

Storage system

Presentation EEPROM Emulation driver

- Presented by Erik Christensen, SW Designer, IFWD

never stop thinking

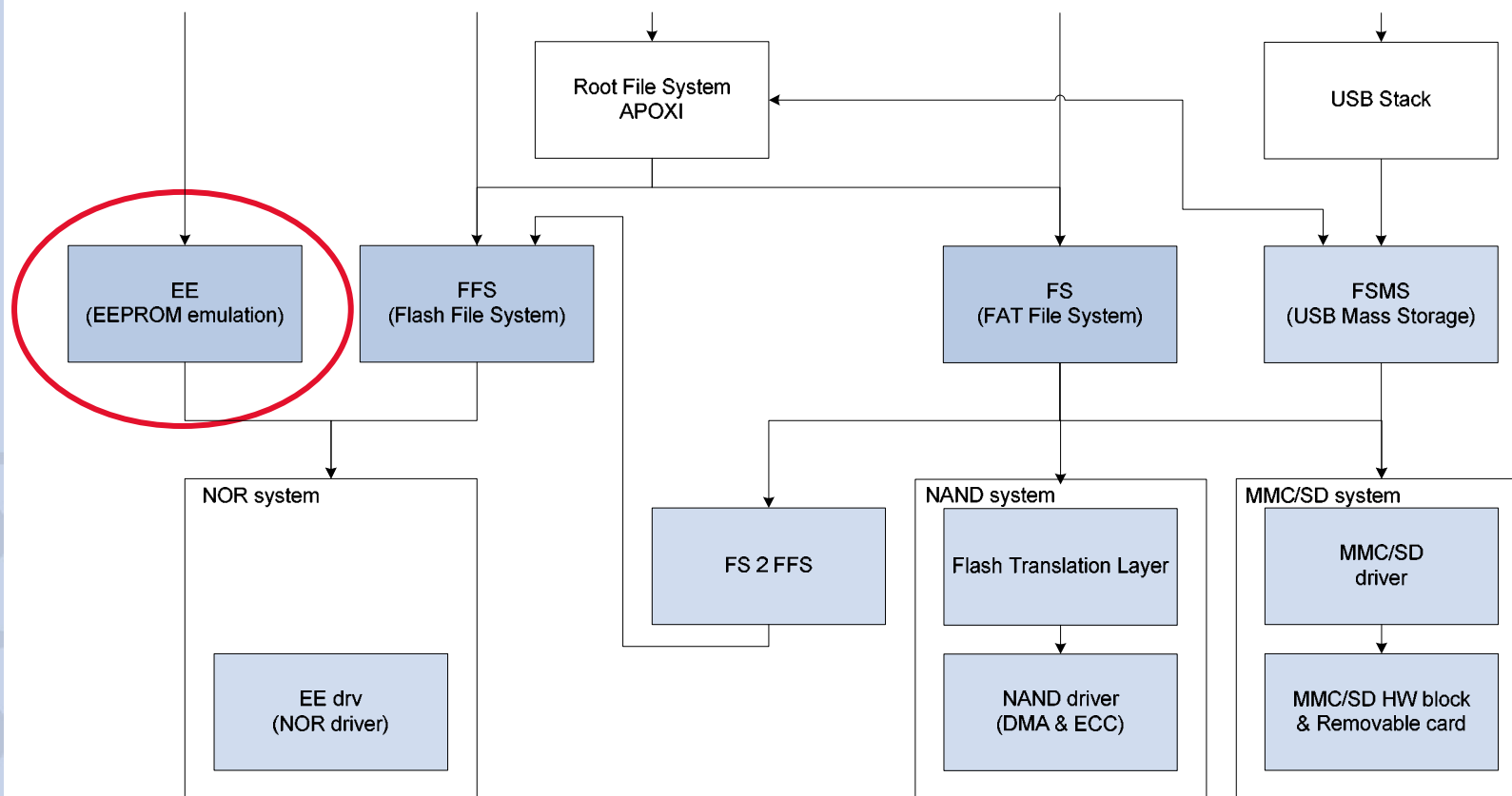


Agenda

- Overview
- Features
- Architecture
- API
- Questions

Overview

- The EE driver is a part of the NOR system in the overall storage system.



- The purpose of the EE driver is to make it possible to store parameters non-volatile in the system.

Features

- Non-volatile parameter storage in flash
- Power loss proof parameter handling.
- Static part: Read only section for calibration and customization parameters
- Dynamic part: For dynamically changed parameters (e.g. settings). Read and write access allowed.
- Factory-settings restore supported for the dynamic part
- Non-volatile Security data handling
- Exception storage handling
- Debug log handling for RAM dumps etc.

Architecture (main parts)

Dynamic Part

These parameters can be changed during runtime in normal operation mode.

E.g.:

If the user changes a menu-setting.

These parameters can also be adjusted by the EEPROM editor.

Static Part

Parameters that are set in the production, and they are not the same for all MS's.

E.g.:

- RF calibration parms.
- Audio parms.

These parameters can be adjusted by the EEPROM editor.

Default Static Part

Parameters that are set in the production, and they are the same for all MS's.

These parameters will never be changed.

Architecture (Flash block allocation)

- **Static EE:** Uses two flash blocks. The flash block size can be 8KB, 32KB or 128KB depending on the physical flash architecture.
- **Dynamic EE:** Uses two flash blocks. The flash block size can be 8K, 2x8KB, 64KB, 128KB or 256KB depending on the physical flash architecture, and the selected mode.
- **Security area:** Uses one or two flash blocks. The flash block size can be 8K, 32KB or 128KB depending on the physical flash architecture.
- **Exception log:** Uses one flash blocks. The flash block size can be 8K, 32KB, 128KB or 256KB depending on the physical flash architecture.
- **Debug log:** 256KB consisting of one or more blocks depending on the physical flash architecture.

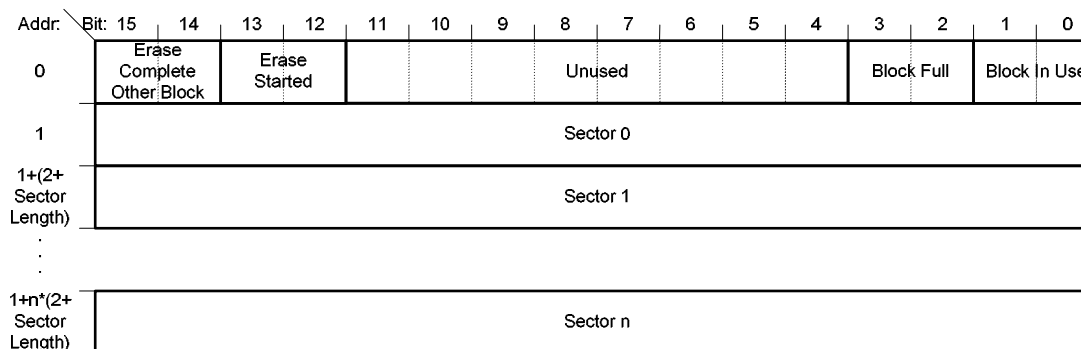
Architecture (dynamic part)

- This part of the EEPROM can be updated in normal mode of the MS.
- E.g. MMI parameters and parameters that can be modified by the user.
- The structure of the dynamic block is given from a common type definition containing all the parameters.
- The dynamic EEPROM is physically stored in a sectorized way. This is to avoid erasing and re-programming the entire block when a dynamic parameter is changed. When a parameter is changed only the sector(s) containing the parameter is re-programmed on the next unprogrammed location in the flash block.
- Because of the real-time issues the dynamic parameters are not stored instantly in flash, but temporarily in a RAM buffer. In the same way parameters are read from this RAM mirror. When the required CPU time is available the RAM mirror is updated in the flash.

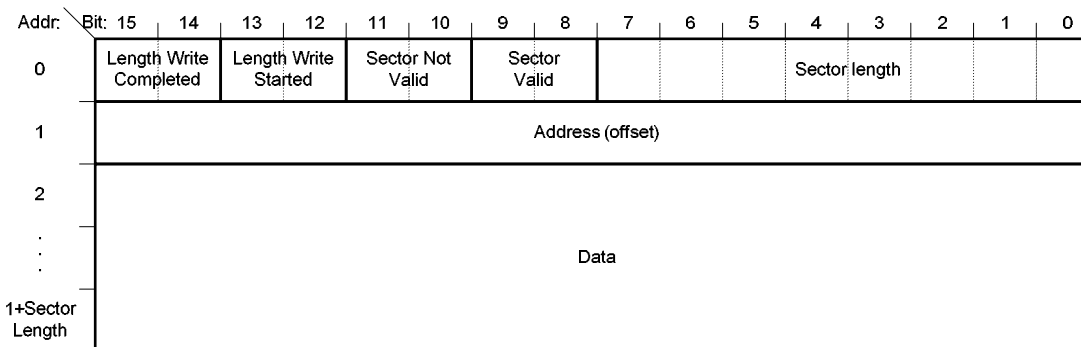
Architecture (dynamic part)

The dynamic EE consists of a factory block (containing default data for all parameters), and two swap blocks. The current parameter values are stored in one of the swap blocks.

Block Layout:



Sector Layout:



Block header control:

- In use bit
- Block full bit
- Erase started bit
- Erase completed other block

Sector header control:

- Sector valid bit
- Sector not valid bit
- Length write started
- Length write completed bit

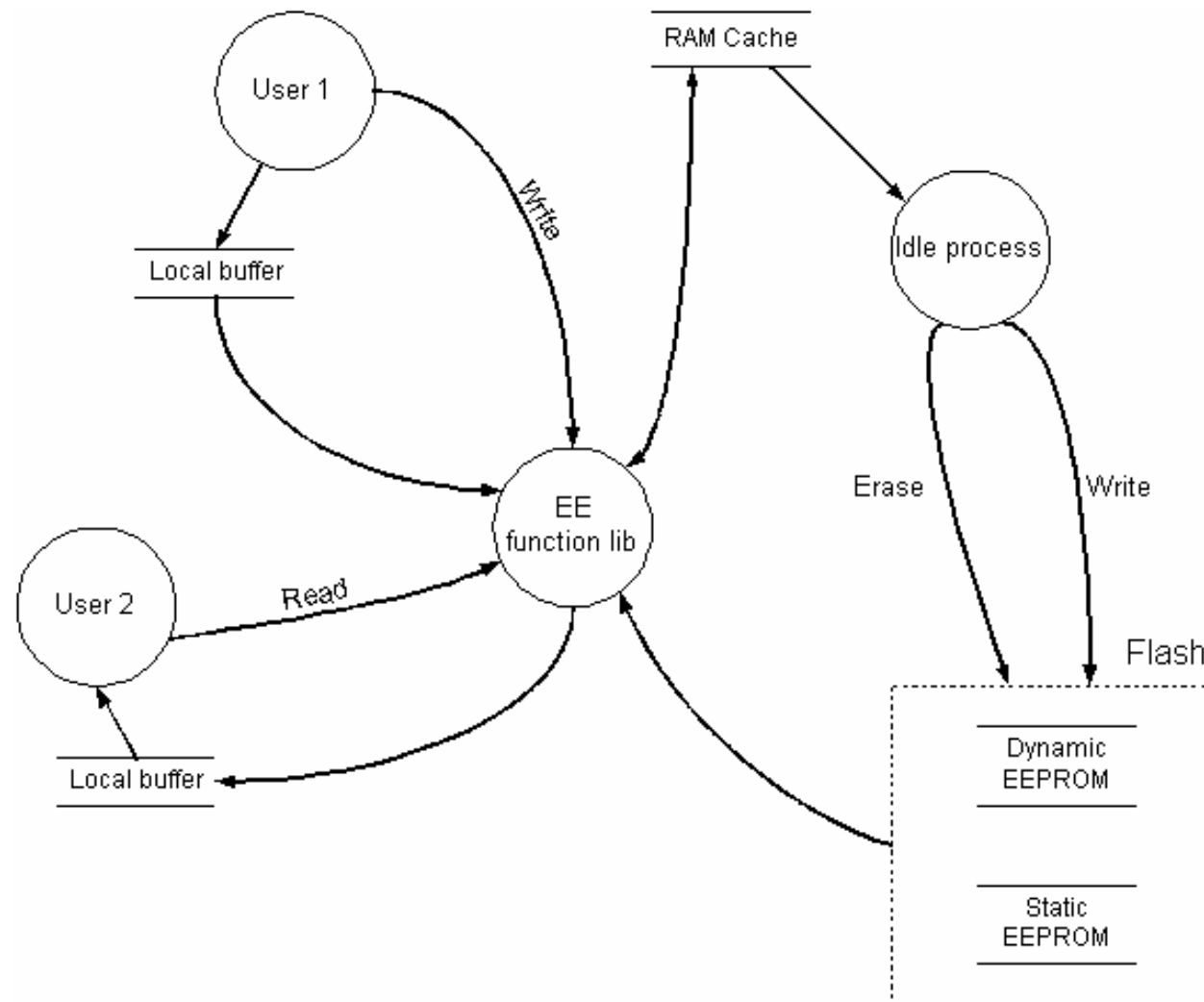
Sector update:

The modified sector is added on the next free flash location, and the old sector is marked invalid.

Block swap:

When the actual used swap block is full, all valid sectors are copied to the other swap block and the old swap block is erased.

Architecture (dynamic part)



Architecture (dynamic part – data structure)

- The dynamic structure is defined in eep.h/eep.c:
 - `const eep_dynamic_type EEP_dynamic`
- The filter structure for factory default function is defined in eep.h/eep.c:
 - `const eep_dynamic_type EEP_dynamic_default_filter`
- The segmented structure for the physical store is defined in eep.h/eep.c:
 - `const unsigned char factory[FACTORY_SIZE]`

Architecture (dynamic part – power loss recovery)

Consistency checks:

Initially (power up) a consistency check of the dynamic EEPROM is done to ensure that the EEPROM is intact and that no update operation were in progress when the MS was powered off (battery removed).

- **All sectors present:**

If no valid swap block containing all the sectors present in the factory block could be found, a swap block is initialized with the contents of the factory block.

- **Unfinished sector update:**

If the MS was turned off by accident in the middle of a sector update, the sector is detected and set to the invalid state.

- **Unfinished block erase:**

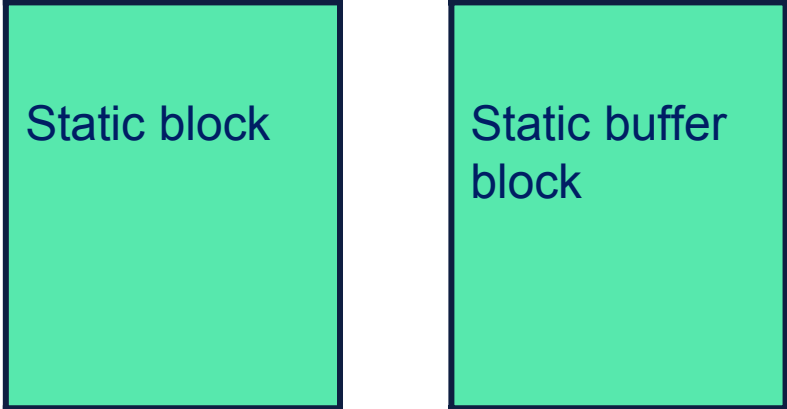
If the MS was turned off by accident in the middle of a block erase operation, the block may be corrupted and needs to be erased before it can be used as swap block.

- **Swap resume:**

If the MS was turned off by accident in the middle of a swap operation this will be detected and the swap operation will be completed.

Architecture (static part)

- The static EEPROM part contains parameters stored during production.
- These parameters have individual values from product to product. Could be RF calibration parameters, audio parameters and other things that needs calibration during production.
- The static parameters are stored in a linear way given from the type definition for the data structure.
- Two flash blocks are used. One for storing the static parameters and one for buffering during update.
- Checksum handling is used to ensure power loss proof updating.



Static block

Static buffer
block

The default static structure is defined in
eep.h/eep.c:

Const eep_static_type EEP_static

Architecture (default static part)

- Is implemented as a structure containing all the default static parameters.
 - A const variable is then initialized with all the values.
 - The default static block is therefore treated as program code and
 - downloaded together with the SW package.
-
- The default static structure is defined in eep.h/eep.c:
 - `const eep_default_type EEP_default`

API – Dynamic part

The interface to the dynamic part is a function interface running on the priority of the calling process. The function-interface:

```
void EE_initialize(void)
```

```
void EE_deactivate_driver(void)
```

```
boolean EE_read_dynamic(unsigned int src, unsigned char *dst, unsigned nof)
```

```
boolean EE_write_dynamic(unsigned char *src, unsigned int dst, unsigned int nof)
```

```
void EE_set_factory_default(void)
```

```
boolean EE_check_write(void)
```

```
boolean EE_check_swap(void)
```

```
boolean EE_check_erase(void)
```

API - Static part

The interface to the static part is a function interface running on the priority of the calling process. All updates are done instantly in the flash before returning to the calling process.

The function-interface:

```
boolean EE_read_static(unsigned int src, unsigned char *dst, unsigned nof)
```

```
boolean EE_update_static(unsigned char *src, unsigned int dst, unsigned int nof)
```

```
boolean EE_update_static_full(unsigned char *src, unsigned int dst, unsigned int nof,  
char cmd)
```

Update of the static part is done through the serial port via the AT-command interface.

Additional API's

In addition to the main API's, EE has some dedicated API's for reading/programming/erasing the following areas:

- Security data
- Exception logging
- Debug logging (for memory dumps for debug purposes)

Questions

?