. The duration of the complete sequence is approximately 1.4ms.

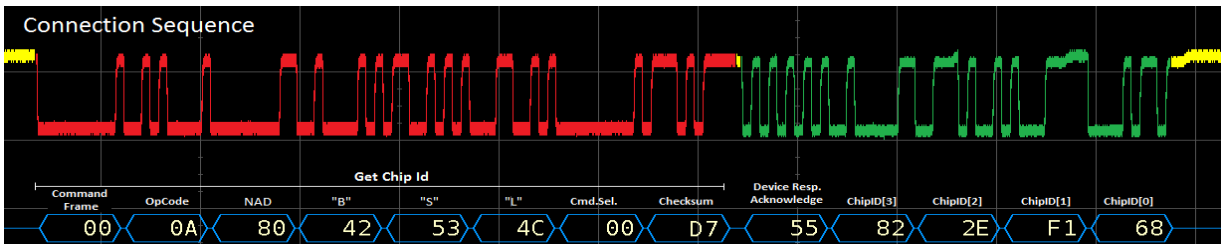

Figure 1 FastLIN conection sequence (yellow: idle, red: host sends, green: target response)

The first byte of the parameter is the NAD (node address). This value is getting compared by the device to the NAD value which is stored inside the code flash. If the NAD value matches the devices replies with acknowledge and the ChipID, otherwise the device will not respond at all. If the NAD value is 0xFF (broadcast) then all connected devices will enter into BSL mode. In **Figure 1** the NAD value of 0x80 is just an example, the NAD value has to match the NAD stored in the device which should be connected. The parameter bytes[1:3] encode the characters “BSL” and are contant inputs to the “ChipID” command for the BSL entry.

00_H (Command Header)	**$0A_H$** (Command OpCode)	**Parameters** (5 bytes)					**Checksum** (1 byte)
		NAD (1 byte)	**‚B‘ 0x42** (1 byte)	**‚S‘ 0x53** (1 byte)	**‚L‘ 0x4C** (1 byte)	**Option $=00_H$** (1 byte)	

In case the connection was not established properly various reasons are possible:

- Host is running with the wrong baudrate, only 115.2kBaud are supported
- Signal integrity on the LIN line is bad
- BSL actitive window (NAC) is either expired already, or disabled completely
- NAD parameter inside the “Get ChipId” command does not match
- “Get Chip Id” command wrong
- wrong checksum for the “Get Chip Id” command

6 FastLIN Commands

The following chapters will list all the commands supported by the FastLIN protocoll. Each command is defined by a header frame, which provides the necessary parameters for the command. Certain commands also require data frames, for transferring the payload of a command to or from the device. For each command the device responds at least with an Acknoledge.

6.1 Transfer Block structure

A transfer block consists of three parts:

Header Type (1 byte)	**Data Area** (X bytes)	**Checksum** (1 byte)

- **Header Type**: the type of block, which determines how the bytes in the data area are interpreted. Implemented header types are:
  - 00_H type “Command Frame”
  - 01_H type “Data Frame”
  - 02_H type “Data End of Transmission Frame”
- **Data area**: A list of Bytes, which represents the data of the block. The length of data area must not exceed 129 Bytes for mode 0 and 2. For mode 2, the length of data area must always be 128 Bytes for data frames, and upt to 129 Bytes for EOT frames.
- **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 FastLIN 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.

6.1.1 Command Frame

The format of a command frame is given as shown in the following figure. The header type is 0x00 (Command Header) followed by an 1-byte OpCode, which identifies the command. The following five bytes are passing the parameters for the selected command. A checksum byte closes the command frame.

00_H (Command Header)	**Command OpCode** (1 byte)	**Command Parameters** (5 bytes)	**Checksum** (1 byte)

6.1.2 Data Frame

The format of a data frame is given as shown in the following figure. The header type is 0x01 (Data Header) followed by the payload. The length of the data frame is the defined by the command which requires a data frame. The maximum length of the data frame can be up to 130 bytes, which makes the payload max. 128 bytes. A data frame is used to transfere up to one entire NVM page (128 bytes) to the target device. The data transfere phase has to be closed by sending a End-of-Transmission frame, see **Chapter 6.1.3**. If less than 128 bytes needs to be transfered an End-of-Transmission frame can be sent immediately, without the need of sending a data frame before. A checksum byte closes the data frame.

01_H (Data Header)	**Payload** Block Length – 2 in Bytes (max. 128 bytes)	**Checksum** (1 byte)

6.1.3 End-of-Transmission (EOT) Frame

The format of the EOT frame is given as shown in the following figure. The header type is 0x02 (EOT Header) followed by a byte which defines the number of payload bytes to be sent with the EOT frame. The length of the EOT frame is defined by the command which requires an EOT frame. The maximum length of the payload can be 128 bytes, which makes the entire EOT frame max. 131 bytes long. In case the remaining bytes to be tranferred within the EOT frame do not fill up the frame completely, then the payload has to be padded with dummy bytes to form a complete frame. The total number of bytes in the frame has to match the “Block Length” parameter of the previous command frame. A checksum byte closes the data frame.

02_H (EOT Header)	**remaining Payload length** (1 byte)	**Payload** (remaining bytes)	**Not Used** Block Length – 3 – remaining Payload length in Bytes	**Checksum** (1 byte)

6.2 Get Chip Id

This command has to be used as entry command for establishing an BSL connection. Beside the function as entry sequence it can be used to identify the target device.

00_H (Command Frame)	**$0A_H$** (Command OpCode)	**Parameters (5 bytes)**		**Checksum** (1 byte)
		Not Used (4 bytes)	**Option $=00_H$** (1 byte)	

For the “Get Chip Id” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x0A. Since the OpCode 0x0A is grouping a selection of commands the 5th byte of the parameters specifies the “Get Chip Id” command by sending a 0x00. The first four bytes of the parameters are not used for this command.

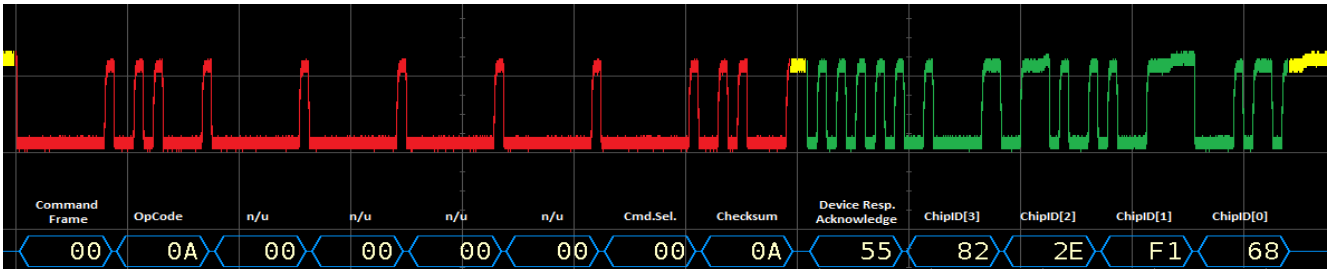

Figure 2 “Get Chip Id” waveform including the device response (yellow: idle, red: host sends, green: target response)

6.2.1 Response

If the command was received and understood the device aswers with an Acknowledge (0x55), followed by the device’s chip id. For the decoding of the chip id, please read the “BootROM Usermanual for TLE986x”.

6.3 NVM Page Verify by Checksum

This command is intened to perform a NVM page verify against a given checksum. The content of the addressed NVM page is summed up (inverted 16bit-XOR sum). The result is compared against the expected checksum provied by the command.

		Parameters (5 bytes)					
00_H (Command Header)	**$0A_H$** (Command OpCode)	**StartAddr [22:15]** (1 byte)	**StartAddr [14:7]** (1 byte)	**Expected CHKSum High** (1 byte)	**Expected CHKSum Low** (1 byte)	**Option $=10_H$** (1 byte)	**Checksum** (1 byte)

For the “NVM Page Verify by Checksum” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x0A. Since the OpCode 0x0A is grouping a selection of commands the 5th byte of the parameters specifies this command by sending a 0x10. The first two bytes of the parameters define the start address of the NVM page as an offset to the NVM start address of 0x11000000. Bytes[3:2] of the parameter defines the expected 16-bit checksum of the selected NVM page. The device internal calculated checksum is compared against the checksum provided in Bytes[3:2] of the parameters.

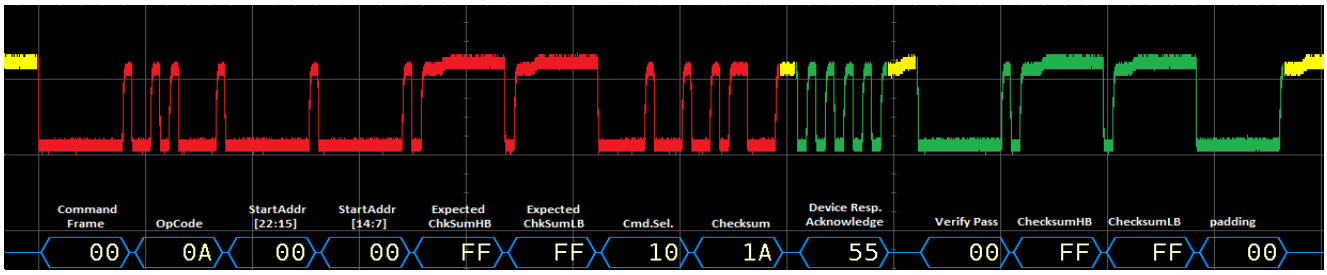

Figure 3 “NVM Page Verify by Checksum” waveform with device pass response (yellow: idle, red: host sends, green: target response)

6.3.1 Response

The device answers with an Acknowledge (0x55) if the command by itself was received correctly and is valid. The following bytemap explains the result sent as response. Byte[0] provides the feedback whether the command was executed pass or fail, a value of 0x00 represents a pass. The calculated checksum among the select page matches the expected checksum given as parameter. A value of 0x80 repesents a fail, the calculated checksum does not match the expected checksum. The Bytes[2:1] provide the calculated checksum back to the host. Byte[3] is always read as 0x00.

Pass/Fail Response Bit7: 1 = fail	**Calc. Checksum** HighByte	**Calc. Checksum** LowByte	**0x00**

In **Figure 4** is an example displayed where the page address parameter is given out of range. The device answered with 0xFF to this command, it will not be processed.

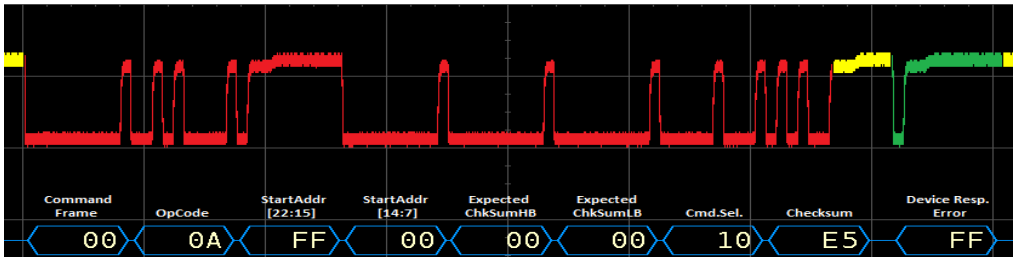

Figure 4 “NVM Page Verify by Checksum” frame with wrong parameter, device response is 0xFF

The second communication example, **Figure 5**, shows a correct command frame, the device response with an Acknowledge (0x55). But the expected checksum as parameter did not match to the checksum internally calculated based on the page data. The first byte of the response is 0x80, telling that the calculated checksum doesn’t match the expected checksum. Bytes[2:1] of the response (here 0xFF, 0xFF) are the calculated checksum. The last byte of the response is 0x00.

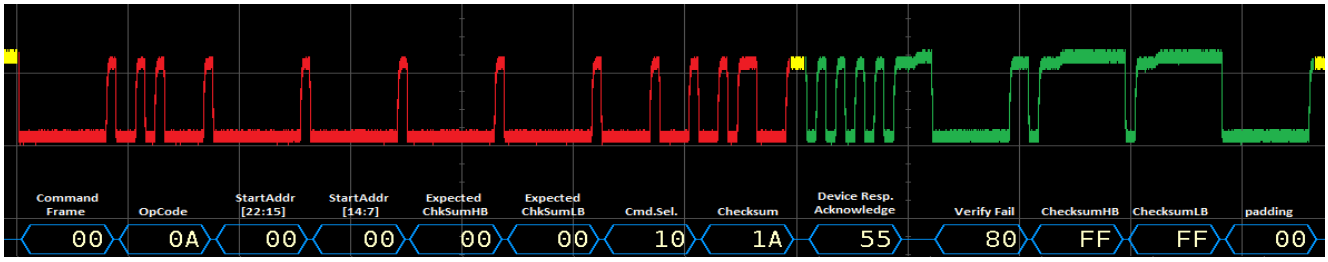

Figure 5 “NVM Page Verify by Checksum” correct frame with wrong expected checksum

6.4 NVM full Chip Verify by Checksum

This command is intened to perform a mass verify against a given checksum. The content of the code NVM section is summed up (inverted 16bit-XOR sum). The content of the date NVM section is not included in the checksum calculation. The result is compared against the expected checksum provied by the command.

00_H (Command Header)	**$0A_H$** (Command OpCode)	**Parameters** (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)	

For the “NVM full Chip Verify by Checksum” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x0A. Since the OpCode 0x0A is grouping a selection of commands the 5th byte of the parameters specifies this command by sending a 0x18. The first two bytes of the parameter is not used as the checksum will be calculated among the entire code NVM region. Bytes[3:2] of the parameter defines the expected 16-bit checksum of the entire code NVM. The device internal calculated checksum is compared against the checksum provided in Bytes[3:2] of the parameters.

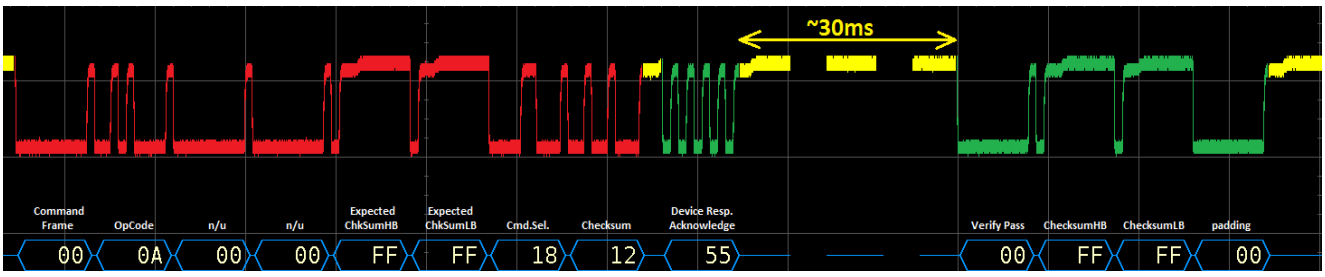

Figure 6 “NVM full Chip Verify by Checksum” waveform with device pass response

6.4.1 Response

The device answers with an Acknowledge (0x55) if the command by itself is received correctly and is valid. The following bytemap explains the result sent as response. Byte[0] provides the feedback whether the command was executed pass or fail, a value of 0x00 represents a pass. The calculated checksum among the select page matches the expected checksum given as parameter. A value of 0x80 repesents a fail, the calculated checksum does not match the expected checksum. The Bytes[2:1] provide the calculated checksum back to the host. Byte[3] is always read as 0x00. Due to the fact that the entire code NVM needs to be checked the sending of the response is delayed by approx. 30ms for a 128KB device.

Pass/Fail Response Bit7: 1 = fail	**Calc. Checksum** HighByte	**Calc. Checksum** LowByte	**0x00**

The communication example in **Figure 5** shows a correct command frame, the device response with an Acknowledge (0x55). But the expected checksum as parameter did not match to the checksum internally calculated based on the page data. The first byte of the response is 0x80, telling that the calculated checksum doesn't match the expected checksum. Bytes[2:1] of the response (here 0xFF, 0xFF) are the calculated checksum. The last byte of the response is 0x00.

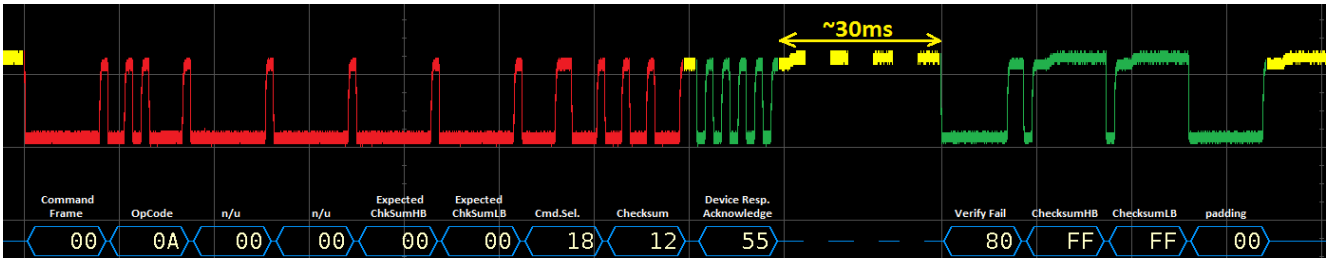

Figure 7 "NVM full Chip Verify by Checksum" correct frame with wrong expected checksum

6.5 100TP-Page Verify by Checksum

This command is intened to perform a verify of a 100TP page against a given checksum. The content of the 100TP page is summed up (inverted 16bit-XOR sum). The result is compared against the expected checksum provied by the command.

00_H (Command Header)	**$0A_H$** (Command OpCode)	**Parameters** (5 bytes)					**Checksum** (1 byte)
		100TP Page (1 byte)	**Not Used** (1 byte)	**Expected CHKSum High** (1 byte)	**Expected CHKSum Low** (1 byte)	**Option $=50_H$** (1 byte)	

For the "100TP-Page Verify by Checksum" command the Header Type is a "Command Frame" (0x00), the OpCode is 0x0A. Since the OpCode 0x0A is grouping a selection of commands the 5th byte of the parameters specifies this command by sending a 0x50. The first byte of the paramteres define the 100TP page to be verified. There are eight 100TP pages available, which are addressed starting from 0x11 to 0x18, where 0x11 adresses 100TP-Page0 and 0x18 addresses 100TP-Page7. Any other value than 0x11..0x18 will lead to an command error response. Bytes[3:2] of the parameter defines the expected 16-bit checksum of the selected NVM page. The device internal calculated checksum is compared against the checksum provided in Bytes[3:2] of the parameters. **Figure 8** shows an example where 100TP-Page0 is checked against the expected checksum of 0x276D. The device accepts the command and responds with a pass response.

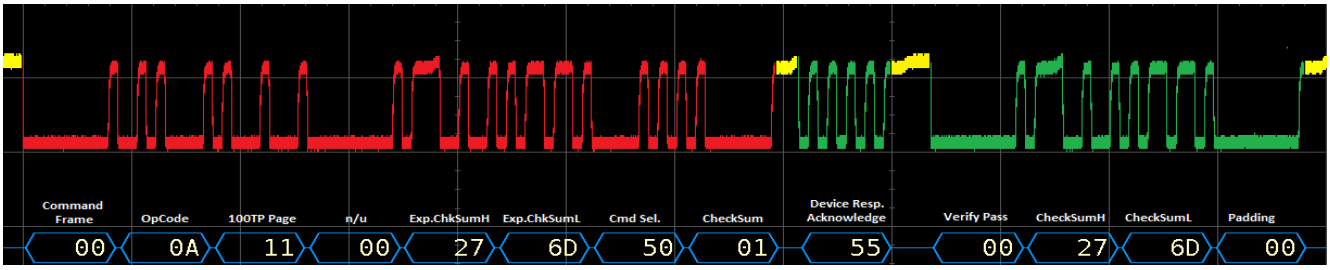

Figure 8 “100TP-Page Verify by Checksum” waveform with device pass response

6.5.1 Response

The device answers with an Acknowledge (0x55) if the command by itself was received correctly and is valid. The following bytemap explains the result sent as response. Byte[0] provides the feedback whether the command was executed pass or fail, a value of 0x00 represents a pass. The calculated checksum among the select page matches the expected checksum given as parameter. A value of 0x80 repesents a fail, the calculated checksum does not match the expected checksum. The Bytes[2:1] provide the calculated checksum back to the host. Byte[3] is always read as 0x00.

Pass/Fail Response Bit7: 1 = fail	**Calc. Checksum** HighByte	**Calc. Checksum** LowByte	**0x00**

In **Figure 9** is an example displayed where the 100TP page address parameter is given out of range, a value of 0x10 is not a valid parameter for selecting a 100TP page. The device answered with 0xFF to this command, it will not be processed.

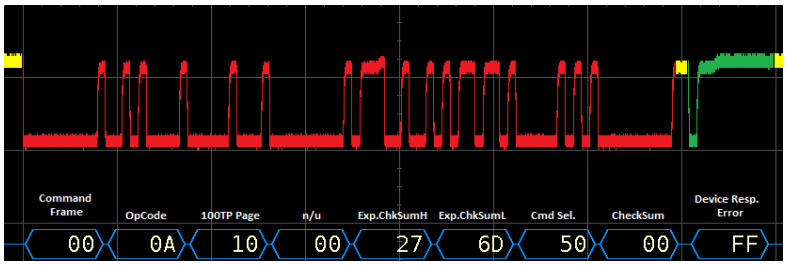

Figure 9 “100TP-Page Verify by Checksum” frame with wrong parameter, device response is 0xFF

The second communication example, **Figure 10**, shows a correct command frame, the device response with an Acknowledge (0x55). But the expected checksum as parameter did not match to the checksum internally calculated based on the page data. The first byte of the response is 0x80, telling that the calculated checksum doesn’t match the expected checksum. Bytes[2:1] of the response (here 0x27, 0x6D) are the calculated checksum. The last byte of the response is 0x00.

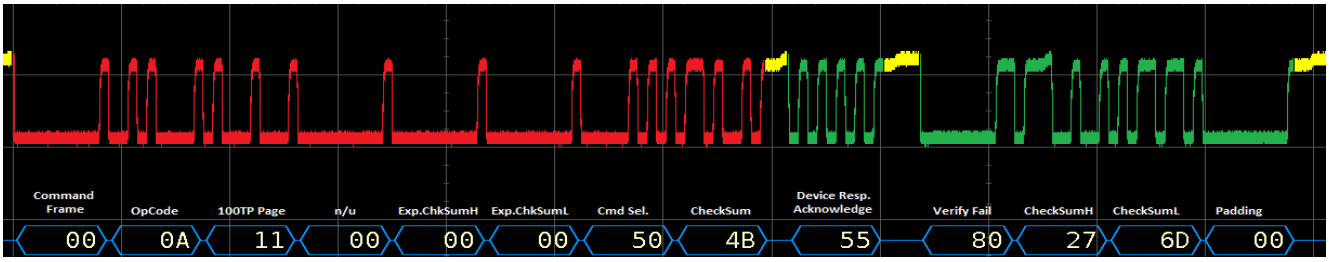

Figure 10 “NVM Page Verify by Checksum” correct frame with wrong expected checksum

6.6 Code Download to RAM

This command has to be used to transfere code from the host to the device internal RAM module.

		Parameters (5 bytes)					
00_H (Command Header)	**00_H** (Command OpCode)	**StartAddr [15:8]** (1 byte)	**StartAddr [7:0]** (1 byte)	**Block Length** (1 byte)	**Not used** (1 byte)	**Option $=00_H$** (1 byte)	**Checksum** (1 byte)

For the “Code Download to RAM” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x00. Since the OpCode 0x00 is grouping a selection of commands the 5th byte of the parameters specifies the “Code Download to RAM” command by sending a 0x00. The first two bytes of the parameters are specifying the start address inside the RAM module as offset to 0x18000000. The parameter “Block Length” (Byte2) defines the length of each following data frame, which transfers the payload of this command into the RAM.

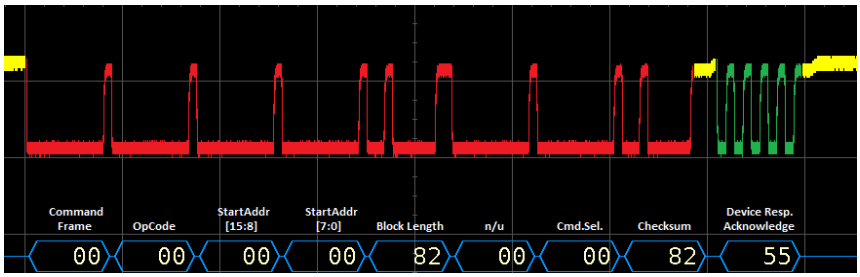

Figure 11 “Code Download to RAM” waveform including the device Acknowledge

Figure 11 shows an example waveform of the “Code Download to RAM” command, the offset to the RAM start address is set to 0x0000, so the code will be downloaded to address 0x18000000 + 0x0000. The “Block Length” is set to be 0x82 (130 decimal). This means each following data frame or EOT frame must have a frame length of 130 bytes. For a successive date frame it means the payload is 128 bytes. For a successive EOT frame it means the maximum payload can be 127 bytes.

The device is now waiting for data frames, by sending an EOT frame the command will be closed. In the case that less then 128 bytes have to be transferred the EOT frame can be sent immediately without data frame before.

6.6.1 Response

The device answers with an Acknowledge (0x55) if the command by itself was received correctly and is valid. The validity of the given address offset is not checked.

6.6.2 Example 1

In the following example the host wants to transfere three bytes of data (0xAA, 0xBB, 0xCC) to the RAM at address 0x18000000. In this sequence after the command frame a data frame with the first part of the data (0xAA, 0xBB) followed by a EOT frame with the rest of data (0xCC) is sent. Figure 12 shows the corresponding waveform for the communication sequence. The “Block Length” in the command frame is set to four, both the data frame as well as the EOT frame have a total length of four bytes each. Each frame is aswered by the device with an Acknowledge (0x55) in case the frame was correct, otherwise the device answeres with 0xFF. After the EOT frame the command “Code Download to RAM” is finished. Further data frames or EOT frames are not handled anymore.

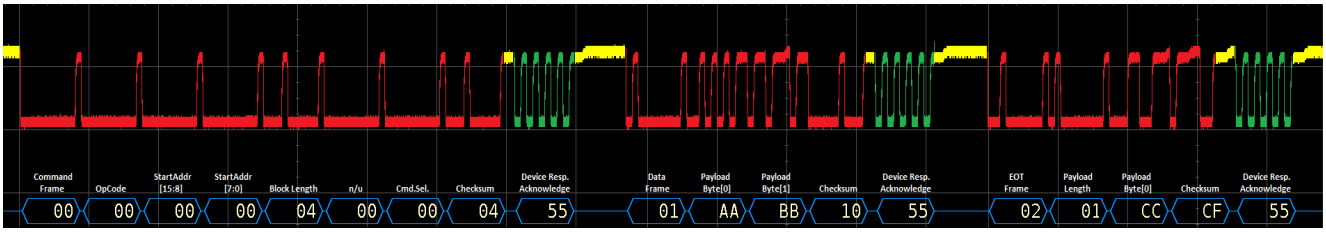

Figure 12 “Code Download to RAM” complete sequence, command frame, data frame and EOT frame

6.6.3 Example 2

This example targets the same as Example 1, three bytes of data (0xAA, 0xBB, 0xCC) needs to be transferred to RAM address 0x18000000. Due to the fact that the amount of data to be transferred is less than 128 bytes, this examples uses the EOT frame right after the command frame. The “Block Length” in the command frame has to be set to a value of six in order to transferre all three user data bytes within the EOT frame at once. After the EOT frame the command is complete.

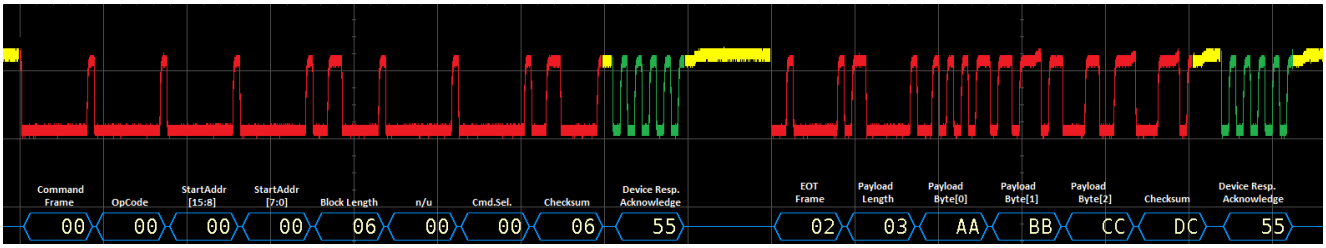

Figure 13 “Code Download to RAM” complete sequence, command frame and only an EOT frame

6.7 Code Download to NVM

This command has to be used to transfere code or data from the host to the device internal NVM module. With this command code/data can only be downloaded to the code NVM section or the data NVM section.

00_H (Command Header)	**02_H** (Command OpCode)	**Parameters** (5 bytes)					**Checksum** (1 byte)
		StartAddr [31:24] (1 byte)	**StartAddr [23:16]** (1 byte)	**StartAddr [15:8]** (1 byte)	**StartAddr [7:0]** (1 byte)	**Block Length** (1 byte)	

For the “Code Download to NVM” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x02. The first four bytes of the parameter are specifying the start address inside the NVM module. The start address has to be given page address aligned, the bits[6:0] are ‘0’. The parameter “Block Length” (Byte5) defines the length of each following data frame, which transfers the payload of this command into the NVM module. Since the smallest granularity of data to be written to the NVM module is 128 bytes (one page), the “Block Length” must be set to 130 (0x82), or if only one EOT frame is about to be sent then the “Block Length” has to be set to 131 (0x83). Any other value for the “Block Length” is not allowed.

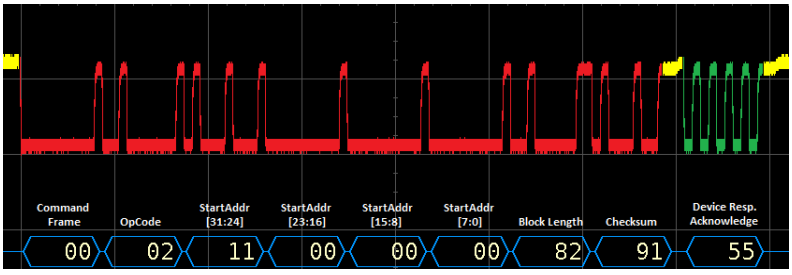

Figure 14 “Code Download to NVM” waveform including the device Acknowledge

6.7.1 Response

If the command frame was successfully received and is valid the device responds with an Acknowledge (0x55). If the address is specified to be out of range of the internal NVM module, the command frame does not report an error, it will sill reply an Acknowledge. The successive data frame is reporting the error message (0xFF) instead. Same for the “Block Length” parameter, it has to be set at 130 or 131, but the command is also excepting any other value and reports an Acknowledge. Again, the data frame afterwards reports an error message (0xFF) in case the data frame length is less then 130 bytes (EOT: 131 bytes). If one of the data frames is reporting an error message the command is aborted, following data frames or the EOT frame will not be handled anymore.

6.7.2 Example 1

The following example writes 128 bytes (one page) of data to the NVM address 0x11000000. The data written are house numbers: Byte0 = 0, Byte1 = 1, ... Byte127 = 127. The **Figure 15** displays the waveform of the example. It shows the command frame (red) followed by the device response (0x55). Then a data frame is being sent (header 0x01), with the 128 bytes of user payload (0x00, 0x01, ..., 0x7F) and the checksum. The device response (0x55) will be sent approx. 7ms after the data frame was received. This is the time needed to perform a NVM page erase and a programming of the new data to the same NVM page. In case the addressed NVM page was already erased, then the delay between the complete reception of the data frame and the device response will be approx. 4ms. The “Block Length” in the command frame is set to be 0x82 (130 bytes). The data frame as well as the EOT frame have to be 130 bytes long. **Figure 16** depicts the mantatory EOT frame which is required to close the command. Since there were only 128 bytes to be written, the entire user data was already being transferred within the data frame (**Figure 15**), the EOT frame actually does not transfere any user data, instead the “Remaining Payload Length” has to be programmed to 0x00 and the payload is padded with 0x00 to reach an EOT frame length of 130 bytes.

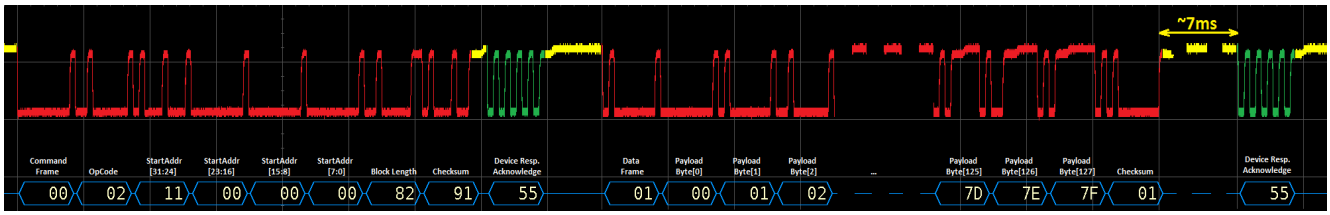

Figure 15 “Code Download to NVM” command and data frame with user payload

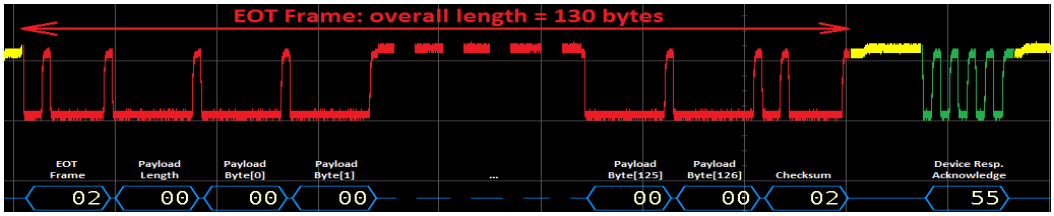

Figure 16 “Code Download to NVM” - mantatory EOT frame, padded with 0x00

6.7.3 Example 2

The second example in principle does the same as example 1. It writes 128 bytes (one page) of data to the NVM address 0x11000000. The data written are house numbers: Byte0 = 0, Byte1 = 1, ... Byte127 = 127. The **Figure 17** displays the waveform of the example. It shows the command frame (red) followed by the device response (0x55). But now only a EOT frame is being sent (header 0x02). The 0x80 as second byte of the EOT frame defines the length of the payload of 128 bytes. Then the 128 bytes of user payload (0x00, 0x01, ..., 0x7F) and the checksum follows. The device response (0x55) will be sent approx. 7ms after the data frame was received. This is the time needed to perform a NVM page erase and a programming of the new data to the same NVM page. In case the addressed NVM page was already erased, then the delay between the complete reception of the data frame and the device response will be approx. 4ms. The “Block Length” in the command frame is set to be 0x83 (131 bytes). The EOT frame has to be 131 bytes long.

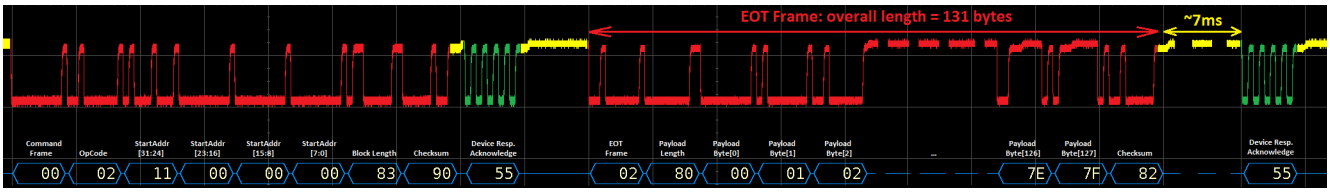

Figure 17 “Code Download to NVM” command and data frame with user payload

6.8 Data Download to 100TP Page

This command has to be used if data needs to be downloaded into the device internal 100TP pages.

00_H (Command Header)	**00_H** (Command OpCode)	**Parameters** (5 bytes)					**Checksum** (1 byte)
		0x04 (1 byte)	**0x00** (1 byte)	**Block Length** (1 byte)	**100TP Page** (1 byte)	**Option $=F0_H$** (1 byte)	

For the “Data Download to 100TP Page” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x00. Since the OpCode 0x00 is grouping a selection of commands the 5th byte of the parameters specifies the “Data Download to 100TP Page” command by sending a 0xF0. The first two bytes of the parameters are specifying the start address inside the RAM module as offset to 0x18000000, they have to be fix 0x0400. Any other value than 0x400 will lead to a command error resonse. The parameter “Block Length” (Byte2) defines the length of each following data frame, which transfers the payload of this command into the RAM. Once the 100TP page update content was transferred into the device internal RAM, the addressed 100TP page is programmed with the given data. The third byte of the paramteres define the 100TP page to be updated. There are eight 100TP pages available, which are addressed starting from 0x11 to 0x18, where 0x11 adresses 100TP-Page0 and 0x18 addresses 100TP-Page7. Any other value than 0x11..0x18 will lead to a command error response.

Table 5 Payload preloading for 100TP page programming

Payload Byte index	Function
00H	Number of Bytes to be programmed (i.e. N, up to a maximum of 127[1)] Bytes)
01H	100TP offset 0
02H	100TP data 0 to be programmed
03H	100TP offset 1
04H	100TP data 1to be programmed
.....	....
01H + ((N-1) x 2)	100TP offset N-1
02H + ((N-1) x 2)	100TP data N-1 to be programmed

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

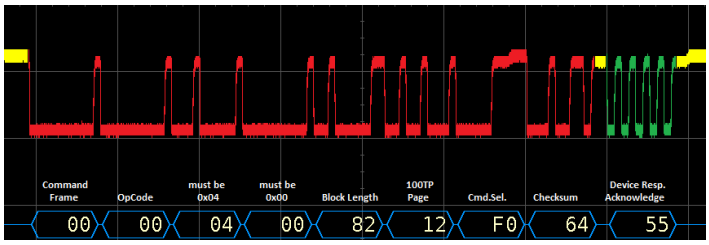

Figure 18 “Data Download to 100TP Page” waveform with device response

6.8.1 Response

If the command was received proverly and if it is valid then the device response with an Acknowledge (0x55).

6.8.2 Example 1

The first example wants to write three values into 100TP page 1, the corresponding 100TP page select would be 0x12 given in Byte[5] of the command. The desired 100TP page date to be written is the following:

Table 6 Example data for 100TP page 1

Offset inside 100TP Page 1	Data values to be written
0x00	0xAA
0x01	0xBB
0x02	0xCC

Figure 19 and **Figure 20** show the complete communication starting with the command frame, followed by the data frame which transferres the actual 100TP data to the device, the command is closed by sending the EOT frame. The “Block Length” (command Byte[4]) is set to 0x09, both the data frame and the mandatory EOT frame are in total nine bytes long each. The complete user data is being transferred inside the data frame. The EOT frame is still needed to close the command, the payload of the EOT frame therefore is padded with 0x00 to reach the “block length” of nine bytes as defined in the command frame. The response of the device after the EOT frame is delayed by approx. 7ms, in this time the device performs the programming of the 100TP page.

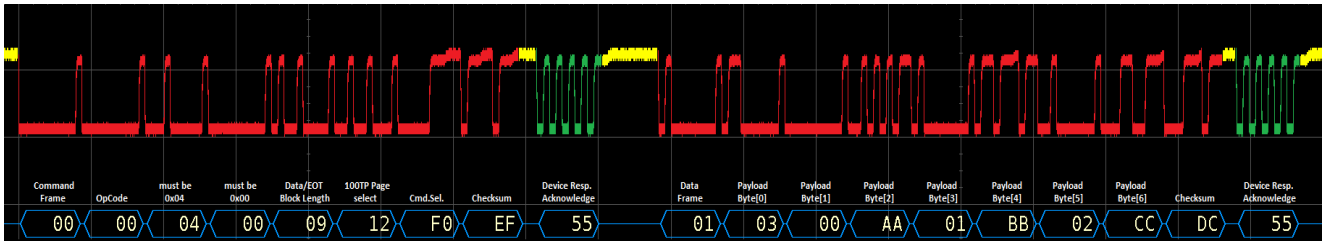

Figure 19 “Data Download to 100TP Page” waveform, command frame and data frame

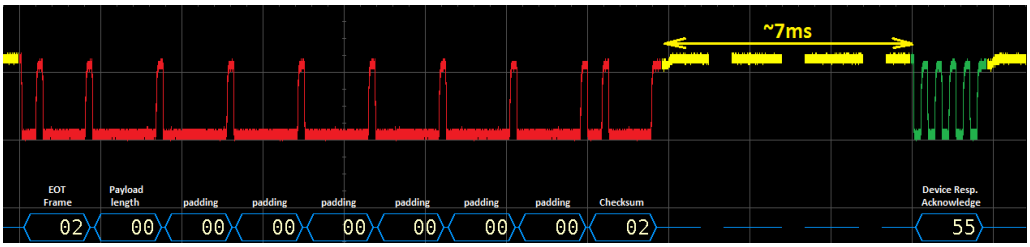

Figure 20 “Data Download to 100TP Page” waveform, EOT frame

6.8.3 Example 2

Example 2 is transferring the same user data into the same 100TP page as described in Example 1. Example 2 is transferring all the user data directly inside the EOT frame, a separate data frame is not needed in this case. The response of the device after the EOT frame is delayed by approx. 7ms, in this time the device performs the programming of the 100TP page.

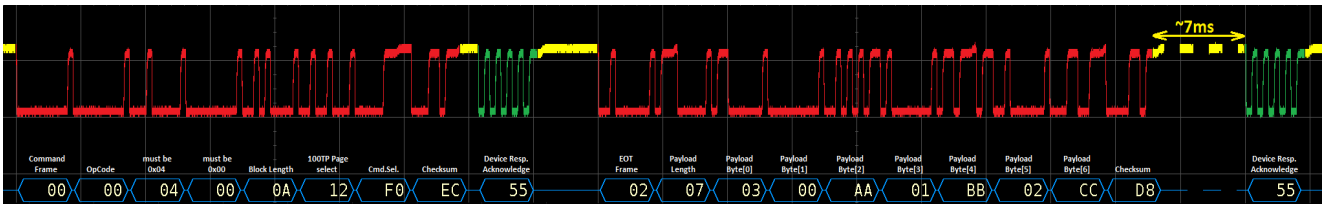

Figure 21 “Data Download to 100TP Page” waveform, command frame and EOT frame

6.9 Execute Code from RAM

This command can be used if code downloaded into the device internal RAM module (see **Chapter 6.6**) shall be executed. By calling this command the instruction pointer is set to 0x18000404 (expected user reset vector). The VTOR register will be initialized to 0x18000400 (start of vector table). The BSL mode is being exited by executing this command.

00_H (Command Header)	**01_H** (Command OpCode)	**Parameters** (5 bytes)					**Checksum** (1 byte)
		Not used (1 byte)	**Not used** (1 byte)	**Not used** (1 byte)	**Not used** (1 byte)	**Not used** (1 byte)	

The command does not need any parameters, the parameter bytes have to by filled up, i.e. with 0x00.

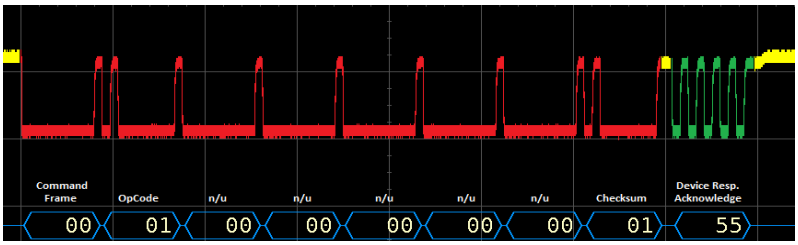

Figure 22 "Execute Code from RAM" waveform with device response

6.9.1 Response

If the command was received proverly and if it is valid then the device response with an Acknowledge (0x55). The device does not perform any checks whether there is a valid code present inside the device RAM. The user shall transferre any valid code before using the command described in **Chapter 6.6**.

6.10 Execute Code from NVM

This command can be used if code downloaded into the device internal NVM module (see **Chapter 6.7**) shall be executed. By calling this command the instruction pointer is set to 0x11000004 (expected user reset vector). The VTOR register will be initialized to 0x11000000 (start of vector table). The BSL mode is being exited by executing this command.

00_H (Command Header)	**03_H** (Command OpCode)	**Parameters** (5 bytes)					**Checksum** (1 byte)
		Not used (1 byte)	**Not used** (1 byte)	**Not used** (1 byte)	**Not used (1 byte)**	**Not used** (1 byte)	

The command does not need any parameters, the parameter bytes have to by filled up, i.e. with 0x00.

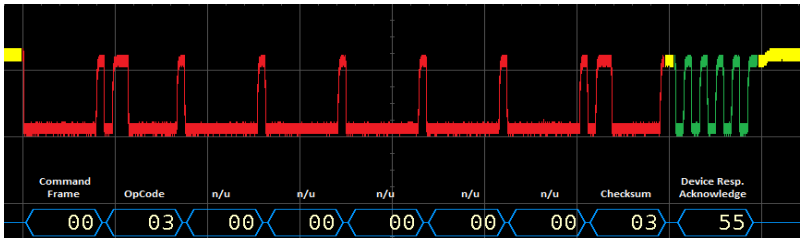

Figure 23 "Execute Code from NVM" waveform with device response

6.10.1 Response

If the command was received proverly and if it is valid then the device response with an Acknowledge (0x55). The device does not perform any checks whether there is a valid code present inside the device RAM. The user shall transferre any valid code before using the command described in **Chapter 6.7**.

6.11 Erasing a NVM Page

This command has to be used if a page (128bytes) within the device NVM module has to be erased.

00_H (Command Header)	**04_H** (Command OpCode)	**Paramerters** (5 bytes)					**Checksum** (1 byte)
		StartAddr [31:24] (1 byte)	**StartAddr [23:16]** (1 byte)	**StartAddr [15:8]** (1 byte)	**StartAddr [7:0]** (1 byte)	**Option $=00_H$** (1 byte)	

For the “Erasing a NVM Page” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x04. Since the OpCode 0x04 is grouping a selection of commands the 5th byte of the parameters specifies the “Erasing a NVM Page” command by sending a 0x00. The first four bytes of the parameters are linear address to the NVM page to be erased. The given address should be page aligned, this means the StartAddr[6:0] are set to zero.

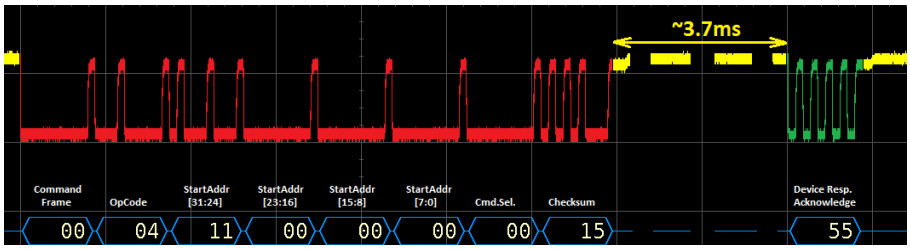

Figure 24 “Erasing a NVM Page” waveform with device response

6.11.1 Response

If the command was received proverly and if it is valid then the device will immediately perform the page erasing function. The positive response with an Acknowledge (0x55) will be sent approx. 3.7ms later. The device checks whether the given device address is a valid NVM page address, otherwise the device responds with an block error (0xFF), see **Figure 25**.

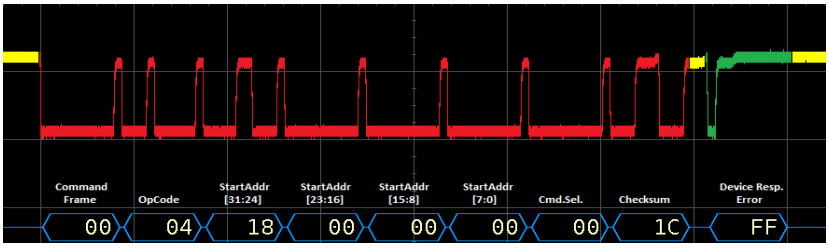

Figure 25 “Erasing a NVM Page” waveform with error response

6.12 Erasing a NVM Sector

This command has to be used if a entire sector (32 pages á 128bytes = 4KB) within the device NVM module has to be erased.

		Paramerters (5 bytes)					
00_H (Command Header)	**04_H** (Command OpCode)	**StartAddr [31:24]** (1 byte)	**StartAddr [23:16]** (1 byte)	**StartAddr [15:8]** (1 byte)	**StartAddr [7:0]** (1 byte)	**Option $=40_H$** (1 byte)	**Checksum** (1 byte)

For the “Erasing a NVM Page” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x04. Since the OpCode 0x04 is grouping a selection of commands the 5th byte of the parameters specifies the “Erasing a NVM Sector” command by sending a 0x40. The first four bytes of the parameters are linear address to the NVM page to be erased. The given address should be sector aligned, this means the StartAddr[11:0] are set to zero.

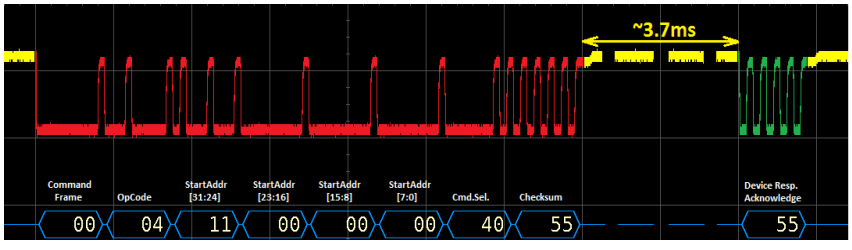

Figure 26 “Erasing a NVM Sector” waveform with device response

6.12.1 Response

If the command was received properly and if it is valid then the device will immediately perform the sector erasing function. The positive response, an Acknowledge (0x55), will be sent approx. 3.7ms later. The device checks whether the given device address is a valid NVM sector address, otherwise the device responds with a block error (0xFF), see **Figure 27**.

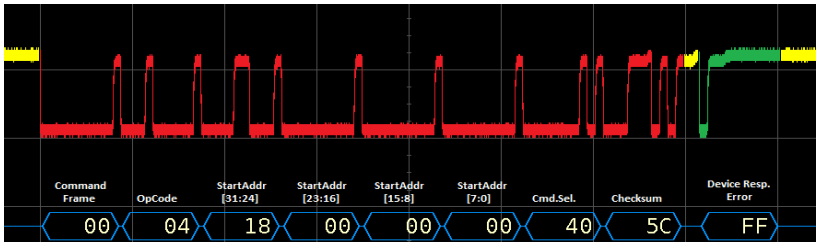

Figure 27 "Erasing a NVM Sector" waveform with error response

6.13 Erasing the full Chip NVM

This command has to be used if the entire code flash and data flash inside the device has to be erased. The content of the 100TP pages is not affected by this command. The NAC parameter is also being erased by this command. After a reset the device is entering the BSL mode (FastLIN protocol) and waits there for a communication attempt.

		Parameters (5 bytes)					
00_H (Command Header)	**04_H** (Command OpCode)	**Not used** (1 byte)	**Not used** (1 byte)	**Not used** (1 byte)	**Not used** (1 byte)	**Option $=C0_H$** (1 byte)	**Checksum** (1 byte)

For the "Erasing the full Chip NVM" command the Header Type is a "Command Frame" (0x00), the OpCode is 0x04. Since the OpCode 0x04 is grouping a selection of commands the 5th byte of the parameters specifies the "Erasing the full Chip NVM" command by sending a 0xC0. The first four bytes of the parameters are not used.

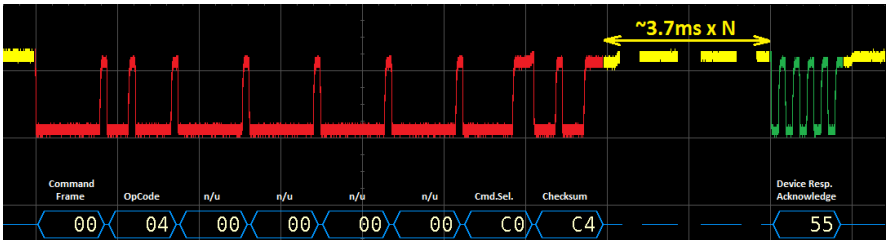

Figure 28 "Erasing the full Chip NVM" waveform with device response, "N" is the number of NVM sectors inside the device.

6.13.1 Response

If the command was received properly and if it is valid then the device will immediately perform the full chip NVM erasing function. The positive response, an Acknowledge (0x55), will be sent approx. 3.7ms times the number of NVM sectors later, i.e. for a 64KB device the delay is approx. 59ms.

6.14 Readout a NVM Page

This command can be used to readout the data stored inside a NVM page. The data of the entire page is being sent to the host.

00_H (Command Header)	$0A_H$ (Command OpCode)	Parameters (5 bytes)					Checksum (1 byte)
		StartAddr [22:15] (1 byte)	StartAddr [14:7] (1 byte)	Not Used (1 byte)	Not Used (1 byte)	Option $=C0_H$ (1 byte)	

For the “Readout a NVM Page” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x0A. Since the OpCode 0x0A is grouping a selection of commands the 5th byte of the parameters specifies the “Readout a NVM Page” command by sending a 0xC0. The first two bytes of the parameters define the start address of the NVM page as an offset to the NVM start address of 0x11000000. The parameter Bytes[3:2] are not used.

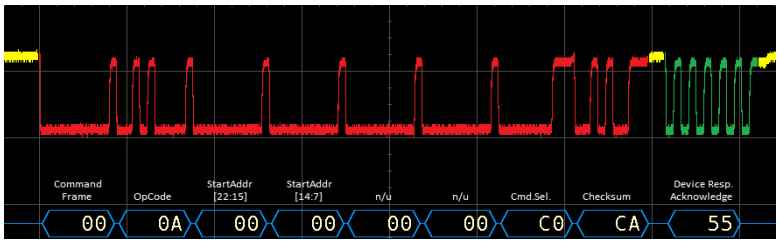

Figure 29 “Readout a NVM Page” waveform with device response

6.14.1 Response

If the command was received properly and if it is valid then the device will respond with an Acknowledge (0x55), and immediately starts sending the 128 data bytes out of the adressed NVM page. **Figure 30** shows the device response after a successfull reception of the command as dispayed in **Figure 29**. The data transferred back is what was written into the first NVM page in the example described in **Chapter 6.7.3**. There is no checksum sent along with the data.

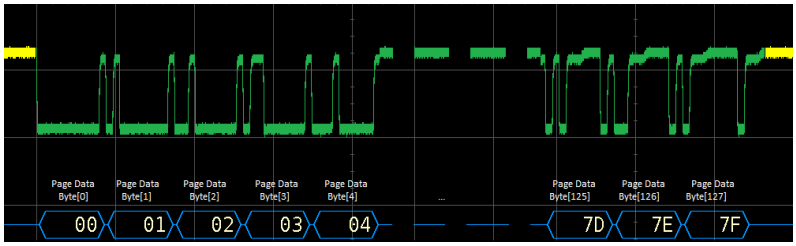

Figure 30 “Readout a NVM Page” waveform device response, 128 bytes of page data

6.15 Readout a 100TP Page

This command can be used to readout the data stored inside a 100TP page. The data of the entire page is being sent to the host.

00_H (Command Header)	$0A_H$ (Command OpCode)	Parameters (5 bytes)					Checksum (1 byte)
		Not Used (1 byte)	Not Used (1 byte)	Not Used (1 byte)	100TP Page (1 byte)	Option $=F0_H$ (1 byte)	

For the “Readout a 100TP Page” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x0A. Since the OpCode 0x0A is grouping a selection of commands the 5th byte of the parameters specifies the “Readout a 100TP Page” command by sending a 0xF0. The first three bytes of the parameters are not used and have to be padded, i.e. with 0x00. The parameter Bytes[3] defines the 100TP page to be read. There are eight

100TP pages available, which are addressed starting from 0x11 to 0x18, where 0x11 adresses 100TP-Page0 and 0x18 addresses 100TP-Page7. Any other value than 0x11..0x18 will lead to a command error response

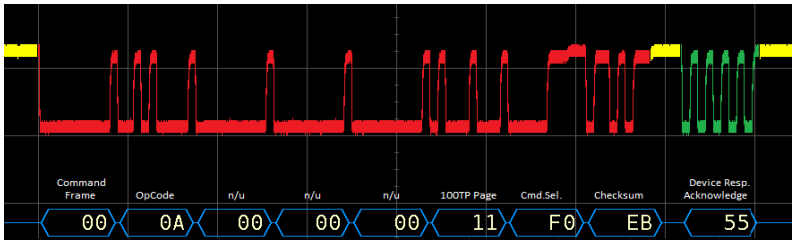

Figure 31 “Readout a 100TP Page” waveform with device response

6.15.1 Response

If the command was received properly and if it is valid then the device will respond with an Acknowledge (0x55), and immediately starts sending the 128 data bytes out of the adressed 100TP page. **Figure 32** shows the device response after a successfull reception of the command as dispayed in **Figure 31**. There is no checksum sent along with the data.

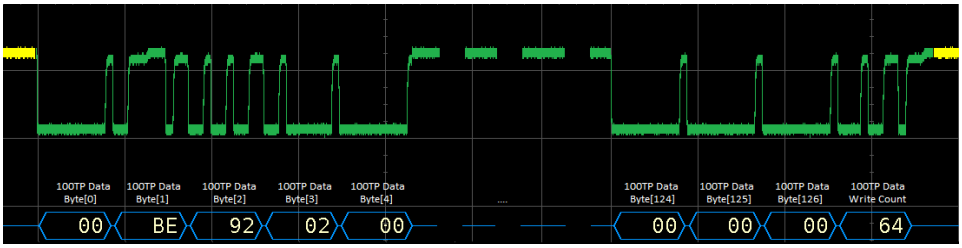

Figure 32 “Readout a 100TPPage” waveform device response, 128 bytes of page data

6.16 Set/Reset NVM Protection

This command can be used to set or reset the NVM protection. In case the NVM protection is not set, then this command sets the protection. In case the protection is set, then it resets the protection.

00_H (Command Header)	**06_H** (Command OpCode)	**Parameters (5 bytes)**					**Checksum** (1 byte)
		User-password (1 byte)	**Not Used** (1 byte)	**Not Used** (1 byte)	**Not Used** (1 byte)	**Not Used** (1 byte)	

For the “Set/Reset NVM Protection” command the Header Type is a “Command Frame” (0x00), the OpCode is 0x06. The first byte of the parameters is the user definable password. It is an 8bit value with a special meaning for Bit7.

- Bit7 = 0: only Code NVM region is erased upon protection reset
- Bit7 = 1: Code NVM region and Data NVM region is erased upon protection reset

The same command is used for both setting as well as resetting the portection. If the device is unprotected by executing this command the user defined password is stored inside the device and the protection is activated. The device will not reply to this command. The device will not reply to any other BSL command. A reset cycle has to be performed.

If the device was already protected then the user password given in the command will be compared to the password stored inside the device, if the password matches a mass erase will be executed. Which NVM regions are erased is defined by Bit7 of the given password. The device has to be power cycled in order to communicate to it.

If the password in the command does not match to the password stored inside the device, the protection removal will not be performed instead the device replies with an error code.

The NVM protection sets the write protection as well as a read protection. All BSL commands which are modifying the NVM content are blocked and return an error code instead. Furthermore all BSL commands which are reading content from the NVM are blocked and return an error code instead. Even the code download into the RAM is prohibited once the protection is set.

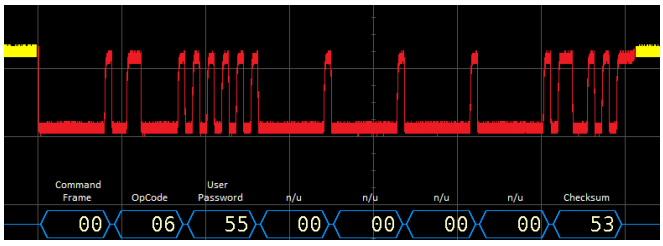

Figure 33 “Protection Set/Reset”, no device response means success

6.16.1 Response

If the command was received properly and if it is valid then the device will not send any response. Instead if a wrong password was given the device will return an error code if the protection was enabled already.

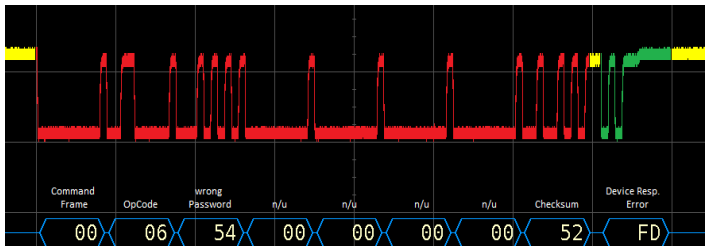

Figure 34 “Protection Reset” waveform device response, wrong password sent to a protected device

7 Revision History

Revision	Date	Changes
1.00	2016-12-05	initial version

Edition 2016-12-05

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

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

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

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