MCAL Supporting User Manual for Fee

32-bit TriCore™ Aurix™ TC3xx microcontroller

About this document

Scope and purpose

This document provides detailed information for about algorithms related to Fee module.

Intended audience

This document is intended for anyone using the Fee module of the TC3xx MCAL software.

Reference documents

This User Manual should be read in conjunction with the following documents:
Aurix ™ TC3xx MCAL User Manual for Fee

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 3
2 Double Sector Algorithm . . . 5
2.1 Overview of state block structure . . . 6
2.2 Overview of NVM Data Block Structure . . . 9
2.3 Overview of garbage collection process . . . 11
3 Quasi Static Algorithm . . . 13
3.1 Quasi static data block structure . . . 13
3.2 Sequence of States . . . 15
3.3 Example - Handling Quasi-Static data blocks . . . 18
4 Conditions to raise production error and illegal state notifications . . . 20
Revision History . . . 23
Disclaimer . . . 24

1 Introduction

Infineon implementation provides services to read and write two types of data blocks to the Flash memory. The first type of data block is called the NVM data block, which is further classified as normal or immediate data blocks. These blocks are read and written by the NVM layer. The FEE implements a double-sector algorithm to handle the NVM data blocks. The second type of data block is called Quasi-Static block and is handled by a separate algorithm. The Quasi-Static data blocks are written very rarely, totalling no more than 500 writes across all Quasi-Static data blocks. Architecturally, the Quasi-Static data blocks are seen to be read and written by the Quasi-Static Data Manager. The implementation of Infineon FEE is designed to handle these two types of blocks differently keeping in mind not to wear off the Flash un-necessarily.

The size of memory configured for double-sector data should always be a multiple of 8KB (2x4KB).

This is because the minimum erasable area supported by the flash hardware is 4KB. The size of memory configured for Quasi-Static data should also be a multiple of 4KB.

The handling of DFlash by FEE can be partitioned into 3 broad areas.

1. Handling standard data (NVM data accessed via MEMIF) using the Double-Sector algorithm.
2. Handling quasi-static data (Quasi-static data accessed by a Quasi Static Data Manager)
3. Handling the co-ordination of standard and QS data handling services.

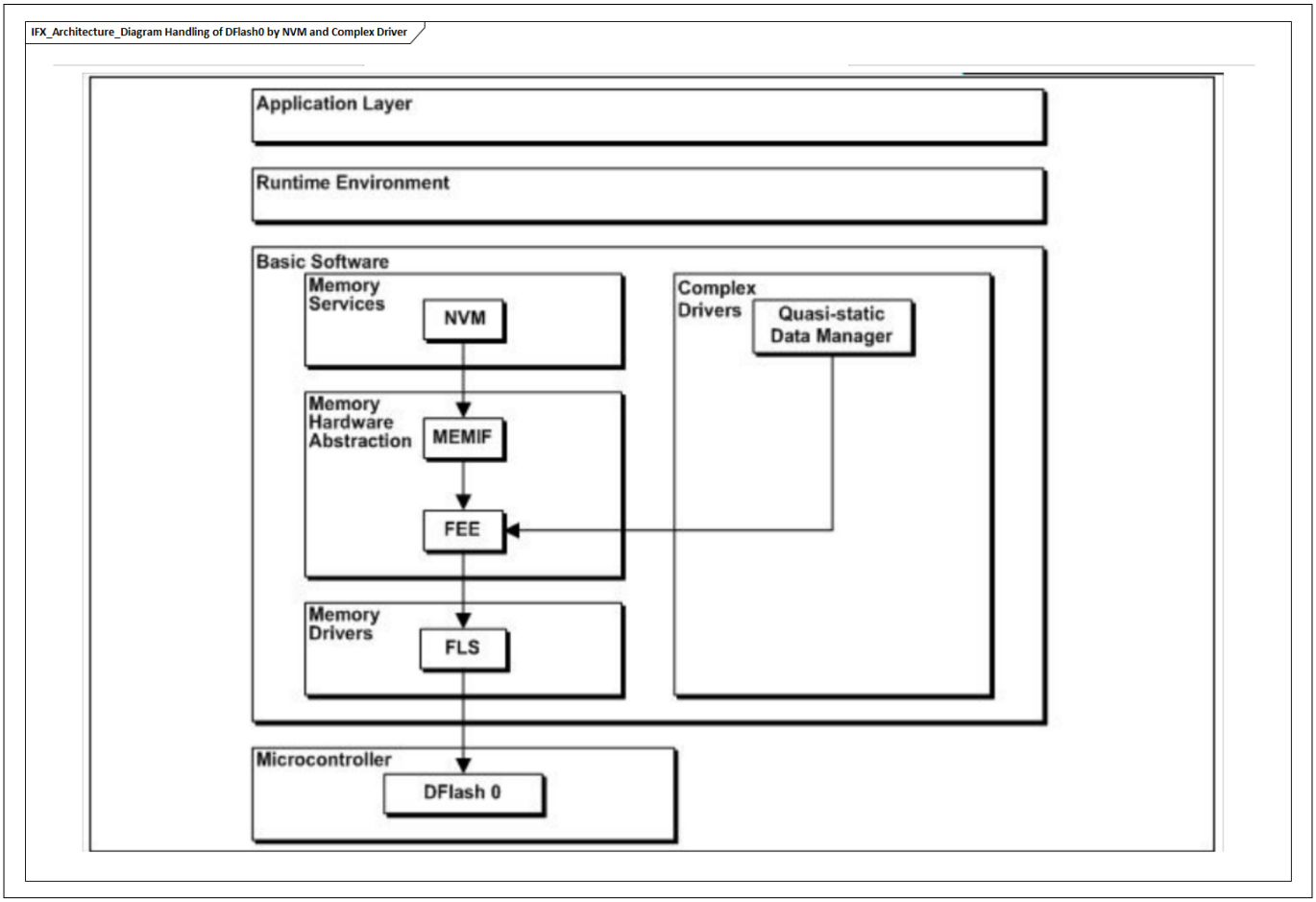

Figure 1 Fee_Handling_of_DFlash0_by_NVM_and_Complex_Driver

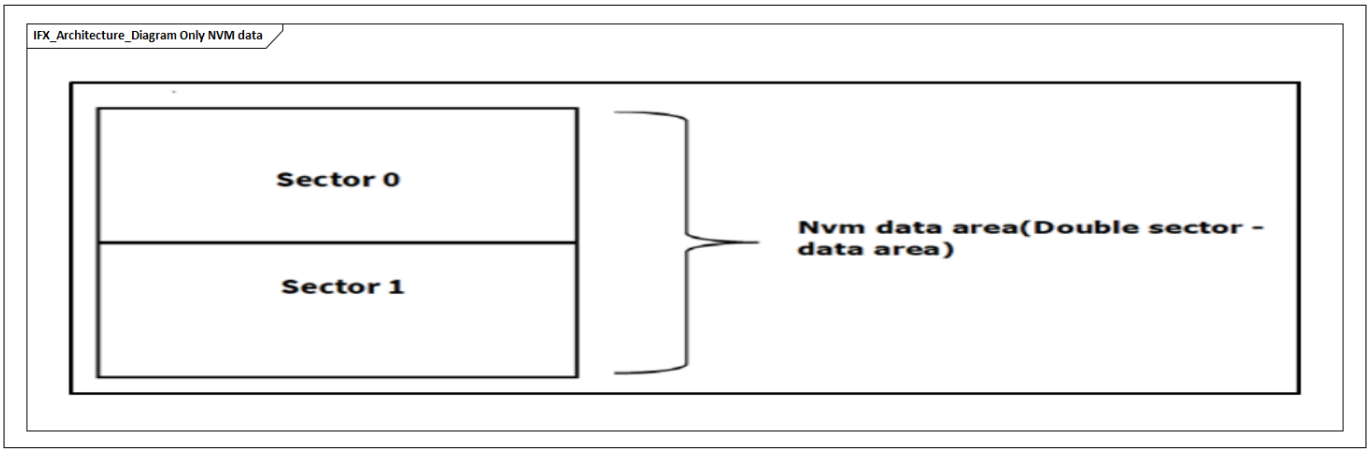

Figure 2 Fee_Only_NVM_data

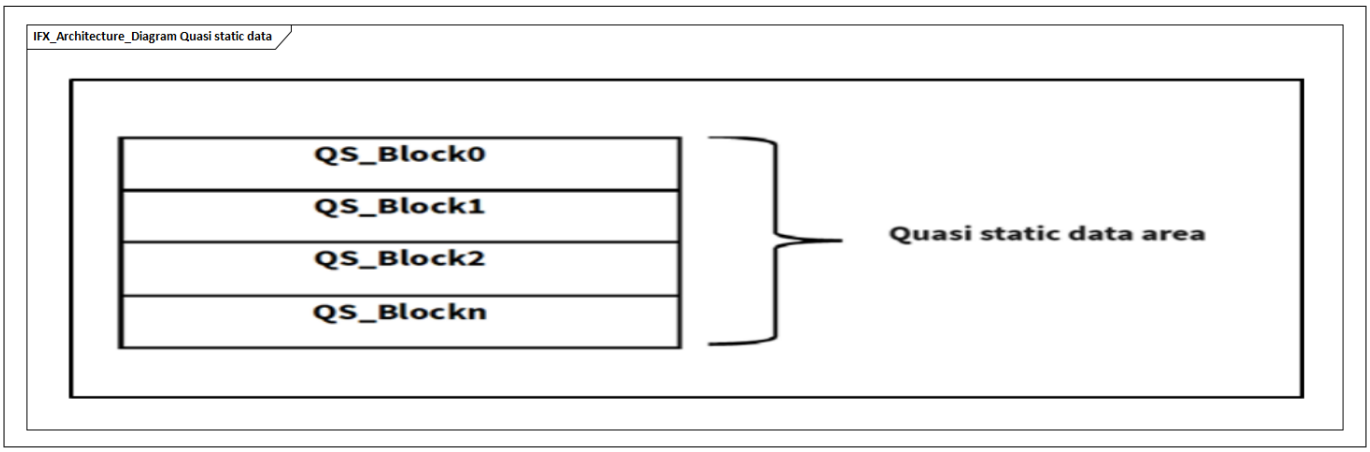

Figure 3 Fee_Quasi_static_data

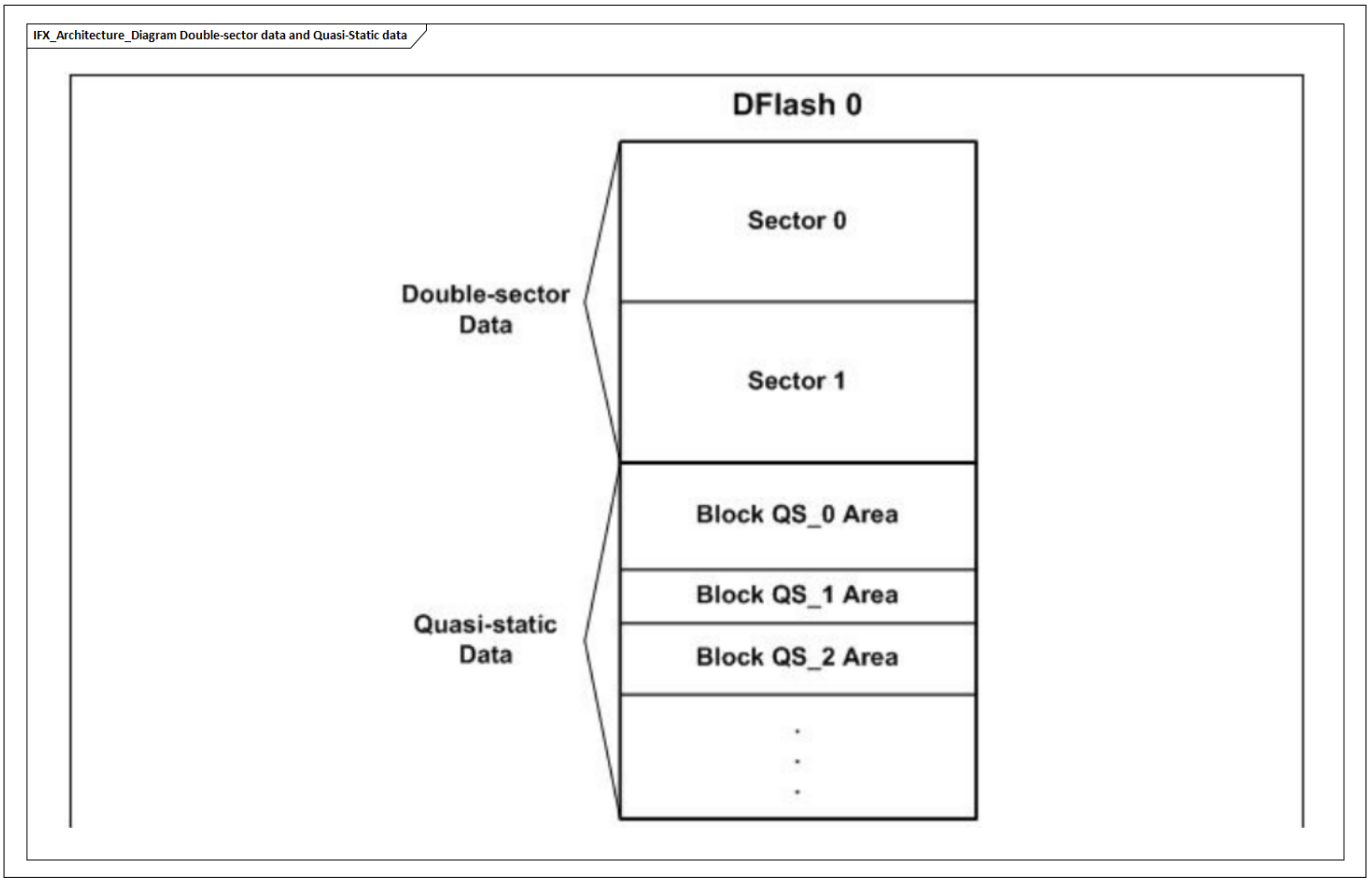

Figure 4 Fee_Double-sector_data_and_Quasi-Static_data

2 Double Sector Algorithm

The Double-Sector algorithm helps to extend the life of the Flash by reducing the number of write-erase cycles.For the Double-Sector algorithm, an area of DFlash (DFLASH0), configured for use by the algorithm, is divided in to two sectors. When write requests are made, the Double-Sector algorithm fills the first DFlash sector with data blocks. The data block is simply appended to the end of the *stack* of data blocks already written to the first sector. When the first sector is filled to a certain threshold, garbage collection is started which copies the most recent data blocks from the first sector to the second sector that is in the erased state. After the data from the first sector is copied to the second sector, the first sector is erased. Further write requests cause the data to be written to the end of the *stack* of blocks in the second sector. The process repeats when more write requests causes the second sector to be filled to the threshold, triggering the garbage collection.

As an initial state it is assumed that sector 0 is active, the data blocks are written into this sector and the threshold has been crossed and sector 1 is erased.

1. Copy the most recent version of all logical blocks from sector 0 to sector 1. Invalidated blocks are copied as well but not inconsistent blocks.
2. Mark the correct end of the copy procedure by programming the state of sector 1 to valid and increment the state counter. From now on, sector 1 is active.
3. Erase sector 0 and verify the erase.
4. Mark the end of the erase by programming the state of sector 0 to erased, record the un-erased WL information in the state block and increment the state Counter.

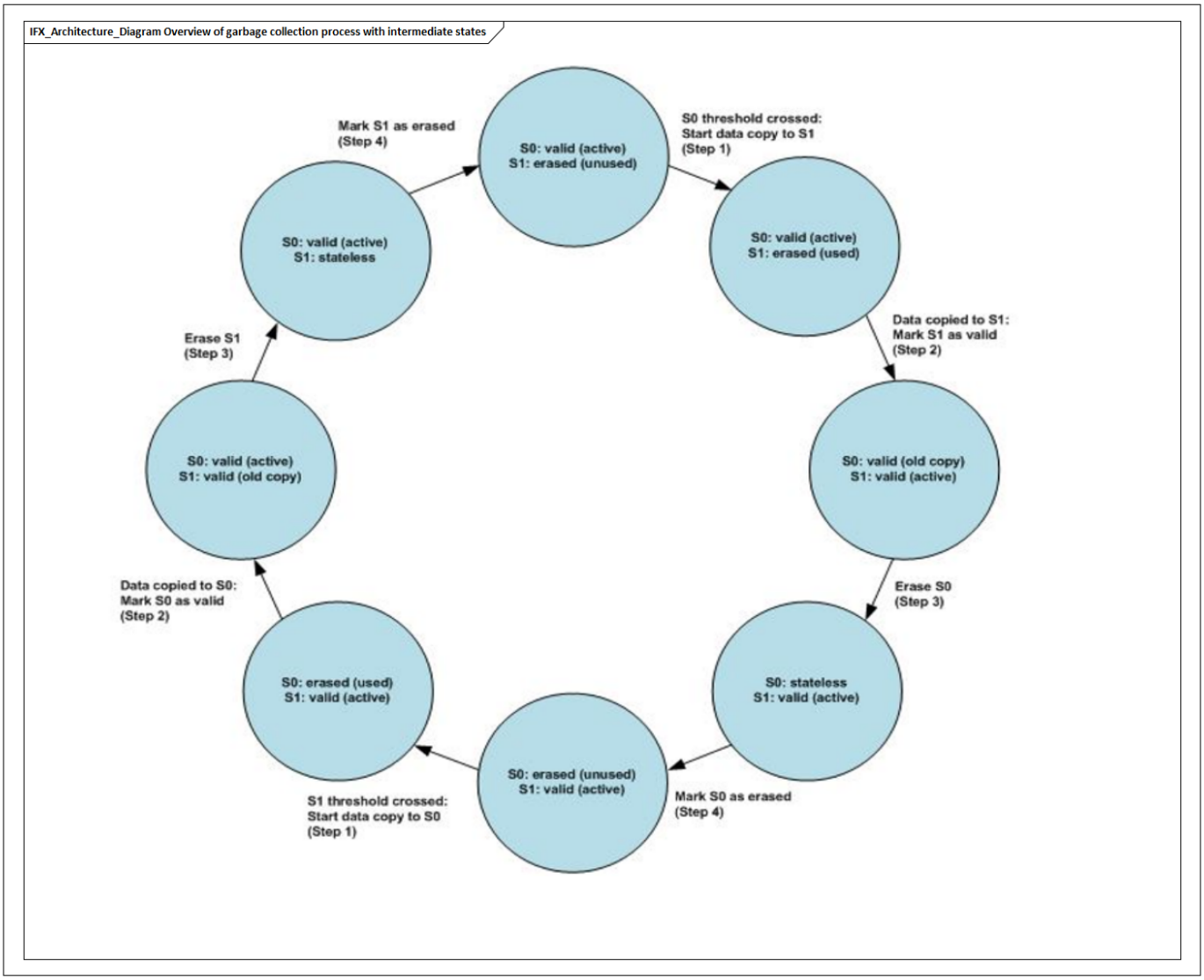

Figure 5 Fee_Overview_of_garbage_collection_process_with_intermediate_states

2.1 Overview of state block structure

The status information of the FEE sector is stored in dedicated set of pages (first 128 bytes of the WL) called State Block. The state block does not have a fixed address within the sector. This concept of “floating state blocks” helps to cope with any WL oriented issues that may occur within the sector.

The status information serves several purposes - It identifies the active sector and the state of the other sector, even after unplanned resets. Additionally, it allows determining the erase count, i.e., the number of FEE sector erasures.

The state block also stores the addresses of up to two un-erasable WLs of the sector that are detected when the sector is erased. These WLs are treated as failed and avoided while performing write operations.

A new state block will be written to a sector either when the sector has been erased or when the sector becomes valid that is, it is used for data storage.

The data content of the State Block has the following format:

1. 8-bit Page Type Identifier which is specific for this type of page and has a big Hamming distance to other Page Type Identifiers. This identifier is required to differentiate between various page types with high safety.

First Page Id: 0x59

Continuation Page Id: 0xC6

Marker Page Id: 0x3A

2. 8-bit Sector State which can be “erased” or “valid”, with a big Hamming distance. This is required to identify the active sector and the state of the other sector, even after unplanned resets.

Sector State “Erased”: 0x1E

Sector State “Valid”: 0xD1

Stored twice for additional reliability. In case of inconsistency, the entire state block will be ignored.

3. 12 bit Offset (in WLs) for quasi-static data check

The offset is used to calculate the address for resuming quasi-static data check and hardening operation. As the check and hardening progresses, the offset value changes. The new offset value is stored back in to the state block so that it can be found on a restart after power interruption. Stored twice for additional reliability. In case of inconsistency, the entire state block will be ignored.

4. 4-bit un-erasable WL Count, which determines the number of un-erasable WLs detected (0 to 2, extendable to 4). This information is required to avoid the use of un-erasable WLs while performing writes and reads.

Stored twice for additional reliability. In case of inconsistency, the entire state block will be ignored

5. 32-bit State Counter which is required to identify the active sector in certain cases after an unplanned reset, and to determine the erase count. It can be assumed that no overflow will occur. When the state is written for the first time, the state counter shall be set to 1. The counter will be incremented when writing a new state. Therefore, two increments are done in a complete cycle for one sector without interrupted GC.

In case of an interruption during the copy process of GC is detected, the state counter will be additionally incremented by 2. If during GC the erase process is interrupted, the state counter will not be additionally incremented. The erase count can be calculated from the maximum state counter value in the state block of both sectors, divided by 2.

Stored twice for additional reliability. In case of inconsistency, the entire state block will be ignored

6. The state block contains the 32-bit addresses of the detected unerasable WLs. The state block can contain 2 unerasable WL address (can be extended to 4 and unused pages (all 0s) will be used to store unerasable WL address). This information is required to avoid the usage of unerasable WLs while performing writes and reads.

Stored twice for additional reliability. In case of inconsistency, the entire state block will be ignored Pages shall have Page Type ID=0xC6 and Unused bytes shall be 0. 2 additional pages are reserved to future use to allow for the possibility to record up to 4 unerasable WL addresses.

7. 4 pages of all-1’s except page id to detect early interrupted erase operation & Pages shall have Page Type ID=0xC6

8. CRC32 field which is the 32 bit CRC value for data from Byte0 to Byte115 to ensure integrity of the State Block, particularly in case of early interrupted erase. Page shall have Page Type ID=0xC6 and special subsequent bytes

9. Additional to the data content, each state block will have an 8-byte marker which is written in a separate operation after verifying the state block data contents. The marker can detect an interrupted write operation of the state block data. The marker pattern shall contain more ones than zeros, i.e., 0x3A, 0xF5, 0xAF, 0xAF, 0xF5, 0xF5, 0xAF, 0xAF.

10. Rest of the State Block data contents i.e., the pages which doesn’t contain WL addresses will have page type id only with no data contents i.e., they will be 0x00s by default.

Remaining pages of the WL containing the State Block will be left unused.

Note: The CRC32 field is based on the standard polynomial defined by IEEE 802.3 CRC32 polynomial with seed value 0. This is calculated using compiler intrinsic which internally uses the Flexible CRC engine implemented within the Aurix TC3X micro controller.

Note: Bitwise little endian ordering is assumed. For example in 'Byte 3' the number of un-erasable WL count is stored in the upper nibble. With reference to byte 43, the number of un-erasable WL count resides in the upper nibble of byte 43.

IFX_Architecture_Diagram State Block

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Page Type ID (0x69)	State	Offset for QS Data Check	Uner. WL Count	L S-Byte	State Counter		M S-Byte
Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
Page Type ID (0xC6)	0	0	0	L S-Byte	0/Unerasable WL 0 Address		M S-Byte
Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
Page Type ID (0xC6)	0	0	0	L S-Byte	0/Unerasable WL 1 Address		M S-Byte
Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	Byte 29	Byte 30	Byte 31
Page Type ID (0xC6)	0	0	0	0	0	0	0
Byte 32	Byte 33	Byte 34	Byte 35	Byte 36	Byte 37	Byte 38	Byte 39
Page Type ID (0xC6)	0	0	0	0	0	0	0
Byte 40	Byte 41	Byte 42	Byte 43	Byte 44	Byte 45	Byte 46	Byte 47
Page Type ID (0xC6)	State	Offset for QS Data Check	Uner. WL Count	L S-Byte	State Counter		M S-Byte
Byte 48	Byte 49	Byte 50	Byte 51	Byte 52	Byte 53	Byte 54	Byte 55
Page Type ID (0xC6)	0	0	0	L S-Byte	0/Unerasable WL 0 Address		M S-Byte
Byte 56	Byte 57	Byte 58	Byte 59	Byte 60	Byte 61	Byte 62	Byte 63
Page Type ID (0xC6)	0	0	0	L S-Byte	0/Unerasable WL 1 Address		M S-Byte
Byte 64	Byte 65	Byte 66	Byte 67	Byte 68	Byte 69	Byte 70	Byte 71
Page Type ID (0xC6)	0	0	0	0	0	0	0
Byte 72	Byte 73	Byte 74	Byte 75	Byte 76	Byte 77	Byte 78	Byte 79
Page Type ID (0xC6)	0	0	0	0	0	0	0
Byte 80	Byte 81	Byte 82	Byte 83	Byte 84	Byte 85	Byte 86	Byte 87
Page Type ID (0xC6)	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
Byte 88	Byte 89	Byte 90	Byte 91	Byte 92	Byte 93	Byte 94	Byte 95
Page Type ID (0xC6)	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
Byte 96	Byte 97	Byte 98	Byte 99	Byte 100	Byte 101	Byte 102	Byte 103
Page Type ID (0xC6)	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
Byte 104	Byte 105	Byte 106	Byte 107	Byte 108	Byte 109	Byte 110	Byte 111
Page Type ID (0xC6)	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
Byte 112	Byte 113	Byte 114	Byte 115	Byte 116	Byte 117	Byte 118	Byte 119
Page Type ID (0xC6)	0x5F	0xFA	0xFA	CRC32			
Byte 120	Byte 121	Byte 122	Byte 123	Byte 124	Byte 125	Byte 126	Byte 127
Marker (0x3A)	Marker (0xF5)	Marker (0xAF)	Marker (0xAF)	Marker (0xF5)	Marker (0xF5)	Marker (0xAF)	Marker (0xAF)

Figure 6 Fee_State_Block

2.2 Overview of NVM Data Block Structure

Each of the data blocks will be stored in DF0_EEPROM as per the following data block structure.

1. First page of the data block is called the header page. Header page structure is as follows:

• 8-bit Page Type Identifier, which is specific for the type of page and has a big Hamming distance to other page type identifiers. Page type identifier is required to differentiate between various page types with high safety.

a) First Page Id: 0xA3

b) Continuation Page Id: 0x9C

c) Marker Page Id: 0x65

• 16-bit Block Number, which identifies the logical block.

• 24-bit Block Cycle Counter, which identifies the instance of the data block and to limit the number of write cycles. When block is written for the first time, its Block Cycle Counter will be set to 1. Each time the block is updated, the counter will be incremented by 1. When the counter reaches the maximum configured number of write cycles, no further writing will occur and the counter will not be incremented any more. Write Cycles Exhausted production error will be raised if enabled. Additionally a job error notification will be called if configured. If the configuration parameter FeeNumberOfWriteCycles is configured as 0, then the data block will always be updated and the block cycle counter will overflow.

• 15-bit Block Page Count, which specifies the number of pages occupied the block in DF0_EEPROM (including the header page and excluding the marker page). It is used to evaluate the block size, particularly for un-configured blocks. As per AUTOSAR 65535 bytes is the maximum block size, hence this can be accommodated within this field.

• 1-bit Valid Information, which specifies whether the block is valid or is it invalidated by the user.

2. Following pages will have 1-byte Page Type Identifier (0x9C) followed by 7-byte data of the logical block. At the end of the block if there are bytes remaining until the end of the page, then they will be filled with 0x00s called stuff bytes.

3. Last page of the data block is the marker page, which marks the completion of data block programming. The marker page also contains 16-bit Block Number, 1-Bit Block validity information and 15-Bits Number of Pages occupied by the logical block, which are described in the header page section. Marker pattern of the first 4 bytes of the marker page: 0x65, 0xAF, 0xF5, 0xF5.

Data block is considered as inconsistent in case of [i] missing pages of the data block or [ii] missing (0x00s) marker page. Data block with a valid marker is considered as a consistent block.

In case of missing marker page i.e., expected marker page containing 0x00s for the last written block, the page where the marker was expected, will be left untouched and its following page is considered as the empty page for future writes. This is to avoid the possibility of treating the block as consistent by treating the failed programming of next block's header page as non-zero marker.

Upon block invalidation request, only 2 pages will be written that is, the header and the marker page, irrespective of the size of the data block being invalidated. Header page with "Valid" bit (MSB of byte 7) as 0 and Number of data block pages as 1 (header and marker page).

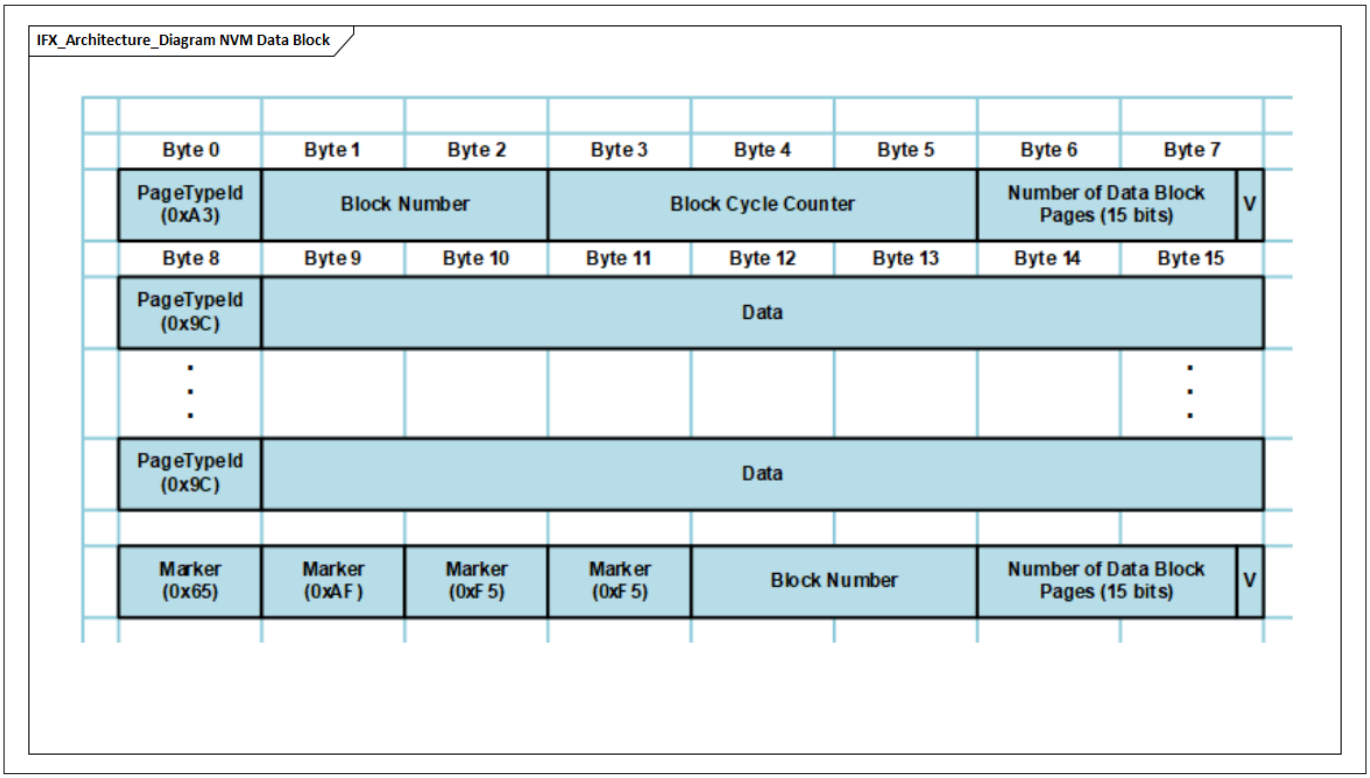

Figure 7 Fee_NVM_Data_Block

2.3 Overview of garbage collection process

Garbage collection is triggered when a write request to an NVM data block crosses the threshold or an immediate write request crosses the threshold. The garbage collection algorithm performs the sequence of steps listed below. As an initial stage it is assumed that sector 0 is active, the filling threshold has been crossed and sector 1 is erased.

1. Copy the most recent version of all logical blocks from sector 0 to sector 1. Invalidated blocks are copied as well but not inconsistent blocks. This copy phase may be interrupted by read/write/cancel_all requests.
2. Mark the correct end of the copy procedure by programming the state of sector 1 to valid and increment the state counter. From now on, sector 1 is active.
3. If QS Blocks are configured then perform check and hardening operation on QS Memory region.
4. Now sector 1 can be filled with data after pending data have been written.
5. Erase sector 0 and verify the erase operation and check for un-erasable WLs.
6. Mark the correct end of the erase procedure by programming the state of sector 0 to erased, update the un-erasable WL information and increment the state counter.

When sector 1 reaches the threshold, the same sequence will start with step 1 and interchanged roles of the sectors.

See also the following figures:

- *Overview of garbage collection process with intermediate states*;
- *Example of sector change from S0 to S1, caused by normal write request*;
- *Example of sector change from S1 to S0, caused by immediate write request.*

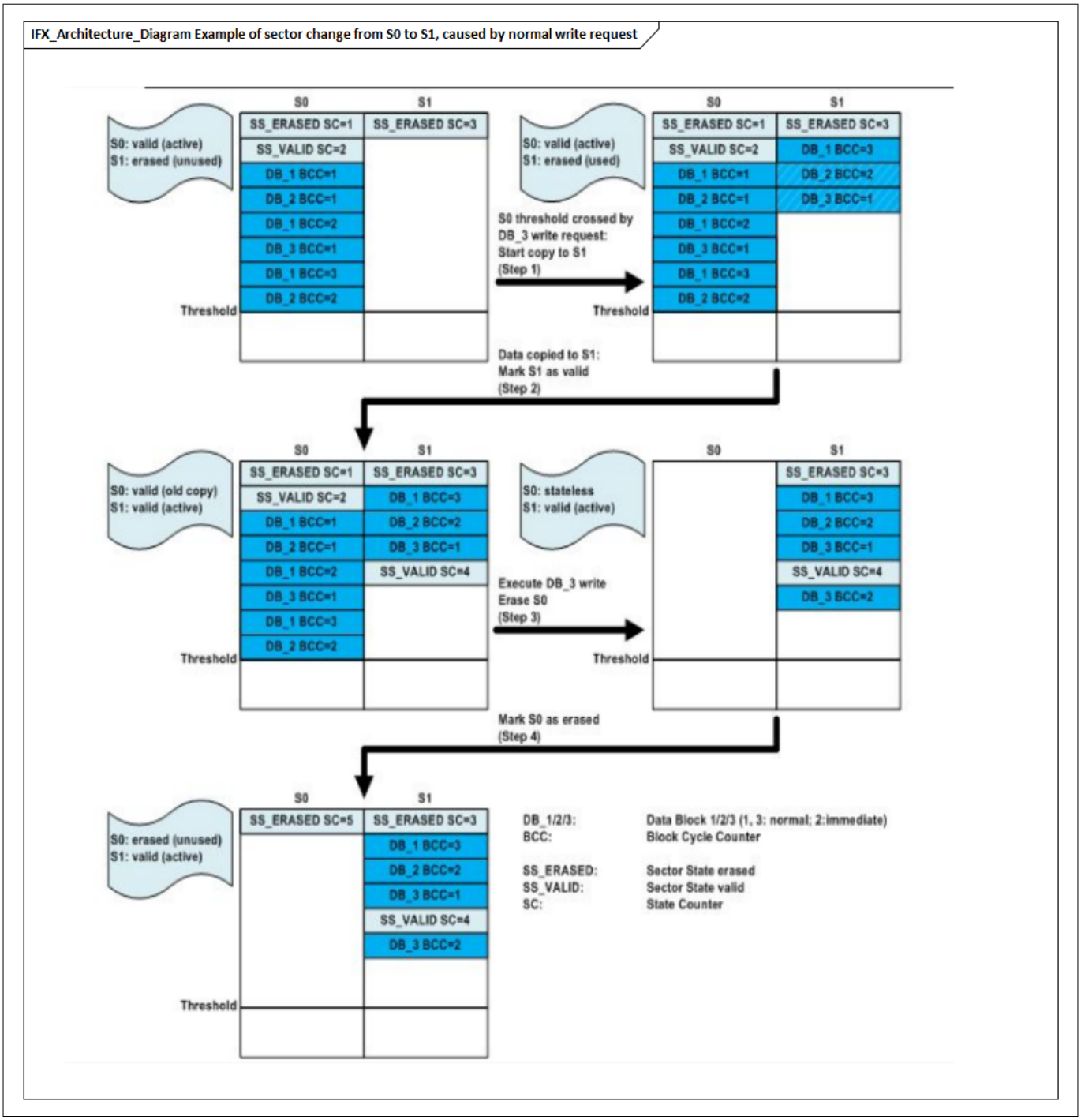

Figure 8 Fee_Example_of_sector_change_from_S0_to_S1,_caused_by_normal_write_request

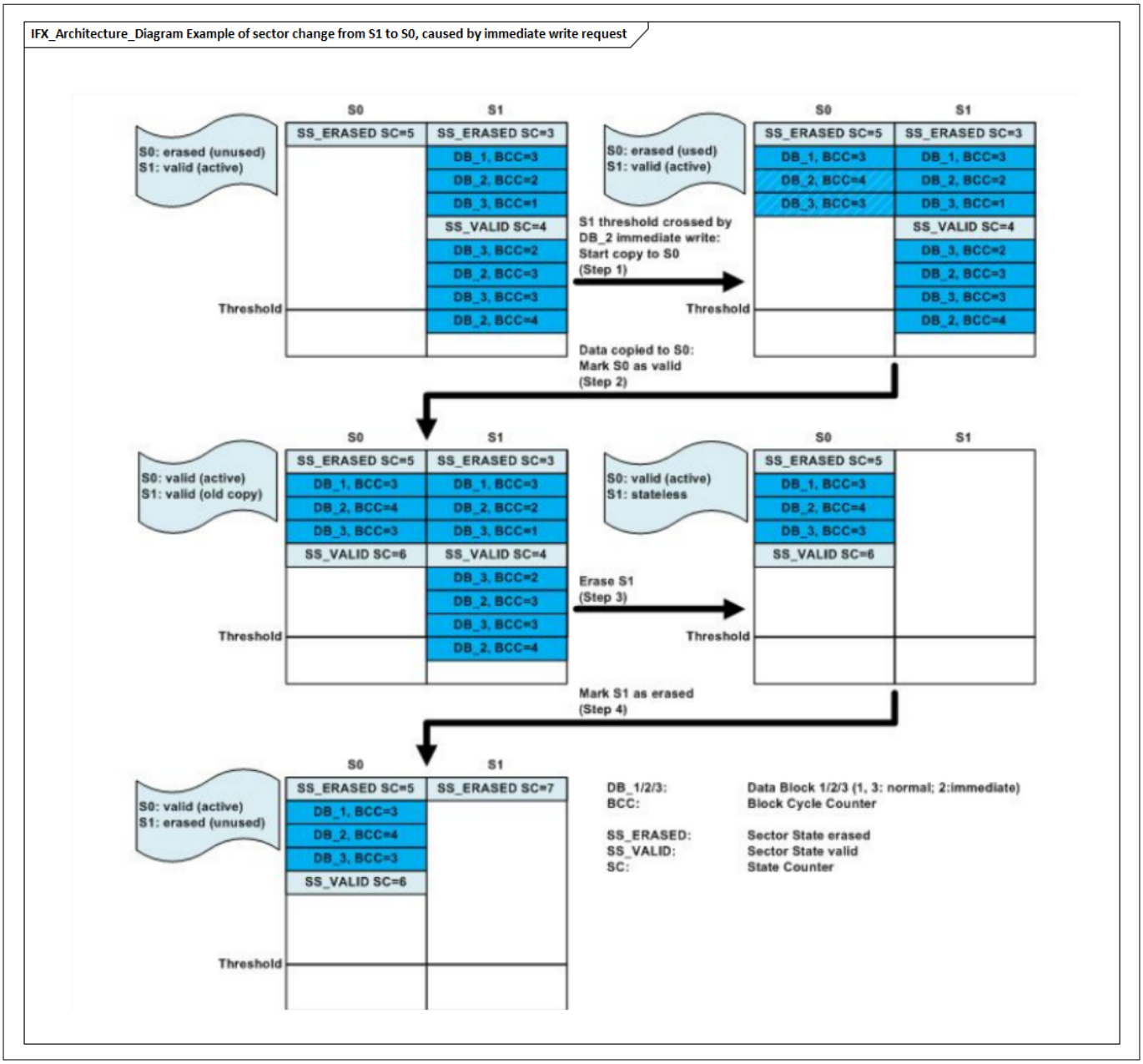

Figure 9 **Fee_Example_of_sector_change_from_S1_to_S0,_caused_by_immediate_write_request**

3 Quasi Static Algorithm

The standard EEPROM emulation algorithm is not efficient for relatively big data blocks with a low update rate (500 erase/write cycles over the entire QS data area). Therefore, in addition to the standard double-sector EEPROM emulation algorithm, a special algorithm for infrequently updated data blocks (called Quasi-Static data) is made available. The algorithm handling quasi-static data writes, reads and erases data blocks located in a separate area of the DFlash0. Quasi-static data is written to a separate area of the DFlash0.

3.1 Quasi static data block structure

Status and data information of a quasi-static data block is stored in the same area. The minimum erasable size, that is 4KB logical sector is used, or a multiple of that for larger data blocks.

The information is stored at fixed addresses given by configuration, without using an identifier.

The status information is contained only in the first mini sector; no CRC protection is needed.

If CRC protection is required for user data, this shall be handled by the application software.

The Quasi Static data block has the following contents:

The Quasi-Static data block consists of a 4 page header that helps determine the state of the block.

• State “erase complete”:

− Indicates whether erase is complete

− Shall be repaired after interrupted write of state (may be treated as a kind of marker)

- If the erase complete state marker repair fails, then the block status will be declared as invalid. The user can choose to either read the data block or erase it.

• State “write started”:

− Indicates whether data writing has been started

− Shall not be repaired after interrupted write of state

• State “write complete”:

− Indicates whether data writing is complete

− Shall be repaired after interrupted write of state (may be treated as a kind of marker)

- If the write complete state marker repair fails, then the block status will be declared as invalid. The user can choose to either read the data block or erase it.

• State “erase started”:

− Indicates whether erasing of data block has been started

− Shall not be repaired after interrupted write of state

- Marker will be written but not verified and re-programmed because the block will be erased.

A page with write or erase started contains

0x7777888811112222U

A page with erase or write complete contains

0x5555666633334444U

• Block Cycle Counter (BCC) and BCC complement:

− Set to maximum of valid BCC values available for all instances of this data block plus 1, or set to 1 if no BCC value is known (example first initialization)

− Indicates sequence and number of write operations for the whole block

− Stored also as complemented value to increase reliability

• Data:

− User data bytes

− Unused bytes in last programmed page filled with 0x00 (stuffing bytes)

• Unused Pages:

− Any unused pages of the QS data block remain as erased pages (all zeros).

The following is the block structure used for Quasi-Static data.

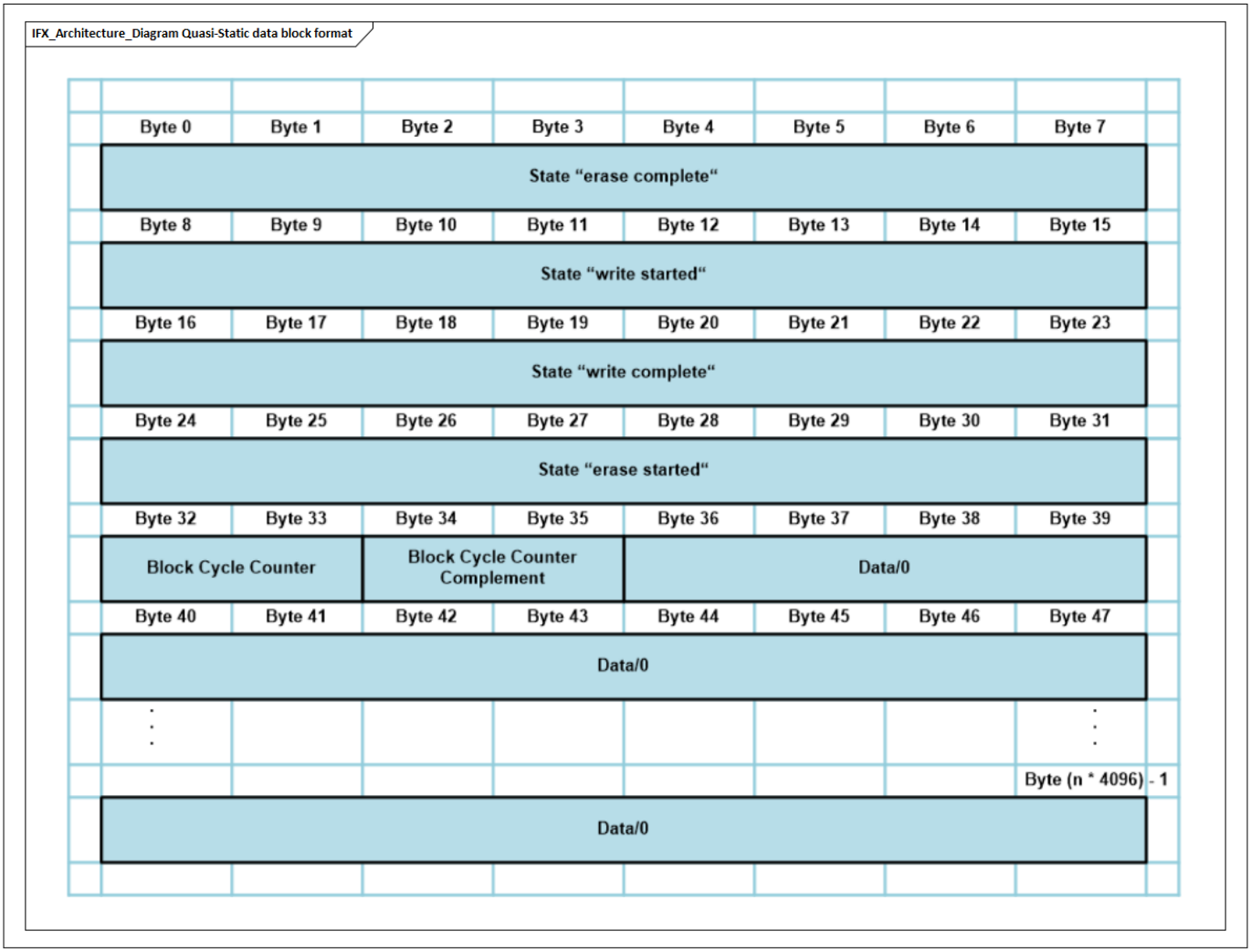

Figure 10 Fee_Quasi-Static_data_block_format

3.2 Sequence of States

Initially, state “erase complete” shall be set. All other states shall be 0 (erased), as well as data information of the block.

Write

On a write request, first the state “write started” is set, then BCC and data are written and finally the state “write complete” is set.

Thus an interrupted write operation can be safely detected by state evaluation; see chapter "Initialization after Reset".

Erase

After an erase request, it is important to avoid problems due to an undetected interrupted erase. Therefore, several steps are taken before the erase is started:

- Set state “erase started”;
- Destroy states “erase complete”, “write started” and “write complete” by setting all state bits to ‘1’.

Basically, state “erase started” is not needed. This state is effective only in the same ignition cycle before reset happens.

After successful erase, state “erase complete” is set.

Thus an interrupted erase operation can be safely detected by state evaluation; see chapter "Initialization after Reset".

The sequence of states for a quasi-static data block instance is shown in the Figure.

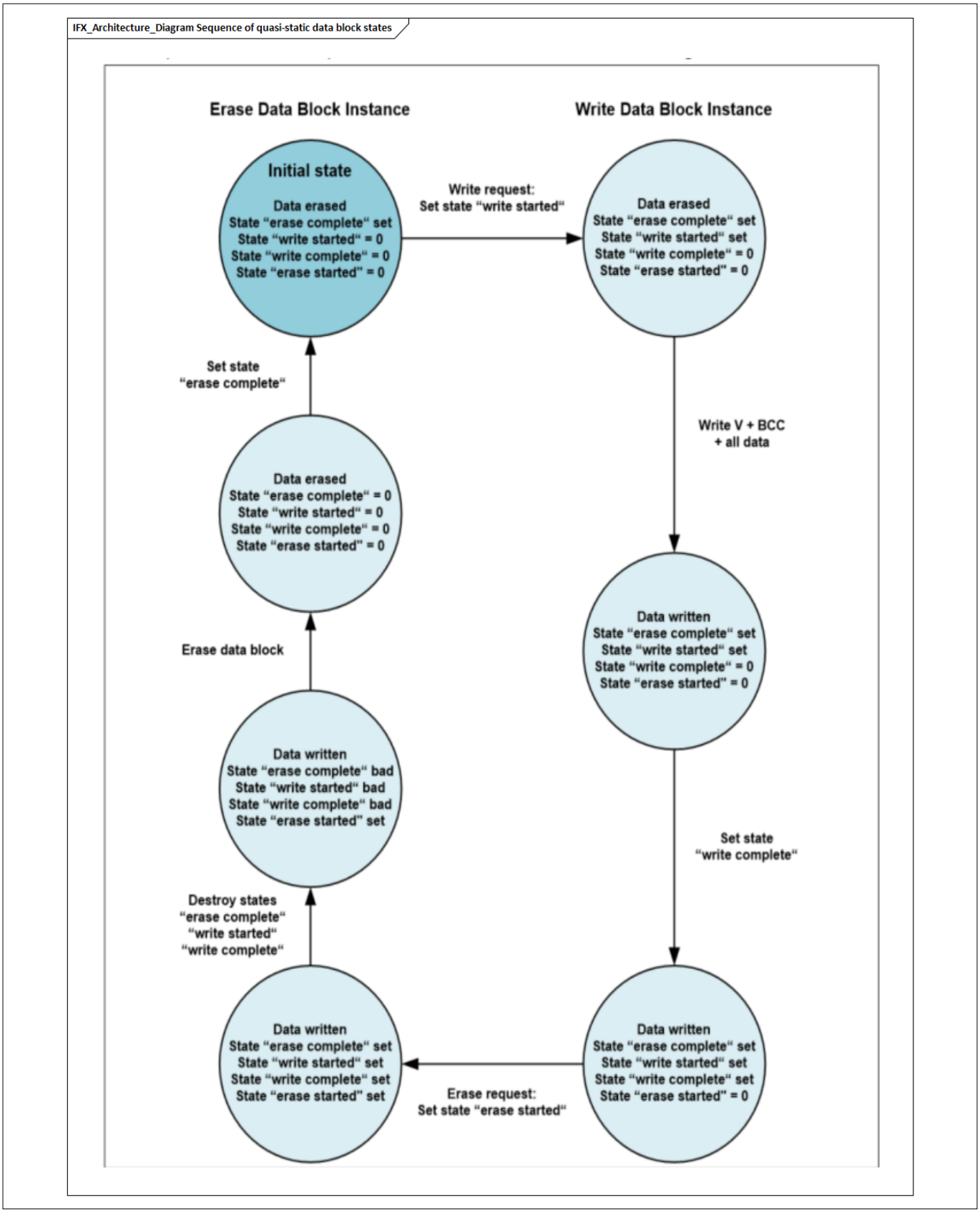

Figure 11 Fee_Sequence_of_quasi-static_data_block_states

3.3 Example - Handling Quasi-Static data blocks

Quasi-Static data blocks can be configured to have one or more instances.

The following figure shows how three different types of Quasi-Static data blocks can be handled by the application software.

The QS data blocks have different attributes:

Data block N Area:

Some old block instances are maintained, and some block instances are free for urgent write operations.

- Data block size: 4 KB
- Old data block instances to be maintained: 6
- Data block instances to be kept free: 3

Data block N+1 Area:

Only one block instance will always be available, like with standard EEPROM emulation. The other block instance has to be cleared before new data can be written.

- Data block size: 16 KB
- Old data block instances to be maintained: 0
- Data block instances to be kept free: 1

Data block N+2 Area:

In this corner case, there is only one block instance which has to be cleared before new data can be written. This makes sense if no backup is required, for example, calibration data.

- Data block size: 12 KB
- Old data block instances to be maintained: 0
- Data block instances to be kept free: 0

In this example, data is updated by writing the data block to a kind of ring buffer. No more needed contents are erased if required.

Reading, writing and erasing of a data block instance is triggered by the application software. Important write operations can be prioritized.

Each instance of a data block has a specific block number. Therefore, the existing API services for reading and writing can be used to access the data block instances individually.

The Block Cycle Counter (BCC) is incremented by the FEE software on each write. It can be used to identify the sequence and the total number of write operations in a block area.

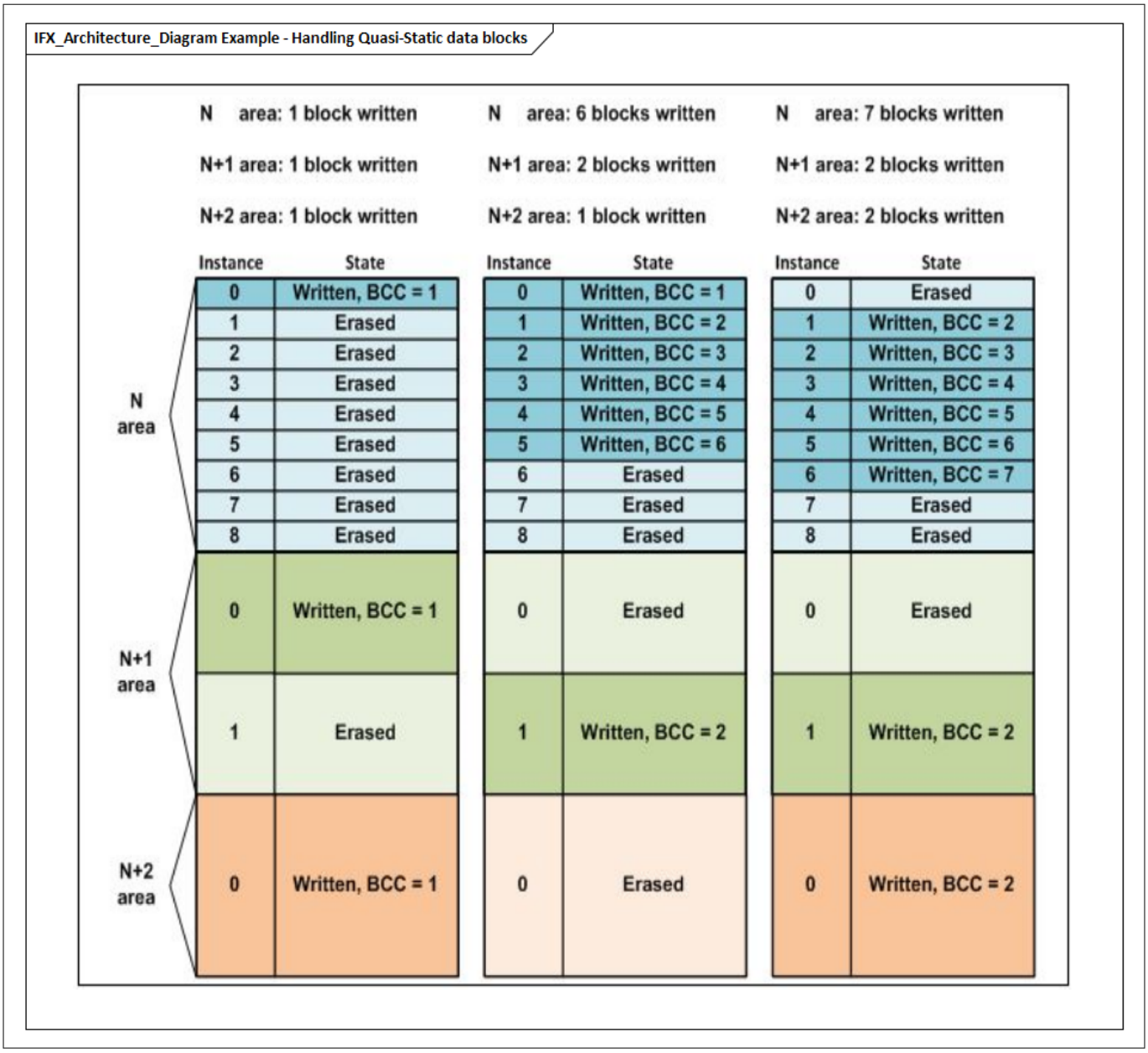

Figure 12 Fee_Example_-_Handling_Quasi-Static_data_blocks

4 Conditions to raise production error and illegal state notifications

IFX_Architecture_Diagram Condition for raising illegal state notification

Table 2: Condition for raising illegal state notification

Condition	Action taken
Init GC fail in following phase: 1. Erase of sector as a part of DFlash preparation /interrupted failed and found more than 2 un-erasable WLs. 2. Writing of state block as a part of DFlash preparation /interrupted failed twice.	Raise the NVM illegal state notification and FEE_E_GC_INIT production error to Mcal_Wrapper.
During GC write of block (configured/un-configured) failed twice	Raise the NVM illegal state notification and FEE_E_GC_WRITE production error to Mcal_Wrapper.
During GC erase of sector failed and found more than 2 un-erasable WL.	Raise the NVM illegal state notification and FEE_E_GC_ERASE production error to Mcal_Wrapper.
During GC when there is no space in the new sector to write valid state block or pending requested block.	Raise the NVM illegal state notification and FEE_E_GC_TRIG production error to Mcal_Wrapper.
While writing use requested block WL failure happens and old written blocks in that WL are affected, so, while re-writing affected block read/write failure occurred.	Raise the NVM illegal state notification and FEE_E_WRITE production error to Mcal_Wrapper.
During initialization virgin state of Flash (NVM/QS) is detected and handling of virgin Flash is disabled.	For NVM: Raise the NVM illegal state notification and FEE_E_GC_INIT production error to Mcal_Wrapper. For QS: Raise the QS illegal state notification.
Handling of QS virgin Flash failed.	Raise QS illegal state notification.
Resume of erase operation is failed.	For NVM: Raise the NVM illegal state notification and FEE_E_GC_ERASE production error to Mcal_Wrapper. For QS: Raise the QS illegal state notification.

Figure 13 Fee_Condition_for_raising_illegal_state_notification

IFX_Architecture_Diagram Conditions to raising production error

Table 1: Conditions to raising production error

FEE_E_GC_TRIG	1.The garbage collection process itself triggers another garbage collection (due to wrong configuration, expected number of WL failures etc.) 2. Erase-suspend feature is enabled and request occurs while sector is being erased 3. During GC when there is no space in the new sector to write valid state block or pending requested block.
FEE_E_GC_READ	1. While reading a block during the copy phase, an un-correctable ECC error is encountered. 2. Configured and un-configured blocks containing ECC errors will be dropped during garbage collection
FEE_E_GC_WRITE	1. During GC, a (configured or un-configured) block was read successfully but failed to be written to the other sector due to a word line failure. 2. During GC write of block (configured/un-configured) failed twice.
FEE_E_GC_ERASE	1.If the resume of a suspended erase operation fails due to hardware reasons, then the following approached shall be taken: For NVM, resume shall be attempted up to 4 times. If resume was not successful then illegal state notification and GC_ERASE production error is raised. For QS, if the resume attempt fails, then there shall be no further attempts made to resume the erase. Illegal state notification and GC_ERASE production error is raised. 2. During GC erase of sector failed and found more than 2 un-erasable WL. 3. Resume of erase operation is failed.

Figure 14 Fee_Conditions_to_raising_production_error

IFX_Architecture_Diagram Conditions to raise production error Contd

FEE_E_GC_INIT	Init GC fail in following phase: 1. Erase of sector as a part of DFlash preparation /interrupted failed and found more than 2 un-erasable WLs. 2. Writing of state block as a part of DFlash preparation /interrupted failed twice. 3. During initialization virgin state of Flash (NVM/QS) is detected and handling of virgin Flash is disabled.
FEE_E_INVALIDATE	Failure during the Block invalidate
FEE_E_READ	Failure during the Block read
FEE_E_UNCONFIG_BLK_EXCEEDED	If the sum of configured block count and un-configured block count exceeds the maximum block count (FeeMaxBlockCount)
FEE_E_WRITE	While writing use requested block WL failure happens and old written blocks in that WL are affected, so, while re-writing affected block read/write failure occurred.
FEE_E_WRITE_CYCLES_EXHAUSTED	Error due to exceeding the configured limit of the write cycles for the given block.

Figure 15 Fee_Conditions_to_raise_production_error_Contd

Revision History

Major changes since the last revision.

Date	Version	Description
18-09-2024	1.0	Released
08-08-2024	0.1	For Inspection

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-09-18

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
IFX-ocr1484806431059

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, 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.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.