

Droplet Platform API

Generated by Doxygen 1.8.8

Mon Oct 13 2014 13:27:45

Contents

1	Droplet Swarm Robotic Platform API	1
1.1	Introduction	1
1.2	Dependencies	1
1.3	Installation	1
1.3.1	Step 1: Obtaining Source Code	1
1.3.2	Step 2: Building DropletSimLibrary	1
1.4	How to contribute	1
1.4.1	Issue Tracker	1
1.4.2	Contacting us.	1
2	Deprecated List	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Data Structure Documentation	9
5.1	list_el Struct Reference	9
5.2	node Struct Reference	9
5.3	rnb_data Struct Reference	9
5.4	task Struct Reference	10
6	File Documentation	11
6.1	boot.h File Reference	11
6.1.1	Detailed Description	11
6.2	delay_x.h File Reference	11
6.2.1	Detailed Description	11
6.3	droplet_init.h File Reference	12
6.3.1	Detailed Description	13
6.4	ecc.h File Reference	13
6.4.1	Detailed Description	13

6.5	eeeprom_driver.h File Reference	14
6.5.1	Detailed Description	14
6.6	flash_api.h File Reference	14
6.6.1	Detailed Description	15
6.6.2	Macro Definition Documentation	16
6.6.2.1	NVM_EXEC	16
6.6.3	Function Documentation	16
6.6.3.1	FLASH_EraseApplicationSections	16
6.6.3.2	FLASH_EraseWriteApplicationPage	16
6.6.3.3	FLASH_FlushFlashPageBuffer	16
6.6.3.4	FLASH_LoadFlashPageBuffer	16
6.6.3.5	FLASH_WaitForNVM	17
6.7	ir_comm.h File Reference	17
6.7.1	Detailed Description	18
6.8	ir_sensor.h File Reference	19
6.8.1	Detailed Description	19
6.8.2	Function Documentation	19
6.8.2.1	check_collisions	19
6.9	sp_driver.h File Reference	20
6.9.1	Detailed Description	21
6.9.2	Function Documentation	22
6.9.2.1	SP_ApplicationCRC	22
6.9.2.2	SP_BootCRC	22
6.9.2.3	SP_EraseApplicationPage	22
6.9.2.4	SP_EraseApplicationSection	22
6.9.2.5	SP_EraseBootPage	22
6.9.2.6	SP_EraseFlashBuffer	23
6.9.2.7	SP_EraseUserSignatureRow	23
6.9.2.8	SP_EraseWriteApplicationPage	23
6.9.2.9	SP_EraseWriteBootPage	23
6.9.2.10	SP_LoadFlashPage	23
6.9.2.11	SP_LoadFlashWord	23
6.9.2.12	SP_LockSPM	24
6.9.2.13	SP_ReadByte	24
6.9.2.14	SP_ReadCalibrationByte	24
6.9.2.15	SP_ReadFlashPage	24
6.9.2.16	SP_ReadFuseByte	24
6.9.2.17	SP_ReadLockBits	25
6.9.2.18	SP_ReadUserSignatureByte	25
6.9.2.19	SP_ReadWord	25

6.9.2.20	SP_WaitForSPM	25
6.9.2.21	SP_WriteApplicationPage	25
6.9.2.22	SP_WriteBootPage	26
6.9.2.23	SP_WriteLockBits	26
6.9.2.24	SP_WriteUserSignatureRow	26

Chapter 1

Droplet Swarm Robotic Platform API

1.1 Introduction

TODO: Add Intro here...

1.2 Dependencies

- ATMEL Studio 6.2+

1.3 Installation

TODO: Add installation instructions here

1.3.1 Step 1: Obtaining Source Code

Source code for this project can be attained from the [cu-droplet](#) GitHub Repository. To download it you will need to have installed a [git client](#).

1.3.2 Step 2: Building DropletSimLibrary

TODO: Add build instructions here...

1.4 How to contribute

TODO: Add policies on contributing.

1.4.1 Issue Tracker

Current known issues with the Droplets project can be found at the [cu-droplet issues tracker on Google Code](#).

1.4.2 Contacting us.

email: john.klingner@colorado.edu

Chapter 2

Deprecated List

File [boot.h](#)

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

list_el	??
node	??
rnb_data	??
task	??

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

boot.h	Code for Droplet bootloader. No longer in use	??
delay_x.h	??
droplet_init.h	Droplet initialization routines and global macros are defin3d here	??
ecc.h	Code for Droplet IR communication message error detection and correction	??
eeprom_driver.h	??
flash_api.h	Code for reading and writing to flash memory directly	??
i2c.h	??
ir_comm.h	Droplet infrared communication subsystem functions are defined here	??
ir_sensor.h	??
main.h	??
motor.h	??
pc_com.h	??
power.h	??
random.h	??
range_algs.h	??
rgb_led.h	??
rgb_sensor.h	??
scheduler.h	??
serial_handler.h	??
sp_driver.h	XMEGA Self-programming driver header file	??

Chapter 5

Data Structure Documentation

5.1 `list_el` Struct Reference

Data Fields

- float **Rx**
- float **Ry**
- float **rijMag**
- uint8_t **e**
- uint8_t **s**
- struct [list_el](#) * **next**

The documentation for this struct was generated from the following file:

- `range_algs.h`

5.2 `node` Struct Reference

Data Fields

- char * **msg**
- uint32_t **arrival_time**
- uint8_t **arrival_dir**
- uint16_t **sender_ID**
- uint8_t **msg_length**
- struct [node](#) * **prev**

The documentation for this struct was generated from the following file:

- [ir_comm.h](#)

5.3 `rnb_data` Struct Reference

Data Fields

- float **range**
- float **bearing**

- float **heading**
- uint8_t(* **brightness_matrix_ptr**)[6]
- uint16_t **id_number**

The documentation for this struct was generated from the following file:

- range_algs.h

5.4 task Struct Reference

Data Fields

- uint32_t **scheduled_time**
- void(* **task_function**)(void *)
- void * **arg**
- struct [task](#) * **next**

The documentation for this struct was generated from the following file:

- scheduler.h

Chapter 6

File Documentation

6.1 boot.h File Reference

Code for Droplet bootloader. No longer in use.

6.1.1 Detailed Description

Code for Droplet bootloader. No longer in use.

Deprecated

6.2 delay_x.h File Reference

```
#include <inttypes.h>
```

Macros

- `#define busy_delay_ns(__ns) _delay_cycles((double)(F_CPU)*((double)__ns)/1.0e9 + 0.5)`
- `#define busy_delay_us(__us) _delay_cycles((double)(F_CPU)*((double)__us)/1.0e6 + 0.5)`
- `#define busy_delay_ms(__ms) _delay_cycles((double)(F_CPU)*((double)__ms)/1.0e3 + 0.5)`
- `#define busy_delay_s(__s) _delay_cycles((double)(F_CPU)*((double)__s)/1.0e0 + 0.5)`

6.2.1 Detailed Description

[delay_x.h](#)

Accurate delays ranging from a single CPU cycle up to more than 500 second (e.g. with 8MHz device):

The idea for the functions below was heavily inspired by the file `<avr/delay.h>` which is part of the excellent WinAVR distribution. Therefore, thanks to Marek Michalkiewicz and Joerg Wunsch.

The idea is to have the GCC preprocessor handle all calculations necessary for determining the exact implementation of a delay algorithm. The implementation itself is then inlined into the user code. In this way it is possible to always get the code size optimized delay implementation.

```
!!=====!!
!! Requires compile time constants for the delay      !!
!! Requires compiler optimization                     !!
!!=====!!
```

Copyright (c) 2005, Hans-Juergen Heinrichs All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the copyright holders nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.3 droplet_init.h File Reference

Droplet initialization routines and global macros are defined here.

```
#include <avr/io.h>
#include <util/crc16.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "scheduler.h"
#include "pc_com.h"
#include "rgb_led.h"
#include "rgb_sensor.h"
#include "power.h"
#include "random.h"
#include "ecc.h"
#include "ir_comm.h"
#include "ir_sensor.h"
#include "i2c.h"
#include "motor.h"
#include "range_algs.h"
#include "serial_handler.h"
```

Macros

- `#define DIR0 0x01`
- `#define DIR1 0x02`
- `#define DIR2 0x04`
- `#define DIR3 0x08`
- `#define DIR4 0x10`
- `#define DIR5 0x20`
- `#define ALL_DIRS 0x3F`

Functions

- uint16_t `get_droplet_id` ()
Returns this Droplet's unique 16-bit identifier.
- void `init_all_systems` ()
Initializes all the subsystems for this Droplet. This function MUST be called by the user before using any other functions in the API.
- void `droplet_reboot` ()
Resets the Droplet's program counter and clears all low-level system buffers.
- void `calculate_id_number` ()
- void `enable_interrupts` ()
- void `startup_light_sequence` ()

Variables

- uint16_t `droplet_ID`
- uint8_t `got_rnb_cmd_flag`

6.3.1 Detailed Description

Droplet initialization routines and global macros are defined here.

It is highly recommended to include ONLY this header file in any user level droplet rather than including header files for each of the subsystems independently.

6.4 ecc.h File Reference

Code for Droplet IR communication message error detection and correction.

```
#include <avr/io.h>
#include <stdint.h>
#include "ir_comm.h"
```

Functions

- uint16_t `manchester_encode` (uint8_t data)
- uint8_t `manchester_decode` (uint16_t cw)
- uint8_t `manchester_verify` (uint16_t cw)
- uint32_t `golay_encode` (uint16_t data)
- uint16_t `golay_decode_fast` (uint32_t cw)
- uint16_t `golay_decode` (uint32_t w, int8_t *errs)
- uint8_t `golay_find_errors` (uint32_t cw)

6.4.1 Detailed Description

Code for Droplet IR communication message error detection and correction.

6.5 eeprom_driver.h File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
```

Macros

- **#define EEPROM_PAGE_SIZE** E2PAGESIZE
- **#define EEPROM_read_byte(addr)** eeprom_read_byte((const uint8_t *)((uint16_t)(addr)))
- **#define EEPROM_write_byte(addr, value)** eeprom_write_byte((uint8_t *)((uint16_t)(addr)), (value))
- **#define EEPROM_read_block(addr, dest, len)** eeprom_read_block((dest), (void *)((uint16_t)(addr)), (len))
- **#define EEPROM_write_block(addr, src, len)** eeprom_write_block((src), (void *)((uint16_t)(addr)), (len))

Functions

- void **EEPROM_erase_all** (void)

6.5.1 Detailed Description

XMEGA EEPROM Driver

eeprom.h

Alex Forencich alex@alexforencich.com

Copyright (c) 2011 Alex Forencich

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.6 flash_api.h File Reference

Code for reading and writing to flash memory directly.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/atomic.h>
#include "sp_driver.h"
#include "eeprom_driver.h"
```

Macros

- #define **FLASH_PAGE_SIZE** 256
- #define **FLASH_FWORD_SIZE** 9
- #define **FRAZIONI_DI_PAGINA_FLASH** 4
- #define **MAX_PAGE_NUMBER** 256
- #define **MIN_PAGE_NUMBER** 0
- #define **IMX_MAX** 20
- #define **PAGE_TRUE** 1
- #define **PAGE_FALSE** 0
- #define **NVM_EXEC()**
Non-Volatile Memory Execute Command.
- #define **XB_SUCCESS** 0
- #define **XB_ERR_NO_API** 1
- #define **XB_ERR_NOT_FOUND** 2
- #define **XB_INVALID_ADDRESS** 3

Functions

- uint8_t **FLASH_ReadByte** (uint32_t)
- void **FLASH_FlushFlashPageBuffer** (void)
Flush temporary FLASH page buffer.
- void **FLASH_LoadFlashPageBuffer** (const uint8_t *)
Load entire page into temporary FLASH page buffer.
- void **FLASH_EraseApplicationSections** (void)
Erase entire application section.
- void **FLASH_EraseWriteApplicationPage** (uint16_t)
Erase and write page buffer to application or application table section at byte address.
- uint32_t **FLASH_ApplicationCRC** (void)
- uint32_t **FLASH_RangeCRC** (uint32_t, uint32_t)
- void **FLASH_WaitForNVM** (void)
Read a byte from flash.
- void **FLASH_ReadFlashPage** (uint8_t *, uint32_t)
- uint16_t **binary_search** (uint16_t, uint16_t)
- uint16_t **flash_compare** (uint32_t)
- uint16_t **midpoint** (uint16_t, uint16_t)
- uint8_t **write_user_signature_row** (uint8_t *data)
- uint8_t **read_user_signature_byte** (uint16_t index)
- void **load_flash_page** (const uint8_t *data)
- void **erase_write_application_page** (uint32_t address)
- void **erase_flash_buffer** ()
- void **read_flash_page** (const uint8_t *data, uint32_t address)
- void **erase_application_page** (uint32_t address)

6.6.1 Detailed Description

Code for reading and writing to flash memory directly.

This file largely derived from the [xboot](#) project, on GitHub.

6.6.2 Macro Definition Documentation

6.6.2.1 #define NVM_EXEC()

Value:

```
asm("push r30"      "\n\t" \
    "push r31"      "\n\t" \
    "push r16"      "\n\t" \
    "push r18"      "\n\t" \
    "ldi r30, 0xCB"  "\n\t" \
    "ldi r31, 0x01"  "\n\t" \
    "ldi r16, 0xD8"  "\n\t" \
    "ldi r18, 0x01"  "\n\t" \
    "out 0x34, r16"  "\n\t" \
    "st Z, r18"      "\n\t" \
    "pop r18"        "\n\t" \
    "pop r16"        "\n\t" \
    "pop r31"        "\n\t" \
    "pop r30"        "\n\t" \
    )
```

Non-Volatile Memory Execute Command.

This macro set the CCP register before setting the CMDEX bit in the NVM.CTRLA register.

Note

The CMDEX bit must be set within 4 clock cycles after setting the protection byte in the CCP register.

6.6.3 Function Documentation

6.6.3.1 void FLASH_EraseApplicationSections (void)

Erase entire application section.

This function erases the entire application and application table section

Note

If the lock bits is set to not allow spm in the application or application table section the erase is not done.

6.6.3.2 void FLASH_EraseWriteApplicationPage (uint16_t page_number)

Erase and write page buffer to application or application table section at byte address.

This function does a combined erase and write to a flash page in the application or application table section.

Parameters

<i>page_number</i>	Flash page number.
--------------------	--------------------

6.6.3.3 void FLASH_FlushFlashPageBuffer (void)

Flush temporary FLASH page buffer.

This function flushes the FLASH page buffers.

6.6.3.4 void FLASH_LoadFlashPageBuffer (const uint8_t * ram_buffer_ptr)

Load entire page into temporary FLASH page buffer.

This function loads an entire FLASH page from an SRAM buffer to the FLASH page buffers. Make sure that the buffer is flushed before starting to load bytes.

Note

Only the lower part of the address is used to address the buffer. Therefore, no address parameter is needed. In the end, the data is written to the FLASH page given by the address parameter to the FLASH write page operation.

Parameters

<i>values</i>	Pointer to SRAM buffer containing an entire page.
---------------	---

6.6.3.5 void FLASH_WaitForNVM (void)

Read a byte from flash.

This function reads one byte from the flash.

Note

Both IAR and GCC have functions to do this, but we include the functions for easier use.

Parameters

<i>address</i>	Address to the location of the byte to read.
----------------	--

Return values

<i>Byte</i>	read from flash.
-------------	------------------

Wait for any NVM access to finish, including FLASH.

This function is blocking and waits for any NVM access to finish, including FLASH. Use this function before any FLASH accesses, if you are not certain that any previous operations are finished yet, like an FLASH write.

6.7 ir_comm.h File Reference

Droplet infrared communication subsystem functions are defined here.

```
#include <avr/io.h>
#include <util/crc16.h>
#include <avr/interrupt.h>
#include "droplet_init.h"
#include "scheduler.h"
```

Data Structures

- struct [node](#)

Macros

- #define **IR_BUFFER_SIZE** 63
- #define **IR_UPKEEP_FREQUENCY** 20
- #define **IR_MSG_TIMEOUT** 10
- #define **IR_STATUS_BUSY_bm** 0x01
- #define **IR_STATUS_COMPLETE_bm** 0x02
- #define **IR_STATUS_ERROR_bm** 0x04
- #define **IR_STATUS_COMMAND_bm** 0x08

- `#define IR_STATUS_TARGETED_bm 0x10`
- `#define IR_STATUS_UNAVAILABLE_bm 0x03`
- `#define DATA_LEN_VAL_bm 0x7F`
- `#define DATA_LEN_CMD_bm 0x80`
- `#define HEADER_POS_SENDER_ID_LOW 0`
- `#define HEADER_POS_SENDER_ID_HIGH 1`
- `#define HEADER_POS_MSG_LENGTH 2`
- `#define HEADER_POS_CRC_LOW 3`
- `#define HEADER_POS_CRC_HIGH 4`
- `#define HEADER_POS_TARGET_ID_LOW 5`
- `#define HEADER_POS_TARGET_ID_HIGH 6`
- `#define HEADER_LEN 7`

Typedefs

- typedef volatile struct `node` `msg_node`

Functions

- void `ir_com_init` ()
- void `perform_ir_upkeep` ()
- void `clear_ir_buffer` (uint8_t dir)
- void `ir_targeted_cmd` (uint8_t dirs, char *data, uint16_t data_length, uint16_t target)
- void `ir_cmd` (uint8_t dirs, char *data, uint16_t data_length)
- void `ir_targeted_send` (uint8_t dirs, char *data, uint16_t data_length, uint16_t target)
- void `ir_send` (uint8_t dirs, char *data, uint8_t data_length)
- void `ir_receive` (uint8_t dir)
- void `ir_transmit` (uint8_t dir)
- void `ir_transmit_complete` (uint8_t dir)
- void `ir_reset_rx` (uint8_t dir)
- void `wait_for_ir` (uint8_t dirs)
- void `print_received_message` (void *dir_star)

Variables

- USART_t * `channel` []
- struct {
 - uint32_t `last_byte`
 - char `buf` [IR_BUFFER_SIZE]
 - uint16_t `data_crc`
 - uint16_t `sender_ID`
 - uint16_t `target_ID`
 - uint16_t `curr_pos`
 - uint8_t `data_length`
 - volatile uint8_t `status`
 } `ir_rtx` [6]
- volatile msg_node * `last_ir_msg`

6.7.1 Detailed Description

Droplet infrared communication subsystem functions are defined here.

6.8 ir_sensor.h File Reference

```
#include <avr/io.h>
#include "scheduler.h"
#include "delay_x.h"
#include "i2c.h"
```

Macros

- **#define IR_SENSOR_PORT** PORTB
- **#define IR_SENSOR_0_PIN_bm** PIN5_bm
- **#define IR_SENSOR_1_PIN_bm** PIN6_bm
- **#define IR_SENSOR_2_PIN_bm** PIN7_bm
- **#define IR_SENSOR_3_PIN_bm** PIN4_bm
- **#define IR_SENSOR_4_PIN_bm** PIN2_bm
- **#define IR_SENSOR_5_PIN_bm** PIN3_bm
- **#define ALL_IR_SENSOR_PINS_bm** (PIN2_bm | PIN3_bm | PIN4_bm | PIN5_bm | PIN6_bm | PIN7_bm)
- **#define ALL_EMITTERS_CARWAV_bm** (PIN0_bm | PIN1_bm | PIN4_bm | PIN5_bm | PIN7_bm | PIN6_bm)
- **#define MUX_IR_SENSOR_0** ADC_CH_MUXPOS_PIN5_gc
- **#define MUX_IR_SENSOR_1** ADC_CH_MUXPOS_PIN6_gc
- **#define MUX_IR_SENSOR_2** ADC_CH_MUXPOS_PIN7_gc
- **#define MUX_IR_SENSOR_3** ADC_CH_MUXPOS_PIN4_gc
- **#define MUX_IR_SENSOR_4** ADC_CH_MUXPOS_PIN2_gc
- **#define MUX_IR_SENSOR_5** ADC_CH_MUXPOS_PIN3_gc
- **#define MUX_SENSOR_CLR** 0b10000111

Functions

- `uint8_t check_collisions ()`
Can be used to check if object(s) are within 1cm of this Droplet.
- `void ir_sensor_init ()`
- `uint8_t get_ir_sensor (uint8_t sensor_num)`
- `int8_t find_median (int8_t *meas)`
- `int8_t ir_bounce_meas (uint8_t dir)`
- `void ir_sensor_enable ()`
- `void ir_sensor_disable ()`

6.8.1 Detailed Description

Low level sensing functions using IR channels. Note that there is no IR communication code in this file.

6.8.2 Function Documentation

6.8.2.1 `uint8_t check_collisions ()`

Can be used to check if object(s) are within 1cm of this Droplet.

Returns

A bit-mask with 1 set in the directions where objects are detected within 1 cm. Direction macros are defined in [droplet_init.h](#).

6.9 sp_driver.h File Reference

XMEGA Self-programming driver header file.

```
#include <avr/io.h>
```

Functions

- uint8_t [SP_ReadByte](#) (uint32_t address)
Read a byte from flash.
- uint16_t [SP_ReadWord](#) (uint32_t address)
Read a word from flash.
- uint8_t [SP_ReadCalibrationByte](#) (uint8_t index)
Read calibration byte at given index.
- uint8_t [SP_ReadFuseByte](#) (uint8_t index)
Read fuse byte from given index.
- void [SP_WriteLockBits](#) (uint8_t data)
Write lock bits.
- uint8_t [SP_ReadLockBits](#) (void)
Read lock bits.
- uint8_t [SP_ReadUserSignatureByte](#) (uint16_t index)
Read user signature at given index.
- void [SP_EraseUserSignatureRow](#) (void)
Erase user signature row.
- void [SP_WriteUserSignatureRow](#) (void)
Write user signature row.
- void [SP_EraseApplicationSection](#) (void)
Erase entire application section.
- void [SP_EraseApplicationPage](#) (uint32_t address)
Erase page at byte address in application or application table section.
- void [SP_EraseWriteApplicationPage](#) (uint32_t address)
Erase and write page buffer to application or application table section at byte address.
- void [SP_WriteApplicationPage](#) (uint32_t address)
Write page buffer to application or application table section at byte address.
- void [SP_LoadFlashWord](#) (uint16_t address, uint16_t data)
Load one word into Flash page buffer.
- void [SP_LoadFlashPage](#) (const uint8_t *data)
Load entire page from SRAM buffer into Flash page buffer.
- void [SP_ReadFlashPage](#) (const uint8_t *data, uint32_t address)
Read entire Flash page into SRAM buffer.
- void [SP_EraseFlashBuffer](#) (void)
Flush Flash page buffer.
- void [SP_EraseBootPage](#) (uint32_t address)
Erase page at byte address in boot section.
- void [SP_EraseWriteBootPage](#) (uint32_t address)
Erase and write page buffer to boot section at byte address.
- void [SP_WriteBootPage](#) (uint32_t address)
Write page buffer to boot section at byte address.
- uint32_t [SP_ApplicationCRC](#) (void)
Generate CRC from application section.

- uint32_t [SP_BootCRC](#) (void)
Generate CRC from boot section.
- void [SP_LockSPM](#) (void)
Lock SPM instruction.
- void [SP_WaitForSPM](#) (void)
Wait for SPM to finish.

6.9.1 Detailed Description

XMEGA Self-programming driver header file.

This file contains the function prototypes for the XMEGA Self-programming driver. If any SPM instructions are used, the linker file must define a segment named BOOT which must be located in the device boot section.

None of these functions clean up the NVM Command Register after use. It is therefore important to write NVMCMD_NO_OPERATION (0x00) to this register when you are finished using any of the functions in this driver.

For all functions, it is important that no interrupt handlers perform any NVM operations. The user must implement a scheme for mutually exclusive access to the NVM. However, the 4-cycle timeout will work fine, since writing to the Configuration Change Protection register (CCP) automatically disables interrupts for 4 instruction cycles.

Application note:

AVR1316: XMEGA Self-programming

Documentation

For comprehensive code documentation, supported compilers, compiler settings and supported devices see [readme.html](#)

Author

Atmel Corporation: <http://www.atmel.com>
Support email: avr@atmel.com

Revision

1691

Date

2008-07-29 13:25:40 +0200 (ti, 29 jul 2008)

Copyright (c) 2008, Atmel Corporation All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of ATMEL may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.9.2 Function Documentation

6.9.2.1 uint32_t SP_ApplicationCRC (void)

Generate CRC from application section.

Return values

24-bit	CRC value
--------	-----------

6.9.2.2 uint32_t SP_BootCRC (void)

Generate CRC from boot section.

Return values

24-bit	CRC value
--------	-----------

6.9.2.3 void SP_EraseApplicationPage (uint32_t address)

Erase page at byte address in application or application table section.

This function erase one page given by the byte address.

Parameters

address	Byte address for flash page.
---------	------------------------------

6.9.2.4 void SP_EraseApplicationSection (void)

Erase entire application section.

This function erases the entire application and application table section

Note

If the lock bits is set to not allow spm in the application or application table section the erase is not done.

6.9.2.5 void SP_EraseBootPage (uint32_t address)

Erase page at byte address in boot section.

This function erase one page given by the byte address.

Parameters

<i>address</i>	Byte address for flash page.
----------------	------------------------------

6.9.2.6 void SP_EraseFlashBuffer (void)

Flush Flash page buffer.

This function flush the Flash page buffer.

6.9.2.7 void SP_EraseUserSignatureRow (void)

Erase user signature row.

This function erase the entire user signature row.

6.9.2.8 void SP_EraseWriteApplicationPage (uint32_t *address*)

Erase and write page buffer to application or application table section at byte address.

This function does a combined erase and write to a flash page in the application or application table section.

Parameters

<i>address</i>	Byte address for flash page.
----------------	------------------------------

6.9.2.9 void SP_EraseWriteBootPage (uint32_t *address*)

Erase and write page buffer to boot section at byte address.

This function does a combined erase and write to a flash page in the boot section.

Parameters

<i>address</i>	Byte address for flash page.
----------------	------------------------------

6.9.2.10 void SP_LoadFlashPage (const uint8_t * *data*)

Load entire page from SRAM buffer into Flash page buffer.

This function load an entire page from SRAM.

Parameters

<i>data</i>	Pointer to the data to put in buffer.
-------------	---------------------------------------

Note

The `__near` keyword limits the pointer to two bytes which means that only data up to 64K (internal SRAM) can be used.

6.9.2.11 void SP_LoadFlashWord (uint16_t *address*, uint16_t *data*)

Load one word into Flash page buffer.

This function Loads one word into the Flash page buffer.

Parameters

<i>address</i>	Position in inside the flash page buffer.
<i>data</i>	Value to be put into the buffer.

6.9.2.12 void SP_LockSPM (void)

Lock SPM instruction.

This function locks the SPM instruction, and will disable the use of SPM until the next reset occurs.

6.9.2.13 uint8_t SP_ReadByte (uint32_t address)

Read a byte from flash.

This function reads one byte from the flash.

Note

Both IAR and GCC have functions to do this, but we include the functions for easier use.

Parameters

<i>address</i>	Address to the location of the byte to read.
----------------	--

Return values

<i>Byte</i>	read from flash.
-------------	------------------

6.9.2.14 uint8_t SP_ReadCalibrationByte (uint8_t index)

Read calibration byte at given index.

This function reads one calibration byte from the Calibration signature row.

Parameters

<i>index</i>	Index of the byte in the calibration signature row.
--------------	---

Return values

<i>Calibration</i>	byte
--------------------	------

6.9.2.15 void SP_ReadFlashPage (const uint8_t * data, uint32_t address)

Read entire Flash page into SRAM buffer.

This function reads an entire flash page and puts it to SRAM.

Parameters

<i>data</i>	Pointer to where to store the data.
<i>address</i>	Address to page to read from flash.

6.9.2.16 uint8_t SP_ReadFuseByte (uint8_t index)

Read fuse byte from given index.

This function reads the fuse byte at the given index.

Parameters

<i>index</i>	Index of the fuse byte.
--------------	-------------------------

Return values

<i>Fuse</i>	byte
-------------	------

6.9.2.17 uint8_t SP_ReadLockBits (void)

Read lock bits.

This function reads the lock bits.

Return values

<i>Lock</i>	bits
-------------	------

6.9.2.18 uint8_t SP_ReadUserSignatureByte (uint16_t index)

Read user signature at given index.

This function reads one byte from the user signature row.

Parameters

<i>index</i>	Index of the byte in the user signature row.
--------------	--

Return values

<i>User</i>	signature byte
-------------	----------------

6.9.2.19 uint16_t SP_ReadWord (uint32_t address)

Read a word from flash.

This function reads one word from the flash.

Note

Both IAR and GCC have functions to do this automatically, but we include the functions for easier use.

Parameters

<i>address</i>	Address to the location of the word to read.
----------------	--

Return values

<i>word</i>	read from flash.
-------------	------------------

6.9.2.20 void SP_WaitForSPM (void)

Wait for SPM to finish.

This routine waits for the SPM to finish and clears the command register.

6.9.2.21 void SP_WriteApplicationPage (uint32_t address)

Write page buffer to application or application table section at byte address.

This function writes the Flash page buffer to a page in the application or application table section given by the byte address.

Note

The page that is written to must be erased before it is written to.

Parameters

<i>address</i>	Byte address for flash page.
----------------	------------------------------

6.9.2.22 void SP_WriteBootPage (uint32_t *address*)

Write page buffer to boot section at byte address.

This function writes the Flash page buffer to a page in the boot section given by the byte address.

Note

The page that is written to must be erased before it is written to.

Parameters

<i>address</i>	Byte address for flash page.
----------------	------------------------------

6.9.2.23 void SP_WriteLockBits (uint8_t *data*)

Write lock bits.

This function changes the lock bits.

Note

It is only possible to change the lock bits to a higher security level.

Parameters

<i>data</i>	The new value of the lock bits.
-------------	---------------------------------

6.9.2.24 void SP_WriteUserSignatureRow (void)

Write user signature row.

This function write the flash buffer in the user signature row.