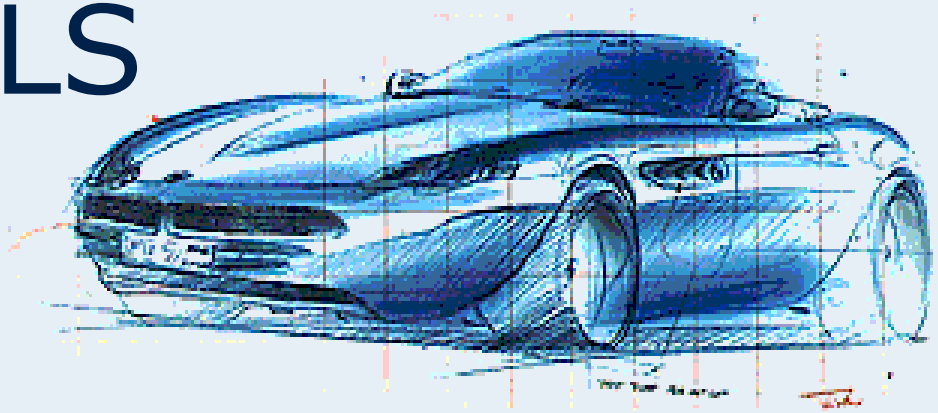
Infineon AUTOSAR

AURIX FEE / FLS



Autosar 4.x

AURIX

ATV MC ACE PDA 32Bit, Stephan Barnikol

IFX_AURIX_AUTOSAR_FEE_TRW_1_1

Never stop thinking

Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- MCAL FEE Data Page Format
- MCAL FEE Garbage Collection
- MCAL FEE Power Failure Scenario – State Page
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- MCAL FEE Data Page Format
- MCAL FEE Garbage Collection
- MCAL FEE Power Failure Scenario – State Page
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

10.2.2.2 Data Flash Features

Summary of Data Flash features:

- Contains 48 EEPROMx sectors commonly used for EEPROM emulation (data storage at application run-time).
- Contains 8 HSMx sectors used by HSM for EEPROM emulation protected from application access.
- UCBx sectors used for protection installation and the erase-cycle counter.
- Read-only UCB configured by IFX with unique chip identifier and trimming data.
- Password based read protection combined with write protection.
- Flash read access based on 64-bit reads (see Data Sheet WS_{DF}).
- Read path separated from PFlash (DFlash reads don't influence PFlash read accesses).
- Burst read is not supported.
- Separate command interface (command interpreter, status register) for HSM.
- Requested read (data reading performed in background, read data supplied in registers).
- Fast programming of 8 byte pages (see Data Sheet t_{PRD}).
- High throughput burst programming of 32 byte units (see Data Sheet t_{PRDB}).
- Erase time per sector: see Data Sheet t_{ERD}.
- High throughput erase by multi-sector erase commands: see Data Sheet t_{MERD}.
- Erase and program performed by a Flash specific control logic independent of the CPU.
- Fast suspend erase to read command.
- End of erase and program operations reported by interrupt.
- Dynamic correction of single-bit, double-bit and triple-bit errors and detection of quad-bit errors ("TEC-QED").
- Error reporting to the SMU, additionally local status flags and bus error generation.
- Margin reads for quality assurance.
- Delivery in the erased state.
- Configurable wait-state configuration (see Data Sheet WS_{DF}).
- The high endurance (see Data Sheet N_E) is granted under the condition of a robust EEPROM emulation algorithm (see PMU chapter).
- Endurance and retention figures are documented in the Data Sheet.
- Pad supply voltage used for program and erase.

10.8.2.1 Robust EEPROM Emulation

A key requirement for an EEPROM emulation algorithm is the reliability of the stored data. The DFlash with its TEC-QED ECC algorithm protects perfectly against bit or bit-line oriented failures. However word-line oriented failures need to be handled by the EEPROM emulation algorithm.

The following hints shall be followed to achieve highest possible robustness:

- Before programming a page save the content of all other pages on the same word-line that contain active data to SRAM.
- Program the new page and compare the content of this page and of the saved pages with their reference data. This can be done with normal read margins. Ignore correctable bit-errors and the program VER flag.
- If the data comparison fails program this page and the saved content of the other pages to a different word-line.

- This procedure can be repeated if the data comparison fails again. The number of repetitions should be limited (e.g. to 3) in case the programming fails because of out-of-spec operating conditions.
- Word-line oriented fails can also have the effect that the affected word-lines can not be erased anymore (other word-lines stay fully functional). A robust EEPROM emulation is immune against such word-lines (e.g. by identifying old data by version counters).

For the TC27x this robust EEPROM algorithm is required for the usage of the DFlash.

Due to the specificity of each application the appropriate usage and implementation of these measures (together with the more elaborate VER handling) must be chosen according to the context of the application.

AURIX HW Features supporting FEE Emulation

- DFLASH Page Programming
  - Programming time significantly reduced : 140 us
    - much lower risk of interruptions of page programming operations
  - Small capacitances can easily ensure stable operating condition during programming jobs
    - no risk of interruptions of page programming operations
- Reset Concept
  - 100us operation between reset request and its activation
- DFLASH ECC
  - ECC detection capability significantly increased
    - Triple error correction
    - Quad error detection

Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- MCAL FEE Data Page Format
- MCAL FEE Garbage Collection
- MCAL FEE Power Failure Scenario – State Page
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

Features of FEE Algorithm

- Standard SW solution
  - Follow AUTOSAR requirements
    - Usage of AUTOSAR Interfaces e.g. Handling of immediate data, Error Signaling (DEM, DET)
    - Configuration Concept
- Handling of Interruptions
  - Detection of not fully programmed data blocks/pages
  - Detection of not fully erased sector
  - SW shall recover after interruption caused by Hardware (Reset, Power loss)
- SW adaptation to Flash Hardware
  - Use HW mechanisms for Error Detection (ECC)
  - Detection/Handling of Wordline oriented failures
  - Strict checking of programmed data (tight0 margin check for data and state pages)

Copyright © Infineon Technologies 2013. All rights reserved.

FEE MCAL Overview

- Double Sector Algorithm
  - has to use all logic sectors round-robin in order to prevent the accumulation of erase disturbs in static sectors
  - Selection of equal sector sizes
    - TC27x: 192 kByte sector size (24 logical sectors)
- Direct State Page Format
- Floating State Pages
- Resistance against Resets and Power Failures
- Robustness against Word-Line Issues

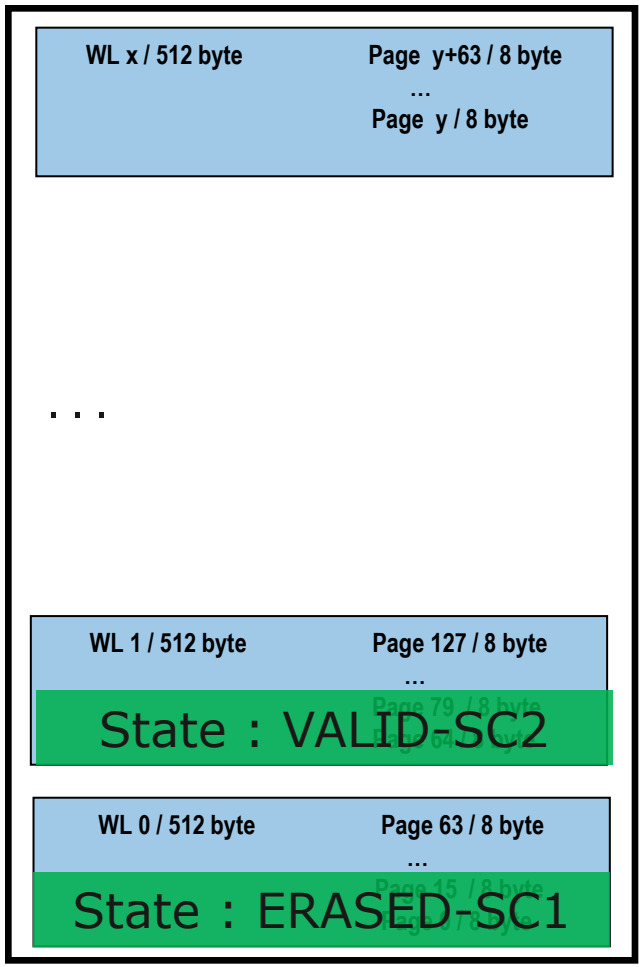
Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- **MCAL FEE State Page Format**
- MCAL FEE Data Page Format
- MCAL FEE Garbage Collection
- MCAL FEE Power Failure Scenario – State Page
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

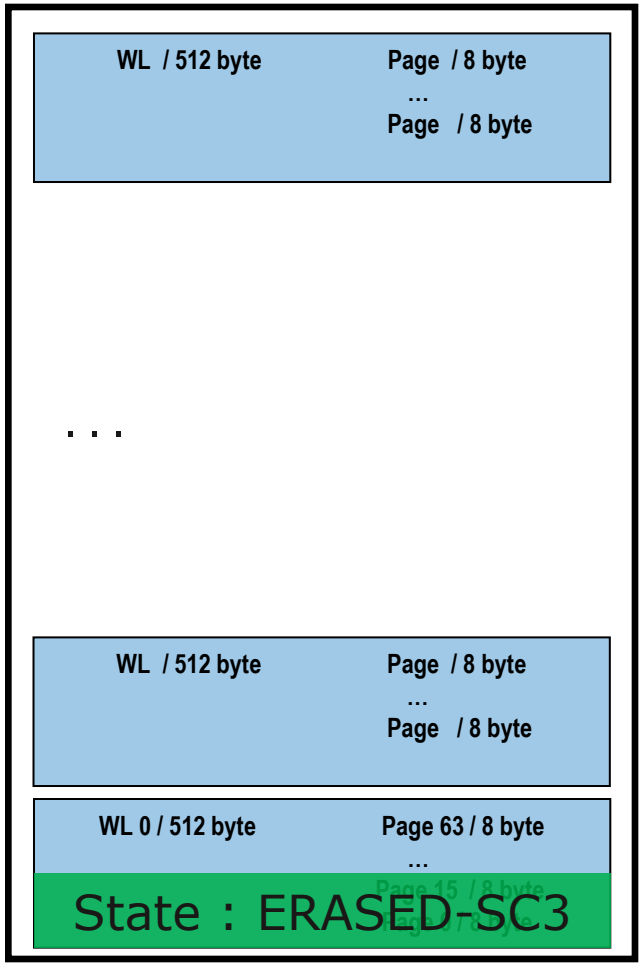
FEE MCAL Overview

- State Page Format
  - Status information of FEE sector is stored in dedicated set of first 16 pages of a WL called ‘State Page’. A state page effectively consists of 16 pages = 128 bytes.
  - Remaining pages of WL containing ‘State Page’ are not further used. A state page effectively uses 64 pages = 512 bytes.
  - State Page has no fixed address – floating state page concept.
  - 2 State Pages typically used per sector.
  - Logical content of State Pages :
    - Sector is erased
    - Sector is valid

TC27x FEE Driver
Initial Status

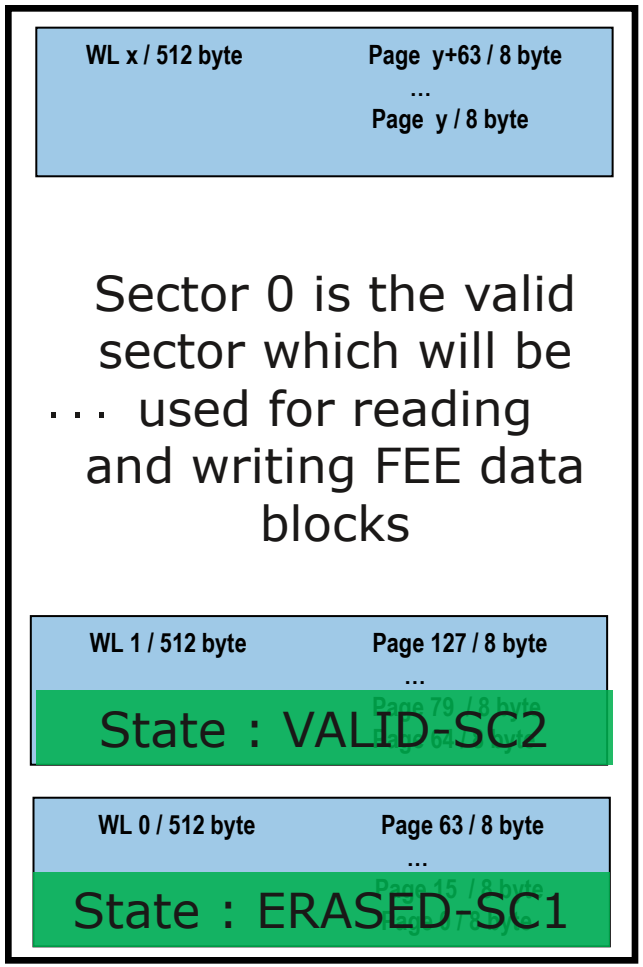


FEE Sector 0 / 192 kByte

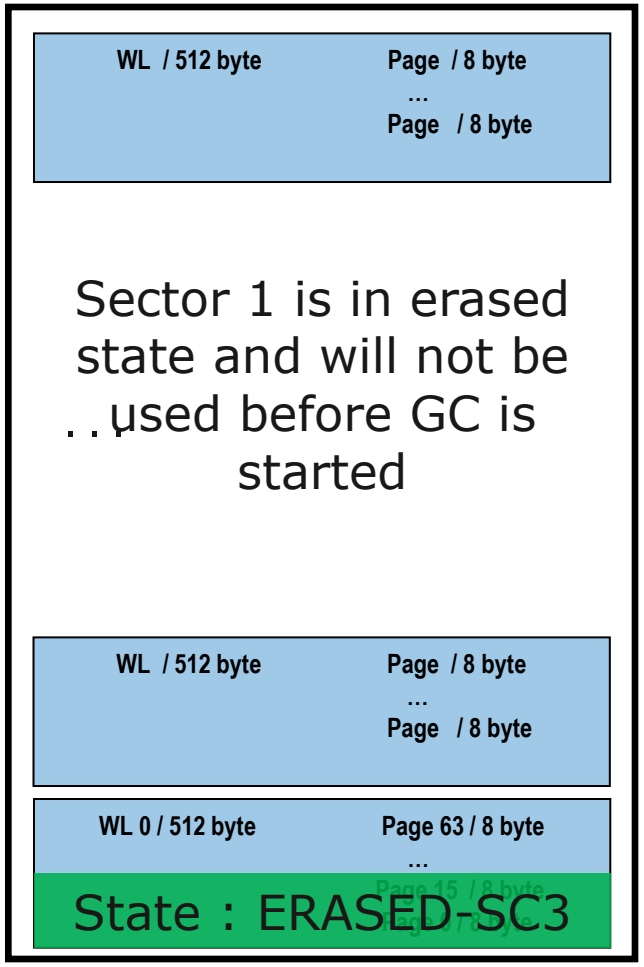


FEE Sector 1 / 192 kByte

TC27x FEE Driver
Initial Status



FEE Sector 0 / 192 kByte



FEE Sector 1 / 192 kByte

FEE MCAL Overview

- State Page Format
  - Status information of FEE sector is stored in dedicated set of first 16 pages of a WL called ‘State Page’
  - State Page has no fixed address

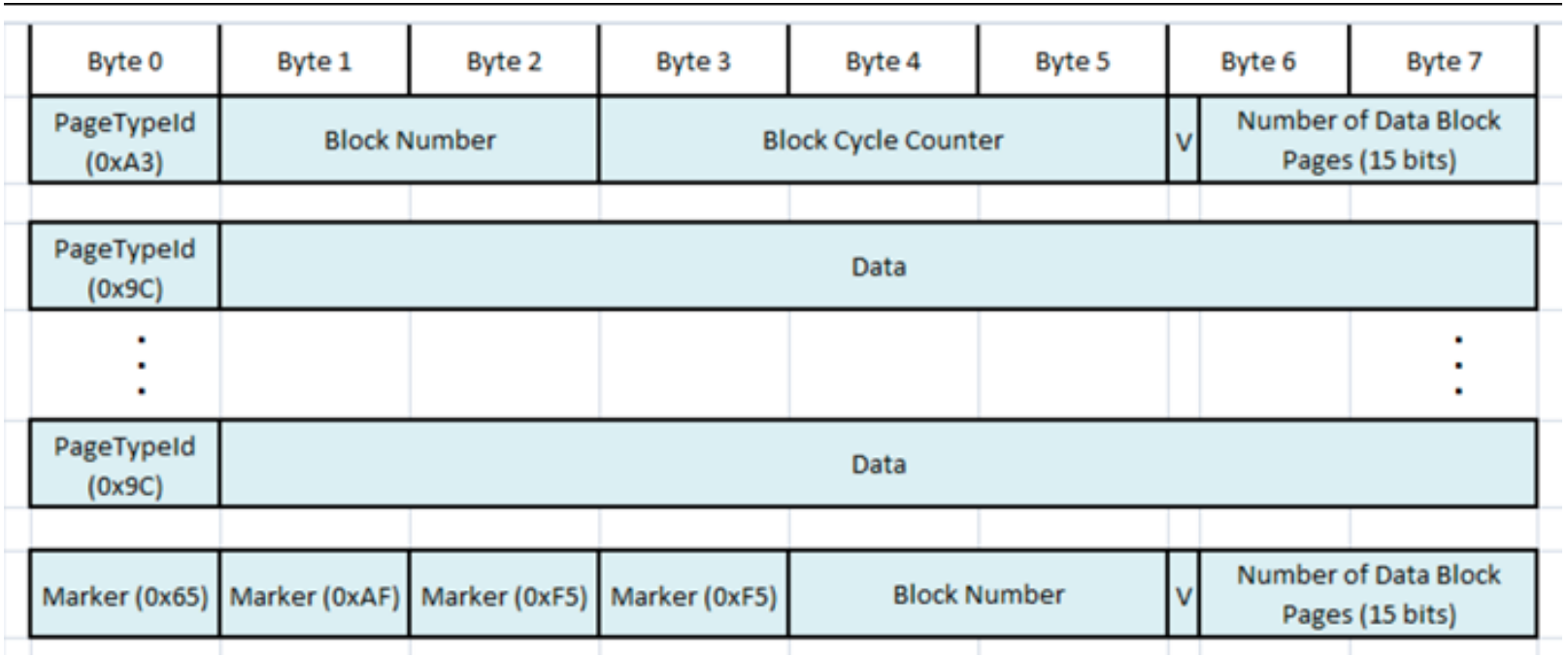
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
PageTypeId (0x59)	State	Unerasable WL Count	0	LS-Byte	State Counter		MS-Byte
Byte 8	Byte9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
PageTypeId (0xC6)	0	0	0	LS-Byte	Unerasable WL 1 Address		MS-Byte
⋮							⋮
PageTypeId (0xC6)	0	0	0	LS-Byte	Unerasable WL N Address		MS-Byte
0	0	0	0	0	0	0	0
⋮							⋮
Byte 112	Byte 113	Byte 114	Byte 115	Byte 116	Byte 117	Byte 118	Byte 119
0	0	0	0	0	0	0	0
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)

Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- **MCAL FEE Data Page Format**
- MCAL FEE Garbage Collection
- MCAL FEE Power Failure Scenario – State Page
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

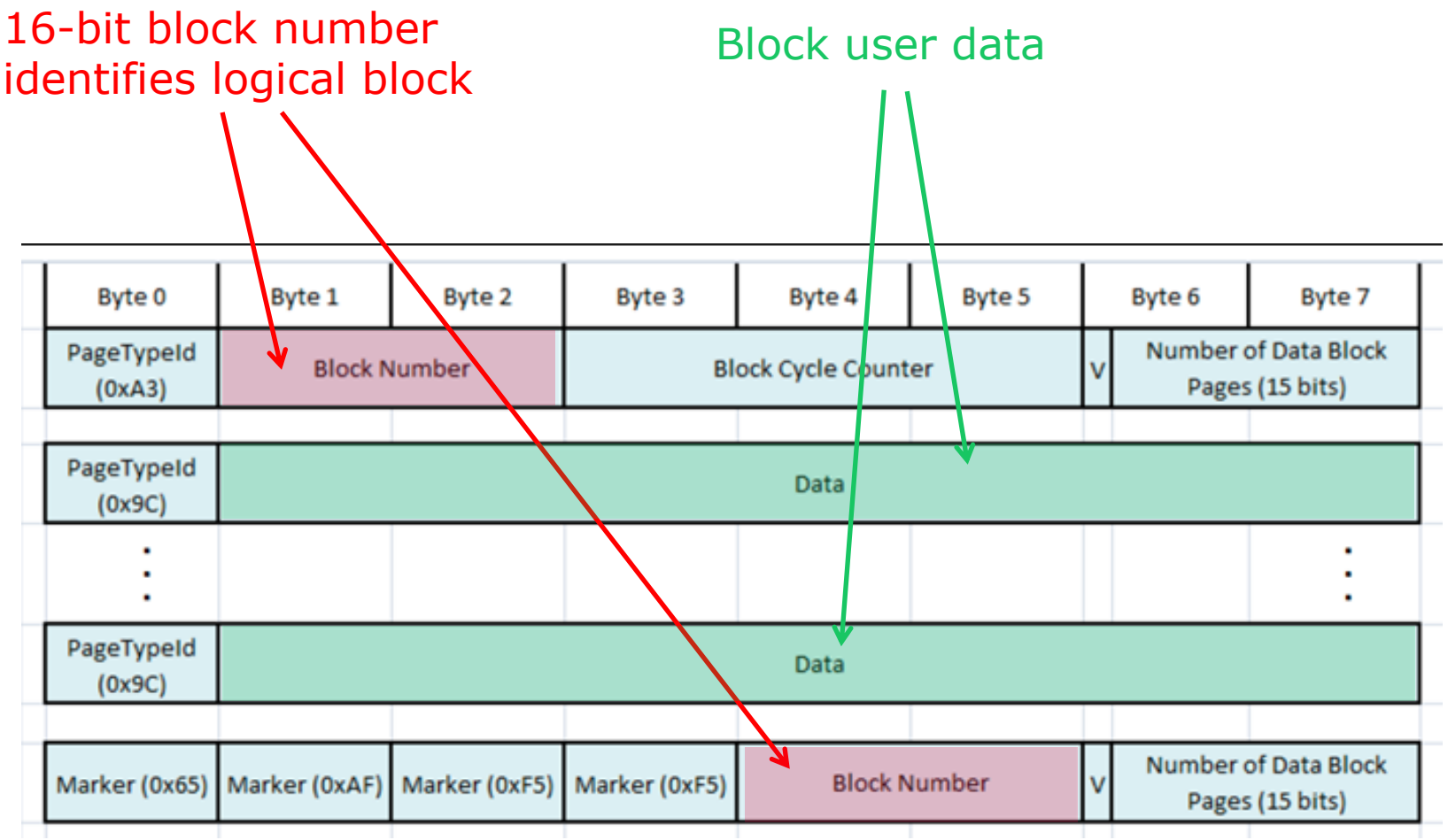
FEE MCAL Overview

- Data Block Format
  - first page : header page with ID 0xA3
  - cont. pages : data pages with ID 0x9C
  - marker page : last page with ID 0x65



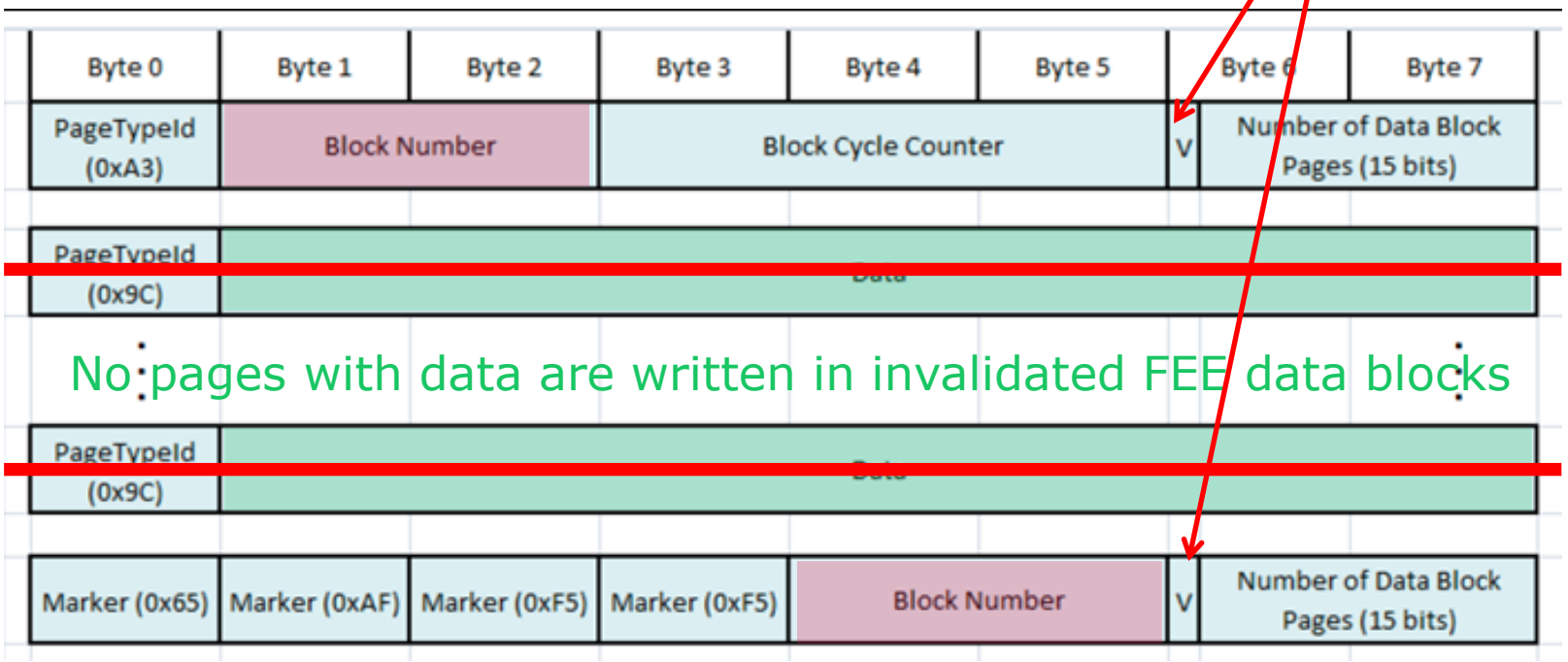
FEE MCAL Overview

- Data Block Format



FEE MCAL : Block Invalidation

- FEE data blocks can be invalidated with FEE API function Fee_InvalidateBlock()
- FEE data block is written with validation bit V=0
- No data inside invalidated data blocks
- 16 byte required for invalidation

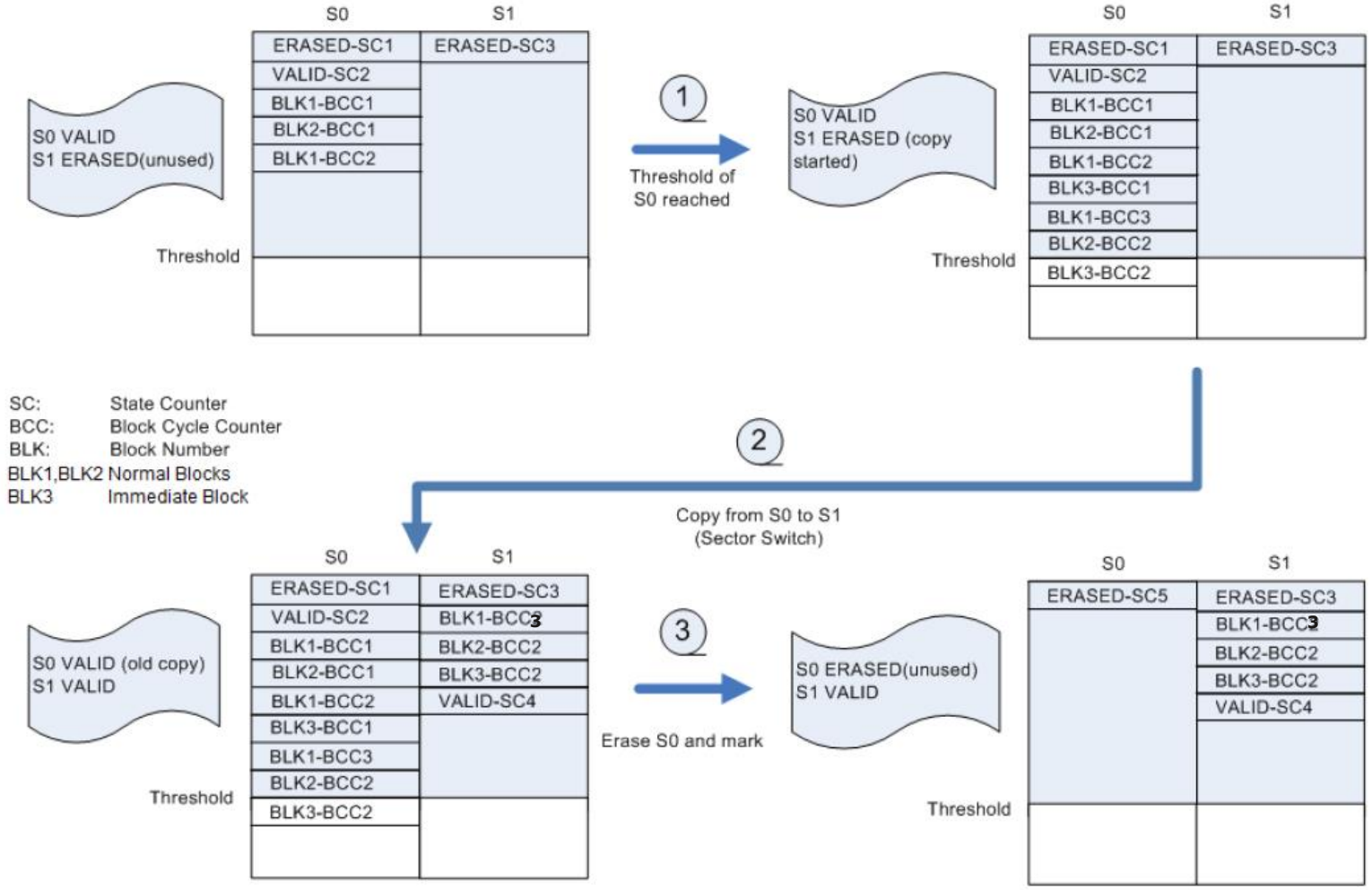


Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- MCAL FEE Data Page Format
- **MCAL FEE Garbage Collection**
- MCAL FEE Power Failure Scenario – State Page
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

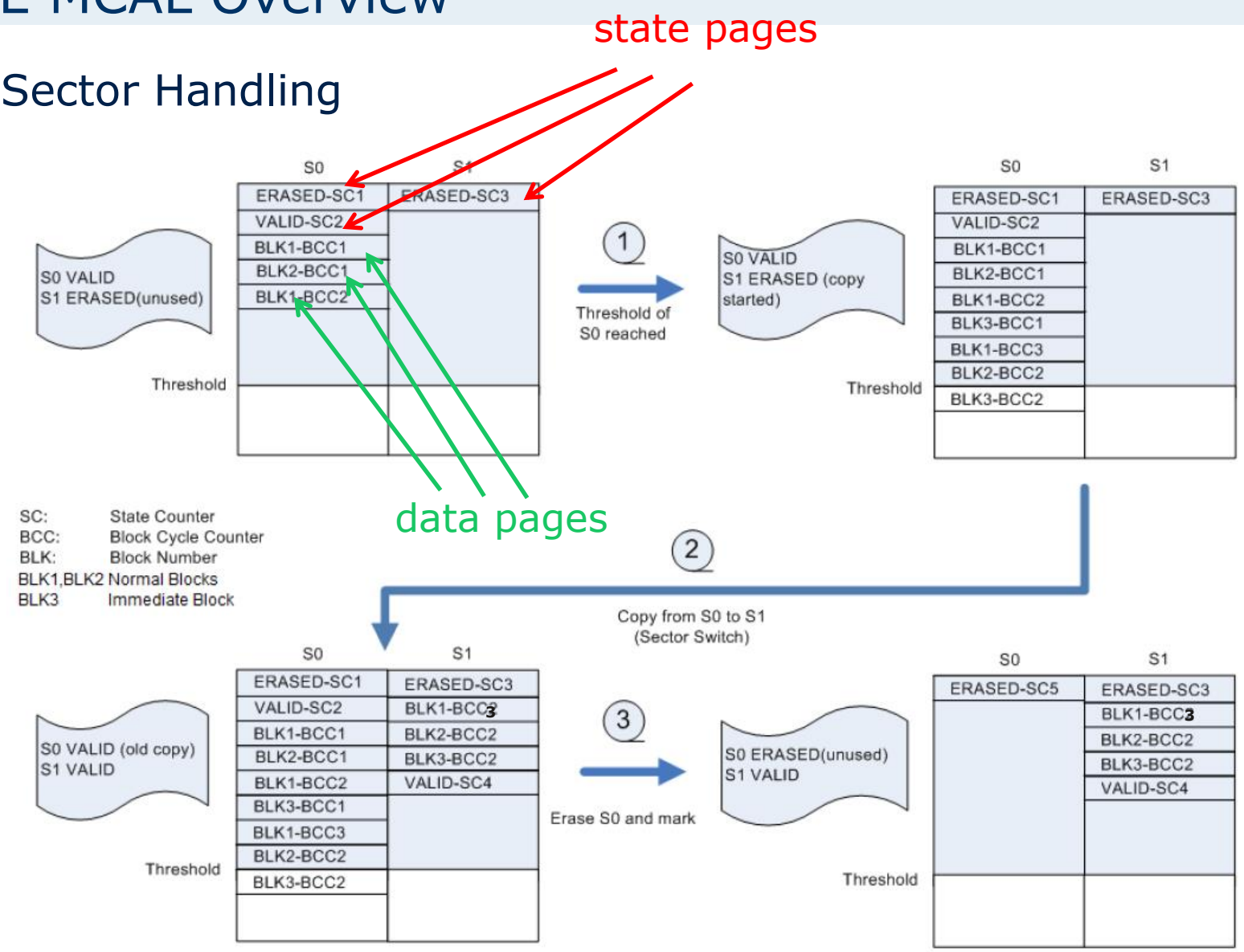
FEE MCAL Overview

- Sector Handling



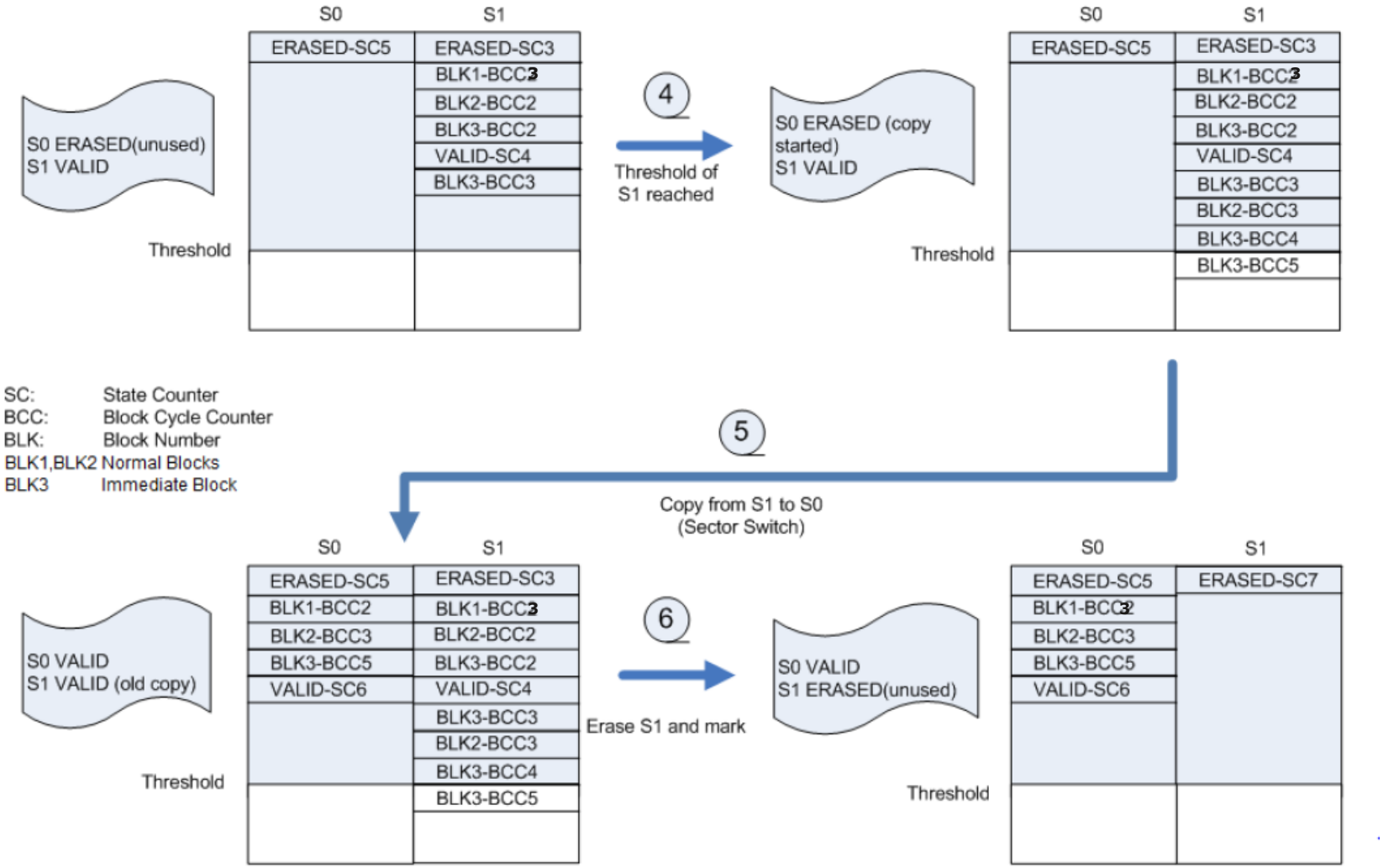
FEE MCAL Overview

- Sector Handling

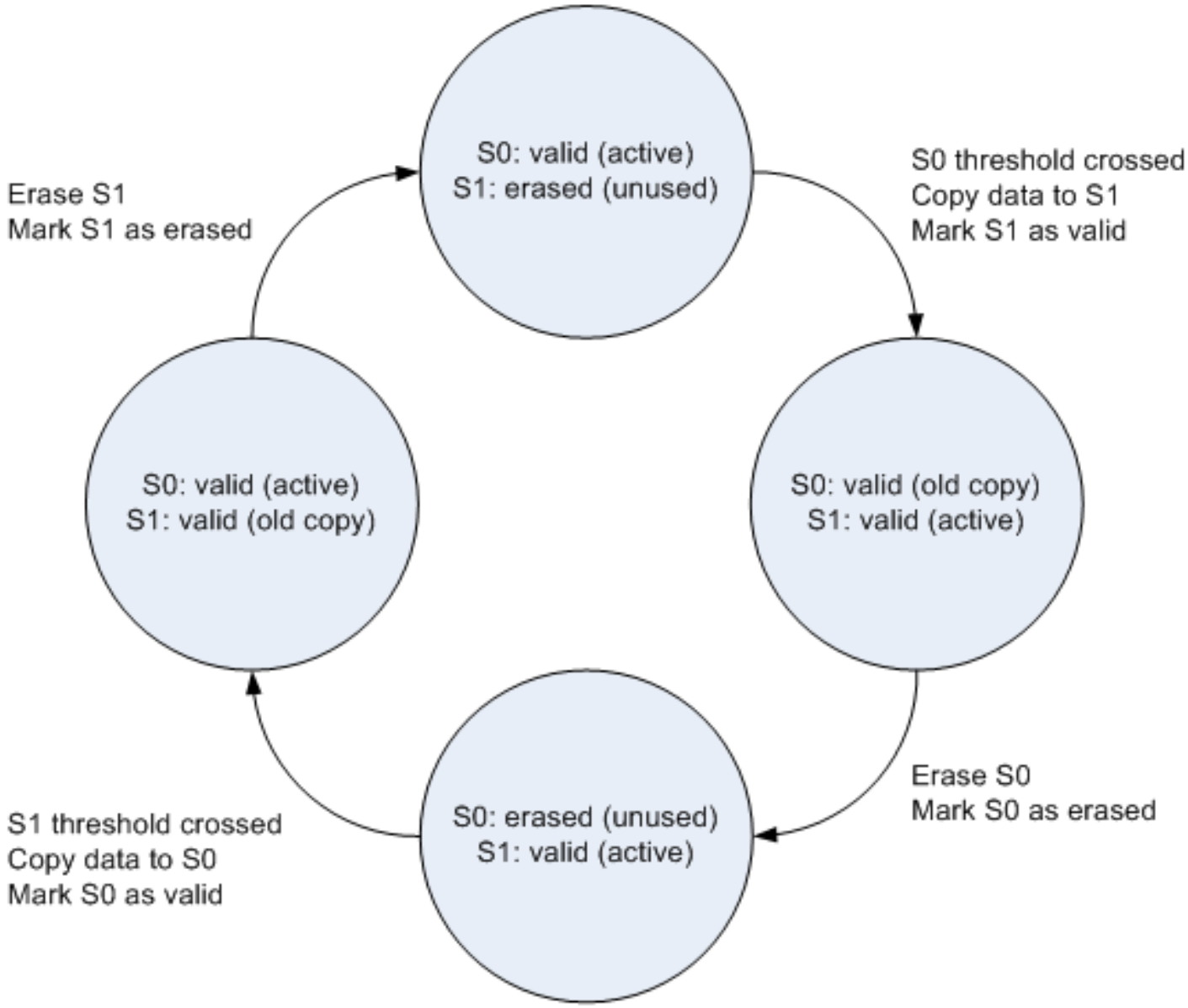


FEE MCAL Overview

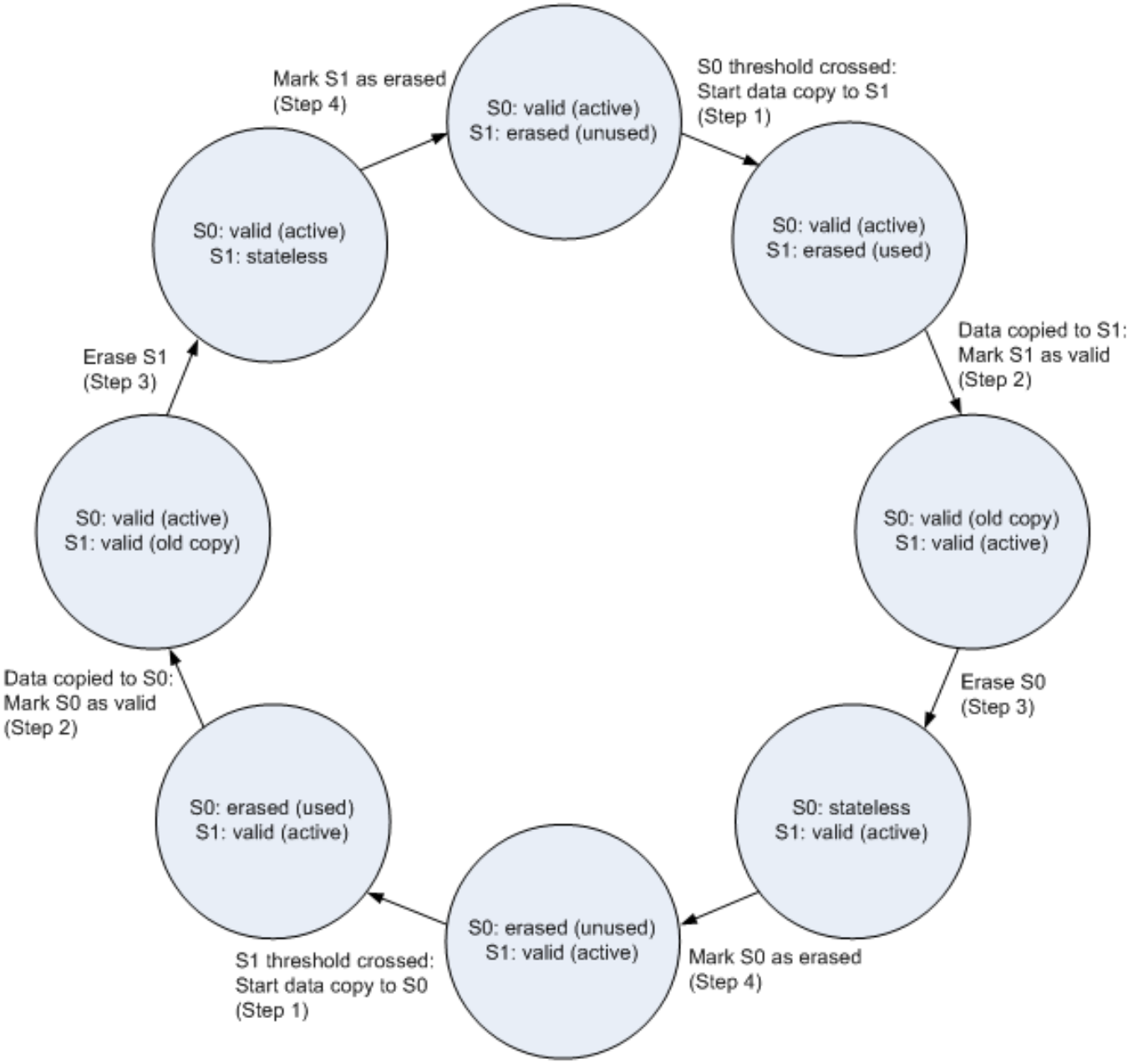
Sector Handling



State Machine – written State Pages



State Machine – including intermediate states



Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- MCAL FEE Data Page Format
- MCAL FEE Garbage Collection
- **MCAL FEE Power Failure Scenario – State Page**
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

Power Failure Scenarios

- Interrupted Write for SP data
  - Detected by marker="0"
  - State is ignored, previous state is taken

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
PageTypeId (0x59)	State	Unerasable WL Count	0	LS-Byte	State Counter		MS-Byte
Byte 8	Byte9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
PageTypeId (0xC6)	0	0	0	LS-Byte	Unerasable WL 1 Address		MS-Byte
⋮							⋮
PageTypeId (0xC6)	0	0	0	LS-Byte	Unerasable WL N Address		MS-Byte
0	0	0	0	0	0	0	0
⋮							⋮
Byte 112	Byte 113	Byte 114	Byte 115	Byte 116	Byte 117	Byte 118	Byte 119
0	0	0	0	0	0	0	0
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)

Power Failure Scenarios

- Interrupted Write for SP data
  - Detected by marker="0"
  - State is ignored, previous state is taken

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
PageTypeId (0x59)	State	Unerasable WL Count	0	LS-Byte	State Counter		MS-Byte

Byte 112	Byte 113	Byte 114	Byte 115	Byte 116	Byte 117	Byte 118	Byte 119
0	0	0	0	0	0	0	0
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)

Power Failure Scenarios

- Interrupted Write for SP marker
  - Detected by marker>”0”
  - Repairing of marker, if this fails re-programming on next empty WL*

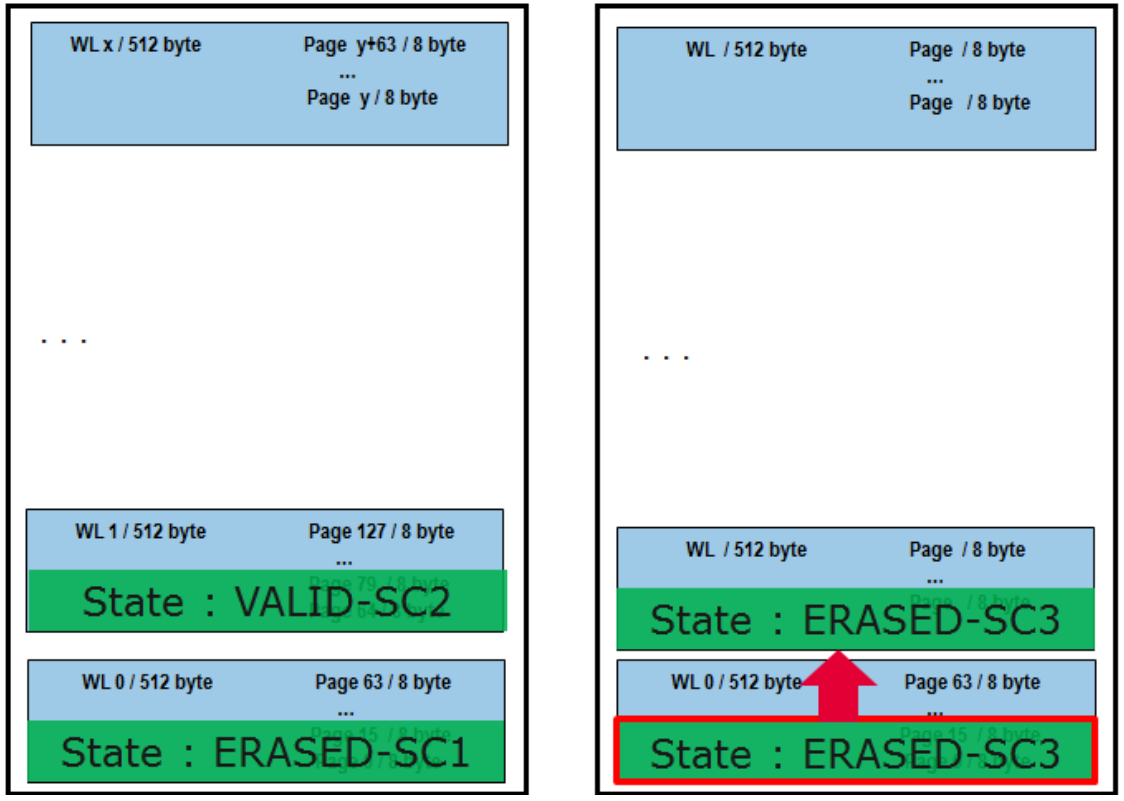
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
PageTypeId (0x59)	State	Unerasable WL Count	0	LS-Byte	State Counter		MS-Byte
Byte 8	Byte9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
PageTypeId (0xC6)	0	0	0	LS-Byte	Unerasable WL 1 Address		MS-Byte
⋮							⋮
PageTypeId (0xC6)	0	0	0	LS-Byte	Unerasable WL N Address		MS-Byte
0	0	0	0	0	0	0	0
⋮							⋮
Byte 112	Byte 113	Byte 114	Byte 115	Byte 116	Byte 117	Byte 118	Byte 119
0	0	0	0	0	0	0	0
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)

*attempt only 1 time

Power Failure Scenarios

- Early interrupted Write (Detection of slightly programmed SP)
  - Strict verification of programmed data with tight margin read “0”
  - Re-programming on next empty WL*

*attempt only 1 time



SP re-programmed

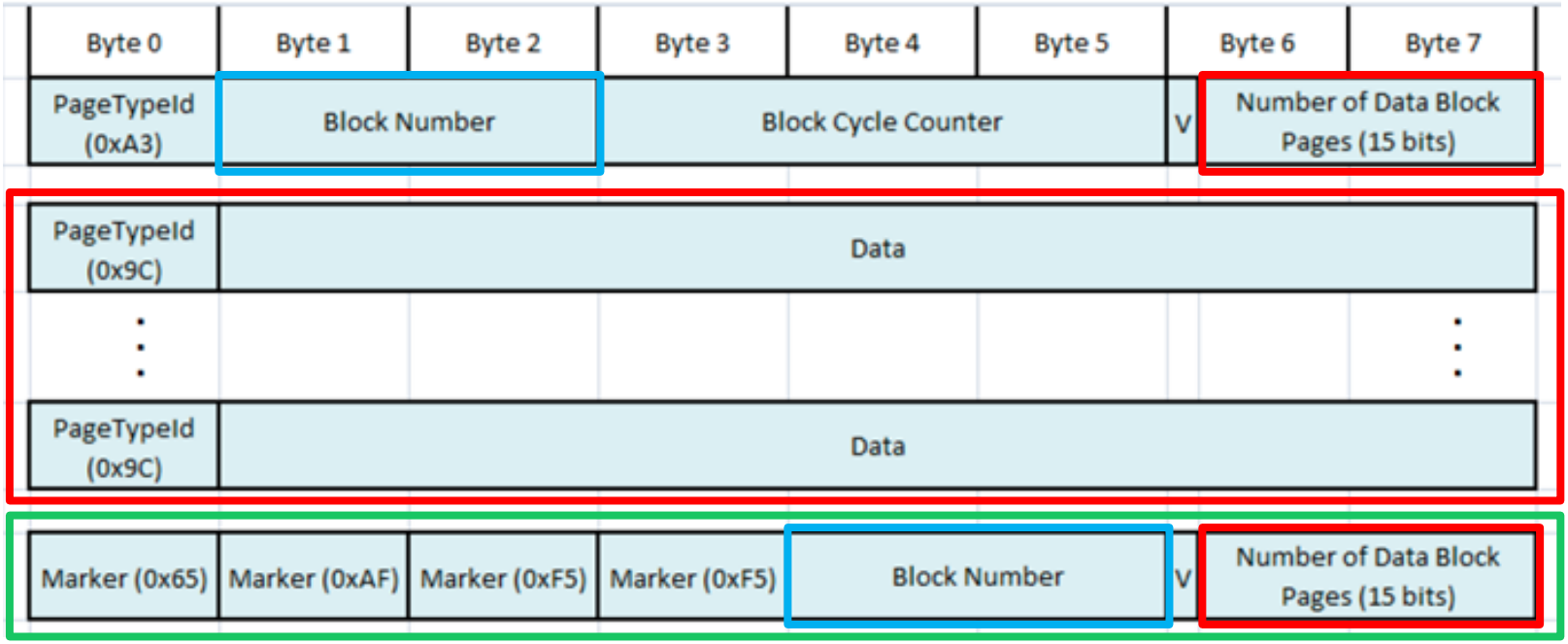
SP early interrupted write

Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- MCAL FEE Data Page Format
- MCAL FEE Garbage Collection
- MCAL FEE Power Failure Scenario – State Page
- **MCAL FEE Power Failure Scenario – Data Page**
- MCAL FEE Power Failure Scenario – Erase Operation

Power Failure Scenarios

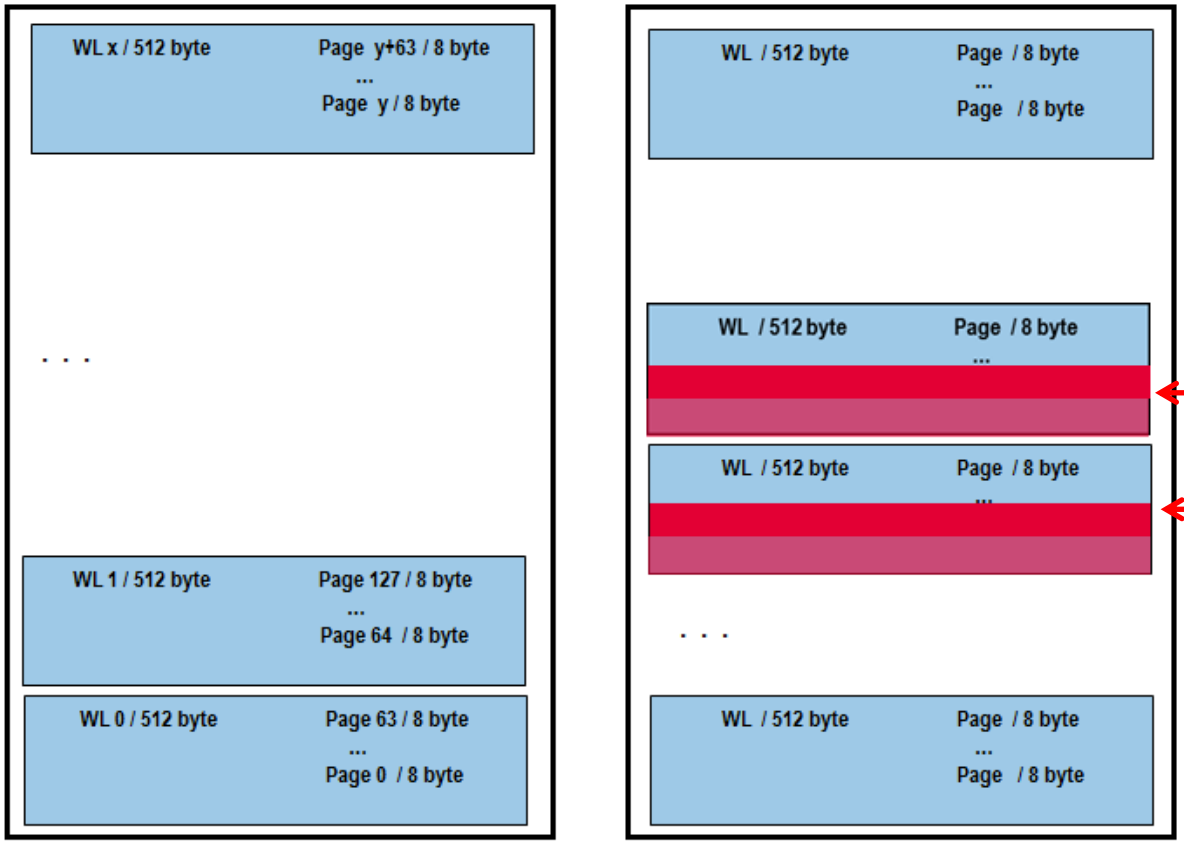
- Interrupted Write for DP
  - Rely on ECC detection and CRC calculated by caller
  - Detected by missing data pages for multi page data blocks
  - Detected by marker="0"
  - Reporting as inconsistent data block upon read request
  - Detected by Block Number inconsistency



Power Failure Scenarios

- Early interrupted Write (Detection of slightly programmed DP)
  - Strict verification of programmed data with tight margin read “0”
  - Re-programming on next empty WL*

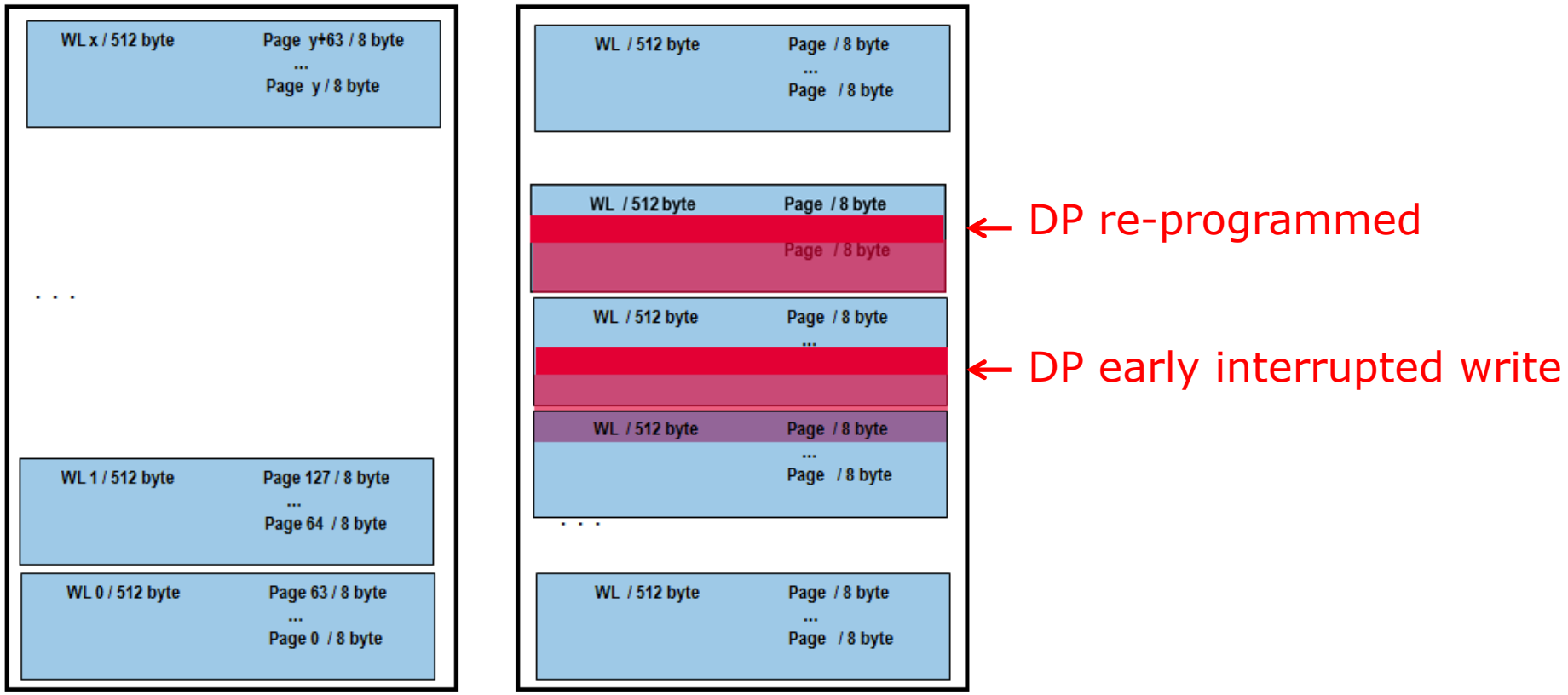
*attempt only 1 time



Power Failure Scenarios

- Early interrupted Write (Detection of slightly programmed DP)
  - Strict verification of programmed data with tight margin read “0”
  - Re-programming on next empty WL*

*attempt only 1 time

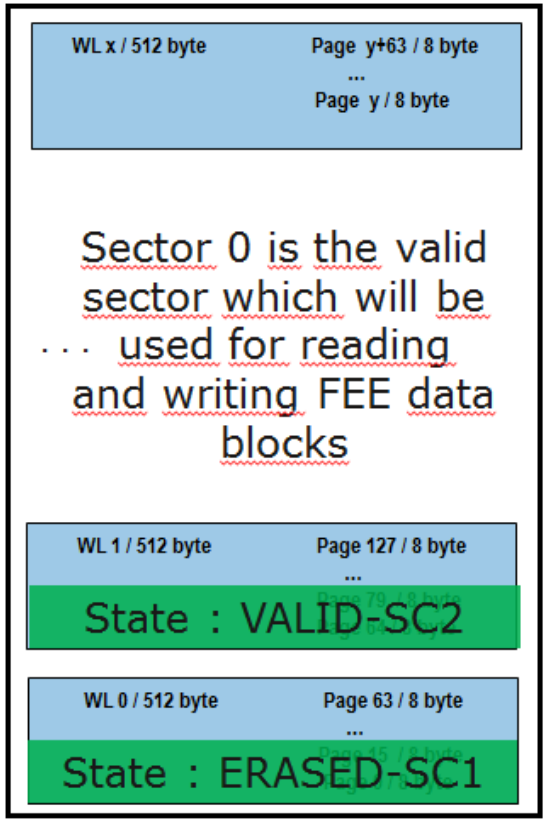


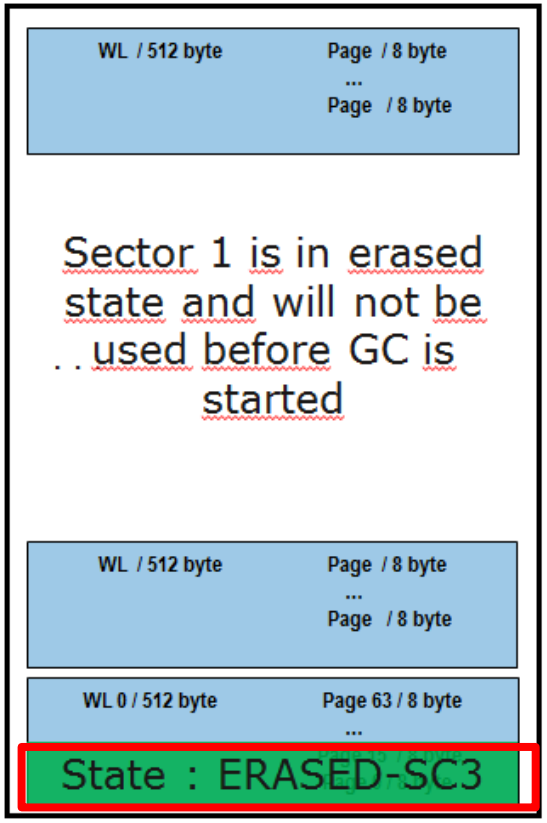
Overview

- Requirements for robust EEPOM Emulation on DFLASH
- MCAL FEE Algorithm - Overview
- MCAL FEE State Page Format
- MCAL FEE Data Page Format
- MCAL FEE Garbage Collection
- MCAL FEE Power Failure Scenario – State Page
- MCAL FEE Power Failure Scenario – Data Page
- MCAL FEE Power Failure Scenario – Erase Operation

Power Failure Scenarios

- Interrupted Erase
  - End of Erase is marked in SP
  - Restart of erase is initiated if marker is missing





ECC Error Handling

ECC Error Handling

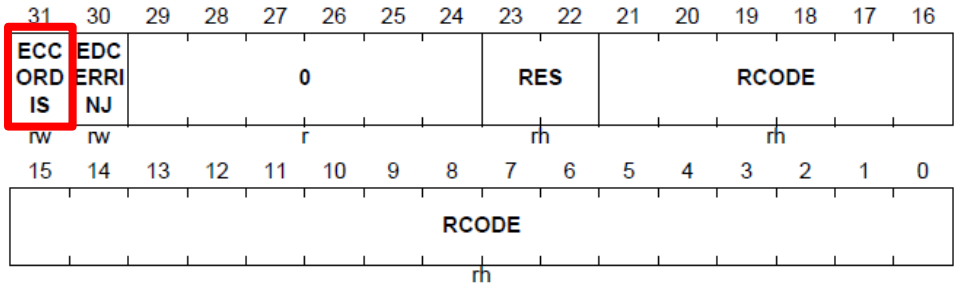
- DFLASH has TEC-QED ECC
  - TEC : triple error correction
  - QED : quad error detection
- ECC is generated per page (8byte / 64 bit)

- Error correction up to triple bit errors is (assumed to be) enabled
  - ECCRD.ECCORDIS = 0
- Single bit ECC error are not evaluated from FLS driver
  - FSR.DFSBER
- Double bit ECC error are not evaluated from FLS driver
  - FSR.DFBER
- Triple bit ECC error are not evaluated from FLS driver
  - FSR.DFTBER

- Quad bit ECC error are taken into account from the FLS driver
  - FSR.DFMBER
- Bus error on quad bit error is (assumed to be) disabled
  - MARD.TRAPDIS = 1

ECC Error Handling

ECCRD
ECC Read Register DF **($10A4_H$)** **Reset Value: $0000\ 0000_H$**

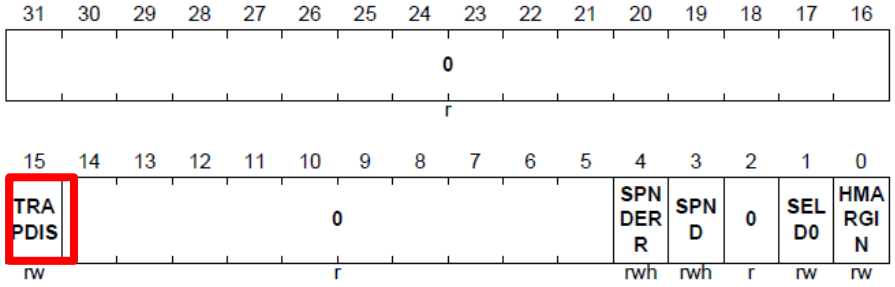


Field	Bits	Type	Description
RCODE	[21:0]	rh	**Error Correction Read Code** ECC code, read from the Flash read buffer with last data read operation.
RES	[23:22]	rh	**Reserved** Internal data.
EDCERRINJ	30	rw	**EDC Error Injection** Setting this bit enforces an error in the ECC error detection supervision circuit. This can be used to check the correct function of this circuit. This bit is only implemented for the PFlash read paths (i.e. ECCRPp). In ECCRD this bit is read-only 0. 0_B EDC logic operates normally. 1_B An error is injected into the EDC logic.
ECCORDIS	31	rw	**ECC Correction Disable** 0_B ECC correction for this read path is enabled. 1_B ECC correction for this read path is disabled.
0	[29:24]	r	**Reserved** Always read as 0.

Error correction up to triple bit errors enabled

ECC Error Handling

MARD
Margin Control Register DFlash (10AC$_H$) **Reset Value: 0000 8000$_H$**



Field	Bits	Type	Description
HMARGIN	0	rw	**Hard Margin Selection** Banks selected by MARD.SELi = 1 or MARP.SELi = 1 are read with: 0_B **Tight0**, Tight margin for 0 (low) level. Suboptimal 0-bits are read as 1s. 1_B **Tight1**, Tight margin for 1 (high) level. Suboptimal 1-bits are read as 0s. The concrete margin values are restored from the configuration sector and are determined by Infineon.
SELD0	1	rw	**DFLASH Bank Selection** Bank DF0 is selected for reading with HMARGIN. 0_B Bank DF0 is read with the standard (default) margin independent of HMARGIN. 1_B Bank DF0 is read with the margin selected by HMARGIN.
SPND	3	rwh	**Suspend** 0_B No suspend requested or pending. 1_B Suspension of the ongoing program/erase process is requested or pending.
SPNDERR	4	rwh	**Suspend Error** 0_B No suspend error. 1_B Last suspend request via MARD.SPND failed (see Chapter 11.5.4.5). Can be cleared by writing '1'.
TRAPDIS	15	rw	**DFLASH Uncorrectable Bit Error Trap Disable** 0_B If an uncorrectable error occurs in DFLASH (setting FSR.DFMBER), a bus error trap is generated[1]. 1_B The uncorrectable error trap is disabled.
0	2, [14:5], [31:16]	r	**Reserved** Always read as 0; should be written with 0.

1) After Boot ROM exit, uncorrectable bit error traps are enabled (TRAPDIS = 0_B).

Bus error on quad bit error disabled

FLS : HW related Error Handling

- Fls_17_Pmu_Erase
  - Sequence error -> EraseFailed DEM if enabled
  - Protection error -> EraseFailed DEM if enabled
- Fls_17_Pmu_Write
  - Sequence error -> WriteFailed DEM if enabled
  - Protection error -> WriteFailed DEM if enabled
- Fls_17_Pmu_Compare, Fls_17_Pmu_Read, Fls_17_Pmu_Cancel, Fls_17_GetStatus, Fls_17_GetJobResult, Fls_17_Pmu_GetBankStatus
  - none
- Fls_17_Pmu_ReadWordsSync
  - Uncorrectable ECC error -> return E_NOT_OK
    ReadFailed DEM if enabled
- Fls_17_CompareWordsSync
  - Uncorrectable ECC error -> return E_NOT_OK
    ReadFailed DEM if enabled
- Fls_17_Pmu_VerifyErase
  - Uncorrectable ECC error -> return E_NOT_OK (>1 not correctable ECC)
    ReadFailed DEM if enabled

HW related Error Handling

- Fls_17_Pmu_SuspendErase
  - MARD.SPNDERR -> return E_NOT_OK
- Fls_17_Pmu_ResumeErase
  - Sequence error -> return E_NOT_OK
  - Protection error -> return E_NOT_OK

FEE Cache Table

FEE MCAL Driver : Cache Table

- FEE driver initially builds a cache table to identify valid data blocks and starting address to be used for subsequent FEE write requests.
- FEE driver reads first 2 pages (2x8 byte) per WL (512 bytes) to check whether WL contains data or seems to be empty.
- If first 2 pages are found to be empty, the complete WL is checked to be empty.

FEE Erase Verify

FEE MCAL Driver : Erase Verify

- After erasing an FEE sector, FEE driver checks the sector for non erasable WL.
- Non erasable WL address is stored in the state page and not taken into account from FEE algorithm for further write requests.
- The state page is written at the end of the erase process after verification was executed.
- Finding an erased sector without erase state page will result in an new erase of the sector.