

# emLib ECC

Bit error correction

## User Guide & Reference Manual

Document: UM12005  
Software Version: 1.10  
Revision: 0  
Date: February 29, 2024



A product of SEGGER Microcontroller GmbH

[www.segger.com](http://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

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2022-2024 SEGGER Microcontroller GmbH, Monheim am Rhein / Germany

## 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](http://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: February 29, 2024

Software	Revision	Date	By	Description
1.10	1	240228	MD	Added <code>ECC_SECEDED_64x32_Apply()</code> function. Added <code>ECC_SECEDED_64x32_Calc()</code> function. Added <code>ECC_SECEDED_64x32_Load()</code> function. Added <code>ECC_SECEDED_64x32_Store()</code> function. Added <code>ECC_SECEDED_64x32_Validate()</code> function. Updated performance values.
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

---

1	Introduction to emLib ECC .....	9
1.1	What is emLib ECC .....	10
1.2	Features .....	11
1.3	What is BCH? .....	12
1.4	Package content .....	13
1.5	Included modules .....	14
1.6	Usage .....	15
1.6.1	Recommended project structure .....	15
1.6.2	Include directories .....	15
2	API reference .....	16
2.1	Core functions .....	17
2.1.1	ECC_SECEDED_64x32_Apply() .....	19
2.1.2	ECC_SECEDED_64x32_Calc() .....	20
2.1.3	ECC_SECEDED_64x32_Load() .....	21
2.1.4	ECC_SECEDED_64x32_Store() .....	22
2.1.5	ECC_SECEDED_64x32_Validate() .....	23
2.1.6	ECC_BCH4_GF8_Apply() .....	24
2.1.7	ECC_BCH4_GF8_Calc() .....	25
2.1.8	ECC_BCH4_GF8_Load() .....	26
2.1.9	ECC_BCH4_GF8_Store() .....	27
2.1.10	ECC_BCH4_GF8_Validate() .....	28
2.1.11	ECC_BCH4_GF13_Apply() .....	29
2.1.12	ECC_BCH4_GF13_ApplyMultiple() .....	30
2.1.13	ECC_BCH4_GF13_Calc() .....	31
2.1.14	ECC_BCH4_GF13_CalcMultiple() .....	32
2.1.15	ECC_BCH4_GF13_Load() .....	33
2.1.16	ECC_BCH4_GF13_Store() .....	34
2.1.17	ECC_BCH4_GF13_Validate() .....	35
2.1.18	ECC_BCH8_GF13_Apply() .....	36
2.1.19	ECC_BCH8_GF13_ApplyMultiple() .....	37
2.1.20	ECC_BCH8_GF13_Calc() .....	38
2.1.21	ECC_BCH8_GF13_CalcMultiple() .....	39
2.1.22	ECC_BCH8_GF13_Load() .....	40
2.1.23	ECC_BCH8_GF13_Store() .....	41
2.1.24	ECC_BCH8_GF13_Validate() .....	42
2.1.25	ECC_BCH24_GF14_Apply() .....	43
2.1.26	ECC_BCH24_GF14_ApplyMultiple() .....	44
2.1.27	ECC_BCH24_GF14_Calc() .....	45
2.1.28	ECC_BCH24_GF14_CalcMultiple() .....	46

2.1.29	ECC_BCH24_GF14_Load()	47
2.1.30	ECC_BCH24_GF14_Store()	48
2.1.31	ECC_BCH24_GF14_Validate()	49
2.1.32	ECC_BCH40_GF14_Apply()	50
2.1.33	ECC_BCH40_GF14_ApplyMultiple()	51
2.1.34	ECC_BCH40_GF14_Calc()	52
2.1.35	ECC_BCH40_GF14_CalcMultiple()	53
2.1.36	ECC_BCH40_GF14_Load()	54
2.1.37	ECC_BCH40_GF14_Store()	55
2.1.38	ECC_BCH40_GF14_Validate()	56
<b>3</b>	<b>Sample usage</b>	<b>57</b>
3.1	Target applications	58
3.1.1	Using API functions to correct bit errors in a data block	58
3.2	PC applications	60
3.2.1	ECCCalcApply	60
<b>4</b>	<b>Performance and resource usage</b>	<b>62</b>
4.1	General information	63
4.2	Target system configuration	64
4.2.1	Performance	64
4.2.2	Resource usage	64
<b>5</b>	<b>Support</b>	<b>65</b>
5.1	Contacting SEGGER support	66
5.2	Where can I find the license number?	67
<b>6</b>	<b>Glossary</b>	<b>68</b>



# 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 transferred via digital networks or of the data stored on storage devices. The error detection and correction is performed using either a BCH linear block code or a Hamming code.

## 1.2 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.
- MISRA C:2012 compliant.

## 1.3 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/products/file-system/em-file/add-ons/device-driver-nand-flash/about-device-driver-nand-flash/>) comes with support for using emLib ECC to correct multiple bit errors on systems that do not support bit error correction in the hardware.

## 1.4 Package content

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

<b>Files</b>	<b>Description</b>
Application	Test applications.
Config	Configuration header files.
Doc	emLib ECC documentation.
LIB	emLib ECC source code.
SEGGER	Software utilities.
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 requirements. The following table lists the attributes of the supported bit error correction modules.

Module	Error correction capability	Parity-check size	Max. information size
ECC_SECCDED_64x32.c	1 bit	3 bytes	256 bytes
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.

#### Warning

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`

#### Warning

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
1-bit error correction	
<code>ECC_SECEDED_64x32_Apply()</code>	Verifies and corrects bit errors of the specified data block using ECC.
<code>ECC_SECEDED_64x32_Calc()</code>	Calculates the ECC of the specified data block.
<code>ECC_SECEDED_64x32_Load()</code>	Loads the ECC from memory.
<code>ECC_SECEDED_64x32_Store()</code>	Stores the ECC in memory.
<code>ECC_SECEDED_64x32_Validate()</code>	Checks if the ECC routines work correctly.
4-bit error correction	
<code>ECC_BCH4_GF8_Apply()</code>	Corrects up to 4 bit errors and detects 5 bit errors in the specified data block.
<code>ECC_BCH4_GF8_Calc()</code>	Calculates the ECC of the specified data block.
<code>ECC_BCH4_GF8_Load()</code>	Loads the ECC from the specified buffer.
<code>ECC_BCH4_GF8_Store()</code>	Stores the ECC to the specified buffer.
<code>ECC_BCH4_GF8_Validate()</code>	Checks if the ECC routines work correctly.
<code>ECC_BCH4_GF13_Apply()</code>	Corrects up to 4 bit errors and detects 5 bit errors in the specified data block.
<code>ECC_BCH4_GF13_ApplyMultiple()</code>	Corrects up to 4 bit errors and detects 5 bit errors in the specified data blocks.
<code>ECC_BCH4_GF13_Calc()</code>	Calculates the ECC of the specified data block.
<code>ECC_BCH4_GF13_CalcMultiple()</code>	Calculates the ECC of the specified data blocks.
<code>ECC_BCH4_GF13_Load()</code>	Loads the ECC from the specified buffer.
<code>ECC_BCH4_GF13_Store()</code>	Stores the ECC to the specified buffer.
<code>ECC_BCH4_GF13_Validate()</code>	Checks if the ECC routines work correctly.
8-bit error correction	
<code>ECC_BCH8_GF13_Apply()</code>	Corrects up to 8 bit errors and detects 9 bit errors in the specified data block.
<code>ECC_BCH8_GF13_ApplyMultiple()</code>	Corrects up to 8 bit errors and detects 9 bit errors in the specified data blocks.
<code>ECC_BCH8_GF13_Calc()</code>	Calculates the ECC of the specified data block.
<code>ECC_BCH8_GF13_CalcMultiple()</code>	Calculates the ECC of the specified data blocks.
<code>ECC_BCH8_GF13_Load()</code>	Loads the ECC from the specified buffer.
<code>ECC_BCH8_GF13_Store()</code>	Stores the ECC to the specified buffer.
<code>ECC_BCH8_GF13_Validate()</code>	Checks if the ECC routines work correctly.
24-bit error correction	
<code>ECC_BCH24_GF14_Apply()</code>	Corrects up to 24 bit errors and detects 25 bit errors in the specified data block.
<code>ECC_BCH24_GF14_ApplyMultiple()</code>	Corrects up to 24 bit errors and detects 25 bit errors in the specified data blocks.

Function	Description
<code>ECC_BCH24_GF14_Calc()</code>	Calculates the ECC of the specified data block.
<code>ECC_BCH24_GF14_CalcMultiple()</code>	Calculates the ECC of the specified data blocks.
<code>ECC_BCH24_GF14_Load()</code>	Loads the ECC from the specified buffer.
<code>ECC_BCH24_GF14_Store()</code>	Stores the ECC to the specified buffer.
<code>ECC_BCH24_GF14_Validate()</code>	Checks if the ECC routines work correctly.
<b>40-bit error correction</b>	
<code>ECC_BCH40_GF14_Apply()</code>	Corrects up to 40 bit errors and detects 41 bit errors in the specified data block.
<code>ECC_BCH40_GF14_ApplyMultiple()</code>	Corrects up to 40 bit errors and detects 41 bit errors in the specified data block.
<code>ECC_BCH40_GF14_Calc()</code>	Calculates the ECC of the specified data block.
<code>ECC_BCH40_GF14_CalcMultiple()</code>	Calculates the ECC of the specified data blocks.
<code>ECC_BCH40_GF14_Load()</code>	Loads the ECC from the specified buffer.
<code>ECC_BCH40_GF14_Store()</code>	Stores the ECC to the specified buffer.
<code>ECC_BCH40_GF14_Validate()</code>	Checks if the ECC routines work correctly.

## 2.1.1 ECC\_SECDED\_64x32\_Apply()

### Description

Verifies and corrects bit errors of the specified data block using ECC.

### Prototype

```
int ECC_SECDED_64x32_Apply(U32 * pData,  
                           U32 * pECC);
```

### Parameters

Parameter	Description
<code>pData</code>	<code>in</code> 256-byte data block to be verified. <code>out</code> Verified and corrected data.
<code>pECC</code>	<code>in</code> Calculated ECC to be used for verification. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$       Number of bit errors corrected.
- $< 0$       Number of bit errors  $> 1$ . Correction not possible.

### Additional information

`pData` has to be organized as 64 32-bit values.

## 2.1.2 ECC\_SECDED\_64x32\_Calc()

### Description

Calculates the ECC of the specified data block.

### Prototype

```
U32 ECC_SECDED_64x32_Calc(const U32 * pData);
```

### Parameters

Parameter	Description
<code>pData</code>	<code>in</code> 256-byte data block.

### Return value

Calculated ECC.

### Additional information

`pData` has to be organized as 64 32-bit values.

### 2.1.3 ECC\_SECEDED\_64x32\_Load()

#### Description

Loads the ECC from memory.

#### Prototype

```
U32 ECC_SECEDED_64x32_Load(const U8 * pBuffer);
```

#### Parameters

Parameter	Description
<code>pBuffer</code>	<code>in</code> Stored ECC.

#### Return value

Loaded ECC.

## 2.1.4 ECC\_SECDDED\_64x32\_Store()

### Description

Stores the ECC in memory.

### Prototype

```
void ECC_SECDDED_64x32_Store(U8 * pBuffer,  
                             U32 ecc);
```

### Parameters

Parameter	Description
<code>pBuffer</code>	<code>out</code> Stored ECC.
<code>ecc</code>	ECC to be stored.

## 2.1.5 ECC\_SECEDED\_64x32\_Validate()

### Description

Checks if the ECC routines work correctly.

### Prototype

```
int ECC_SECEDED_64x32_Validate(void);
```

### Return value

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

## 2.1.6 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
<code>pData</code>	<code>in</code> Data to be checked and corrected. <code>out</code> Corrected data.
<code>NumBytes</code>	Number of bytes in the block ( $\leq 27$ bytes).
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$      Number of bit errors corrected.
- $< 0$      Number of bit errors  $> 4$ . Correction not possible.



## 2.1.7 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
<code>pData</code>	<code>in</code> Data to be protected by ECC.
<code>NumBytes</code>	Number of data bytes to be protected by ECC ( $\leq 27$ bytes).

### Return value

The calculated ECC value.

### Example

Please see *Using API functions to correct bit errors in a data block* on page 58.

## 2.1.8 ECC\_BCH4\_GF8\_Load()

### Description

Loads the ECC from the specified buffer.

### Prototype

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

### Parameters

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

### Return value

ECC in memory format.

## 2.1.9 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
<code>pBuffer</code>	<b>out</b> Encoded ECC. The buffer has to be at least 4 bytes large.
<code>ecc</code>	ECC in memory format.

## 2.1.10 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.

## 2.1.11 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
<code>pData</code>	<code>in</code> Data to be checked and corrected. <code>out</code> Corrected data.
<code>NumBytes</code>	Number of bytes in the data block ( $\leq 1017$ bytes).
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$       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 58.

## 2.1.12 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
<code>pDataItem</code>	<code>in</code> List of data block descriptors.
<code>NumDataItems</code>	Number of data block descriptors.
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$       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.

## 2.1.13 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
<code>pData</code>	<code>in</code> Data to be protected by ECC.
<code>NumBytes</code>	Number of data bytes to be protected by ECC ( $\leq 1017$ bytes).

### Return value

The calculated ECC value.

### Example

Please see *Using API functions to correct bit errors in a data block* on page 58.

## 2.1.14 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
<a href="#">pDataItem</a>	<b>in</b> List of data block descriptors.
<a href="#">NumDataItems</a>	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.



## 2.1.15 ECC\_BCH4\_GF13\_Load()

### Description

Loads the ECC from the specified buffer.

### Prototype

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

### Parameters

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

### Return value

ECC in memory format.

## 2.1.16 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
<code>pBuffer</code>	<b>out</b> Encoded ECC. The buffer has to be at least 7 bytes large.
<code>ecc</code>	ECC in memory format.

## 2.1.17 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

## 2.1.18 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
<code>pData</code>	<code>in</code> Data to be checked and corrected. <code>out</code> Corrected data.
<code>NumBytes</code>	Number of bytes in the data block ( $\leq 1010$ bytes).
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

$\geq 0$       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 58.

## 2.1.19 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
<code>pDataItem</code>	<code>in</code> List of data block descriptors.
<code>NumDataItems</code>	Number of data block descriptors.
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$  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.

## 2.1.20 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
<code>pData</code>	<b>in</b> Data block to be protected by ECC.
<code>NumBytes</code>	Number of bytes in the data block ( $\leq 1010$ bytes).
<code>pECC</code>	<b>out</b> Calculated ECC.

### Example

Please see *Using API functions to correct bit errors in a data block* on page 58.

## 2.1.21 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
<code>pDataItem</code>	<code>in</code> List of data block descriptors.
<code>NumDataItems</code>	Number of data block descriptors.
<code>pECC</code>	<code>out</code> 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.

## 2.1.22 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
<code>pBuffer</code>	<b>in</b> Encoded ECC. The buffer has to be at least 13 bytes large.
<code>pECC</code>	<b>out</b> ECC in memory format.



## 2.1.23 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
<code>pBuffer</code>	<b>out</b> Encoded ECC. The buffer has to be at least 13 bytes large.
<code>pECC</code>	<b>in</b> ECC in memory format.

## 2.1.24 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

## 2.1.25 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
<code>pData</code>	<code>in</code> Data to be checked and corrected. <code>out</code> Corrected data.
<code>NumBytes</code>	Number of bytes in the data block ( $\leq 2005$ bytes).
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$  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 58.

## 2.1.26 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
<code>pDataItem</code>	<code>in</code> List of data block descriptors.
<code>NumDataItems</code>	Number of data block descriptors.
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$       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.

## 2.1.27 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
<code>pData</code>	<code>in</code> Data block to be protected by ECC.
<code>NumBytes</code>	Number of bytes in the data block ( $\leq 2005$ bytes).
<code>pECC</code>	<code>out</code> Calculated ECC.

### Example

Please see *Using API functions to correct bit errors in a data block* on page 58.

## 2.1.28 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
<code>pDataItem</code>	<code>in</code> List of data block descriptors.
<code>NumDataItems</code>	Number of data block descriptors.
<code>pECC</code>	<code>out</code> 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.

## 2.1.29 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
<code>pBuffer</code>	<b>in</b> Encoded ECC. The buffer has to be at least 42 bytes large.
<code>pECC</code>	<b>out</b> ECC in memory format.

## 2.1.30 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
<code>pBuffer</code>	<b>out</b> Encoded ECC. The buffer has to be at least 42 bytes large.
<code>pECC</code>	<b>in</b> ECC in memory format.



## 2.1.31 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.

## 2.1.32 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
<code>pData</code>	<code>in</code> Data to be checked and corrected. <code>out</code> Corrected data.
<code>NumBytes</code>	Number of bytes in the data block ( $\leq 1977$ bytes).
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$       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 58.

## 2.1.33 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
<code>pDataItem</code>	<code>in</code> List of data block descriptors.
<code>NumDataItems</code>	Number of data block descriptors.
<code>pECC</code>	<code>in</code> Calculated ECC. <code>out</code> Corrected ECC.

### Return value

- $\geq 0$  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.

## 2.1.34 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
<code>pData</code>	<b>in</b> Data block to be protected by ECC.
<code>NumBytes</code>	Number of bytes in the data block ( $\leq 1977$ bytes).
<code>pECC</code>	<b>out</b> Calculated ECC.

### Example

Please see *Using API functions to correct bit errors in a data block* on page 58.

## 2.1.35 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
<code>pDataItem</code>	<code>in</code> List of data block descriptors.
<code>NumDataItems</code>	Number of data block descriptors.
<code>pECC</code>	<code>out</code> 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.

## 2.1.36 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
<code>pBuffer</code>	<b>in</b> Encoded ECC. The buffer has to be at least 70 bytes large.
<code>pECC</code>	<b>out</b> ECC in memory format.

## 2.1.37 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
<code>pBuffer</code>	<b>out</b> Encoded ECC. The buffer has to be at least 70 bytes large.
<code>pECC</code>	<b>in</b> ECC in memory format.

## 2.1.38 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.

```

/*****
 *          SEGGER MICROCONTROLLER GmbH & Co KG
 *          Solutions for real time microcontroller applications
 *****/
 *
 *          (c) 2017 SEGGER Microcontroller GmbH & Co KG
 *
 *          www.segger.com      Support: support@segger.com
 *
 *****/
 *
 *          emLIB * Library collection
 *
 *          Please note:
 *
 *          Knowledge of this file may under no circumstances
 *          be used to write a similar product or a real-time
 *          operating system for in-house use.
 *
 *          Thank you for your fairness !
 *
 *****/
 *
 *          emLIB version: Internal
 *
 *          Current version number will be inserted here
 *          when shipment is built.
 *
 *****/
-----
File       : ECC_Start.c
Purpose    : Sample application that shows the usage of the functions
              for bit error correction.

Additional information:
  Preparations:
    Works out-of-the-box with any project.

  Expected behavior:
    The application performs the following actions:
    - Calculates the 4-bit ECC of an character array
      that contains the string "SEGGER".
    - Forces four bit errors by toggling bits of this array.
    - Displays the information about the forced bit errors.
    - Corrects the bit errors.
    - Displays the information about the corrected bit errors.

  Sample output:
    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
-----END-OF-HEADER-----
*/

/*****
 *
 *          #include section

```

```

*
*****
*/
#include <string.h>
#include <stdio.h>
#include "ECC.h"

/*****
*
*   Public code
*
*****
*/

/*****
*
*   MainTask
*
*/
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] ^= 0x20u;
    abData[2] ^= 0x20u;
    abData[3] ^= 0x20u;
    abData[4] ^= 0x20u;
    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");
    for (;;) {
        ;
    }
}

/***** End of file *****/

```

## 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. <a href="#">ECCAlgo</a> can be one of: BCH4GF8, BCH4GF13, BCH8GF13, BCH24GF14 or BCH40GF14. Default is BCH4GF13.
-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 <a href="#">DataFile.ecc</a>
-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.

## 4.1 General information

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.

## 4.2 Target system configuration

The following table provides information about the hardware and the software that were used to measure the performance and the resource usage:

Detail	Description
CPU	Cortex-M7, 400 MHz
IDE	SEGGER Embedded Studio V8.10
Compiler options	Highest speed optimization

### 4.2.1 Performance

The following table shows the performance values for calculation and correction of emLib ECC. All values are specified in MB/sec.

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)
1-bit	221.0	185.9	-	-	-	-	-	-
4-bit	47.0	37.0	8.4	5.3	4.4	-	-	-
8-bit	37.0	23.3	7.1	5.5	3.6	1.9	-	-
24-bit	9.7	3.4	2.4	2.1	1.7	1.0	0.2	-
40-bit	5.5	1.2	1.1	1.0	0.9	0.6	0.2	0.1

### 4.2.2 Resource usage

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

Error correction capability	ROM (code)	ROM (division LUT)	ROM (GF LUTs)	RAM (stack)
1-bit	500 bytes	-	-	-
4-bit	7.8 KB	2 KB	32 KB	144 bytes
8-bit	7.9 KB	4 KB	32 KB	144 bytes
24-bit	6.7 KB	12.2 KB	64 KB	272 bytes
40-bit	8.7 KB	18.4 KB	64 KB	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

There are different ways to contact the emLib ECC support if you are a registered user:

1. You can send an email to [support@segger.com](mailto:support@segger.com)\*
2. You can create a support ticket at [segger.com/ticket](https://segger.com/ticket).

Please include the following information in the email or ticket:

- Used emLib ECC version.
- Used emLib ECC license number.
- If you are unsure about the above information you can also send the name of the emLib ECC shipment file.
- A detailed description of the problem.
- Optionally, a test project that can be used to reproduce the problem.

---

\*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/>.

## 5.2 Where can I find the license number?

The license number is part of the shipped .zip file name. For example the number of license shipped in the file `emLib_ECC_V100_LIB-01234_AJqI3dkD_231220.zip` file is `LIB-01234`.

The license number is also stored in the header of all the \*.c and \*.h files that are included in the emLib ECC shipment. For example, if you open `ECC.h` you should find the license number as in the example below:

```
-----  
Licensing information  
Licensor:                SEGGER Microcontroller GmbH  
Licensed to:             MyCompany  
Licensed SEGGER software: emLib  
License number:         LIB-01234  
License model:          SSL  
Licensed product:       Any  
Licensed platform:      ARM Cortex M  
Licensed number of seats: -  
-----  
Support and Update Agreement (SUA)  
SUA period:              2024-02-29 - 2024-08-29  
Contact to extend SUA:  sales@segger.com  
-----  
File           : ECC.h  
Purpose        : Error correction routines.  
----- END-OF-HEADER -----
```

# 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.