emLib ECC

Bit error correction

User Guide & Reference Manual

Document: UM12005
Software Version: 1.00
Revision: 1
Date: November 8, 2022

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

The information written in this document is assumed to be accurate without guarantee. The information in this manual is subject to change for functional or performance improvements without notice. SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions in this document. SEGGER disclaims any warranties or conditions, express, implied or statutory for the fitness of the product for a particular purpose. It is your sole responsibility to evaluate the fitness of the product for any specific use.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel. +49 2173-99312-0
Fax. +49 2173-99312-28
E-mail: support@segger.com*
Internet: www.segger.com

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: November 8, 2022

Software	Revision	Date	By	Description
1.00	1	221108	MD	Updated company information.
1.00	0	170130	MD	First release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0--13--1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

Table of contents

Chapter 1

Introduction to emLib ECC

This section presents an overview of emLib ECC, its structure, and its capabilities.

1.1 What is emLib ECC

emLib ECC is a library that provides functions for detection and correction of bit errors. The library can be employed to ensure the reliability of data transfered via digital networks or of the data stored on storage devices. The error detection and correction is performed using a BCH linear block code.

1.2 What is BCH?

BCH is a widely used linear block error-correcting code that is capable of detecting and correcting multiple random bit errors. The name of the code comes from the names of scientists that discovered this code: Raj Bose, D. K. Ray-Chaudhuri, and Alexis Hocquenghem. Error-correcting codes work by adding redundant parity-check bits to the information bits which have to be protected against bit errors. The information bits together with the parity-check bits form a codeword. The parity-check bits are a linear combination (XOR and shift operations) of the information bits and are calculated by the means of a generator polynomial. The generator polynomial is chosen based on the capabilities of the BCH code such as number of correctable bit errors and the number of information bits. The error correction process uses the property that the Hamming distance (that is the number of bit positions that have different values) between any two different valid codewords (that is without bit errors) is exactly two times the number of bit errors that can be corrected plus one. Using this property the error correction procedure is able to map a codeword with bit errors to a valid codeword and thus to correct the occurred bit errors. The BCH code is also able to detect but not correct a number of bit errors equal to error correction capability plus one.

Given the capability of BCH code to correct random bit errors this error-correcting code is typically employed for the correction of data stored on NAND flash devices. The NAND flash driver provided by SEGGER (`https://www.segger.com/emfile_driver_nand_flash.html`) comes with support for using emLib ECC to correct multiple bit errors on systems that do not support bit error correction in the hardware.

1.3 Features

emLib ECC is written in standard ANSI C and can run on virtually any CPU. Here’s a list summarizing the main features of emLib ECC:

- Clean ISO/ANSI C source code.
- Easy-to-understand and simple-to-use API.
- Same modules and same API can be used in PC programs and on embedded targets.
- Includes validation and further sample applications.
- Royalty free.

1.4 Package content

emLib ECC is provided in source code and contains everything needed. The following table shows the contents of the emLib ECC Package:

Files	Description
`Config`	Configuration header files.
`Doc`	emLib ECC documentation.
`LIB`	emLib ECC source code.
`Windows`	Executable application samples and their source code.

1.5 Included modules

emLib ECC comes with implementation of bit error correction algorithms with different correction capabilities and different block sizes designed to meet different application requirement. The following table lists the attributes of the supported bit error correction modules.

Module	Error correction capability	Parity-check size	Max. information size
ECC_BCH4_GF8.c	4 bits	4 bytes	27 bytes
ECC_BCH4_GF13.c	4 bits	7 bytes	1017 bytes
ECC_BCH8_GF13.c	8 bits	13 bytes	1010 bytes
ECC_BCH24_GF14.c	24 bits	42 bytes	2005 bytes
ECC_BCH40_GF14.c	40 bits	70 bytes	1977 bytes

1.6 Usage

emLib ECC has a simple yet powerful API. It can be easily integrated into an existing application. The code is completely written in ANSI-C.

All functionality can be verified with standard test patterns using the validation API. The functions for generating the lookup tables used for specific polynomials are also included for full transparency.

Separate functions are provided for the calculation of the ECC and for the correction of bit errors. To simply calculate the ECC for contiguous raw data, the application would only need to call one function. If two or more distinct data areas need to be processed protected by the same ECC, a separate function is provided which takes a list of data areas as parameter.

1.6.1 Recommended project structure

We recommend keeping emLib ECC separate from your application files. It is good practice to keep all the program files (including the header files) together in the `LIB` subdirectory of your project's root directory. This practice has the advantage of being very easy to update to newer versions of emLib ECC by simply replacing the `LIB` directory. Your application files can be stored anywhere.

> **Note**
>
> When updating to a newer emLib ECC version: as files may have been added, moved or deleted, the project directories may need to be updated accordingly.

1.6.2 Include directories

You should make sure that the include path contains the following directories (the order of inclusion is of no importance):

- `Config`
- `LIB`

> **Note**
>
> Always make sure that you have only one version of each file!

It is frequently a major problem when updating to a new version of emLib ECC if you have old files included and therefore mix different versions. If you keep emLib ECC in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing up (or at least renaming) the `LIB` directories before updating.

Chapter 2

API reference

This section describes the public API for emLib ECC. Any functions or data structures that are not described here but are exposed through inclusion of the `ECC.h` header file must be considered private and subject to change.

2.1 Core functions

Function	Description
4-bit error correction	
ECC_BCH4_GF8_Apply()	Corrects up to 4 bit errors and detects 5 bit errors in the specified data block.
ECC_BCH4_GF8_Calc()	Calculates the ECC of the specified data block.
ECC_BCH4_GF8_Load()	Loads the ECC from the specified buffer.
ECC_BCH4_GF8_Store()	Stores the ECC to the specified buffer.
ECC_BCH4_GF8_Validate()	Checks if the ECC routines work correctly.
ECC_BCH4_GF13_Apply()	Corrects up to 4 bit errors and detects 5 bit errors in the specified data block.
ECC_BCH4_GF13_ApplyMultiple()	Corrects up to 4 bit errors and detects 5 bit errors in the specified data blocks.
ECC_BCH4_GF13_Calc()	Calculates the ECC of the specified data block.
ECC_BCH4_GF13_CalcMultiple()	Calculates the ECC of the specified data blocks.
ECC_BCH4_GF13_Load()	Loads the ECC from the specified buffer.
ECC_BCH4_GF13_Store()	Stores the ECC to the specified buffer.
ECC_BCH4_GF13_Validate()	Checks if the ECC routines work correctly.
8-bit error correction	
ECC_BCH8_GF13_Apply()	Corrects up to 8 bit errors and detects 9 bit errors in the specified data block.
ECC_BCH8_GF13_ApplyMultiple()	Corrects up to 8 bit errors and detects 9 bit errors in the specified data blocks.
ECC_BCH8_GF13_Calc()	Calculates the ECC of the specified data block.
ECC_BCH8_GF13_CalcMultiple()	Calculates the ECC of the specified data blocks.
ECC_BCH8_GF13_Load()	Loads the ECC from the specified buffer.
ECC_BCH8_GF13_Store()	Stores the ECC to the specified buffer.
ECC_BCH8_GF13_Validate()	Checks if the ECC routines work correctly.
24-bit error correction	
ECC_BCH24_GF14_Apply()	Corrects up to 24 bit errors and detects 25 bit errors in the specified data block.
ECC_BCH24_GF14_ApplyMultiple()	Corrects up to 24 bit errors and detects 25 bit errors in the specified data blocks.
ECC_BCH24_GF14_Calc()	Calculates the ECC of the specified data block.
ECC_BCH24_GF14_CalcMultiple()	Calculates the ECC of the specified data blocks.
ECC_BCH24_GF14_Load()	Loads the ECC from the specified buffer.
ECC_BCH24_GF14_Store()	Stores the ECC to the specified buffer.
ECC_BCH24_GF14_Validate()	Checks if the ECC routines work correctly.
40-bit error correction	

Function	Description
ECC_BCH40_GF14_Apply()	Corrects up to 40 bit errors and detects 41 bit errors in the specified data block.
ECC_BCH40_GF14_ApplyMultiple()	Corrects up to 40 bit errors and detects 41 bit errors in the specified data block.
ECC_BCH40_GF14_Calc()	Calculates the ECC of the specified data block.
ECC_BCH40_GF14_CalcMultiple()	Calculates the ECC of the specified data blocks.
ECC_BCH40_GF14_Load()	Loads the ECC from the specified buffer.
ECC_BCH40_GF14_Store()	Stores the ECC to the specified buffer.
ECC_BCH40_GF14_Validate()	Checks if the ECC routines work correctly.

2.1.1 ECC_BCH4_GF8_Apply()

Description

Corrects up to 4 bit errors and detects 5 bit errors in the specified data block.

Prototype

```
int ECC_BCH4_GF8_Apply(U8       * pData,
                       unsigned   NumBytes,
                       U32      * pECC);
```

Parameters

Parameter	Description
pData	in Data to be checked and corrected. out Corrected data.
NumBytes	Number of bytes in the block (≤ 27 bytes).
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit errors corrected.
< 0 Number of bit errors > 4. Correction not possible.

2.1.2 ECC_BCH4_GF8_Calc()

Description

Calculates the ECC of the specified data block.

Prototype

```
U32 ECC_BCH4_GF8_Calc(const U8       * pData,
                      unsigned   NumBytes);
```

Parameters

Parameter	Description
pData	in Data to be protected by ECC.
NumBytes	Number of data bytes to be protected by ECC (≤ 27 bytes).

Return value

The calculated ECC value.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.3 ECC_BCH4_GF8_Load()

Description

Loads the ECC from the specified buffer.

Prototype

```
U32 ECC_BCH4_GF8_Load(const U8 * pBuffer);
```

Parameters

Parameter	Description
pBuffer	in Encoded ECC. The buffer has to be at least 4 bytes large.

Return value

ECC in memory format.

2.1.4 ECC_BCH4_GF8_Store()

Description

Stores the ECC to the specified buffer.

Prototype

```
void ECC_BCH4_GF8_Store(U8  * pBuffer,
                        U32   ecc);
```

Parameters

Parameter	Description
pBuffer	out Encoded ECC. The buffer has to be at least 4 bytes large.
ecc	ECC in memory format.

2.1.5 ECC_BCH4_GF8_Validate()

Description

Checks if the ECC routines work correctly.

Prototype

```
int ECC_BCH4_GF8_Validate(void);
```

Return value

= 0	Routines verified successfully.
≠ 0	An error occurred.

2.1.6 ECC_BCH4_GF13_Apply()

Description

Corrects up to 4 bit errors and detects 5 bit errors in the specified data block.

Prototype

```
int ECC_BCH4_GF13_Apply(U8       * pData,
                        unsigned   NumBytes,
                        U64      * pECC);
```

Parameters

Parameter	Description
pData	in Data to be checked and corrected. out Corrected data.
NumBytes	Number of bytes in the data block (≤ 1017 bytes).
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit errors corrected.
< 0 Number of bit errors > 4. Correction not possible.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.7 ECC_BCH4_GF13_ApplyMultiple()

Description

Corrects up to 4 bit errors and detects 5 bit errors in the specified data blocks.

Prototype

```
int ECC_BCH4_GF13_ApplyMultiple(ECC_DATA_ITEM * pDataItem,
                                unsigned        NumDataItems,
                                U64           * pECC);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit errors corrected.
< 0 Number of bit errors > 4. Correction not possible.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 1017 bytes. The first data block descriptor in the list is processed first.

2.1.8 ECC_BCH4_GF13_Calc()

Description

Calculates the ECC of the specified data block.

Prototype

```
U64 ECC_BCH4_GF13_Calc(const U8       * pData,
                       unsigned   NumBytes);
```

Parameters

Parameter	Description
pData	in Data to be protected by ECC.
NumBytes	Number of data bytes to be protected by ECC (≤ 1017 bytes).

Return value

The calculated ECC value.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.9 ECC_BCH4_GF13_CalcMultiple()

Description

Calculates the ECC of the specified data blocks.

Prototype

```
U64 ECC_BCH4_GF13_CalcMultiple(const ECC_DATA_ITEM * pDataItem,
                               unsigned              NumDataItems);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.

Return value

The calculated ECC.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 1017 bytes. The first data block descriptor in the list is processed first.

2.1.10 ECC_BCH4_GF13_Load()

Description

Loads the ECC from the specified buffer.

Prototype

```
U64 ECC_BCH4_GF13_Load(const U8 * pBuffer);
```

Parameters

Parameter	Description
pBuffer	in Encoded ECC. The buffer has to be at least 7 bytes large.

Return value

ECC in memory format.

2.1.11 ECC_BCH4_GF13_Store()

Description

Stores the ECC to the specified buffer.

Prototype

```
void ECC_BCH4_GF13_Store(U8  * pBuffer,
                         U64   ecc);
```

Parameters

Parameter	Description
pBuffer	out Encoded ECC. The buffer has to be at least 7 bytes large.
ecc	ECC in memory format.

2.1.12 ECC_BCH4_GF13_Validate()

Description

Checks if the ECC routines work correctly.

Prototype

```
int ECC_BCH4_GF13_Validate(void);
```

Return value

= 0	Routines verified successfully
≠ 0	An error occurred

2.1.13 ECC_BCH8_GF13_Apply()

Description

Corrects up to 8 bit errors and detects 9 bit errors in the specified data block.

Prototype

```
int ECC_BCH8_GF13_Apply(U8                * pData,
                        unsigned            NumBytes,
                        ECC_BCH8_GF13_ECC * pECC);
```

Parameters

Parameter	Description
pData	in Data to be checked and corrected. out Corrected data.
NumBytes	Number of bytes in the data block (≤ 1010 bytes).
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit error corrected
< 0 Number of bit errors > 8. Correction not possible.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.14 ECC_BCH8_GF13_ApplyMultiple()

Description

Corrects up to 8 bit errors and detects 9 bit errors in the specified data blocks.

Prototype

```
int ECC_BCH8_GF13_ApplyMultiple(ECC_DATA_ITEM     * pDataItem,
                                unsigned            NumDataItems,
                                ECC_BCH8_GF13_ECC * pECC);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit error corrected.
< 0 Number of bit errors > 8. Correction not possible.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 1010 bytes. The first data block descriptor in the list is processed first.

2.1.15 ECC_BCH8_GF13_Calc()

Description

Calculates the ECC of the specified data block.

Prototype

```
void ECC_BCH8_GF13_Calc(const U8                * pData,
                        unsigned                  NumBytes,
                        ECC_BCH8_GF13_ECC * pECC);
```

Parameters

Parameter	Description
pData	in Data block to be protected by ECC.
NumBytes	Number of bytes in the data block (≤ 1010 bytes).
pECC	out Calculated ECC.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.16 ECC_BCH8_GF13_CalcMultiple()

Description

Calculates the ECC of the specified data blocks.

Prototype

```
void ECC_BCH8_GF13_CalcMultiple(const ECC_DATA_ITEM     * pDataItem,
                                unsigned                  NumDataItems,
                                ECC_BCH8_GF13_ECC * pECC);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.
pECC	out Calculated ECC.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 1010 bytes. The first data block descriptor in the list is processed first.

2.1.17 ECC_BCH8_GF13_Load()

Description

Loads the ECC from the specified buffer.

Prototype

```
void ECC_BCH8_GF13_Load(const U8                * pBuffer,
                        ECC_BCH8_GF13_ECC * pECC);
```

Parameters

Parameter	Description
pBuffer	in Encoded ECC. The buffer has to be at least 13 bytes large.
pECC	out ECC in memory format.

2.1.18 ECC_BCH8_GF13_Store()

Description

Stores the ECC to the specified buffer.

Prototype

```
void ECC_BCH8_GF13_Store(      U8                * pBuffer,
                         const ECC_BCH8_GF13_ECC * pECC);
```

Parameters

Parameter	Description
pBuffer	out Encoded ECC. The buffer has to be at least 13 bytes large.
pECC	in ECC in memory format.

2.1.19 ECC_BCH8_GF13_Validate()

Description

Checks if the ECC routines work correctly.

Prototype

```
int ECC_BCH8_GF13_Validate(void);
```

Return value

= 0	Routines verified successfully
≠ 0	An error occurred

2.1.20 ECC_BCH24_GF14_Apply()

Description

Corrects up to 24 bit errors and detects 25 bit errors in the specified data block.

Prototype

```
int ECC_BCH24_GF14_Apply(U8                  * pData,
                         unsigned              NumBytes,
                         ECC_BCH24_GF14_ECC  * pECC);
```

Parameters

Parameter	Description
pData	in Data to be checked and corrected. out Corrected data.
NumBytes	Number of bytes in the data block (≤ 2005 bytes).
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit error corrected.
< 0 Number of bit errors > 24. Correction not possible.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.21 ECC_BCH24_GF14_ApplyMultiple()

Description

Corrects up to 24 bit errors and detects 25 bit errors in the specified data blocks.

Prototype

```
int ECC_BCH24_GF14_ApplyMultiple(ECC_DATA_ITEM      * pDataItem,
                                 unsigned             NumDataItems,
                                 ECC_BCH24_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit error corrected.
< 0 Number of bit errors > 24. Correction not possible.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 2005 bytes. The first data block descriptor in the list is processed first.

2.1.22 ECC_BCH24_GF14_Calc()

Description

Calculates the ECC of the specified data block.

Prototype

```
void ECC_BCH24_GF14_Calc(const U8                 * pData,
                         unsigned                   NumBytes,
                         ECC_BCH24_GF14_ECC       * pECC);
```

Parameters

Parameter	Description
pData	in Data block to be protected by ECC.
NumBytes	Number of bytes in the data block (≤ 2005 bytes).
pECC	out Calculated ECC.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.23 ECC_BCH24_GF14_CalcMultiple()

Description

Calculates the ECC of the specified data blocks.

Prototype

```
void ECC_BCH24_GF14_CalcMultiple(const ECC_DATA_ITEM      * pDataItem,
                                       unsigned             NumDataItems,
                                       ECC_BCH24_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.
pECC	out Calculated ECC.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 2005 bytes. The first data block descriptor in the list is processed first.

2.1.24 ECC_BCH24_GF14_Load()

Description

Loads the ECC from the specified buffer.

Prototype

```
void ECC_BCH24_GF14_Load(const U8                 * pBuffer,
                         ECC_BCH24_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pBuffer	in Encoded ECC. The buffer has to be at least 42 bytes large.
pECC	out ECC in memory format.

2.1.25 ECC_BCH24_GF14_Store()

Description

Stores the ECC to the specified buffer.

Prototype

```
void ECC_BCH24_GF14_Store(      U8                 * pBuffer,
                          const ECC_BCH24_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pBuffer	out Encoded ECC. The buffer has to be at least 42 bytes large.
pECC	in ECC in memory format.

2.1.26 ECC_BCH24_GF14_Validate()

Description

Checks if the ECC routines work correctly.

Prototype

```
int ECC_BCH24_GF14_Validate(void);
```

Return value

= 0	Routines verified successfully.
≠ 0	An error occurred.

2.1.27 ECC_BCH40_GF14_Apply()

Description

Corrects up to 40 bit errors and detects 41 bit errors in the specified data block.

Prototype

```
int ECC_BCH40_GF14_Apply(U8                 * pData,
                         unsigned             NumBytes,
                         ECC_BCH40_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pData	in Data to be checked and corrected. out Corrected data.
NumBytes	Number of bytes in the data block (≤ 1977 bytes).
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit error corrected.
< 0 Number of bit errors > 40. Correction not possible.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.28 ECC_BCH40_GF14_ApplyMultiple()

Description

Corrects up to 40 bit errors and detects 41 bit errors in the specified data block.

Prototype

```
int ECC_BCH40_GF14_ApplyMultiple(ECC_DATA_ITEM      * pDataItem,
                                 unsigned             NumDataItems,
                                 ECC_BCH40_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.
pECC	in Calculated ECC. out Corrected ECC.

Return value

≥ 0 Number of bit error corrected.
< 0 Number of bit errors > 40. Correction not possible.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 1977 bytes. The first data block descriptor in the list is processed first.

2.1.29 ECC_BCH40_GF14_Calc()

Description

Calculates the ECC of the specified data block.

Prototype

```
void ECC_BCH40_GF14_Calc(const U8                 * pData,
                         unsigned                   NumBytes,
                         ECC_BCH40_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pData	in Data block to be protected by ECC.
NumBytes	Number of bytes in the data block (≤ 1977 bytes).
pECC	out Calculated ECC.

Example

Please see *Using API functions to correct bit errors in a data block* on page 54.

2.1.30 ECC_BCH40_GF14_CalcMultiple()

Description

Calculates the ECC of the specified data blocks.

Prototype

```
void ECC_BCH40_GF14_CalcMultiple(const ECC_DATA_ITEM      * pDataItem,
                                       unsigned             NumDataItems,
                                       ECC_BCH40_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pDataItem	in List of data block descriptors.
NumDataItems	Number of data block descriptors.
pECC	out Calculated ECC.

Additional information

The total number of bytes in all the data block descriptors has to be smaller than or equal to 1977 bytes. The first data block descriptor in the list is processed first.

2.1.31 ECC_BCH40_GF14_Load()

Description

Loads the ECC from the specified buffer.

Prototype

```
void ECC_BCH40_GF14_Load(const U8                 * pBuffer,
                         ECC_BCH40_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pBuffer	in Encoded ECC. The buffer has to be at least 70 bytes large.
pECC	out ECC in memory format.

2.1.32 ECC_BCH40_GF14_Store()

Description

Stores the ECC to the specified buffer.

Prototype

```
void ECC_BCH40_GF14_Store(      U8                 * pBuffer,
                          const ECC_BCH40_GF14_ECC * pECC);
```

Parameters

Parameter	Description
pBuffer	out Encoded ECC. The buffer has to be at least 70 bytes large.
pECC	in ECC in memory format.

2.1.33 ECC_BCH40_GF14_Validate()

Description

Checks if the ECC routines work correctly.

Prototype

```
int ECC_BCH40_GF14_Validate(void);
```

Return value

= 0	Routines verified successfully
≠ 0	An error occurred

Chapter 3

Sample usage

This chapter explains the usage of the emLib ECC API.

3.1 Target applications

3.1.1 Using API functions to correct bit errors in a data block

This sample shows the how use the emLib ECC API functions to calculate the ECC of a data block and then how to use the calculated ECC to correct 4 bit errors.

```
#include <stdio.h>
#include "ECC.h"

void MainTask(void);
void MainTask(void) {
  U8  abData[] = "SEGGER";      // Data to be protected by ECC
  U64 ecc;                      // Calculated ECC
  int r;                        // Correction result

  printf("Start\n");
  //
  // Calculate the ECC of the original data.
  //
  printf("Calculate ECC...");
  ecc =  ECC_BCH4_GF13_Calc(abData, sizeof(abData));
  printf("OK\n");
  printf("  Data:            %s\n", abData);
  printf("  ECC:             0x%16llu\n", ecc);
  //
  // Force 4 bit errors in the data protected by ECC.
  //
  printf("Force bit errors...");
  abData[1] ^= 0x20;
  abData[2] ^= 0x20;
  abData[3] ^= 0x20;
  abData[4] ^= 0x20;
  printf("OK\n");
  printf("  Data:            %s\n", abData);
  printf("  Num. bit errors: 4\n");
  //
  // Correct the bit errors using ECC.
  //
  printf("Correct bit errors...");
  r = ECC_BCH4_GF13_Apply(abData, sizeof(abData), &ecc);
  if (r < 0) {
    printf("Error (Too many bit errors)\n");
  } else {
    printf("OK\n");
    printf("  Data:            %s\n", abData);
    printf("  Num. bit errors: %d\n", r);
  }
  printf("Finished\n");
  while (1) {
    ;
  }
}
```

The sample application outputs the following messages during the execution:

```
Start
Calculate ECC...OK
  Data:            SEGGER
  ECC:             0x1192622231858287
Force bit errors...OK
  Data:            SeggeR
  Num. bit errors: 4
Correct bit errors...OK
  Data:            SEGGER
```



```
  Num. bit errors: 4
Finished
```


3.2 PC applications

emLib ECC includes sample applications to demonstrate its functionality and to provide an easy to use starting point for your application. The applications' source code is included in the shipment. The following applications are delivered with emLib ECC:

Application name	Target platform	Description
ECCCalcApply.exe	Windows	Command line tool to calculate and correct bit errors.

3.2.1 ECCCalcApply

ECCCalcApply is a Windows console application that can be used to calculate the ECC of data stored in a file and to use the calculated ECC to correct bit errors in the same file. The calculated ECC is stored to a separate file in hexadecimal ASCII format. If the number of bytes stored to file is larger than the block size supported by the ECC algorithm one ECC is calculated for each data block.

Usage

`ECCCalcApply` [-a <ECCAlgo>] [-c] [-e <ECCFile>] [-h] [-q] <DataFile>

Parameter	Description
-a <ECCAlgo>	(Optional) Type of the ECC algorithm. ECCAlgo can be one of:
-c	(Optional) Used to indicate the bit error correction should be performed. Default is to calculate the ECC.
-e <ECCFile>	(Optional) Name of the file that contains the ECC in ASCII hexadecimal format. Default is DataFile.ecc
-h	(Optional) Shows information about the usage of the application.
-q	(Optional) Used to disable any output messages.
<DataFile>	Name of the file containing the data to be protected by ECC.

The following screen capture shows the messages generated by the application during the ECC calculation.

```
C:>ECCCalcApply Test.txt
(c) 2017 SEGGER Microcontroller GmbH & Co. KG
         www.segger.com

ECCCalcApply V1.00 ('?' or '-h' for help)
Compiled on Dec 23 2016, 14:23:43

Data file:                Test.txt
ECC file:                 Test.txt.ecc
ECC algorithm:            BCH4GF13
ECC correction capability: 4 bits
ECC block size:           1017 bytes

Calculate ECC...OK (1 ECC values(s) stored to file)
C:> _
```

The following screen capture shows the messages generated by the application during the bit error correction procedure.

```
C:>ECCCalcApply -c Test.txt
(c) 2017 SEGGER Microcontroller GmbH & Co. KG
         www.segger.com
```



```
ECCCalcApply V1.00 ('?' or '-h' for help)
Compiled on Dec 23 2016, 14:23:43

Data file:                Test.txt
ECC file:                 Test.txt.ecc
ECC algorithm:            BCH4GF13
ECC correction capability: 4 bits
ECC block size:           1017 bytes

Apply ECC...OK (4 bit error(s) corrected in 1 ECC block(s))

C:> _
```


Chapter 4

Performance and resource usage

This chapter covers the performance and resource usage of emLib ECC. Contained information may be used to obtain sufficient estimates for most target systems.

emLib ECC is designed to cater for many different embedded design requirements, from constrained microcontrollers to high performance microprocessors. Each single module might be excluded from build in order to construct a highly compact, minimal system.

4.1 Target system configuration

The following table shows the hardware and toolchain details used to measure the given values:

Detail	Description
CPU	Cortex-M7, 217 MHz
Tool chain	IAR Embedded Workbench for ARM V7.60
Model	Thumb-2 instructions
Compiler options	Highest speed optimization

4.1.1 Performance

The following table shows performance values for emLib ECC:

Error correction capability	Calculation speed	Correction speed (1 bit)	Correction speed (2 bits)	Correction speed (3 bits)	Correction speed (4 bits)	Correction speed (8 bits)	Correction speed (24 bits)	Correction speed (40 bits)
4-bit	25 MByte/ sec	18.5 MByte/ sec	2.4 MByte/ sec	1.8 MByte/ sec	1.4 MByte/ sec	-	-	-
8-bit	13 MByte/ sec	7.2 MByte/ sec	1.2 MByte/ sec	0.8 MByte/ sec	0.5 MByte/ sec	0.3 MByte/ sec	-	-
24-bit	2 MByte/ sec	0.9 MByte/ sec	0.5 MByte/ sec	0.4 MByte/ sec	0.4 MByte/ sec	0.2 MByte/ sec	54 KBytes/s	-
40-bit	1.5 MByte/ sec	0.4 MByte/ sec	0.3 MByte/ sec	0.2 MByte/ sec	0.2 MByte/ sec	0.1 MByte/ sec	51 KBytes/s	36 KBytes/s

4.1.2 Resource usage

The following table shows ROM and RAM requirements for emLib ECC:

Error correction capability	ROM (code)	ROM (division LUT)	ROM (GF LUTs)	RAM (stack)
4-bit	7.8 KBytes	2 KBytes	32 KBytes	144 bytes
8-bit	7.9 KBytes	4 KBytes	32 KBytes	144 bytes
24-bit	6.7 KBytes	12.2 KBytes	64 KBytes	272 bytes
40-bit	8.7 KBytes	18.4 KBytes	64 KBytes	424 bytes

Chapter 5

Support

This chapter should help if any problem occurs, e.g. with the use of the emLib ECC functions, and describes how to contact the SEGGER support.

5.1 Contacting SEGGER support

If you are a registered emLib ECC user and need to contact the SEGGER support, please send the following information via email to `support@segger.com`:

- The emLib ECC version.
- Your emLib ECC registration number.
- If you are unsure about the above information, you may also use the name of the shipped emLib ECC ZIP-file (which contains the above information).
- A detailed description of the problem.
- (Optional) A project with which we can reproduce the problem.

Chapter 6

Glossary

BCH
A type of error correcting code that can detect and correct multiple bit errors.

ECC
Error-Correcting Code. An algorithm that can be used to detect and correct bit errors. In this manual it is used also as synonym for the parity-check bits.

Generator polynomial
The divisor in the polynomial division that is used to calculate the ECC.

GF
Galois Field. A finite set of values on which operations of addition, subtraction, multiplication and division are defined.

Information bits
Data that has to be protected against bit errors.

Parity-check bits
Redundant bits that are used by the error-correcting code to detect and correct bit errors.

LUT
Look-Up Table. An array containing precalculated values that is used to increase the performance of certain calculations.

