Eeprom Emulation Driver

Subject: Interface specification

Revision: 1.0
Author:
Last Updated: 01/02/2005 09:55
Last Updated By:
Printout Date: 14/02/2006 02:58
File: C:\TEMP\NV\EE_drv_InterfaceSpecification.doc

History

Date	Author	Revision	Comments
2004-01-31	PS	1.0	EEPROM Specification rev. 0.8 split into module spec. and interface spec. Various updates.

Contents

1 Overview

This document describes the public interface functions provided by the Eeprom Emulation Driver (ee.c).

2 Interface specification

This module is used to store and retrieve data from the emulated eeprom in the flash.

2.1 Driver Start and Stop Interfaces

2.1.1 EE_initialize()

Prototype: void EE_initialize(void)
Input parameters: None
Output parameters: None
Description: After power up the EE_initialize() must be called before the RTOS is started. In this initialization there will be a check for if the last power off was a "structured power off" or there is a need to do some clean up work. It is checked if **Swap resume** is needed and as well the **Consistence check** is done.
Furthermore EE_initialize() handles to copy the functions that have to be located in external RAM from flash to RAM.
A flash type auto-detect between Intel and Fujitsu is done from here.

2.1.2 EE_deactivate_driver()

Prototype: void EE_deactivate_driver(void)
Input parameters: None.
Output parameters: None.
Description: Should be called before power off is activated. If the dynamic eeprom needs to be updated the update is done and it is checked if any SW exceptions has to be stored.

2.2 Dynamic Parameters Interfaces

2.2.1 EE_check_write_dynamic()

Prototype: boolean EE_check_write_dynamic(void)
Input parameters: None
Output parameters: TRUE if any programming has been done else FALSE.
Description: Write is done from IDLE process (lowest process level) by the use of function *EE_check_write_dynamic()*. In this function it is checked if write is needed i.e. If that is the case, write is started and the function will first return when there is no more words to be programmed. Interrupts is allowed in between word writes.

2.2.2 EE_write_dynamic()

Prototype: boolean EE_write_dynamic(unsigned char *src, unsigned int dst, unsigned int nof)
Input parameters: src: Absolute address of data to write
dst: Offset related to start address of Eeprom dynamic data RAM structure where to write data.
nof: Number of bytes to write
Output parameters: TRUE if all data were written correctly else FALSE
Description: **This function can be called from only one process level due it is not implemented reentrant**. The function copies data from caller RAM to RAM Eeprom mirror.

2.2.3 EE_write_dynamic_to_flash()

Prototype: boolean EE_write_dynamic_to_flash(void)
Input parameters: None
Output parameters: TRUE if any programming has been done else FALSE.
Description: This function copies data supplied by the caller, into the local RAM mirror of the *Dynamic Parameters*, and writes the RAM mirror to the flash.

2.2.4 EE_dynamic_updates_pending()

Prototype: boolean EE_dynamic_updates_pending(void)
Input parameters: None
Output parameters: TRUE if any pending updates.
Description: This function returns the status of whether the RAM mirror has been stored in flash.

2.2.5 EE_dynamic_updates_completed()

Prototype: boolean EE_dynamic_updates_completed(void)
Input parameters: None
Output parameters: TRUE if any pending updates.
Description: This function returns the status of whether the RAM mirror has been stored in flash since last time the function was called.

2.2.6 EE_check_swap_dynamic()

Prototype: boolean EE_check_swap_dynamic(void)
Input parameters: None
Output parameters: TRUE if swap has been done else FALSE.
Description: Swap is done from IDLE process (lowest process level) by the use of function *EE_check_swap_dynamic()*. In this function it is checked if swap is needed i.e. If that is the case, swap is started and the function will first return when the swap is completed. Interrupts is allowed in between word writes.

2.2.7 EE_read_dynamic()

Prototype: boolean EE_read_dynamic(unsigned int src, unsigned char *dst, unsigned int nof)
Input parameters:
- src: Offset related to start address of Eeprom dynamic data structure from where to read data.
- dst: Absolute address where to copy data read in Eeprom
- nof: Number of bytes to read

Output parameters: TRUE if all data were read correctly else FALSE
Description: This function can be called from more processes with different process levels priority. **Anyway it is not allowed to make read access from a process level with lower priority than the priority of the writing process.** The function copies data from Eeprom RAM mirror to RAM location given by input *dst*.

2.2.8 EE_set_factory_default()

Prototype: void EE_set_factory_default(void)
Input parameters: None.
Output parameters: None.
Description: This function defaults the dynamic parameters, but only the parameters that are allowed to be defaulted.

2.2.9 EE_set_factory_default_all()

Prototype: void EE_set_factory_default_all(void)
Input parameters: None.
Output parameters: None.
Description: This function defaults all the dynamic parameters.

2.2.10 EE_check_erase()

Prototype: boolean EE_check_erase(void)
Input parameters: None
Output parameters: TRUE if any erase has been done else FALSE.
Description: Erase is done from IDLE process (lowest process level) by the use of function *EE_check_erase()*. In this function it is checked if erase is needed i.e. the "*Erase Started*" bit is set in one of the two dynamic blocks. If that is the case, erase is started and the function will first return when erase is finished. Interrupt requests are monitored and allowed during erase.

2.3 Static Parameters Interfaces

2.3.1 EE_write_static()

Prototype: boolean EE_write_static(unsigned char *src, unsigned int dst, unsigned int nof)
Input parameters: src: Absolute address of data to write
dst: Offset related to start address of Eeprom static data structure where to write data.
nof: Number of bytes to write
Output parameters: TRUE if all data were written correctly else FALSE
Description: The function copies data from RAM to Eeprom (static block). There is no bookkeeping that guarantee the static page to be erased in the location area where to write i.e. the write function just writes data without any kind of check. The function is not implemented reentrant.

2.3.2 EE_read_static()

Prototype: boolean EE_read_static(unsigned int src, unsigned char *dst, unsigned int nof)
Input parameters: src: Offset related to start address of the Eeprom static data structure where to read data
dst: Absolute address where to copy data read in Eeprom
nof: Number of bytes to read
Output parameters: TRUE if all data were read correctly else FALSE
Description: The function copies data from Eeprom (static block) to RAM location given by input *dst*.

2.3.3 EE_erase_static()

Prototype: void EE_erase_static(void)
Input parameters: None
Output parameters: None
Description: Erase the static block within this function call on the priority of the calling process. The erase procedure is allowing interrupt to be handled.

2.3.4 EE_update_static()

Prototype: boolean EE_update_static(unsigned char *src, unsigned int dst, unsigned int nof)
Input parameters: src: Absolute address of data to write
dst: Offset related to start address of Eeprom static data structure where to write data.
nof: Number of bytes to write
Output parameters: TRUE if all data were written correctly else FALSE
Description: The function is erasing the static block, and copying data from RAM to Eeprom (static block) in the selected area (specified by the src and nof parameters). The previous contents of the static block is if necessary buffered in the “Static update buffer block” and rewritten (leaves unaffected parameters unchanged). The function is not implemented reentrant.

2.3.5 EE_update_static_full()

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

Input parameters: src: Absolute address of data to write
dst: Offset related to start address of Eeprom static data structure where to write data.
nof: Number of bytes to write
cmd: 'S' = Start update in buffer block
'U' = Update in buffer block
'E' = End update and copy contents to static block

Output parameters: TRUE if all data were written correctly else FALSE

Description: By this function the static block can be fully updated by several function-calls with only 1(2) block erases. The update is initiated by a function-call with cmd = 'S' and the buffer block is erased if necessary and the block is updated with the new data in the specified range. Any subsequent calls with cmd = 'U' will update the specified area of the buffer block. When called with cmd = 'E' the specified area is updated and the contents of the buffer block is copied to the static block.
Note: The entire block must be updated when using this function. Can be called from one process level only.

2.4 Exception Log Interfaces

2.4.1 EE_store_exception()

Prototype: void EE_store_exception(void *src)

Input parameters: src: Absolute address of data to write

Output parameters: None.

Description: Is writing an element of EE_exception_type specified by src in non-volatile memory. RAM buffered SW exceptions are also stored (if any). Should be called from the trap service routine.

2.4.2 EE_get_nof_stored_exceptions()

Prototype: unsigned int EE_get_nof_stored_exceptions(void)

Input parameters: None.

Output parameters: Returns the number of stored exceptions.

Description: Is returning the actual number of exceptions in the exception store.

2.4.3 EE_get_exception()

Prototype: void* EE_get_exception(unsigned int number)

Input parameters: number: The number in the exception store of the exception to read.

Output parameters: Returns a pointer to the exception in flash.

Description: Gets the exception from flash specified by “number” in the exception store.

2.4.4 EE_clr_exception_store()

Prototype: void EE_clr_exception(unsigned int number)
Input parameters: number: The number in the exception store of the exception to read.
Output parameters: none.
Description: This function erases all exceptions stored in the system.

2.4.5 EE_get_exception_block_start_address()

Prototype: void* EE_get_exception(unsigned int number)
Input parameters: number: The number in the exception store of the exception to read.
Output parameters: Returns a pointer to the exception in flash.
Description: This function returns a pointer to the exception store in NOR flash.

2.5 Security Interfaces

2.5.1 EE_write_imei_sp_lock()

Prototype: boolean EE_write_imei_sp_lock(unsigned char *src, unsigned int type);
Input parameters: src: Physical address of data to write.
Type: Determines whether IMEI or SP lock shall be written.
Output parameters: Returns a pointer to the exception in flash.
Description: TRUE if all data were written correctly else FALSE.

2.5.2 EE_write_sec_data()

Prototype: boolean EE_write_sec_data(unsigned int16 offset, unsigned char *src, unsigned int16 nof);
Input parameters: offset: Offset to write to.
src: Physical address of data to write.
nof: Number of byte to write.
Output parameters: TRUE: write success.
Description: This function writes the *Security Data* supplied by the caller into the security block.

2.5.3 EE_write_sec_data_full()

Prototype: boolean EE_write_sec_data_full(unsigned int16 offset, unsigned char *src, unsigned int16 nof, char cmd);
Input parameters: offset: Offset to write to.
src: Physical address of data to write.
nof: Number of byte to write.
cmd: Update type
Output parameters: TRUE: write success.
Description: By this function the sec block can be fully updated by several function-calls with only 1(2) block erases. The update is initiated by a function-call with cmd = 'S' and the buffer block is erased if necessary and the block is updated with the new data in the specified range. Any subsequent calls with cmd = 'U' will update the specified area of the buffer block. When called with cmd = 'E' the specified area is updated and the contents of the buffer block is copied to the sec block.
Note: The entire block must be updated when using this function.

Can be called from one process level only.

2.5.4 EE_buffer_sec_data()

Prototype: boolean EE_buffer_sec_data(unsigned int16 offset, unsigned char *src, unsigned int nof);
Input parameters: offset: Offset to write to.
src: Physical address of data to write.
nof: Number of bytes to write.
Output parameters: TRUE: write success.
Description: This function writes the *Security Data* supplied by the caller into the buffer block.

2.5.5 EE_update_sec_data()

Prototype: boolean EE_update_sec_data (EE_sec_elem_type *elem,
unsigned int16 nof)
Input parameters: elem: Element to update.
nof: Number of byte to update.
Output parameters: TRUE: write success.
Description: This function writes the *Security Data* supplied by the caller into the security block

2.5.6 EE_write_seccode()

Prototype: boolean EE_write_seccode(unsigned int16 offset, unsigned char *src,
unsigned int16 nof)
Input parameters: offset: Offset to write data at.
src: source of data to write.
nof: umber of bytes to write.
Output parameters: TRUE: write success.
Description: By this function the sec code block can be fully updated by
several function-calls with only 1 block erase.
The update is initiated by a function-call with cmd = 'S' and the
block is erased if necessary and the block is updated with
the new data in the specified range.
Any subsequent calls with cmd = 'U' will update the specified
area of the buffer block.
Note: The entire block must be updated when using this function.
Can be called from one process level only.

2.5.7 EE_erase_seccode()

Prototype: void EE_write_sec_data(void)
Input parameters: none
Output parameters: none
Description: This function erases the *Security Data.*

2.5.8 EE_write_uhsp_data()

Prototype: boolean EE_write_uhsp_data(unsigned char *src)
Input parameters: src: uhsp data to write.
Output parameters: TRUE: write succes
Description: Copy data from source to the uhsp block.

2.6 Test and Debug Interfaces

2.6.1 EE_write_debug_data()

Prototype: boolean EE_write_debug_data(unsigned char *src, unsigned int offset, unsigned nof);
Input parameters: src: Physical address of data to write.
offset: Address offset in block to write data.
nof: Number of bytes to write.
Output parameters: TRUE: Write success.
Description: Writes the data into the debug data block, used for development debugging.

2.6.2 EE_erase_debug_data()

Prototype: boolean EE_erase_debug_data(void)
Input parameters: none
Output parameters: TRUE: erase success.
Description: Erases the block used for debug data.

2.6.3 EE_ptest_generic_func()

Prototype: boolean EE_ptest_generic_func(atctst_ee_generic_func_req_type *func_rec_ptr)
Input parameters: func_rec_ptr: Pointer to opcode test request.
Output parameters: Returns number of return data in buffer
Description: Entry for the GMTS to execute on target tests.