Never stop thinkin

Data Flash Handling

Application Note for TLE987x

BE-/BF-Step

Application Note

Rev. 1.1, 2019-03-20

Automotive Power

Table of Contents

1 Abstract . . . 1

2 Data Flash Page Internals . . . 1
2.1 Data Flash Cell . . . 1
2.2 Flash Cell Effects . . . 3
2.2.1 Cycling Effect . . . 3
2.2.2 Retention Loss . . . 4
2.2.3 Relaxation . . . 5
2.3 Read Logic Effects . . . 6
2.3.1 Read Voltage Drift . . . 6
2.3.2 Frequency Jump . . . 6
2.4 Flash Cell Summary . . . 7

3 Data Flash Handling in Hardware and Firmware . . . 8
3.1 Data Flash Concept . . . 8
3.2 Data Flash Initialization . . . 9
3.2.1 Random Number Generator . . . 9
3.2.2 Spare Page Pointer Selection . . . 9
3.2.3 MapRAM Init . . . 10
3.2.3.1 MapRAM Init Aborts . . . 11
3.2.4 Service Algorithm . . . 13
3.2.4.1 Service Algorithm Limits . . . 14
3.2.4.2 Service Algorithm Summary . . . 15

4 User Application Considerations . . . 16
4.1 After USER MODE Entry . . . 16
4.2 USER_MAPRAM_INIT function . . . 17
4.3 USER MODE Entry after Soft Reset . . . 18
4.4 Program Page Flow . . . 18
4.4.1 Voltage and Temperature Check . . . 20
4.4.2 MEMSTAT . . . 20
4.4.3 Interrupts . . . 20
4.4.4 Watchdog Handling . . . 20
4.4.5 OpenAB . . . 20
4.4.6 Filling Data into Assembly Buffer . . . 21
4.4.7 USER_PROG . . . 21
4.4.8 Program Page Flow return handling . . . 22

5 Summary . . . 24

6 Additional Information . . . 25

7 Revision History . . . 1

1 Abstract

This application note is a hint to the user of the TLE987x BE-/BF-Step device family and provides further information about the data flash handling. If a flash operation like write or erase does not complete properly, then a loss of data integrity in the data flash is possible. A Service Algorithm designed to recover from such events runs during device start-up. The Service Algorithm acts as an exception handler which can recover from many, but not all data integrity issues. Therefore, certain precautions should be taken by the user application to properly handle the data flash in order to reduce the risk of loss of data due to data integrity issues. This application note describes the physical effects of a data flash cell which influences the repair capability of the Service Algorithm. Furthermore, it describes the data flash handling process during start-up. Finally it provides user application recommendations to ensure correct handling of the data flash operations.

2 Data Flash Page Internals

A data flash page consists of 128 bytes user data, secured by eight ECC bits per 8 user data bytes. Furthermore, the flash page contains control information about the mapping of this page, the MapBlock. The MapBlock contains the link to the logical page in true and 1's-complement, control bits for the Disturb Handling, the Page Counter and a ECC word (6 bits) to secure the MapBlock information. In total a data flash page holds 1.218 bits, control bits, user data bits and ECC bits. If a flash page gets erased or programmed all 1.218 bits are handled at once. It is not possible to erase individual bits separately.

2.1 Data Flash Cell

Each bit of the data flash page is stored inside a data flash cell. The logical value of a flash cell is represented by the charge of the floating gate of the flash cell. By erasing a flash cell, charge will be removed. By programming a flash cell charge will be transferred onto the flash cell. The process of charging or discarging a flash cell takes several milliseconds. If the process of charging or discharging a flash cell is interrupted, the charge and resulting read voltage stays at the level where it was interrupted. This means that due to the fact that a flash cell during programming or erasing passes through the entire range of charge a cell can represent, if an erase or write process gets interrupted the flash cell can take any intermediate state in terms of cell charge. **Figure 1** displays the distribution of the effective read voltage of a flash cell based on the cell charge, and the corresponding logical interpretation.

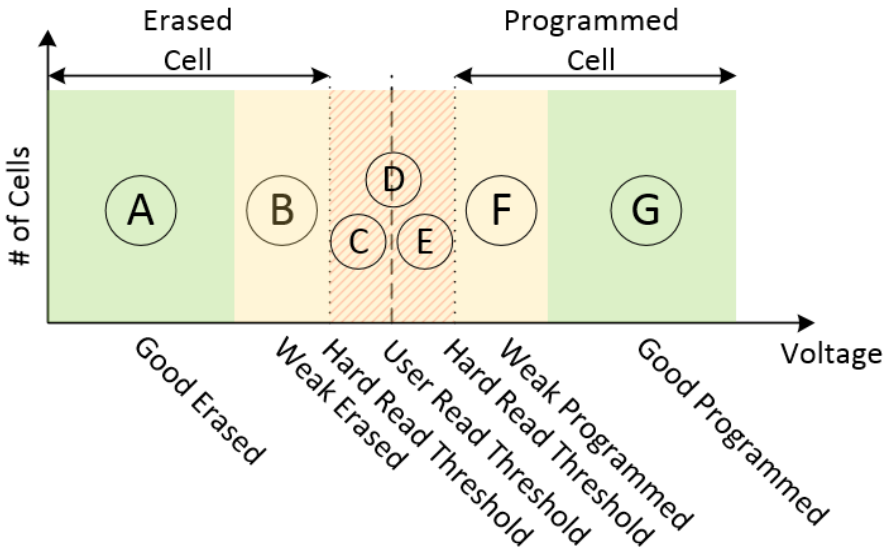

Figure 1 Voltage distribution of a flash cell

Figure 2 provides a zoomed in view of the voltage distribution around the User Read Threshold to make it able to recognize the difference between the Hard Read Thresholds and the Soft Read Thresholds.

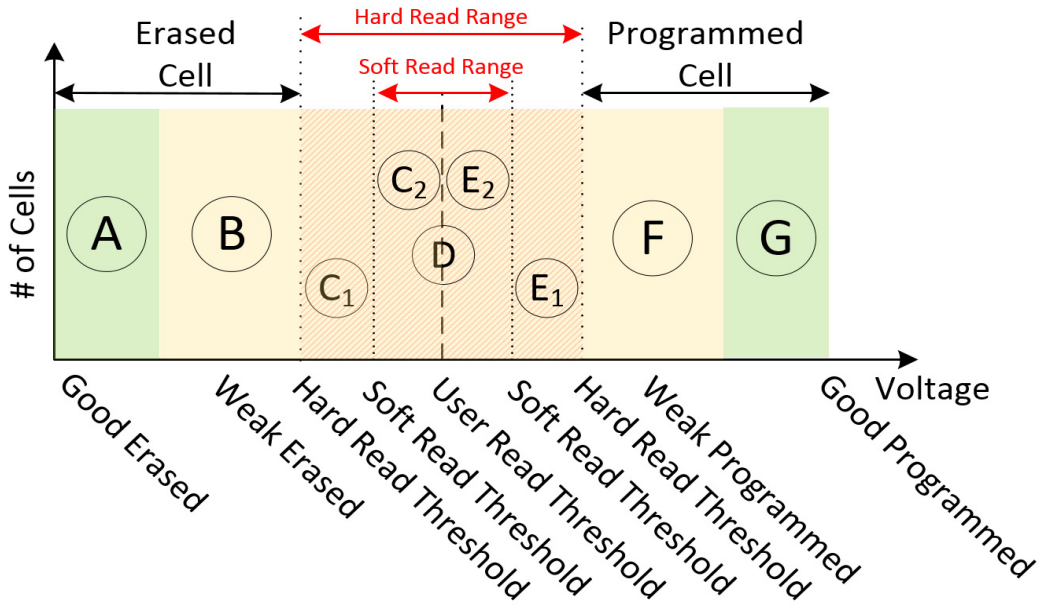

Figure 2 Voltage distribution of a flash cell, zoomed in to differentiate between Hard Read Thresholds and Soft Read Thresholds

In order to determine the logical value of a flash cell, the flash cell is read with a certain read voltage. In the normal case, the voltage of a flash cell is compared against the “User Read Threshold”. If the voltage of a flash cell is below the User Read Threshold, then the cell is interpreted as logical ‘1’ (erased). If the voltage of a flash cell is above the User Read Threshold, then the cell is interpreted as logical ‘0’ (programmed). Extended read margings, employing the Hard Read Thresholds, are used to verify the cell voltage, e.g. during the USER_PROG function, or during the Service Algorithm execution.

While a flash cell gets erased or programmed it passes through the voltage ranges, and for each instance the cell voltage represents a certain logical state of the flash cell, which are (see **Figure 1**):

- Section A: the flash cell is in excellent erased state, it has plenty of margin to the User Read Threshold
- Section B: the flash cell is in weakly erased state, it has reduced margin to the User Read Threshold
- Section C: the flash cell is in marginaly erased state, it is already above the Hard Read Thresholds
- Section D: the flash cell is in faulty state, it more or less does not have any margin to the User Read Threshold
- Section E: the flash cell is in marginaly programmed state, it is still below the Hard Read Threshold
- Section F: the flash cell is in weakly programmed state, it has reduced margin to the User Read Threshold
- Section G: the flash cell is in excellent programmed state, it has plenty of margin to the User Read Threshold

If a flash cell gets programmed, it walks all the way through the sections from section A to section G. If a flash cell gets erased, it walks all the way through from section G back to section A. For programming a flash page, only those cells to be set to logical ‘0’ are moved from section A to section G, see **Figure 3**.

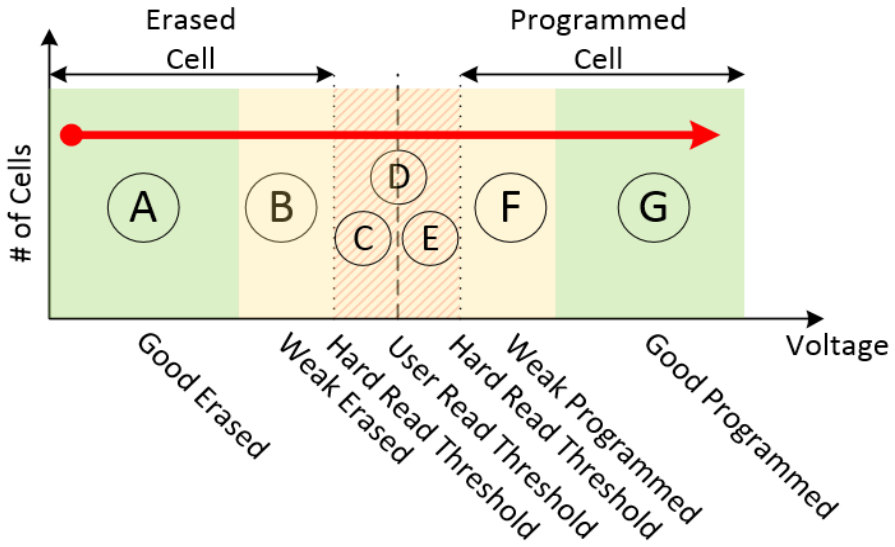

Figure 3 Voltage distribution of a flash cell, page programming

For erasing a flash page, to avoid overerase, all cells which still represent the logical value ‘1’ (erased) have to be moved to section G first, then all cells together are moved back to section A, see **Figure 4**.

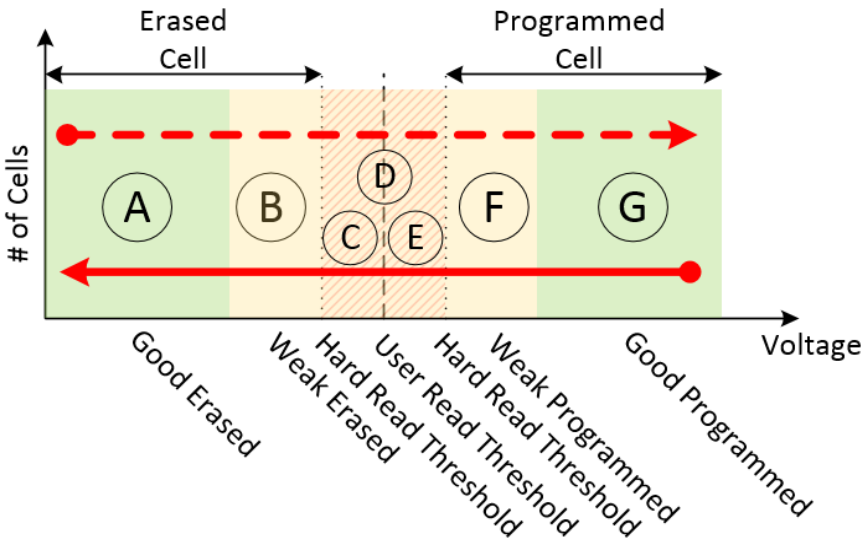

Figure 4 Voltage distribution of a flash cell, page erasing

2.2 Flash Cell Effects

Certain well known physical effects are influencing the behavior of a flash cell. The following section will briefly describe those effects.

2.2.1 Cycling Effect

By cycling it is meant to program a flash cell and later erase the same flash cell again. The time between the programming and erasing of the flash cell is not of interest for this effect. With every erase process, some charge gets trapped inside the cell, which lets the state of that cell slightly move towards the programmed state. When this cell has trapped so much charge that it cannot be discharged below the User Read Threshold anymore, this cell will then be faulty permanently. This effect only applies to erased cells. This is a permanent effect.

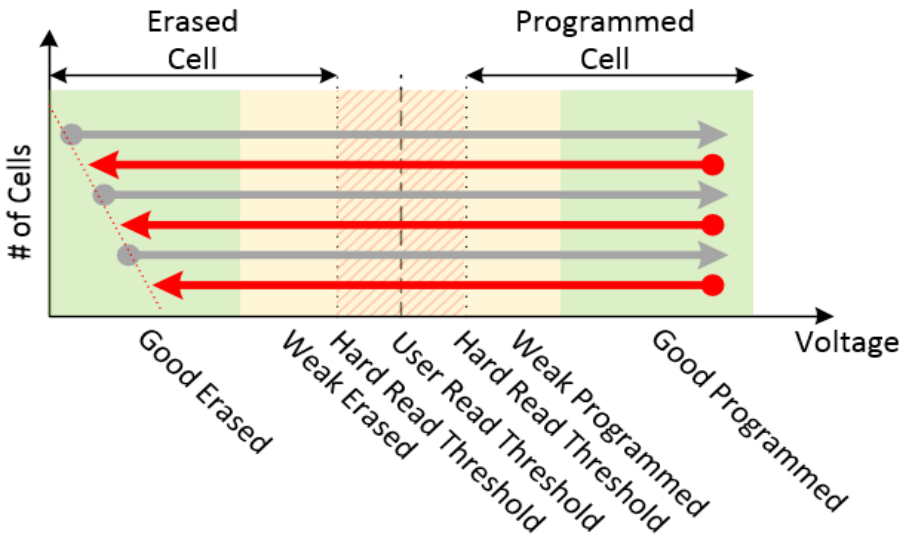

Figure 5 Cycling Effect - drift of the erased cell voltage based on write/erase cycles

2.2.2 Retention Loss

By retention loss the charge loss of a programmed flash cell over time is meant. Due to unavoidable leakage of a flash cell, the charge will get lost over time. While this effect occurs in µs time frames for DRAMs, for flash cells this effect operates on time frames of years. But nevertheless, a programmed flash cell loses charge over time and moves towards the erased state. When this cell has lost so much charge that it does not have enough margin to the User Read Threshold it cannot be recognized as programmed anymore. By erasing and reprogramming the cell can be fully charged again. The effect only applies to programmed cells. It is a non-permanent effect.

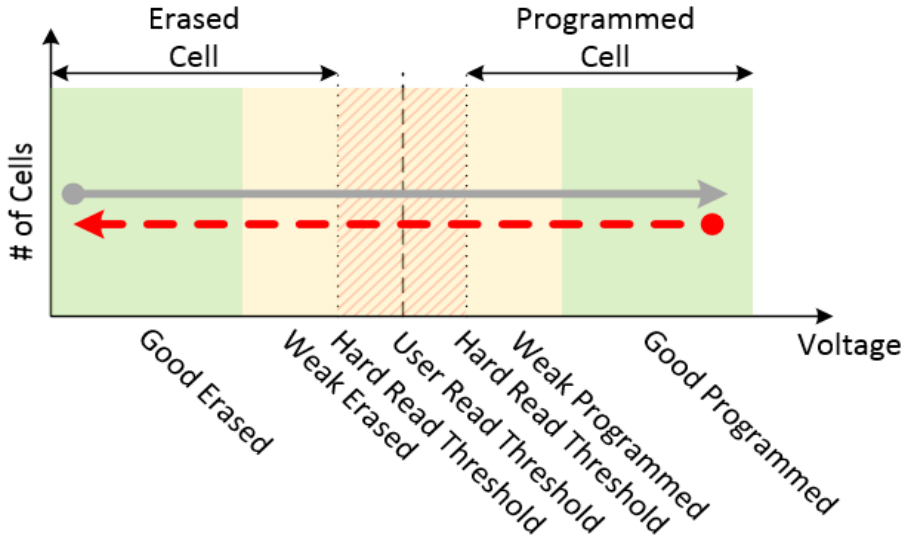

Figure 6 Retention Loss - moving of a programmed cell towards erased state over time

2.2.3 Relaxation

Relaxation is a similar effect as the retention loss, the cell loses some charge. This effect occurs within a few milliseconds after the programming of a flash cell. It immediately starts once the programming operation has been stopped. The cell loses a little amount of charge, but contrary to the Retention Loss, for the relaxation effect the loss of charge stops after a few milliseconds. The cells are relaxing a bit. This relaxing effect also happens if the device gets powered down immediately after the programming has stopped, or even if the programming has been stopped due to a power down of the device. The relaxation effects moves programmed cells slightly towards the erased state for a short period of time. For erased cells this effect applies as well. Here the cells do drift slightly towards the programmed state for a few milliseconds right after the erase process has been finished. This is a temporary effect.

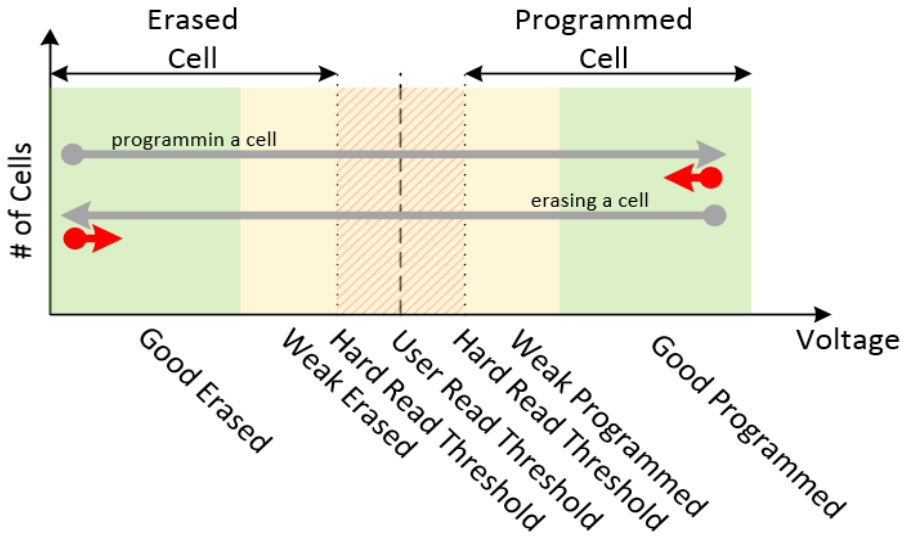

Figure 7 Relaxation - minimal moving of a programmed or erased cell towards the opposit state

2.3 Read Logic Effects

The read logic is interpreting the cell state by evaluation of the cell current, a function of the cell charge, for a given read voltage. By this the logical value of the flash cell gets generated. The read logic also is affected by some physical effects.

2.3.1 Read Voltage Drift

The read voltage is an analog voltage that is chosen according to the user read condition or the hard read condition. The analog read voltage will vary slightly over time and temperature. This influences the margin of a flash cell during read. This effect applies to erased cells, as well as to programmed cells. This is a non-deterministic always present effect.

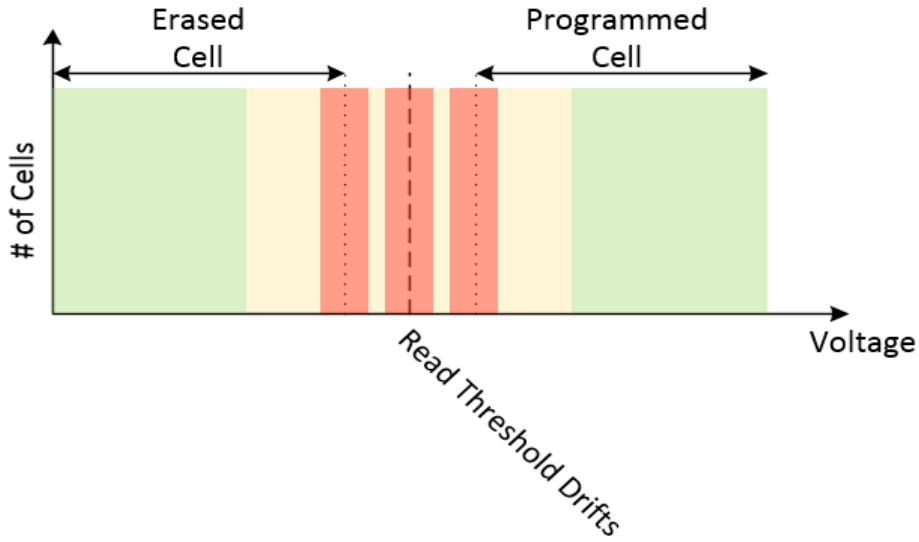

Figure 8 Read Voltage Drifts - Variation in analog camparison voltage

2.3.2 Frequency Jump

Depending on the NVM access clock frequency the cell voltage applied to the read comparator appears slightly shifted. The higher the readout frequency is the higher the cell voltage shift is. This virtual shift in cell voltage is only present or noticeable if the same cell is read twice, once with lower NVM access frequency, and again with higher NVM access frequency. In this case the cell slighly shifts towards programmed state. If the cell is read first with higher NVM access clock and secondly with lower NVM access clock then the cell apparently shifts slightly towards erased state. This applies to erased cells as well as to programmed cells. This effect is not present if the NVM access clock is not changed.

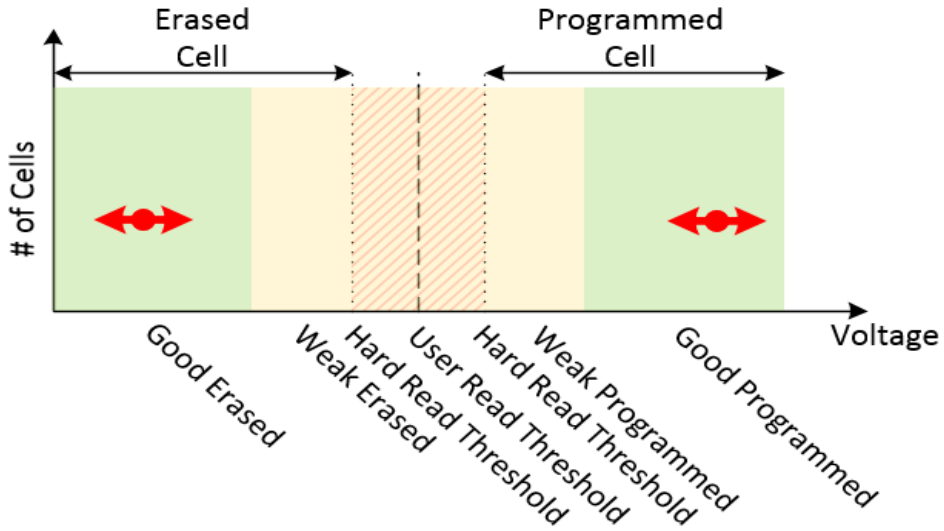

Figure 9 Frequency Shift - Virtual variation if the cell voltage based on NVM access clock frequency

2.4 Flash Cell Summary

A flash cell is an analog storage element. It is affected by various physical effects which are more or less effective under various circumstances. Furthermore, the interpretation of a flash cell done by the read out logic is providing its own effects and is influencing the flash content in another way. Overall, in order to minimize these physical effects it is required to always let the NVM module finish a NVM operation, like Erase or Program, and not to interrupt Erase or Program operations before they have finished. A good erased cell, or a good programmed cell provides enough margin to the User Read Theshold that all the effects mention in the **Chapter 2.2** and the **Chapter 2.3** are negligible.

3 Data Flash Handling in Hardware and Firmware

The Data Flash as a storage container for user data which is supposed to be changed frequently needs some special handling in order to maintain the data integrity. The Data Flash is handled in bootrom during the start up of the device as well as during the execution of the user code. The specifics of these handling are described in the chapters below.

3.1 Data Flash Concept

The TLE987x family provides 4KB, corresponding to one sector, of emulated EEPROM to the user. The EEPROM emulation is achieved by implementing an address translation between logical data flash address and physical data flash address. The logical address is given by the user application and used to access the data flash module by the microcontroller core. An address translation table, the so called MapRAM, is used to translate the flash page address from logical address to physical address. The physical address is then being used to access the physical data flash module itself. **Figure 10** displays the address translation scheme.

The MapRAM is used to establish a link between the logical address and the physical address. This link can be dynamically modified by the USER_PROG function.

Within the data flash sector 32 flash pages of 128 Bytes each are accessible by the user application. Physically the data flash sector provides 33 pages. One page cannot be accessed directly by the user application. One page more is implemented to always have one empty page left to store the next data set by the user application. This one additional physical flash page is addressable by a special MapRAM entry, the so called Spare Page Pointer.

The Spare Page Pointer always points to an empty page which will be used next for storing the new application data being written by the next USER_PROG function call. At the end of the USER_PROG function the link between the logical address and the newly programmed physical page is established. Any former page which was linked to the same logical address will be erased, and the link to the logical page will be removed by the erase process. In order to be prepared for the next USER_PROG function call the Spare Page Pointer has to point to a new empty page.

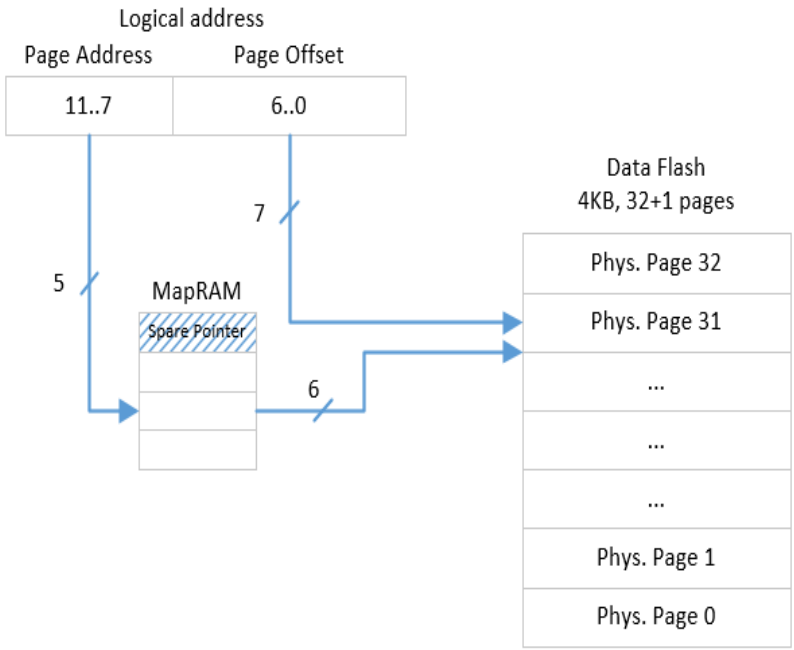

Figure 10 Address translation scheme for the Data Flash

With this method the user can operate the Data Flash logically as a normal flash page, while in the background by using the address translation done by the MapRAM the write accesses are distributed all over the entire data

flash sector. The mapping of the physical data flash pages to the logical data flash pages used by the user application is absolutely transparent to the user.

3.2 Data Flash Initialization

In order to access the data flash from a user application the address translation table, MapRAM, has to be initialized. The process of the MapRAM initialization performed during startup is starting from a random position within the MapRAM, how these random numbers are generated is described in the following chapter. During the MapRAM initialization the Spare Page pointer is selected as well, pointing to the next physical page used for write operations to the data flash.

3.2.1 Random Number Generator

In order to provide a random start for the Spare Page Pointer selection a random number generator is implemented inside the device. The Random Number Generator is realized as Logical Feedback Shift Register (LFSR). The principle of such a LFSR can be seen in **Figure 11**. From its functional principle a LFSR is also called a pseudo random number generator, because the random sequence generated by an LFSR is deterministic and therefore not purely random.

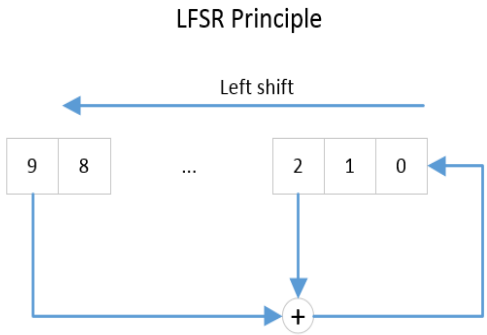

Figure 11 The principle of a Logical Feedback Shift Register - LSFR

The LFSR implemented in the TLE987x as pseudo random number generator is a 10 bit wide LFSR.

3.2.2 Spare Page Pointer Selection

The Spare Page Pointer selection function is implemented as a hardware function inside the NVM module, the MapRAM Init function. As initial value for the Spare Page Pointer selection only 5 bits are taken out of the 10 bit of the LFSR. The 5 Bits taken from the LFSR are resulting in values between 31 and 0, and acting as physical page selection in the data flash sector. The physical page 32 cannot be selected by the random number generator directly. Starting from the randomized Spare Page Pointer the next empty page is being searched, by reading the physical page control information successively page by page. If the end of the data flash module has been reached (page 32) and no empty page has been found yet, a wrap around to page 0 takes place and the searching for an empty page keeps going on.

In case the Spare Page Pointer search reaches the starting point, given by the Random Number Generator, without finding an empty page, then the Spare Page Pointer will be invalidated, it basically points to a not existing page address. Due to the fact that the data flash sector has one physical page more than being able to be addressed by the user, an empty page will always be found by the Spare Page Pointer search under normal operating condition. However, a Spare Page Pointer search may fail if the data integrity inside the physical data flash sector is lost and no erased page is found. The register SYS_STRTUP_STS reflects in bit[1] the result of the failed MapRAM initialization. If no Spare Page Pointer is available the user function USER_OpenAB will return a fail and no further write operations to the data flash will be possible. In such an event it is recommended to reset the device in order to start the Service Algorithm. The Service Algorithm tries to resolve the data integrity issue in

the data flash sector. In case the Service Algorithm might not be able to resolve the data integrity issue inside the data flash sector then a data flash sector erase is recommended.

Figure 12 provides three examples of how the Spare Page Pointer selection is working. The SP Init defines the starting point for the Spare Page Pointer search given by the Random Number Generator. From that page on the NVM state machine searches for the next empty (erased) page. In Example 1 already the physical page addressed by the SP Init is available and will be selected as Spare Page. In Example 2 the same starting point is assumed, but in that case the page 2 is occupied (written), the Spare Page Pointer will now be incremented by one. It points now to page 3 which is empty (erased), so physical page 3 will now be the resulting Spare Page. Example 3 displays the wrap around. Here the SP Init value points to page 31. Since page 31 is occupied, by linear upcounting to the next page, page 32 is checked. Due to the fact that page 32 is occupied too, a wrap around to page 0 takes place. In the example 3 page 0 is empty, it will be taken as the next Spare Page. In the green box the selected Spare Page for each example is displayed.

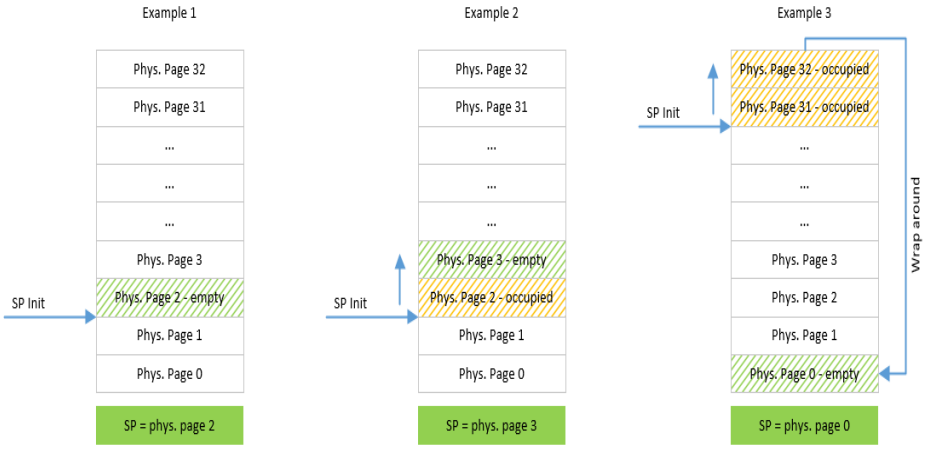

Figure 12 Spare Page Pointer selection examples

3.2.3 MapRAM Init

The MapRAM as address translation look-up table is a volatile SRAM, it has to be reconstructed after each Power-On Reset, PIN Reset, WDT1 Reset, wake-up from SleepMode and wake-up from Stop Mode (with reset only). The NVM module provides a hardware function for this purpose, the MAPRAM_INIT function. After these resets the content of the MapRAM is invalid, i.e. none of the entries contains any mapping to any physical data flash page, the MapRAM is “empty”. The MapRAM content is being reconstructed out of the MapBlock. The MapBlock is the control information which is stored along with the 128 bytes of user data within every physical data flash page. The MapBlock content is copied over from the data flash page into the MapRAM one by one for mapped pages only. The readout of the MapBlock information is done by using the User Read Threshold as well as Soft Read Thresholds. Using Soft Read Thresholds for evaluating the MapBlock provides more margin to the User Read Thresholds used in user mode. Physical data flash pages which are in erased state (section A) or weakly erased state (section B) are skipped. The initialization of the MapRAM then continues with the next physical data flash page. The initialization of the MapRAM always starts at the index (physical data flash page) which is addressed by the initial starting point of Spare Page Pointer. The initialization of the MapRAM ends after a wrap-around at physical data flash page 33 when the initial starting point for the Spare Page Pointer has been reached again. **Figure 13** depicts an example of the principle of the MapRAM initialization. The randomized Spare Page Pointer (SP Init, here page 31) acts as starting point for the MapRAM initialization. It points to the first physical data flash page, where the link inside the MapBlock is used to address an entry of the MapRAM. At the addressed entry of the MapRAM the current physical data flash page number is stored. The logical address and the physical data flash page are now linked together. The pointer to the physical data flash page is now incremented to physical page 32. Again the link inside the MapBlock if physical data flash page 32 is addressing an entry inside the MapRAM, this MapRAM entry is now initialized with the value 32 pointing to the physical data flash page 32. By

incrementing the physical data flash pointer a wrap around to physical data flash page 0 takes place. The physical data flash page 0 is empty, this page will be skipped and it continues with physical data flash page 1. The entire process stops once the initial starting point (SP Init) has been reached again.

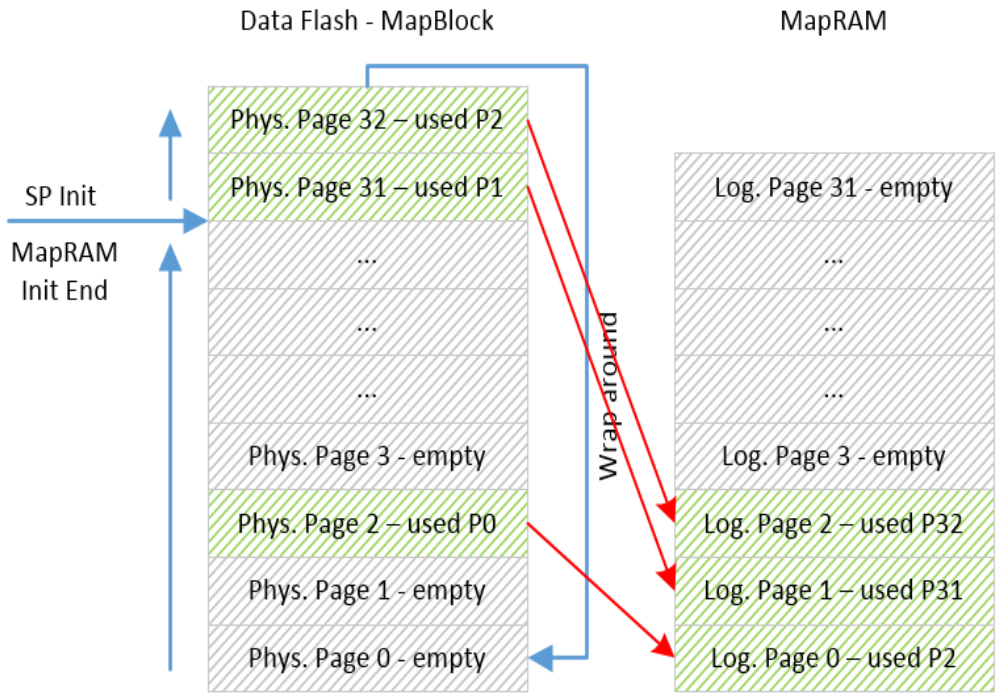

Figure 13 MapRAM initialization example (principle)

The logical-to-physical address translation for the data flash is reconstructed and an consistent mapping has been established.

3.2.3.1 MapRAM Init Aborts

The MapRAM initialization is performed by the NVM hardware module, there is no code execution required, except for starting the hardware function and evaluating the return value of the hardware function. There are a few conditions under which the hardware function MAPRAM_INIT aborts, these are:

- faulty page: a physical data flash page has a MapBlock information inconsistencies (e.g. ECC)
- double-mapping: the link of an addressed physical data flash page is addressing an already initialized entry inside the MapRAM

If the MAPRAM_INIT function aborts due to one of the reasons mentioned above the initialization of the MapRAM has only been performed until the erroneous physical data flash page has been detected. All physical data flash pages later in this order are not initialized anymore (i.e. they are not mapped). Please see **Figure 14** as an example. Here the MapRAM initialization starts at physical flash page 31, proceeds with page 32 and performs a wrap-around to physical flash page 0. Page 0 can be skipped as it is humpy, but physical page 1 is read faulty. At this point the MAPRAM_INIT function aborts. All physical pages from physical page 2 onwards until physical page 30 are not considered anymore. In this example, physical flash page 2 is not entered into the MapRAM.

Due to the fact that the initialization of the MapRAM starts at the randomized Spare Page Pointer initial value, it is indeterminable which pages are mapped or whether any page is mapped at all. Anything in between “all pages mapped successfully” and “no pages mapped at all” is possible.

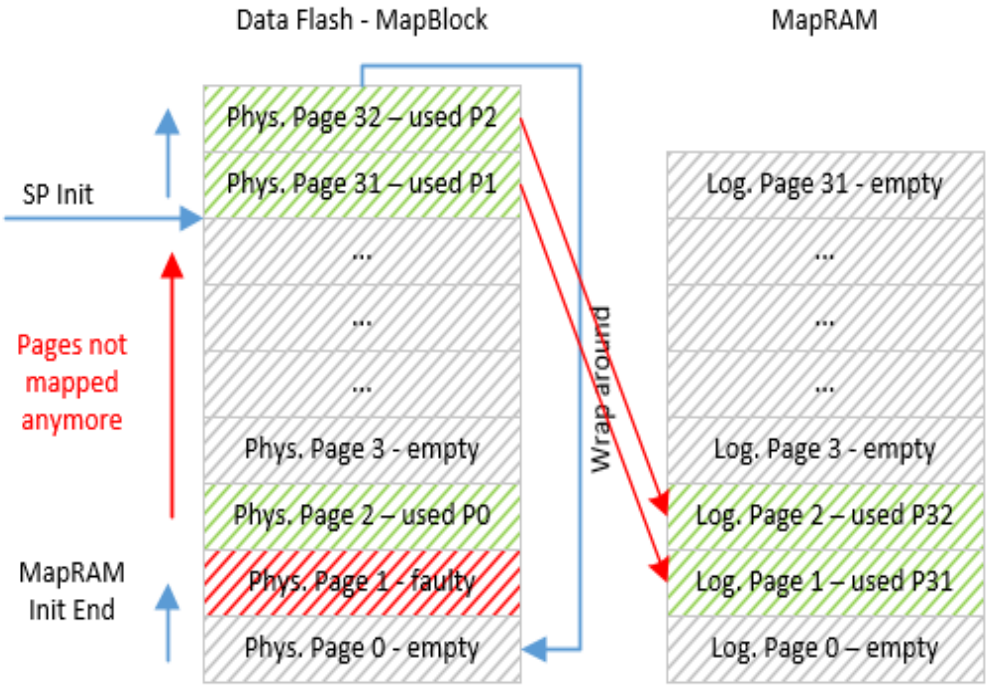

Figure 14 MapRAM initialization example with abort (principle)

If the MAPRAM_INIT function aborts during the start-up, the “Service Algorithm” is executed in order to try to fix the data flash integrity issue.

Figure 15 shows an simplified start-up flow which shows the MAPRAM_INIT and the call of the Service Algorithm in case of MAPRAM_INIT abort.

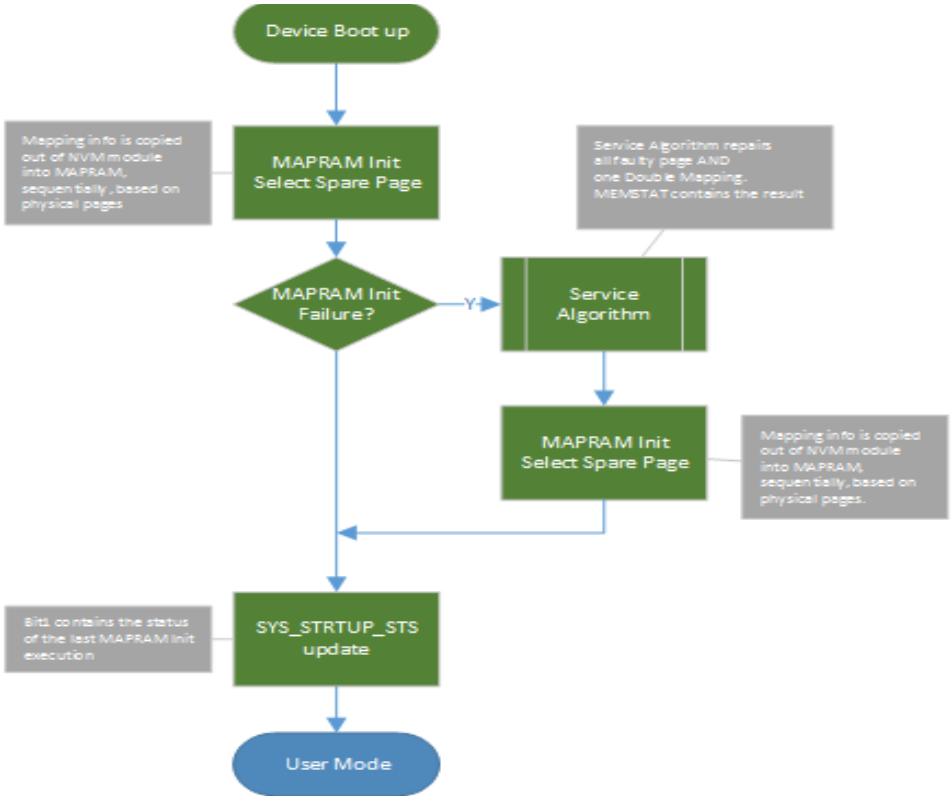

Figure 15 Simplified Start-up flow with MAPRAM_INIT and Service Algorithm

3.2.4 Service Algorithm

The Service Algorithm (SA) is meant to be executed in the exceptional event of losing the data integrity inside the data flash, e.g. due to an interrupted programming operation. The SA is by concept an exception handler, to handle unlikely events of interrupted NVM operations. It is not meant to be part of the regularly used user application flow.

The SA is called due to an abort of the MAPRAM_INIT hardware function. The SA is able to resolve two kind of data flash integrity error. It is able to solve as many “faulty”-pages as found in the data flash sector. Furthermore the SA is capable of solving one double-mapping issue. As soon as more than one double-mapping, or a higher mapping, is detected, the SA quits without performing the solving of the multimapping issue. In this case the SA returns a value 0b10xxxxxx inside the MEMSTAT register.

A repair for the SA always means “Page Erase”, as the SA is not able to predict the intended user data inside the erroneous data flash page.

The SA scans through the entire physical data flash page by page and searches for faulty pages. When doing this the SA uses the User Read Threshold. Any faulty page found will be erased. Then the SA scans the entire sector again, page by page, but using Soft Read Thresholds now. Soft Read Thresholds are in between the User Read Thresholds and the Hard Read Thresholds. Soft Read Thresholds are used in order to gain more guard band to the User Read Threshold used during the user mode execution. Any faulty page found by using Soft Read Thresholds will be erased.

After the faulty pages scans the SA continues searching for double-mappings. In order to detect double-mappings, meaning more than one physical page is mapped to the same logical page, the SA starts scanning the data flash sector starting at physical page 32. The SA counts the mappings for each logical page.

The SA is capable of resolving a maximum of ONE double-mapping.

If more than one double-mapping or even a multiple-mapping exists then the SA quits without performing the correction of the multimapping issue. In this case the SA returns a value 0b10xxxxxx inside the MEMSTAT register.

In order to resolve the double-mapping the SA reads each page contributing to the double mapping again with Hard Read Thresholds. By this the SA tries to figure out which of the two pages provides the best signal quality of the flash cells. If there is one page recognized to have worse signal quality compared to the other page, then the page with the worse signal quality is erased. The double-mapping is resolved.

If both pages contributing to the double-mapping have the same signal quality based on the Hard Read Threshold readout, then the page counter is evaluated to determine which of the two pages is the newer one. The page counter always counts in a row like: 1-2-3-1-2-3 ... and so on. With every USER_PROG function execution the page counter inside the MapBlock is incremented according to the counting scheme mentioned. The SA compares the page counter for the two pages to determine the older and the newer page. The older page will be erased by the SA.

In the exceptional case that even the page counter of the two pages hold the same value then the SA has no chance to resolve the double mapping, it will quit without performing any corrective actions to the physical data flash sector. In this case the SA returns a value 0b10xxxxxx inside the MEMSTAT register.

The SA provides its return values in the register MEMSTAT. The lower 5 bits of the MEMSTAT register, MEMSTAT[4:0], provide the sector number of the data flash sector. This depends on the used derivate:

- 0x09: for 36K flash devices
- 0x10: for 64K flash devices
- 0x20: for 128K flash devices

The upper two bits, MEMSTAT[7:6], provide the actual return value of the SA. These bits are only valid if the lower 5 bits of the MEMSTAT register are different from 0. The bits MEMSTAT[7:6] encode whether the SA was able to resolve an data integrity issue or not:

- 0x0: SA was executed, either no integrity issue was detected, or ONE integrity issue was resolved
- 0x1: more than one integrity issue was resolved

- 0x2: SA was not able to resolve the integrity issue inside the data flash sector
- 0x3: SA was not able to resolve the integrity issue inside the data flash sector

If the SA was not executed at all then the MEMSTAT register is cleared to 0x00.

3.2.4.1 Service Algorithm Limits

By concept the Service Algorithm (SA) is intended as an exception handler. NVM operations like erase or programming of a page shall not be interrupted in any way. In the unlikely event that an erase operation or a programming operation could not be finished completely, the SA at the next start-up of the device will be executed. Due to the fact that a flash cell by physics is an analog element and certain effects of the flash cell might make it impossible for the SA to restore the lost data integrity due to the interrupted NVM operation under every circumstance. By design the SA is able to repair as many faulty pages and up to one double-mapping error inside the data flash. Resolving an error in the data flash for the SA always means “Page Erase”, as the SA is not able to infer the intended user data inside the erroneous data flash page.

Due to the physical effects of a flash cell as described in **Chapter 2.2**, in rare cases it might be possible that the SA is not able to detect any erroneous page, though it exists in the data flash sector. A successive execution of the MAPRAM_INIT function might still abort due to this erroneous page. Especially a double-mapping caused by an interrupted USER_PROG function might lead to the erasing of the “wrong” page being part of the double-mapping which will be explained with the help of **Figure 16**. As described in the previous chapter, the SA checks both pages with the Hard Read Thresholds, and since in the example both pages are beyond the Hard Read Thresholds the SA takes the page counter into account. The older page, obviously the page with the better signal quality is getting erased. The newer page of the two was the interrupted page, but at the time the SA checked this page with Hard Read Thresholds the page was of good programming quality. Effects like relaxation of the page, or drift in the read thresholds might make this page become a faulty page later on, which could lead to an abortion of a successive MAPRAM_INIT after the SA execution. **Figure 16** shows the cell distribution of two pages contributing to a double mapping. The programming of the “new page” could not be finished as the USER_PROG function was interrupted for some reason. After the restart of the device, the MAPRAM_INIT function recognizes the double-mapping cause by the “old page” and the “new page”, it calls the SA in order to resolve the double mapping.

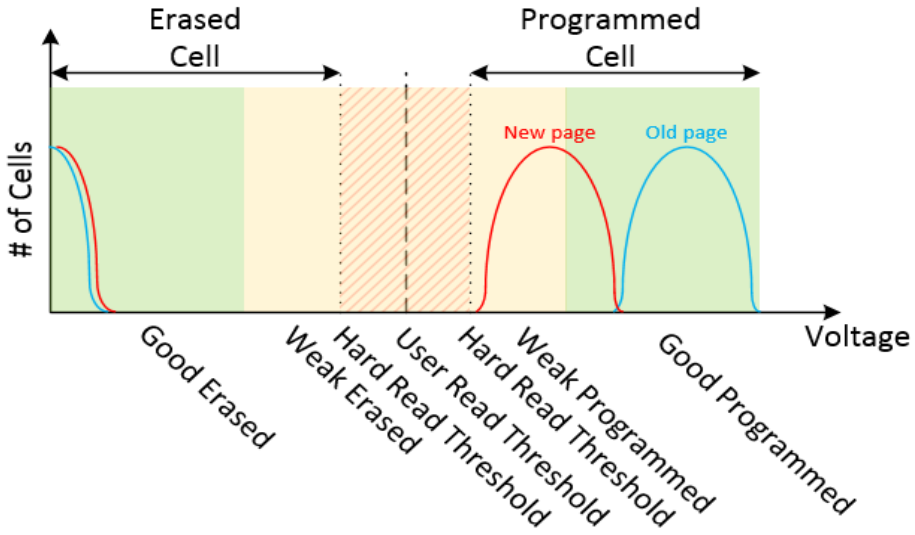

Figure 16 Double-Mapping with the new page as weakly programmed page before SA execution

The SA erases the “old page” based on the evaluations described before. **Figure 17** shows now the remaining “new page”. The “new page” might now drift into the marginal state due to not yet finished relaxation effect, in addition the User Read Threshold might drift as well, making this page temporary a faulty page. The subsequent MAPRAM_INIT function after the SA might then abort due to this “new page” now being a faulty page. The result

of the subsequent MAPRAM_INIT function will be reflected in the register SYS_STRTUP_STS.Bit1. The user has to check this flag before any NVM operation (write/erase) is performed and apply appropriate corrective actions.

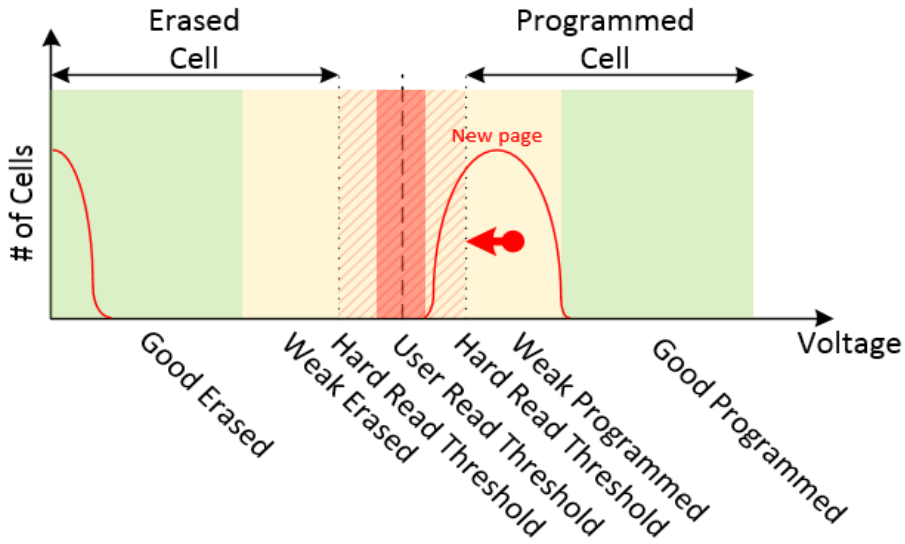

Figure 17 Double-Mapping with the new page as weakly programmed page after SA execution

3.2.4.2 Service Algorithm Summary

The Service Algorithm (SA) as an exception handler for unlikely events of interrupted NVM operations is not able to provide a 100% coverage for all kinds of data flash signal quality possibilities. Nevertheless it is able to reduce the loss of data integrity of the data flash sector. The **Chapter 4** describes the action to be taken by the user application in order to maintain or regain the data integrity in the data flash sector.

4 User Application Considerations

Due to the fact that the Service Algorithm is not able to resolve every data flash integrity issue caused by interrupted NVM operations (refer to **Chapter 3**), erase or programming, certain actions in the user application might be worth considering.

4.1 After USER MODE Entry

Figure 18 displays the simplified start-up flow. It is recommended to implement the test of the SYS_STRTUP_STS.Bit1 upon USER MODE entry, prior to any NVM operation. If the bit is clear then the data flash mapping is consistent NVM write/erase operation can be performed. To see if the Service Algorithm might have been active the user has to check the MEMSTAT register. If the Service Algorithm was active the user has to expect that expected logical data flash pages are not present anymore. The user has to take care of this and reconstruct any missing page. Furthermore it might be possible that the Service Algorithm reports an unrecoverable failure inside the Data Flash, then the same corrective actions shall be applied as described in the following paragraph for the case that SYS_STRTUP_STS.Bit1 is set.

If the SYS_STRTUP_STS.Bit1 is set, then the data flash mapping is inconsistent, the mapping might not be complete and any NVM operation like write or erase is not safe and might cause further inconsistencies inside the data flash. As corrective actions the user might reset the device (cold reset) in order to give the Service Algorithm a chance to repair the data flash sector. If this attempt fails again, then a sector erase is needed to reinitialize the data flash sector and to remove any mapping inconsistency. After the data flash sector has been erased the user has to take care of reconstructing the expected logical data flash pages.

Attention: Due to that fact that it might be needed to erase the data flash sector in case on unrecoverable failures the user shall consider not to store constant operation parameters necessary to operate the ECU inside the data flash. Those kind of data should be placed inside the code flash along with the application code. Inside the data flash only dynamic runtime data should be stored, which could be surrendered in case of data flash inconsistencies without affecting the functionality of the ECU in a bad way.

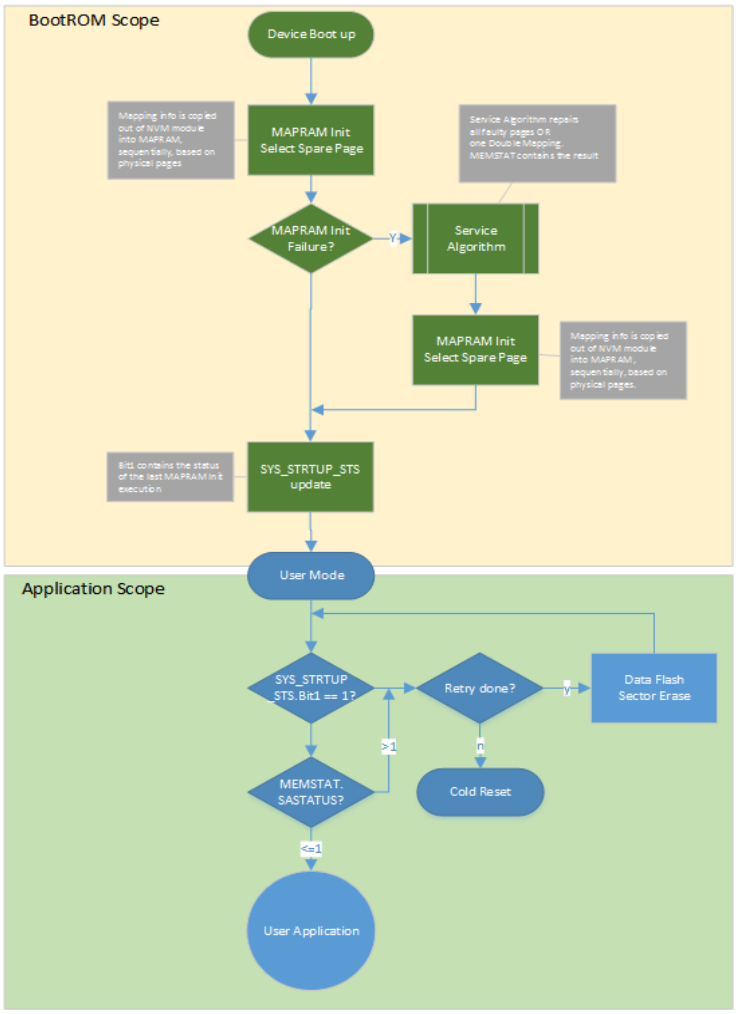

Figure 18 Simplified start-up flow with SYS_STRTUP_STS evaluation in USER MODE

4.2 USER_MAPRAM_INIT function

In order to perform a data flash mapping consistency check at runtime the user mode function USER_MAPRAM_INIT is provided. It provides the same functionality as the MAPRAM_INIT function executed during start-up. It triggers the MapRAM initialization function in the NVM hardware module, which performs two actions:

- MapRAM initialization, copying over the control information out of the data flash pages into the MapRAM
- Random Spare Page Pointer selection

The USER_MAPRAM_INIT function provides a return value to the user which allows the user to judge the data integrity of the data flash. The USER_MAPRAM_INIT function provides the following encoding of the return value:

```
/*****************************************************************************
** Function USER_MAPRAM_INIT: return value decoding                         **
*****************************************************************************/
typedef union {
       uint8   reg;
       struct {
              uint8 GlobFail      :  1; /* [0:0] */
              uint8               :  4; /* [4:2] */
              uint8 DoubleMapping :  1; /* [5:5] */
```

```
            uint8 FaultyPage    :  1; /* [6:6] */
            uint8 ExecFail      :  1; /* [7:7] */
      } bit;
} TUser_MAPRAM_Init;
```

Any subsequent NVM write/erase operations might cause further inconsistencies in the data flash which limits the possibility for the Service Algorithm to resolve them. It would be wise to the user to restart the device (cold reset) to execute the Service Algorithm instead.

4.3 USER MODE Entry after Soft Reset

A soft reset, triggered by calling the CMSIS function NVIC_SystemReset, does not run through the data flash initialization as shown in **Figure 18**. The data flash mapping will not be reinitialized again during soft reset. In case the soft reset is executed by the user code during RAM branching (USER_PROG with RAM branch) or inside an interrupt handler any preloading of the MapRAM done by the function USER_OpenAB has to be undone before any new call of a NVM write/erase function will be done. The simples method to do so is calling the USER_MAPRAM_INIT function, which will invalidate the MapRAM and rebuild it out of the MapBlock. In case the USER_MAPRAM_INIT function returns a fail the device shall be reset (cold reset) in order to execute the Service Algorithm. See also **Chapter 4.2**.

4.4 Program Page Flow

Figure 19 displays a complete flow for programming a flash page. The following chapters will describe each part of the flow more deeply.

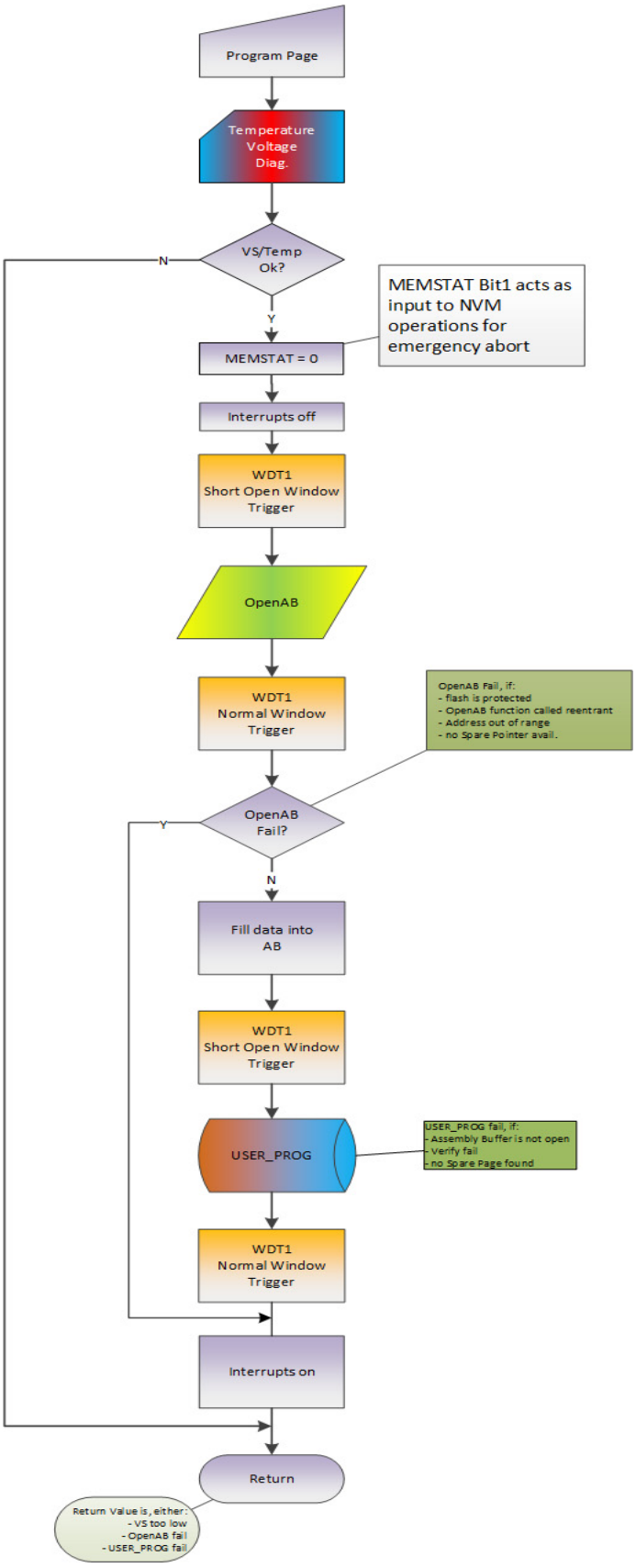

Figure 19 Program Page Flow

4.4.1 Voltage and Temperature Check

Initially the Program Page Flow performs a check of the device supply voltage and device temperature. It is highly recommended to only perform NVM write/erase operations if the device supply voltage is high enough in order to successfully finish the NVM write/erase operation. The capacitance of buffer capacitor connected to the device supply should be selected based on the power consumption of the device and the time needed to finish a started NVM operation in time. The same applies to the device temperature, because if the device temperature exceeds the specified Tj = 150°C it must be expected that the device will be shut down for safety reasons. If an NVM operation has just been triggered before, the notification about the safety shutdown cannot be handled by the core, the system might be shut down right in the middle of the NVM write/erase operation, which could lead to a corrupted data flash. The checking of the supply voltage and the device temperature can be done by reading the current result values out of the ADC2's result registers and comparing the digital values against application defined boundaries.

4.4.2 MEMSTAT

The register MEMSTAT has a dual-function, it serves as return value for the Service Algorithm, and also it acts as input value for the NVM operations to indicate an Emergency Operation. If an Emergency Operation, which exits an NVM operation as quick as possible, is not wanted the MEMSTAT register has to be reset to 0 prior a NVM operation is called.

4.4.3 Interrupts

In order to not interfere with the page programming process it is recommended to disable all system interrupts. It is not forbidden to have interrupts enabled during NVM operations, but the user has to ensure that the interrupted NVM operation will not be influenced, i.e. that no other NVM operation is triggered inside the interrupt handling. Nested NVM operation are not allowed at all.

At the end of the page programming flow the interrupts can be enabled again.

4.4.4 Watchdog Handling

In order to avoid WDT1 resets during the NVM operation it is recommended to trigger a Short-Open-Window (SOW) service for the WDT1. The SOW-service can be placed at any time on an ongoing normal WDT1 period. It opens up an open window period of 30ms. Enough time to safely perform one NVM operation. After the NVM operation returns the SOW is supposed to be closed by servicing the WDT1 in normal window mode. The WDT1 handling reoccurs all over the flow before and after a NVM operation.

4.4.5 OpenAB

The NVM function OpenAB is called in order to open the assembly buffer. The function OpenAB distinguishes between an operation on a new logical data flash page and on an already existing logical data flash page. For a new page the assembly buffer will be filled with all 0xFF, the physical contents of an erased page. Furthermore it preloads the MapRAM for the addresses logical page with the index of the page pointed to by the Spare Page Pointer. This is required in order to link the assembly buffer to the logical address range given by the logical page selected for writing. Only then the user will be able to fill the assembly buffer in the next step by copying the user data to the logical address of the page.

Attention: *At this point in time the Spare Page Pointer and the MapRAM entry for that logical data flash page do point to the same physical data flash page. If the device will be reset by a soft-reset (or reset of the internal WDT) then it is highly recommended to perform a USER_MAPRAM_INIT function call in order to remove the already mapped page from the MapRAM, since otherwise the loss of single page data can be expected if further write operations will be performed to the data flash sector.*

For used pages the assembly buffer will be filled with the physical data of the already mapped data flash page. The assembly buffer will be linked to the logical address range of the addressed logical data flash page.

In order to reverse the actions performed by the function OpenAB, the function USER_ABORTPROG can be used, which is not part of this flow.

The function OpenAB provides return values to the user application to handle certain failure scenarios, which are:

- wrong logical address range - the given address is not part of the NVM module of the device
- page protected - the addressed data flash page is write protected
- no valid Spare Page Pointer - if no valid Spare Page Pointer is available

In either of these failure cases it is not possible to execute the USER_PROG function. While the corrective actions for the first two failure reasons are obvious, for the last one “no valid Spare Page Pointer” the function USER_MAPRAM_INIT should be executed in order to check the data integrity of the data flash sector and to randomize a new Spare Page Pointer. In case of a USER_MAPRAM_INIT fail the user shall implement effective corrective actions.

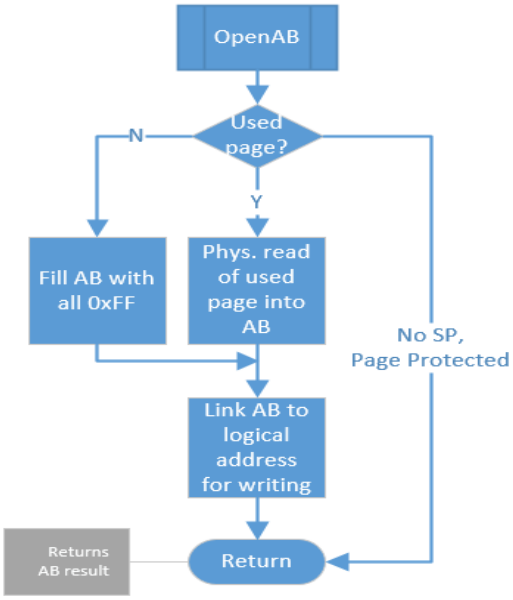

Figure 20 simplified OpenAB function

4.4.6 Filling Data into Assembly Buffer

Once the assembly buffer has been opened successfully the user is able to fill data into the assembly buffer by copying data to the logical address of the desired logical data flash page.

4.4.7 USER_PROG

The function USER_PROG writes the prepared assembly buffer data into the physical data flash page addressed by the Spare Page Pointer. Once the writing is complete the data of the physical data flash page is verified against the content of the assembly buffer. Function parameters determine the response to a verify failure. If Bit1 is set corrective actions are taken. Bit2 determines if page data is erased, and if old or new data is retained if the page has already been used.

For used pages the Spare Page Pointer and the MapRAM content for the addresses logical page are flipped. The logical page is now linked to the new physical data flash page. At this point in time a double-mapping exists inside the data flash sector. A page erase will now be performed on the older page, and by this the previous double-mapping is cleared.

For a new page the erase of the old page is skipped as there is no former page available for erase.

A new Spare Page Pointer is now be selected by reading the pseudo-random number generator as a starting point for the new Spare Page Pointer. The MAPRAM content does not get influenced by the selection of a new Spare

Page Pointer. In case there was no new Spare Page Pointer to be found the USER_PROG function returns this failure in its return value. It is not recommended to perform any further data flash write/erase operations instead the user shall execute the USER_MAPRAM_INIT function to check the consitency of the data flash and react to its return value accordingly. If the Dirsturb-Handling feature is enabled (Bit1) then based on the value of the pseudo random number generator in average every 1000th USER_PROG call the Disturb-Handling is executed.

If the Retry/Distrub-Handling feature is enabled then the maximum execution time for the USER_PROG function has to be doubled compared to if this feature is disabled.

Attention: If the RAM-Branch feature is used then the user has to ensure that no processor core registers are modified upon return back to the USER_PROG function, otherwise a reliably functionality of the USER_PROG function cannot be ensured.

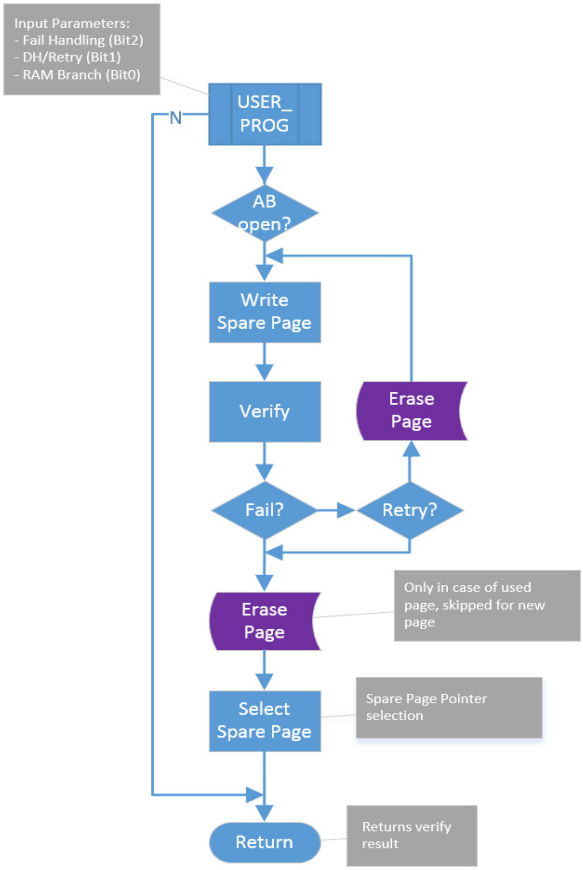

Figure 21 simplified USER_PROG function

4.4.8 Program Page Flow return handling

The program page flow does not implement any error handling, instead the return values of each sub-functions is being handed over to the caller. It can consist of the return values of the USER_OpenAB functions, the USER_PROG function and the supply/temperature check. The caller function (i.e. main) has to implement proper handling of these failure cases.

If the voltage is not stable in the system, or the temperature is maybe too high, then a data flash write might not be able to finish correctly (unintented interruption). The caller might recall the Program Page Flow at some later point in time, when the supply voltage has gained sufficient stability, or the device temperature has been lowered.

In case of USER_OpenAB fails the caller should check failure cases for the USER_OpenAB function, which are:

- Address out of range
- Data Flash protected

- nested call to the USER_OpenAP function
- no valid Spare Page Pointer available

The first three failure cases should not happen in a well designed user application.

The fourth failure, no Spare Page Pointer available, hints to some issues with the integrity of the data flash. It is highly recommended to execute the USER_MAPRAM_INIT function in order to check the mapping integrity of the data flash. If the USER_MAPRAM_INIT returns a pass, then the Program Page Flow can simply be recalled, if it returns a fail then the device could be reset in order to start the Service Algorithm, otherwise the write/erase access to the data flash shll be prevented in the further execution of the user application.

If the USER_OpenAB function was successfully executed in the Program Page Flow, then the USER_PROG provides its failure cases as return value, which are:

- Assembly buffer not open
- Verify fail
- no new Spare Page Pointer found

The first fail, Assembly Buffer not open, should not happen, because then the Program Page Flow should have exited already based on a USER_OpenAB failure.

The Verify fail happens if the data inside the Assembly Buffer and the data inside the physical data page which was just written do not match. A valid user reaction would be to simply execute the Program Page Flow again to give it another try. If it still fails, then a USER_MAPRAM_INIT call might make sense to check if the data flash is in a consistent state. If so the desired logical data flash page could be erased, before that the content of the page shall be saved in some RAM buffer, and then written again to the data flash.

The fourth fail, no new Spare Page Pointer found, points to some integrity issues inside the data flash. It is not recommended to perform any further data flash write/erase operations instead the user shall execute the USER_MAPRAM_INIT function to check the consitency of the data flash and react to its return value accordingly.

5 Summary

In order to maintain a consistent data flash mapping the user shall ensure that no flash operations which modify the content of the flash, like write and erase, get interrupted at any time. In order to ensure this it is recommend to evaluate the device supply voltage and device temperature before any NVM operation (write/erase). For the minimum required supply voltage necessary to finish the NVM operation the user must consider the current consumption of the device and the time needed to finish the NVM operation and calculate the required supply buffer capacitor accordingly. The user shall avoid to perform NVM operations if no sufficient supply voltage is available to finish the NVM operation.

Other interruptions like PIN reset, WDT1 reset or fail-safe SleepMode entries will also interrupt the NVM operation with the potential consequence of the data integrity issues inside the data flash sector.

In order to handle data integrity issues in the very first place it is recommended to check the MEMSTAT register and the SYS_STRTUP_STS.Bit1 upon start-up. NVM operations (write/erase) shall only be applied to a valid mapped data flash sector.

Furthermore, the recommendation of not interrupting a flash operation does also apply to code flash pages as well as to flash pages of the 100TP NVM. Here special care has to be take not to interrupt these operations as there is no mechanism implemented to handle interrupted write/erase operations on these flash regions.

6 Additional Information

In order to learn more about the data flash handling please review the following sources of information:

- For further information you may contact **http://www.infineon.com/**

7 Revision History

Revision	Date	Changes
1.1	2019-03-20	BF-Step support added
1.00	2016-04-18	initial version

www.infineon.com

Edition 2019-03-20

Published by

Infineon Technologies AG

81726 Munich, Germany

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

Email: erratum@infineon.com

Document reference
Doc_Number

Legal Disclaimer

THE INFORMATION GIVEN IN THIS APPLICATION NOTE (INCLUDING BUT NOT LIMITED TO CONTENTS OF REFERENCED WEBSITES) 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.