NEC

User's Manual

EEPROM Emulation Library

32-/16-bit Single-Chip Microcontroller

Document No. U18398EE1V0UM00
Date Published September 2006

Printed in Germany

NOTES FOR CMOS DEVICES

① VOLTAGE APPLICATION WAVEFORM AT INPUT PIN

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (MAX) and V_{IH} (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (MAX) and V_{IH} (MIN).

② HANDLING OF UNUSED INPUT PINS

Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

③ PRECAUTION AGAINST ESD

A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

④ STATUS BEFORE INITIALIZATION

Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

⑤ POWER ON/OFF SEQUENCE

In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current.

The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

⑥ INPUT OF SIGNAL DURING POWER OFF STATE

Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements.

Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

For further information,
please contact:

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
http://www.eu.necel.com/

Hanover Office
Podbielskistrasse 166 B
30177 Hannover
Tel: 0 511 33 40 2-0

Munich Office
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

Stuttgart Office
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

Succursale Française
9, rue Paul Dautier, B.P. 52180
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven
The Netherlands
Tel: 040 265 40 10

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
Tel: 010-8235-1155
http://www.cn.necel.com/

NEC Electronics Shanghai Ltd.
Room 2509-2510, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai P.R. China P.C:200120
Tel: 021-5888-5400
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
12/F., Cityplaza 4,
12 Taikoo Wan Road, Hong Kong
Tel: 2886-9318
http://www.hk.necel.com/

Seoul Branch
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600
http://www.tw.necel.com/

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
http://www.sg.necel.com/

G06.8A

- **The information in this document is current as of September, 2006. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)
(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

Introduction

Readers

This User's Manual is intended for users who want to understand the functionality of the EEPROM emulation for devices of the V850 core family.

Purpose

This User's Manual explains the background and handling of the EEPROM emulation on data and on code Flash.

Organization

This application note contains the major sections:

- Emulation strategy
- Emulation library structure
- Emulation library usage
- Emulation library APIs

Legend

Symbols and notation are used as follows:

Weight in data notation	:	Left is high-order column, right is low order column
Active low notation	:	$\overline{xxx}$ (pin or signal name is over-scored) or /xxx (slash before signal name)
Memory map address:	:	High order at high stage and low order at low stage
Note	:	Explanation of (Note) in the text
Caution	:	Information requiring particular attention
Remark	:	Supplementary explanation to the text
Numeric notation	:	Binary... xxxx or xxxB Decimal... xxxx Hexadecimal... xxxxH or 0x xxxx

Prefixes representing powers of 2 (address space, memory capacity)

K (kilo): $2^{10} = 1024$

M (mega): $2^{20} = 1024^2 = 1,048,576$

G (giga): $2^{30} = 1024^3 = 1,073,741,824$

Table of Contents

List of Figures

Chapter 1 Overview

This User's Manual describes the usage of NEC's EEPROM emulation library (EEELib) featuring EEPROM emulation on NEC's microcontroller with embedded single voltage data flash.
The present document describes the higher layer structure of the NEC's EEPROM emulation approach, whereas the lower layer DFALib is described in a separate document.
The EEELib realises the emulation related features, such as data set handling, fail-safe functionality, start-up handling and call of the DFALib.

1.1 Naming Conventions

Certain terms, required for the description of the Flash and EEPROM emulation are long and too complicated for good readability of the document. Therefore, special names and abbreviations will be used in the course of this document to improve the readability.

Background Section
Section of the EEPROM emulation that is currently not holding valid data and is not used for any operation.

Data Flash
Data flash is a separate flash macro on the device. Due to its implementation the dual operation is supported. That means code operation in the code flash while the data flash is being operated, e.g. write or erase. Additionally the ID-Tag is supported, that allows a hardware supported search.

DFALib
DFALib is the short form for "Data Flash access library".

EEELib
EEELib is the short form of "EEPROM emulation library".

Flash Environment
This is device internal hardware (charge pumps, sequencer,...), required to do any Flash programming. It is not continuously activated, only during Flash modification operations. Activation and deactivation is done implicitly by the *DFALib*.

Foreground Section
The active section of the EEPROM emulation that is currently holding valid data and is not used for any operation such as read or write.

1.2 Limitations

- During data flash operation no power safe mode should be applied as it will leave the flash in an undefined state when the clock is stopped. For details on the power safe modes please refer to the appropriate chapter of the device UM

- The EEELib in combination with the DFALib is only to be used on the device series this UM appendix is related to. For other usage NEC Electronics can not take any reliability.

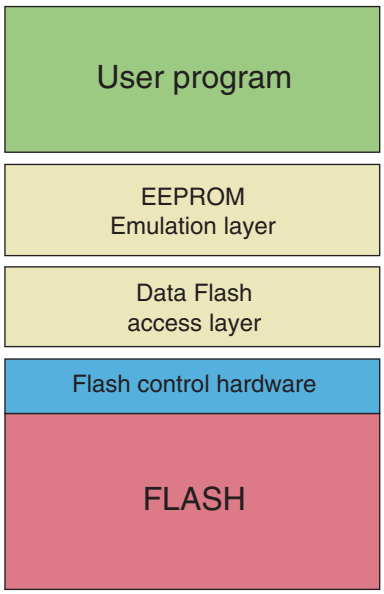
Caution: Please read all chapters of the users manual carefully and follow exactly the given sequences and recommendations. This is required in order to make full use of the high level of security and safety provided by the devices

[MEMO]

Chapter 2 Emulation Library - Structure

2.1 Layer Model

Figure 2-1: EEPROM Emulation Layer Model



The EEPROM emulation is implemented in a strictly layered manner. So that it is in principle possible, to set-up a user dedicated EEPROM emulation strategy, differing from the NEC EEPROM emulation layer, but based on the Data Flash access layer.

2.1.1 Flash control hardware

The Flash hardware can do simple Flash operations, like Flash block erase, word write, data search (Data Flash only) etc. in background after initial configuration. The upper layers and the user program can then continue operation, while the Flash hardware works. Operation end can either be polled or signalled by an interrupt.
The operation and the configuration of the hardware is not open to the user, so the usage of the data Flash access layer is mandatory.

Note: Please refer to the *DeviceInfo* to check, whether the operation end interrupt is also available on the used device

2.1.2 Data Flash access layer

The data Flash access layer (*DFALib*) configures the Flash hardware to do the basic Flash operations, This layer is transparent for the user, if the NEC EEPROM emulation layer is used.
The *DFALib* is explained in a separate document.

2.1.3 EEPROM emulation layer

The EEPROM emulation layer is represented by the *EEELib*. This library calls the *DFALib* for all Flash related operations.
The library contains a software state machine, controlling all EEPROM emulation related operations. State machine operations are initiated by a central function. An input structure defines the kind of operation and the parameters that are required.
Depending on the library configuration, the function returns after operation completion or after operation initiation. In the later case the user program need to frequently call a handler routine which then handles the state machine.

Chapter 3 Emulation Strategy

3.1 Basic strategy

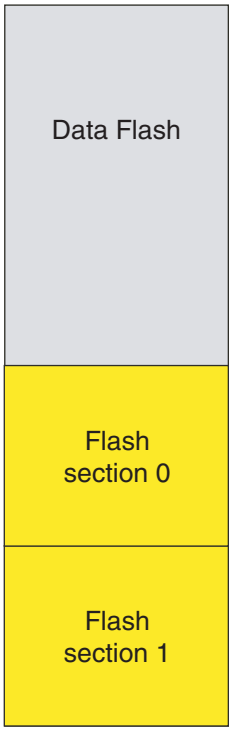
3.1.1 Difference between real EEPROM and emulated EEPROM

In opposite to classical EEPROM's, where the data is stored on a fixed address and so can always be found on the same location, EEPROM emulation need to store data on changing locations. Indeed the physical behaviour of a flash cell does not allow to program different values to the same cell without erasing before re-writing a value. Furthermore, write and erase granularity differs. While write operations are done on word base (32 bit), the erase operations are done on block base (2 kBytes).
NEC's EEPROM Emulation does not support the random byte access, but allows to store sets of data with a unique ID.
Writing new data sets is easily done, by appending the data to the Data Zone (Please see below for details on the content and the structure).

3.1.2 Emulation Concept

The NEC EEPROM emulation concept on the V850 series with data flash is based on two sections. The part of the data flash, that should be used for the EEPROM emulation is divided in two equal sections that are used in alternating manner. With the chosen concept and the given specification of the Data Flash a high number of write cycles can be achieved.

Figure 3-1: EEPROM Emulation Sections in the Data Flash



Notes: 1. A Flash section must always consist of a number of Flash blocks following the rule **NoBlocks = 2^n** with n=0, 1, 2, 3,... ==> NoBlocks = 1, 2, 4, 8,...

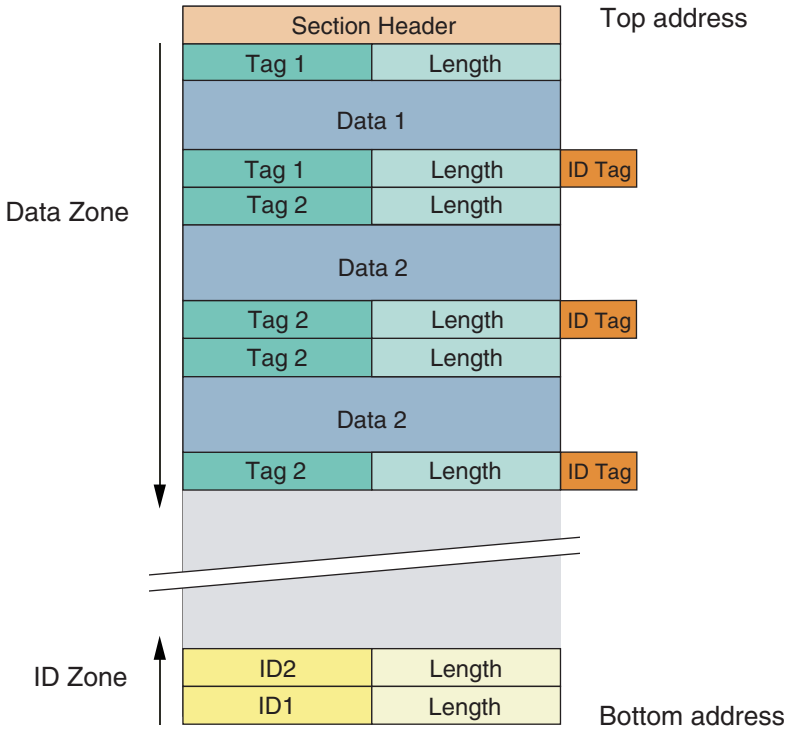
2. Furthermore, the section must start on a block number following the rule: **StartBlock = NoBlocks * n** with n=0, 1, 2, 3,...

3.1.3 Section overview

Each Flash section consists of 3 parts:

- Section Header
- Data Zone
- ID Zone

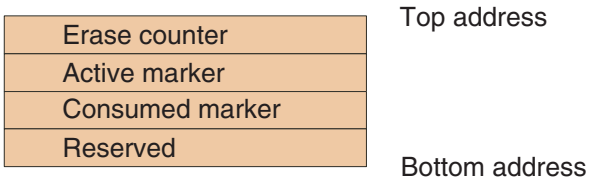
Figure 3-2: EEPROM Emulation Section Overview



(1) Section header

Section header size is 16 Bytes (4 words), where currently 3 words are used to define the current section status.

Figure 3-3: Section Header



The erase counter word consists of the 16-bit counter and a 16-bit inverse value for protection. It describes the number of erase cycles on the data section.

When set, the active marker or the consumed marker have the value 0x55555555. The value of a marker which is cleared is 0xFFFFFFFF (Blank Word)

The meaning of the section header information is described below. Please refer to 3.5 “Basic Flow” on page 19

(2) Data zone

The data zone contains the data sets. New data sets are simply appended by the EEELib after the last written data set. So the data zone grows down in the address space.
The section is full and a *Refresh* is required as soon as there is not enough erased space between ID zone and Data Zone (last written address) for the data set to be written.
That *Refresh* can be performed at any time, when the free Data Zone space is under a certain - user defined - limit. That limit can be inspected by a service function of the EEELib that allows to check the remaining space in the data zone.

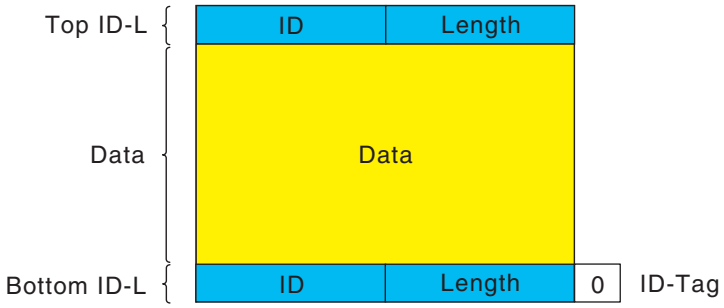
(3) ID zone

The ID-zone is a list of IDs of the data sets written to the section so far. The ID-list is dynamically generated during writing new data sets into the section.
An ID-List is required in order to simplify and to speed up the processing. The solution to keep the ID-list dynamically in the EEPROM Emulation Flash space instead of keeping it statically in code the Flash area has been chosen in order to allow new/added applications to store data sets with new IDs.
The ID-zone grows in case of writing new data sets with new IDs.

3.2 Data Set Representation in the Flash

A data set is always embraced by a word containing ID and length information on the top and on the bottom. This word is called ID-L words furtheron.

Figure 3-4: Data Set Representation



The implemented NEC Data Flash is 33-bit wide. The 33-rd bit is called the ID-Tag, indicating that the word contains internal information and is not part of the data set itself.
The current implementation in the EEPROM emulation layer sets this on the bottom ID-L, while keeping it un-set on the data bits and on the top ID-L.
Dedicated hardware on the data Flash macro can search for a certain ID in the ID-L words by using this ID-Tag in background while code execution from code flash continues.

Note: Usage of the IDs 0xFFFF and 0x0000 for normal data sets is not allowed, as these IDs are used for special emulation strategy internal purposes.

3.3 Basic Operation

3.3.1 Startup1 / 2 operation

For details on the *Startup1* and *Startup2* phase please refer to 3.6 “Init Phase” on page 21.

3.3.2 Refresh operation

The *Refresh* operation is used to copy the latest data sets of all valid IDs from a full Flash section to a prepared Flash section and to activate this section for subsequent *Read* and *Write* operations.
Once the refresh operation is called, the previously prepared section becomes active and the previously active section becomes inactive. All following read/write operations will be done in the newly active section.

3.3.3 Prepare operation

The *Prepare* operation checks, whether the non active section is already prepared.
If the section is already prepared it returns immediately, if not it does a blank check, followed by erase and write the erase counter as indication, that the section is now prepared.
Prepared in this term means:

- The Flash section is erased
- The section header contains the necessary information

Note: As the *Prepare* operation returns immediately without Flash operation when the section is already prepared, the *Prepare* operation can be called frequently, e.g. time driven.

3.3.4 Write operation

The data set *Write* operation writes a complete data set to the active section.
The data is written first, followed by the top ID-L and the bottom ID-L. Last the ID zone is checked whether the ID-L is already contained. If not, the ID-L is added to the ID-zone.

3.3.5 Read operation

The *Read* operation searches the current valid data set according to the ID in the function call parameter and copies the data set to the destination location.
In some cases it may make sense to read only parts of bigger data sets in order to save read buffer space. Therefore, the *Read* operation has a parameter to define an offset within the data set and a length parameter to define the number of bytes to be read.

3.3.6 Format operation

Note: The *Format* operation is not needed in the normal application operation therefore it is not mentioned in above flow.

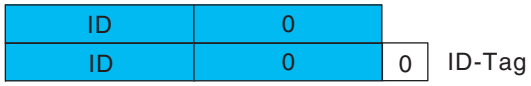
The *Format* operation prepares both Flash sections and activates one of them for data set *Read* and *Write* operations. This operation has to be used for completely erased Data Flash. After the *Format* operation all data sets are deleted. This operation has to be used, if the application itself prepares an erased Data Flash for usage.

Alternatives to using the *Format* operation in the application are using a Debugger (e.g. GHS) or a Flash programmer (e.g. the NEC PG-FP4) to format the data Flash and to copy initial data sets into the sections. Tools to support that are available.

3.3.7 Invalidation of data sets

Data sets can be invalidated before writing a data set new.
A *Read* operation on an invalidated data set does not return the latest values, but returns with an error, indicating that the data is obsolete.
The user application can e.g. react on this by taking default values for this data set. Invalidating a data set is done by writing this data set with the length information 0 (Please refer to 5.2.2 “Operation initiation” on page 31 for further explanation).
The data set is then represented in Flash as the following:

Figure 3-5: Invalidated Data Set



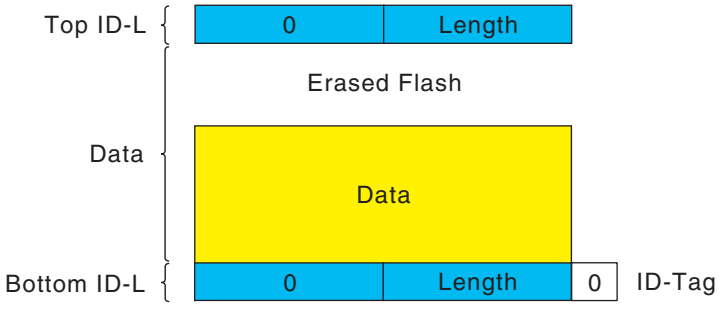
3.3.8 Operation abort

Some emulation operation require quite some time to finish, like flash erase, write, ID search.
In case of an emergency the time for finishing the operation is not given, but some emergency data must be written. So the operation are aborted at appropriate state:

- ID search is just stopped
- Flash erase during a prepare operation is stopped, resulting in an invalid background section. A later prepare fixes this.

A special case is the *Write* operation. This operation can not be directly interrupted as it would lead the active section in an undefined state.
The data set *Write* operation can be aborted until writing the Top ID-L has started. After that point of time, the operation is finished, as this is faster than the abort procedure.
Abort is done by finishing writing the current data word, then writing an invalid top ID and an invalid bottom ID. This data set is then invalid, but the overall data set structure is still consistent.
This kind of abort is called data encapsulation, as invalid data is encapsulated by ID-L’s indicating invalid data.

Figure 3-6: Encapsulated (still erased) Data



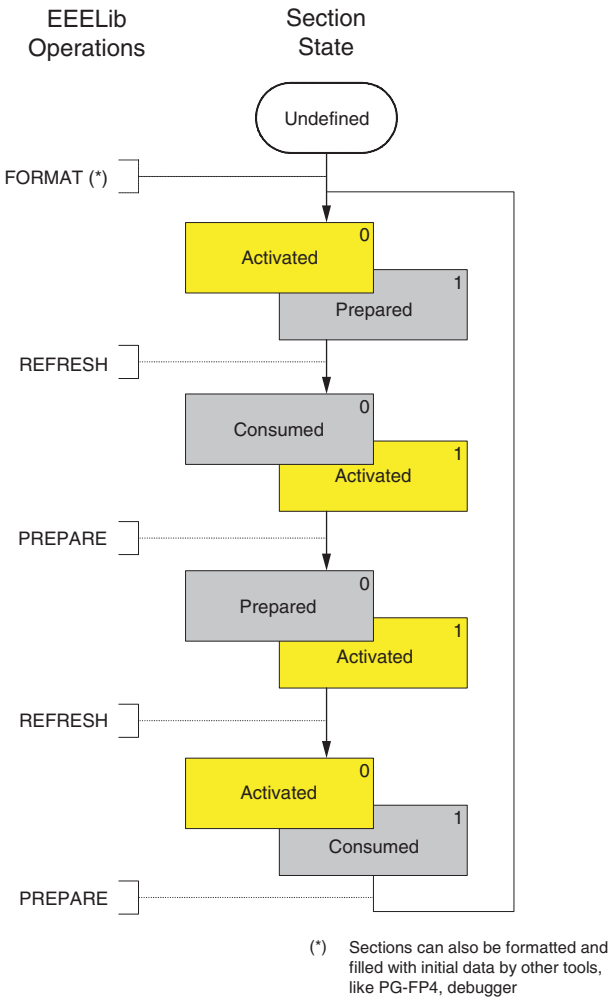
3.4 Data Flash Interrupt

In polling mode the EEELIb_Handler is frequently called by the application. With support of the device specific interrupt “upon data flash operation completion” that frequent call to the EEELib_Handler is not necessary, as the EEELib_Handler has to be called in the interrupt context.

Note: For details on the interrupt please refer to the appropriate chapter of the device. The availability of the interrupt depends on the delivery package’s compile options.

3.5 Basic Flow

Figure 3-7: Section State Transition During Normal Operation



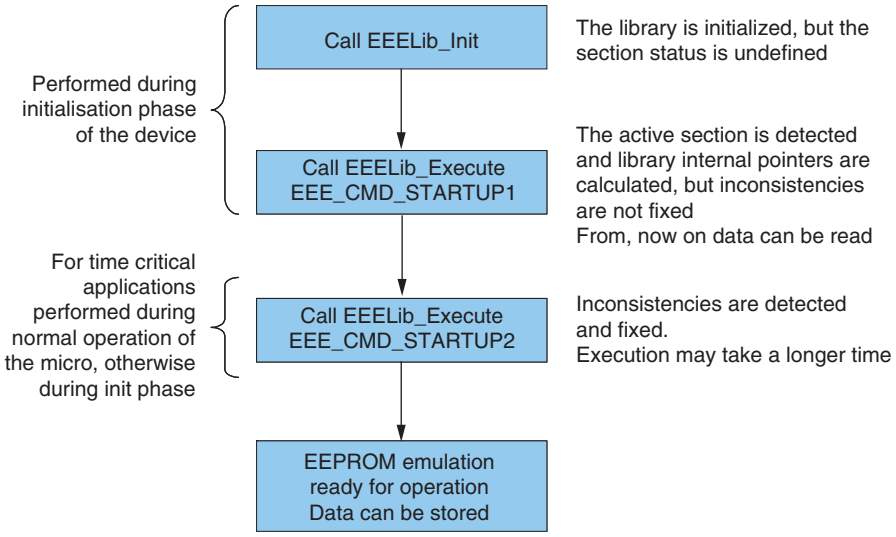
The following section states are possible:

- Prepared:
  This section is ready for activation by a *Refresh* operation. The section has been erased, and in the section header the erase counter is written. Active marker and consumed marker are cleared.
- Active:
  The section contains the latest data sets. The EEPROM emulation can read and write data to this section.
  In the section header additionally to the erase counter also the active marker is set. Consumed marker is cleared.
- Consumed:
  After activating the next section and copying the latest data sets there, the *Refresh* operation marked the “old” active section as consumed.
  In the section header additionally to the erase counter and active marker, also the consumed marker is set.
  Next step is to *Prepare* the section.
- Invalid:
  This is no valid state. During operation it can only occur in case of interruption of a Flash operation like erase to this section.
  The section header may contain any other data except as described in the three states before. Also a completely erased Flash (default factory delivery state) is invalid from EEPROM emulation point of view.

3.6 Init Phase

The start-up procedure is split up into different parts. This splitting is chosen because of the requirement to perform any read operation on EEPROM data after a very short time after reset release. The principle flow is described in below figure.

Figure 3-8: Startup Procedure



Before being in normal operation, the *EEELib* has to sequentially pass dedicated initialisation phases with different conditions and possible operations.

In the explanation below two different device operational phases are distinguished. This is application dependent. However, most applications can be separated into these phases:

- Device initialisation phase
  The device hardware and application software is initialized. An operating system is not yet started, so the code is executed sequential. No restrictions regarding the *EEELib* handler (See 5.2.3 “State machine handler” on page 34) run-time, just the *EEELib* operation overall run-time is important.
- Device operational phase
  An operating system is started. If the *EEELib* is executed in a task (e.g. timed or interrupt driven), beside the overall *EEELib* operation run-time, also the function run-time (E.g. *EEELib* handler) must be limited.

The following phases have to be passed sequentially: **<1> → <2> → <3> → <4> → <5>**:

<1> Library is not initialized
This phase should be handled during the device initialisation phase

Status:
EEELib status is undefined. No EEPROM Emulation operation is allowed.

User application actions:
Call the function EEELib_Init

<2> Library is initialized, Section status is undefined
This phase should be handled during the device initialisation phase

Status:
EEELib is initialised, but the section status is undefined. All EEPROM Emulation operations except *Start-up*1 or *Format* are locked and result in a Flow error, if trying to execute them

User application actions:
In the standard case of sections contain valid data sets, even in that case, that previous program execution was interrupted by a power fail. In this case, execute the *Start-up*1 operation.
In case of blank *EEELib* sections, execute *Format* operation first and after operation end, execute *Start-up*1

Conditions:
The *Start-up*1 operation parses all data sets in the active section and if required also in the consumed section in order to calculate library internal pointers. Depending on how many data sets have to be parsed, the execution time of *Start-up*1 is in the range of several hundred us. During that time, the CPU is loaded to 100%.

<3> Library passed the *Start-Up* Phase 1
This phase should be handled during the device initialisation phase

Status:
The data section status is defined, but inconsistencies (not completed former operations due to e.g. power fail) are not fixed. All EEPROM Emulation operations except *Start-up*2, *Read* or *Format* are locked and result in a Flow error, if trying to execute them

User application actions:
In case of early required EEPROM data sets, these data sets can be read using the *Read* operation. If no EEPROM Emulation data is required in this phase, it can be skipped

Conditions
In this phase the *Read* operation differs from the normal operation:
- The data set search uses a different mechanism. While in normal operation on data Flash a background hardware does the search operation, here the CPU jumps from data set to data set to find the latest data. So, the timing is different and the CPU is loaded to 100% during the *Read* operation
- In case of a preceding interrupted *Refresh* for writing emergency data followed by a RESET and *Start-up*1, the written emergency data cannot be read, as the *Read* operates on the elder section in case of detection of the interrupted *Refresh*. So, it cannot be assured, that emergency data can be read in this phase. Possibility to read these is given after *Start-up*2 phase

<4> Early required data sets are read
This phase should be handled during the device initialisation phase and/or the device operational phase (See below)

Status:
as <3>

User application actions:
The *Start-up*2 operation is executed

Conditions:
The *Start-up*2 operation requires a long execution time. As the long lasting operations are executed by the Flash hardware in background, the CPU load is low.
Up to the given user application conditions this phase may be started and handled (Call the Handler function, see) during the device initialisation phase if enough time is given
Another option is to start the *Start-up*2 operation in the device initialisation phase (one long lasting operation is executed in background) and to call the handler in the device operational phase. So, the *EEELib* is operational relatively early and the device start-up time is not extended more than necessary

<5> Start of normal operation
This phase is handled during the device operational phase (regarding operation refer to Figure 3-7, “Section State Transition During Normal Operation,” on page 19)

Status:
The *EEELib* is up and running, all operations are possible.
However, due to eventually necessary operations during *Start-up*2, the active section might be full and/or the background section might be not prepared.
In order to avoid *Write* operation errors, it should be considered to execute a *Prepare* and a *Refresh* operation first

Conditions:
Normal operation. After starting a *EEELib* operation, the *EEELib* Handler must be called frequently in order to finish the operation timely.

[MEMO]

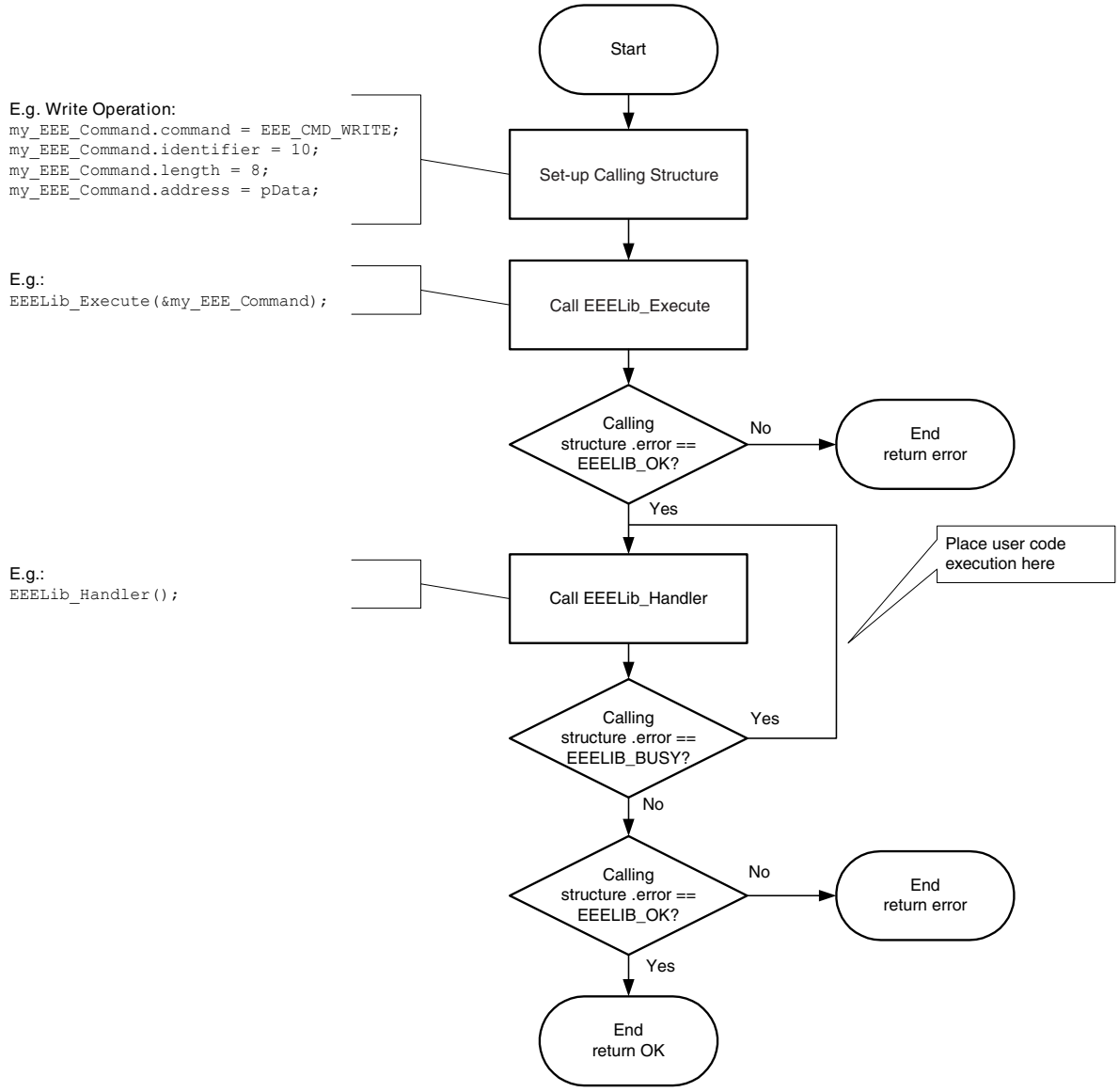
Chapter 4 Emulation Library - Usage

4.1 Application Execution During EEPROM Emulation

4.1.1 Principle Flow of the EEPROM emulation function call

The EEELib_Handler function is called frequently by the user application. The *EEELib* operation initialization call (EEELib_Execute) as well as the EEELib_Handler calls require only short time. The user function can execute any code in between the EEELib_Handler function calls.

Figure 4-1: Principle Flow of an EEPROM Emulation Function Call



4.1.2 User Interrupts

Any interrupt function can be executed at any time, even when operation on the data flash is active. As the code Flash is accessible during data flash operation, e.g. erase or write, also the interrupt vector table is available.

4.2 Library Software

The EEELib is delivered as a pre-compiled version. Due to that some compile-time configurations in the source code are fixed.

Parameter	Description
DFALib Status check / *EEELib* state machine handler	The Flash background operations (e.g. Flash erase) need some time. Before continuing with any Flash related operation, like stepping forward the *EEELib* internal state machine, the SW need to check the end of the Flash operation. The *EEELib* handler responsible for these checks can either be called by the user application until the EEPROM emulation operation is finished or it can be called inside the *EEELib*.
Flash Block Size	The Flash block size of the used device is set with this parameter. Current implementations is 2 kBytes (0x800)
User call-back functions	In case of library internal status check, the library does a status check loop to wait for the end of a Flash operation. In that loop a user function can be called.
Flash related information for the EEPROM emulation	The Flash base address depends on the used Flash type. Data Flash start address depends on the device. Please refer to the device documentation

Please check the delivery package regarding the setting of these configurations. Please also refer to the application note of the EEELib.

4.2.1 Supported compilers

General code from the application note supports compilers from GHS, IAR and NEC.
Pre-compiled libraries according to this document are compiled and tested with one dedicated compiler version and dedicated compiler switches that are explicitly listed up in the release package.
The user application must be compiled according not to interfere with these settings.

4.2.2 Library delivery packages

The *DFALib* and *EEELib* are strictly separated, so that the *DFAlib* can be used without the *EEELib*. However, using *EEELib* without *DFALib* is not possible.

EEELib delivery package

[root]

nec_types.h	NEC types convention compile time
[*EEELib*]	
EEELib.a	Pre-compiled EEELib and DFALib library code

4.3 Section Handling

The following sections are EEPROM emulation library related:

- DFALib_Text, DFALib_TextRAM
  This is *DFALib* code.
  This section is executed from code flash.

- EEELib_Text
  This is *EEELib* code.
  This section is executed from code flash.

- EEELib_Data
  This is the section containing *EEELib* internal data. It must be located to (internal) RAM.

Note: The above named section have to be added to the projects linker file.

4.4 MISRA Compliance

The EEELib and DFALib have been tested regarding MISRA compliance.

The used tool is the GHS V4.0.7 MULTI development environment, which tests on the predecessor of the MISRA 2004 standard.

The following rules had to be disabled, as enabling would result in significant disadvantages in terms of code size, performance or code readability:

- Rule 23: Module local functions shall be defined as static
  The checker did not recognize static functions as local
- Rule 45: Casting from and to pointers not allowed
  This is intensively used for all sfr accesses due to performance and code size reasons
- Rule 90: c-macros shall be used for constants, function-like or specifier/Qualifier only
  This rule prohibits defines, assigning former defines. This is used in the libraries for better code readability and reduction of possible user mis-configuration
- Rule 96: Use of parenthesis in function like macros
  This rule prohibits macros with parameters defined as empty macros. This is used in the libraries for all debug outputs definitions.

[MEMO]

Chapter 5 Emulation Library - API

The EEPROM Emulation library provides two types of user functions:

- Operational functions
  Operational functions control the standard EEPROM emulation operations like *Read*, *Write*, *Prepare*, etc.

- Service functions
  Providing service information to the user.

5.1 Used Types

The structure is passed to the function EEELib_Execute to control the state machine and to read the status and operation results.

```
typedef struct
        {
        u32*        address;
        u16         identifier;
        u16         length;
        u16         offset;
        tEEE_COMMAND command;
        tEEE_ERROR    error;
        } tEEE_REQUEST;
```

command — Distinguishes between the different operations *Start-up1, Start-up2*, *Read*, *Write*, *Refresh*, *Format* and *Prepare*

error — various errors are possible, depending on the operation.
The errors are either set by the initial function EEELib_Execute or by the handler

EEELib_Handler

Table 5-1: Error States

Error Name	Description
EEE_OK	The operation finished successfully
EEE_DRIVER_BUSY	The state machine is busy This value is set, when a new operation has been successfully started by EEELib_Execute
EEE_ERR_PARAMETER	Wrong parameters have been passed to the *EEELib* (See the different commands)
EEE_ERR_SECTION_FULL	A data set *Write* attempt failed, as the section is full A *Refresh* operation is required
EEE_ERR_POOL_FULL	A *Refresh* operation failed, as the background section is not prepared
EEE_ERR_COMMAND_UNKNOWN	An unknown command has been passed to the *EEELib*
EEE_ERR_FLASH_ERROR	A Flash operation of the *DFALib* (called by *EEELib*) failed due to a Flash problem
EEE_ERR_USERABORT	The last operation has been aborted (Only applicable in case of user status check)
EEE_ERR_READ_UNKNOWNID	A *Read* operation did not find a data set with the requested ID
EEE_ERR_READ_OBSOLETE	The data set with the ID is marked invalid. The data is not read, instead the error is returned
EEE_ERR_NOACTIVESECTION	This error is returned in case of unformatted Flash
EEE_ERR_FLOW	During startup an operation, that is not allowed according to the *EEELib* initialisation status was called
EEE_ERR_OPERATION_REJECTED	A new operation is tried to be initiated although the state machine is still busy (Only applicable in case of user status check)

address
identifier
length
offset — These parameters are individually used by the different operations (See operations description).

5.2 Operational Functions

EEPROM operation is controlled by an *EEELib* internal state machine. The operations are initiated by the function EEELib_Execute.
The function EEELib_Handler controls the state machine after operation initiation and forwards the states. The handler function is called by the user application.

5.2.1 State machine initialization

Function prototype
void EEELib_Init(void);

Required parameters
-

Return value
-

Function Description
This function initializes the *EEELib* state machine. The function must be called once before the 1st EEPROM emulation operation.

5.2.2 Operation initiation

Function prototype
void EEELib_Execute(tEEE_REQUEST* request);

Required parameters
request Command structure as described below

Return value
request.error Values as described in 5.1 “Used Types” on page 29.

Function Description
Initiates a EEPROM emulation operation.
The return value request.error generated by the EEELib_Execute are marked with (1), whereas the One’s generated by the EEELib_Handler are marked with (2)

- *Start-up*1

  required parameters:
  request.command EEE_CMD_STARTUP1

  return (See 5.1 “Used Types” on page 29):
  request.error EEE_DRIVER_BUSY (1)
  EEE_ERR_OPERATION_REJECTED (1)
  EEE_OK (2)
  EEE_ERR_NOACTIVESECTION (2)

- *Start-up*2

  required parameters:
  request.command EEE_CMD_STARTUP2

  return (See 5.1 “Used Types” on page 29):
  request.error EEE_DRIVER_BUSY (1)
  EEE_ERR_OPERATION_REJECTED (1)
  EEE_ERR_FLOW (1)
  EEE_OK (2)
  ERR_FLASH_ERROR (2)

- *Prepare*
  required parameters:

  request.command EEE_CMD_PREPARE

  return (See 5.1 “Used Types” on page 29):
  request.error EEE_DRIVER_BUSY (1)
  EEE_ERR_FLOW (1)
  EEE_ERR_OPERATION_REJECTED (1)
  EEE_OK (2)
  ERR_USERABORT (2)
  EEE_ERR_FLASH_ERROR (2)

- *Refresh*
  required parameters:

  request.command EEE_CMD_REFRESH

  return (See 5.1 “Used Types” on page 29):
  request.error EEE_DRIVER_BUSY (1)
  EEE_ERR_FLOW (1)
  EEE_ERR_OPERATION_REJECTED (1)
  EEE_OK (2)
  EEE_ERR_USERABORT (2)
  EEE_ERR_POOL_FULL (2)
  EEE_ERR_FLASH_ERROR (2)

- *Write*

  required parameters:
  request.command EEE_CMD_WRITE
  request.address Source address of the *Write* operation
  request.identifier Identifier of the data set to be written in bytes
  request.length Length of the data set to be written.
  The write granularity is 4 Bytes for Data Flash
  Length=0 defines an invalidation data set.
  (See 3.3.7 “Invalidation of data sets” on page 18)

  return (See 5.1 “Used Types” on page 29):
  request.error EEE_DRIVER_BUSY (1)
  EEE_ERR_PARAMETER (The write granularity of 4 Bytes
  on
  Data Flash responsible 8 Bytes for Code Flash is not met
  or the ID is 0x0000 or > 0xFFFE) (1)
  EEE_ERR_FLOW (1)
  EEE_ERR_OPERATION_REJECTED (1)
  EEE_OK (2)
  EEE_ERR_USERABORT (2)
  EEE_ERR_SECTION_FULL (2)
  EEE_ERR_FLASH_ERROR (2)

- *Read*

  required parameters:

request.command	EEE_CMD_READ
request.address	Destination address of the *Read* operation. The data set (part) is copied here
request.identifier	Identifier of the data set to be read
request.length	Number of bytes to read
request.offset	Offset within the data set to read. Together with length information this is used to read data set fragments

  return (See 5.1 “Used Types” on page 29):

request.error	EEE_DRIVER_BUSY (1) EEE_ERR_PARAMETER (data set length < (request.length + request.offset) (1) EEE_ERR_FLOW (1) EEE_ERR_OPERATION_REJECTED (1) EEE_OK (2) EEE_ERR_READ_UNKNOWNID (2) EEE_ERR_READ_OBSOLETE (2) EEE_ERR_USERABORT (2)

- *Format*
  required parameters:

request.command	EEE_CMD_FORMAT

  return (See 5.1 “Used Types” on page 29):

request.error	EEE_DRIVER_BUSY (1) EEE_ERR_OPERATION_REJECTED (1) EEE_EEE_OK (2) EEE_ERR_USERABORT (2) EEE_ERR_FLASH_ERROR (2)

- Other commands
  Other commands are not available, an error is returned

request.command	Any other unknown command number

  return (See 5.1 “Used Types” on page 29):

request.error	EEE_ERR_COMMAND_UNKNOWN (1)

5.2.3 State machine handler

Function prototype
void EEELib_Handler(void);

Required parameters
The request structure, previous passed to the function EEELib_Execute, is used.

Return value
The request structure, previous passed to the function EEELib_Execute, is modified.
request.error Values as described in 5.1 “Used Types” on page 29.

Function Description
Handles the *EEELib* state machine and forwards the states.

Notes: 1. Depending on the definition for the handler execution, this handler function is either called *EEELib* library internally or by the user application. In case of library internal call, this function is not required and may not be used in the user application.

2. Do not change the command structure information until the *EEELib* operation is finished (requestStructure.error != EEE_DRIVER_BUSY)

5.2.4 State machine abort

Function prototype
void EEELib_Abort(void);

Required parameters
-

Return value
-

Function Description
This function indicates the user abort request to the state machine. The state machine aborts an EEPROM emulation operation as soon as possible.

Notes: 1. After the abort instruction call the handler function must still be called until the state machine has completed the abort procedure. This is indicated by the request.error accordingly.

2. Aborts before finish of the *Start-up*2 operation are not possible, so ignored.

5.3 Service Functions

5.3.1 Check free section space

Function prototype
tEEE_ERROR EEELib_GetSpace(u32 *space);

Required parameters
space pointer to the variable, that shall receive the section space information

Return value

EEE_OK	The operation finished successfully
EEE_DRIVER_BUSY	The *EEELib* state machine is busy, the size information could not be calculated

Function Description
This function calculates the free space inside the active section for new data sets. This is the space between the end of the data zone and the end of the ID zone.

5.3.2 Get the erase counter

Function prototype
tEEE_ERROR EEELib_GetEraseCounter(u32 *eraseCnt);

Required parameters
eraseCnt pointer to the variable, that shall receive the erase counter information

Return value

EEE_OK	The operation finished successfully
EEE_DRIVER_BUSY	The *EEELib* state machine is busy, the size information could not be calculated

Function Description
This function reads the erase counter of the EEPROM emulation sections.

5.3.3 Get the EEELib version

Function prototype
u32 EEELib_LibVersion(void);

Required parameters
-

Return value
EEELib version as decimal number.

Function Description
This function returns the *EEELib* version as decimal number.

[MEMO]

Chapter 6 Electrical Specification

Timing Characteristics

(T_A = -40 to +125°C,
$V_{DD} = EV_{DD} = BV_{DD}$, $4.0 \leq AV_{REF0} \leq 5.5$ V, $V_{SS} = EV_{SS} = BV_{SS} = AV_{SS}$ = 0 V,
C_L = 50 pF,
f_{XX} = 48 MHz)

Parameter	Conditions	MIN.	TYP.	MAX.	Unit
Write	4 Bytes	396	495	2300	us
	8 Bytes	508	635	3037	us
	12 Bytes	621	776	3774	us
	16 Bytes	486	608	2436	us
	24 Bytes	711	889	3911	us
	28 Bytes	823	1029	4648	us
	32 Bytes	690	863	3310	us
	64 Bytes	1092	1365	5051	us
	128 Bytes	1897	2371	8533	us
Read	Last - 64	128	160	192	us
	First - 64	1754	2193	2632	us
Read by SW	Last - 4	118	148	178	us
	First - 4	753	941	1129	us
Prepare		21	26	89	ms
Refresh	1 ID	1,2	1,5	6	ms
	10 IDs	9,6	12	45	ms
	20 IDs	20	25	88	ms
	40 IDs	40	51	177	ms
Start-up1	Number of IDs=0 Fill ratio 0	120	151	182	us
	Number of IDs=10 Fill ratio 0,6	352	441	530	us
Start-up2	Number of IDs=0 Fill ratio 0	30	38	46	ms
	Number of IDs=10 Fill ratio 0,6	29	37	45	ms

Note: All measurements are done with 16 kBytes section size.

[MEMO]