SHA204 Library Examples for AVR 8-Bit Target

0.1.0

Generated by Doxygen 1.8.4

Mon Jun 24 2013 16:47:10

Contents

Chapter 1

Building The Projects

1.1 Work Space and Project Structure

```
The source files for the ATECC108 library are contained in a single folder "src".
```

1.1.1 Hardware Independent Modules

ecc108_example_main.c

ecc108_examples.c

ecc108_examples.h

ecc108_examples.c

ecc108_helper.c

ecc108_helper.h

ecc108_comm_marshaling.c

ecc108_comm_marshaling.h

ecc108_comm.c

ecc108_comm.h

ecc108_i2c.c

ecc108_swi.c

ecc108_lib_return_codes.h

ecc108_config.h

ecc108_physical.h

timer_utilities.c

timer_utilities.h

1.1.2 Hardware Dependent Modules

Hardware dependent modules are provided that support 8-bit AVR micro-controllers. If you are not using an AVR CPU, either implement the functions listed in ecc108_physical.h or choose the appropriate module for the physical implementation of the communication with the device from one of the communication related modules:

Since SWI support comes in two flavors, UART and GPIO, a common header file is provided, swi_phys.h.

- bitbang_phys.c: Physical implementation as single wire interface (SWI) using GPIO.
- uart_phys.c: Physical implementation as single wire interface (SWI) using a UART (includes avr_compatible.h).
- i2c_phys.c: Physical implementation as two wire interface (I2C).

1.1.3 Projects

A solution file (.sln) is supplied for the Atmel Studio 6.1 IDE that contains three projects (.cproj). This solution file and folders (src, output, etc.) are located in the ecc108_90usb1287 folder. Choose the project that fits the communication interface you like to use.

If you don't use Atmel Studio you can easily create a project under the IDE you are using. You need the following modules and compilation switch depending on the interface and its implementation, SWI using UART, SWI using GPIO, or I2C.

- **SWI Using UART**

  ecc108_example_main.c

  ecc108_examples.c

  ecc108_examples.h

  ecc108_examples.c

  ecc108_helper.c

  ecc108_helper.h

  ecc108_comm_marshaling.c

  ecc108_comm_marshaling.h

  ecc108_comm.c

  ecc108_comm.h

  ecc108_swi.c

  ecc108_lib_return_codes.h

  ecc108_config.h

  ecc108_physical.h

  swi_phys.h

  avr_compatible.h

  uart_phys.c

  timer_utilities.c

  timer_utilities.h

  Compilation switches: ECC108_SWI, ECC108_SWI_UART, F_CPU=[your CPU clock in Hz]

- **SWI Using GPIO**

  ecc108_example_main.c

  ecc108_examples.c

  ecc108_examples.h

  ecc108_examples.c

ecc108_helper.c

ecc108_helper.h

ecc108_comm_marshaling.c

ecc108_comm_marshaling.h

ecc108_comm.c

ecc108_comm.h

ecc108_swi.c

ecc108_lib_return_codes.h

ecc108_config.h

ecc108_physical.h

timer_utilities.c

timer_utilities.h

swi_phys.h

bitbang_phys.c

Compilation switches: ECC108_SWI, ECC108_SWI_BITBANG, F_CPU=[your CPU clock in Hz]

In bitbang_config.h, you can change the GPIO pin definition for SDA.

- **I^2 C**

  ecc108_example_main.c

  ecc108_examples.c

  ecc108_examples.h

  ecc108_examples.c

  ecc108_helper.c

  ecc108_helper.h

  ecc108_comm_marshaling.c

  ecc108_comm_marshaling.h

  ecc108_comm.c

  ecc108_comm.h

  ecc108_i2c.c

  ecc108_lib_return_codes.h

  ecc108_config.h

  ecc108_physical.h

  i2c_phys.c

  timer_utilities.c

  timer_utilities.h

  Compilation switches: ECC108_I2C, F_CPU=[your CPU clock in Hz]

Follow the few steps listed below to build an ATECC108 project.

- Supply communication interface independent modules by adding ecc108_example_main.c, ecc108_examples.∗, ecc108_helper.∗, and ecc108_comm∗ to the project. Be aware that all hardware independent modules include ecc108_lib_return_codes.h and ecc108_physical.h

- Supply communication interface hardware independent modules. For SWI add ecc108_swi.∗, for I2C add ecc108-_i2c.∗. You might have to also modify ecc108_i2c.c, especially for 32-bit CPUs, since their I2C peripherals implement such functionality in hardware. For instance, they might not support the generation of individual Start and Stop conditions.
- Supply communication interface hardware dependent modules. If you do not use an AVR CPU, you have to implement the functions in these modules. For SWI using UART add uart_phys.c, for SWI using GPIO add bitbang_phys.c, and for I2C add i2c_phys.∗. Be aware that uart_phys.c includes avr_compatible.h. Also, both SWI modules include swi_phys.h.
- Supply a timer utility module. You can either use the provided timer_utilities.∗ files or supply your own. The AT-ECC108 library uses two delay functions, delay_ms(uint8_t) and delay_10us(uint8_t). The delay_ms function is used to determine command response timeouts. The delay_10us function is used to create a wake-up pulse and wake-up delay. The timer functions do not use hardware timers but loop counters. The supplied module is tuned for an AT90USB1287 CPU running at 16 MHz, but you can easily tune it for other micro-controllers as long as one loop iteration (decrement, compare, and jump) does not take longer than 10 μs.

1.2 Tools

1.2.1 Integrated Development Environment

Atmel Studio 6.1.2562

AVR Toolchain 8 Bit, Version: 3.4.2.939 - GCC 4.7.2

`http://www.atmel.com/Microsite/atmel_studio6/default.aspx`

1.3 Doxygen Generated Documentation

Most comments outside functions (functions, type and macro definitions, groups, etc.) follow a syntax that the Doxygen document generator for source code can parse (www.doxygen.org).

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 Module 01: Command Marshaling

A function is provided for every ATECC108 command in the final release. These functions check the parameters, assemble a command packet, send it, receive its response, and return the status of the operation and the response.

Functions

- uint8_t ecc108m_check_mac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t key_id, uint8_t *client-_challenge, uint8_t *client_response, uint8_t *other_data)

  This function sends a CheckMAC command to the device.
- uint8_t ecc108m_derive_key (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t random, uint8_t target_key, uint8_t *mac)

  This function sends a DeriveKey command to the device.
- uint8_t ecc108m_info (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t gpio_state)

  This function sends an Info command to the device.
- uint8_t ecc108m_gen_dig (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint8_t key_id, uint8_t *other_data)

  This function sends a GenDig command to the device.
- uint8_t ecc108m_hmac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint16_t key_id)

  This function sends an HMAC command to the device.
- uint8_t ecc108m_lock (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t summary)

  This function sends a Lock command to the device.
- uint8_t ecc108m_mac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint16_t key_id, uint8_t *challenge)

  This function sends a MAC command to the device.
- uint8_t ecc108m_nonce (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t *numin)

  This function sends a Nonce command to the device.
- uint8_t ecc108m_pause (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t selector)

  This function sends a Pause command to the device.
- uint8_t ecc108m_random (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode)

  This function sends a Random command to the device.
- uint8_t ecc108m_read (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t address)

  This function sends a Read command to the device.
- uint8_t ecc108m_update_extra (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t new_value)

  This function sends an UpdateExtra command to the device.

- uint8_t ecc108m_write (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t zone, uint16_t address, uint8_t ∗value, uint8-_t ∗mac)

    This function sends a Write command to the device.
- uint8_t ecc108m_execute (uint8_t op_code, uint8_t param1, uint16_t param2, uint8_t datalen1, uint8_t ∗data1, uint8_t datalen2, uint8_t ∗data2, uint8_t datalen3, uint8_t ∗data3, uint8_t tx_size, uint8_t ∗tx_buffer, uint8_t rx_-size, uint8_t ∗rx_buffer)

    This function creates a command packet, sends it, and receives its response.

Codes for ATECC108 Commands

- #define ECC108_CHECKMAC ((uint8_t) 0x28)

    CheckMac command op-code.
- #define ECC108_DERIVE_KEY ((uint8_t) 0x1C)

    DeriveKey command op-code.
- #define ECC108_INFO ((uint8_t) 0x30)

    DevRev command op-code.
- #define ECC108_GENDIG ((uint8_t) 0x15)

    GenDig command op-code.
- #define ECC108_HMAC ((uint8_t) 0x11)

    HMAC command op-code.
- #define ECC108_LOCK ((uint8_t) 0x17)

    Lock command op-code.
- #define ECC108_MAC ((uint8_t) 0x08)

    MAC command op-code.
- #define ECC108_NONCE ((uint8_t) 0x16)

    Nonce command op-code.
- #define ECC108_PAUSE ((uint8_t) 0x01)

    Pause command op-code.
- #define ECC108_RANDOM ((uint8_t) 0x1B)

    Random command op-code.
- #define ECC108_READ ((uint8_t) 0x02)

    Read command op-code.
- #define ECC108_UPDATE_EXTRA ((uint8_t) 0x20)

    UpdateExtra command op-code.
- #define ECC108_WRITE ((uint8_t) 0x12)

    Write command op-code.

Definitions of Data and Packet Sizes

- #define ECC108_RSP_SIZE_VAL ((uint8_t) 7)

    size of response packet containing four bytes of data
- #define ECC108_KEY_SIZE (32)

    size of key

Definitions for Command Parameter Ranges

- #define ECC108_KEY_ID_MAX ((uint8_t) 15)

  maximum value for key id
- #define ECC108_OTP_BLOCK_MAX ((uint8_t) 1)

  maximum value for OTP block

Definitions for Indexes Common to All Commands

- #define ECC108_COUNT_IDX (0)

  command packet index for count
- #define ECC108_OPCODE_IDX (1)

  command packet index for op-code
- #define ECC108_PARAM1_IDX (2)

  command packet index for first parameter
- #define ECC108_PARAM2_IDX (3)

  command packet index for second parameter
- #define ECC108_DATA_IDX (5)

  command packet index for second parameter

Definitions for Zone and Address Parameters

- #define ECC108_ZONE_CONFIG ((uint8_t) 0x00)

  Configuration zone.
- #define ECC108_ZONE_OTP ((uint8_t) 0x01)

  OTP (One Time Programming) zone.
- #define ECC108_ZONE_DATA ((uint8_t) 0x02)

  Data zone.
- #define ECC108_ZONE_MASK ((uint8_t) 0x03)

  Zone mask.
- #define ECC108_ZONE_COUNT_FLAG ((uint8_t) 0x80)

  Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.
- #define ECC108_ZONE_ACCESS_4 ((uint8_t) 4)

  Read or write 4 bytes.
- #define ECC108_ZONE_ACCESS_32 ((uint8_t) 32)

  Read or write 32 bytes.
- #define ECC108_ADDRESS_MASK_CONFIG (0x001F)

  Address bits 5 to 7 are 0 for Configuration zone.
- #define ECC108_ADDRESS_MASK_OTP (0x000F)

  Address bits 4 to 7 are 0 for OTP zone.
- #define ECC108_ADDRESS_MASK (0x007F)

  Address bit 7 to 15 are always 0.

Definitions for the CheckMac Command

- #define CHECKMAC_MODE_IDX ECC108_PARAM1_IDX

  CheckMAC command index for mode.
- #define CHECKMAC_KEYID_IDX ECC108_PARAM2_IDX

  CheckMAC command index for key identifier.
- #define CHECKMAC_CLIENT_CHALLENGE_IDX ECC108_DATA_IDX

  CheckMAC command index for client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_IDX (37)

  CheckMAC command index for client response.
- #define CHECKMAC_DATA_IDX (69)

  CheckMAC command index for other data.
- #define CHECKMAC_COUNT (84)

  CheckMAC command packet size.
- #define CHECKMAC_MODE_CHALLENGE ((uint8_t) 0x00)

  CheckMAC mode 0: first SHA block from key id.
- #define CHECKMAC_MODE_BLOCK2_TEMPKEY ((uint8_t) 0x01)

  CheckMAC mode bit 0: second SHA block from TempKey.
- #define CHECKMAC_MODE_BLOCK1_TEMPKEY ((uint8_t) 0x02)

  CheckMAC mode bit 1: first SHA block from TempKey.
- #define CHECKMAC_MODE_SOURCE_FLAG_MATCH ((uint8_t) 0x04)

  CheckMAC mode bit 2: match TempKey.SourceFlag.
- #define CHECKMAC_MODE_INCLUDE_OTP_64 ((uint8_t) 0x20)

  CheckMAC mode bit 5: include first 64 OTP bits.
- #define CHECKMAC_MODE_MASK ((uint8_t) 0x27)

  CheckMAC mode bits 3, 4, 6, and 7 are 0.
- #define CHECKMAC_CLIENT_CHALLENGE_SIZE (32)

  CheckMAC size of client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_SIZE (32)

  CheckMAC size of client response.
- #define CHECKMAC_OTHER_DATA_SIZE (13)

  CheckMAC size of "other data".
- #define CHECKMAC_CLIENT_COMMAND_SIZE (4)

  CheckMAC size of client command header size inside "other data".

Definitions for the DeriveKey Command

- #define DERIVE_KEY_RANDOM_IDX ECC108_PARAM1_IDX

  DeriveKey command index for random bit.
- #define DERIVE_KEY_TARGETKEY_IDX ECC108_PARAM2_IDX

  DeriveKey command index for target slot.
- #define DERIVE_KEY_MAC_IDX ECC108_DATA_IDX

  DeriveKey command index for optional MAC.
- #define DERIVE_KEY_COUNT_SMALL ECC108_CMD_SIZE_MIN

  DeriveKey command packet size without MAC.
- #define DERIVE_KEY_COUNT_LARGE (39)

  DeriveKey command packet size with MAC.

- #define DERIVE_KEY_RANDOM_FLAG ((uint8_t) 4)

  DeriveKey 1. parameter; has to match TempKey.SourceFlag.
- #define DERIVE_KEY_MAC_SIZE (32)

  DeriveKey MAC size.

Definitions for the GenDig Command

- #define GENDIG_ZONE_IDX ECC108_PARAM1_IDX

  GenDig command index for zone.
- #define GENDIG_KEYID_IDX ECC108_PARAM2_IDX

  GenDig command index for key id.
- #define GENDIG_DATA_IDX ECC108_DATA_IDX

  GenDig command index for optional data.
- #define GENDIG_COUNT ECC108_CMD_SIZE_MIN

  GenDig command packet size without "other data".
- #define GENDIG_COUNT_DATA (11)

  GenDig command packet size with "other data".
- #define GENDIG_OTHER_DATA_SIZE (4)

  GenDig size of "other data".
- #define GENDIG_ZONE_CONFIG ((uint8_t) 0)

  GenDig zone id config.
- #define GENDIG_ZONE_OTP ((uint8_t) 1)

  GenDig zone id OTP.
- #define GENDIG_ZONE_DATA ((uint8_t) 2)

  GenDig zone id data.

Definitions for the HMAC Command

- #define HMAC_MODE_IDX ECC108_PARAM1_IDX

  HMAC command index for mode.
- #define HMAC_KEYID_IDX ECC108_PARAM2_IDX

  HMAC command index for key id.
- #define HMAC_COUNT ECC108_CMD_SIZE_MIN

  HMAC command packet size.
- #define HMAC_MODE_MASK ((uint8_t) 0x74)

  HMAC mode bits 0, 1, 3, and 7 are 0.

Definitions for the Info Command

- #define INFO_PARAM1_IDX ECC108_PARAM1_IDX

  Info command index for 1. parameter.
- #define INFO_PARAM2_IDX ECC108_PARAM2_IDX

  Info command index for 2. parameter.
- #define INFO_COUNT ECC108_CMD_SIZE_MIN

  Info command packet size.
- #define INFO_MODE_REVISION ((uint8_t) 0x00)

Info mode Revision.
- #define INFO_MODE_KEY_VALID ((uint8_t) 0x01)

  Info mode KeyValid.
- #define INFO_MODE_STATE ((uint8_t) 0x02)

  Info mode State.
- #define INFO_MODE_GPIO ((uint8_t) 0x03)

  Info mode GPIO.
- #define INFO_MODE_MAX ((uint8_t) 0x03)

  Info mode maximum value.
- #define INFO_NO_STATE ((uint8_t) 0x00)

  Info mode is not the state mode.
- #define INFO_OUTPUT_STATE_MASK ((uint8_t) 0x01)

  Info output state mask.
- #define INFO_DRIVER_STATE_MASK ((uint8_t) 0x02)

  Info driver state mask.
- #define INFO_PARAM2_MAX ((uint8_t) 0x03)

  Info param2 (state) maximum value.

Definitions for the Lock Command

- #define LOCK_ZONE_IDX ECC108_PARAM1_IDX

  Lock command index for zone.
- #define LOCK_SUMMARY_IDX ECC108_PARAM2_IDX

  Lock command index for summary.
- #define LOCK_COUNT ECC108_CMD_SIZE_MIN

  Lock command packet size.
- #define LOCK_ZONE_NO_CONFIG ((uint8_t) 0x01)

  Lock zone is OTP or Data.
- #define LOCK_ZONE_NO_CRC ((uint8_t) 0x80)

  Lock command: Ignore summary.
- #define LOCK_ZONE_MASK (0x81)

  Lock parameter 1 bits 2 to 6 are 0.

Definitions for the MAC Command

- #define MAC_MODE_IDX ECC108_PARAM1_IDX

  MAC command index for mode.
- #define MAC_KEYID_IDX ECC108_PARAM2_IDX

  MAC command index for key id.
- #define MAC_CHALLENGE_IDX ECC108_DATA_IDX

  MAC command index for optional challenge.
- #define MAC_COUNT_SHORT ECC108_CMD_SIZE_MIN

  MAC command packet size without challenge.
- #define MAC_COUNT_LONG (39)

  MAC command packet size with challenge.
- #define MAC_MODE_CHALLENGE ((uint8_t) 0x00)

MAC mode 0: first SHA block from data slot.
- #define MAC_MODE_BLOCK2_TEMPKEY ((uint8_t) 0x01)

  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_BLOCK1_TEMPKEY ((uint8_t) 0x02)

  MAC mode bit 1: first SHA block from TempKey.
- #define MAC_MODE_SOURCE_FLAG_MATCH ((uint8_t) 0x04)

  MAC mode bit 2: match TempKey.SourceFlag.
- #define MAC_MODE_PASSTHROUGH ((uint8_t) 0x07)

  MAC mode bit 0-2: pass-through mode.
- #define MAC_MODE_INCLUDE_OTP_88 ((uint8_t) 0x10)

  MAC mode bit 4: include first 88 OTP bits.
- #define MAC_MODE_INCLUDE_OTP_64 ((uint8_t) 0x20)

  MAC mode bit 5: include first 64 OTP bits.
- #define MAC_MODE_INCLUDE_SN ((uint8_t) 0x40)

  MAC mode bit 6: include serial number.
- #define MAC_CHALLENGE_SIZE (32)

  MAC size of challenge.
- #define MAC_MODE_MASK ((uint8_t) 0x77)

  MAC mode bits 3 and 7 are 0.

Definitions for the Nonce Command

- #define NONCE_MODE_IDX ECC108_PARAM1_IDX

  Nonce command index for mode.
- #define NONCE_PARAM2_IDX ECC108_PARAM2_IDX

  Nonce command index for 2. parameter.
- #define NONCE_INPUT_IDX ECC108_DATA_IDX

  Nonce command index for input data.
- #define NONCE_COUNT_SHORT (27)

  Nonce command packet size for 20 bytes of data.
- #define NONCE_COUNT_LONG (39)

  Nonce command packet size for 32 bytes of data.
- #define NONCE_MODE_MASK ((uint8_t) 3)

  Nonce mode bits 2 to 7 are 0.
- #define NONCE_MODE_SEED_UPDATE ((uint8_t) 0x00)

  Nonce mode: update seed.
- #define NONCE_MODE_NO_SEED_UPDATE ((uint8_t) 0x01)

  Nonce mode: do not update seed.
- #define NONCE_MODE_INVALID ((uint8_t) 0x02)

  Nonce mode 2 is invalid.
- #define NONCE_MODE_PASSTHROUGH ((uint8_t) 0x03)

  Nonce mode: pass-through.
- #define NONCE_NUMIN_SIZE (20)

  Nonce data length.
- #define NONCE_NUMIN_SIZE_PASSTHROUGH (32)

  Nonce data length in pass-through mode (mode = 3)

Definitions for the Pause Command

- #define PAUSE_SELECT_IDX ECC108_PARAM1_IDX

  Pause command index for Selector.
- #define PAUSE_PARAM2_IDX ECC108_PARAM2_IDX

  Pause command index for 2. parameter.
- #define PAUSE_COUNT ECC108_CMD_SIZE_MIN

  Pause command packet size.

Definitions for the Random Command

- #define RANDOM_MODE_IDX ECC108_PARAM1_IDX

  Random command index for mode.
- #define RANDOM_PARAM2_IDX ECC108_PARAM2_IDX

  Random command index for 2. parameter.
- #define RANDOM_COUNT ECC108_CMD_SIZE_MIN

  Random command packet size.
- #define RANDOM_SEED_UPDATE ((uint8_t) 0x00)

  Random mode for automatic seed update.
- #define RANDOM_NO_SEED_UPDATE ((uint8_t) 0x01)

  Random mode for no seed update.

Definitions for the Read Command

- #define READ_ZONE_IDX ECC108_PARAM1_IDX

  Read command index for zone.
- #define READ_ADDR_IDX ECC108_PARAM2_IDX

  Read command index for address.
- #define READ_COUNT ECC108_CMD_SIZE_MIN

  Read command packet size.
- #define READ_ZONE_MASK ((uint8_t) 0x83)

  Read zone bits 2 to 6 are 0.
- #define READ_ZONE_MODE_32_BYTES ((uint8_t) 0x80)

  Read mode: 32 bytes.

Definitions for the UpdateExtra Command

- #define UPDATE_MODE_IDX ECC108_PARAM1_IDX

  UpdateExtra command index for mode.
- #define UPDATE_VALUE_IDX ECC108_PARAM2_IDX

  UpdateExtra command index for new value.
- #define UPDATE_COUNT ECC108_CMD_SIZE_MIN

  UpdateExtra command packet size.
- #define UPDATE_CONFIG_BYTE_86 ((uint8_t) 0x01)

  UpdateExtra mode: update Config byte 86.

Definitions for the Write Command

- #define WRITE_ZONE_IDX ECC108_PARAM1_IDX

  Write command index for zone.
- #define WRITE_ADDR_IDX ECC108_PARAM2_IDX

  Write command index for address.
- #define WRITE_VALUE_IDX ECC108_DATA_IDX

  Write command index for data.
- #define WRITE_MAC_VS_IDX (9)

  Write command index for MAC following short data.
- #define WRITE_MAC_VL_IDX (37)

  Write command index for MAC following long data.
- #define WRITE_COUNT_SHORT (11)

  Write command packet size with short data and no MAC.
- #define WRITE_COUNT_LONG (39)

  Write command packet size with long data and no MAC.
- #define WRITE_COUNT_SHORT_MAC (43)

  Write command packet size with short data and MAC.
- #define WRITE_COUNT_LONG_MAC (71)

  Write command packet size with long data and MAC.
- #define WRITE_MAC_SIZE (32)

  Write MAC size.
- #define WRITE_ZONE_MASK ((uint8_t) 0xC3)

  Write zone bits 2 to 5 are 0.
- #define WRITE_ZONE_WITH_MAC ((uint8_t) 0x40)

  Write zone bit 6: write encrypted with MAC.

Response Size Definitions

- #define CHECKMAC_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of DeriveKey command
- #define DERIVE_KEY_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of DeriveKey command
- #define INFO_RSP_SIZE ECC108_RSP_SIZE_VAL

  response size of Info command returns 4 bytes
- #define GENDIG_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of GenDig command
- #define HMAC_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of HMAC command
- #define LOCK_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of Lock command
- #define MAC_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of MAC command
- #define NONCE_RSP_SIZE_SHORT ECC108_RSP_SIZE_MIN

  response size of Nonce command with mode[0:1] = 3
- #define NONCE_RSP_SIZE_LONG ECC108_RSP_SIZE_MAX

  response size of Nonce command

- #define PAUSE_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of Pause command
- #define RANDOM_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of Random command
- #define READ_4_RSP_SIZE ECC108_RSP_SIZE_VAL

  response size of Read command when reading 4 bytes
- #define READ_32_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of Read command when reading 32 bytes
- #define UPDATE_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of UpdateExtra command
- #define WRITE_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of Write command

Definitions of Typical Command Execution Times

The library starts polling the device for a response after these delays.

- #define CHECKMAC_DELAY ((uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  CheckMAC typical command delay.
- #define DERIVE_KEY_DELAY ((uint8_t) (14.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  DeriveKey typical command delay.
- #define INFO_DELAY ((uint8_t) (1))

  DevRev typical command delay.
- #define GENDIG_DELAY ((uint8_t) (11.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  GenDig typical command delay.
- #define HMAC_DELAY ((uint8_t) (27.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  HMAC typical command delay.
- #define LOCK_DELAY ((uint8_t) (5.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Lock typical command delay.
- #define MAC_DELAY ((uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  MAC typical command delay.
- #define NONCE_DELAY ((uint8_t) (22.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Nonce typical command delay.
- #define PAUSE_DELAY ((uint8_t) (1))

  Pause typical command delay.
- #define RANDOM_DELAY ((uint8_t) (11.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Random typical command delay.
- #define READ_DELAY ((uint8_t) (1))

  Read typical command delay.
- #define UPDATE_DELAY ((uint8_t) (8.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  UpdateExtra typical command delay.
- #define WRITE_DELAY ((uint8_t) (4.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Write typical command delay.

Definitions of Maximum Command Execution Times

- #define CHECKMAC_EXEC_MAX ((uint8_t) (38.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  CheckMAC maximum execution time.
- #define DERIVE_KEY_EXEC_MAX ((uint8_t) (62.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  DeriveKey maximum execution time.
- #define INFO_EXEC_MAX ((uint8_t) (2.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  DevRev maximum execution time.
- #define GENDIG_EXEC_MAX ((uint8_t) (43.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  GenDig maximum execution time.
- #define HMAC_EXEC_MAX ((uint8_t) (69.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  HMAC maximum execution time.
- #define LOCK_EXEC_MAX ((uint8_t) (24.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Lock maximum execution time.
- #define MAC_EXEC_MAX ((uint8_t) (35.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  MAC maximum execution time.
- #define NONCE_EXEC_MAX ((uint8_t) (60.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Nonce maximum execution time.
- #define PAUSE_EXEC_MAX ((uint8_t) (2.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Pause maximum execution time.
- #define RANDOM_EXEC_MAX ((uint8_t) (50.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Random maximum execution time.
- #define READ_EXEC_MAX ((uint8_t) (4.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Read maximum execution time.
- #define UPDATE_EXEC_MAX ((uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  UpdateExtra maximum execution time.
- #define WRITE_EXEC_MAX ((uint8_t) (42.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Write maximum execution time.

5.1.1 Detailed Description

A function is provided for every ATECC108 command in the final release. These functions check the parameters, assemble a command packet, send it, receive its response, and return the status of the operation and the response. If available code space in your system is tight, or this version of the library does not provide a wrapper function for the command you like to use, you can use the ecc108m_execute function for any command. It is more complex to use, though. Modern compilers can garbage-collect unused functions. If your compiler does not support this feature and you want to use only the ecc108m_execute function, you can just delete the command wrapper functions. If you do use the command wrapper functions, you can respectively delete the ecc108m_execute function.

5.1.2 Function Documentation

5.1.2.1 uint8_t ecc108m_check_mac (uint8_t ∗ *tx_buffer,* uint8_t ∗ *rx_buffer,* uint8_t *mode,* uint8_t *key_id,* uint8_t ∗ *client_challenge,* uint8_t ∗ *client_response,* uint8_t ∗ *other_data*)

This function sends a CheckMAC command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*mode*	selects the hash inputs
in	*key_id*	slot index of key
in	*client_challenge*	pointer to client challenge (ignored if mode bit 0 is set)
in	*client_response*	pointer to client response
in	*other_data*	pointer to 13 bytes of data used in the client command

Returns

status of the operation

5.1.2.2 uint8_t ecc108m_derive_key (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *random*, uint8_t *target_key*, uint8_t * *mac*)

This function sends a DeriveKey command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*random*	type of source key (has to match TempKey.SourceFlag)
in	*target_key*	slot index of key (0..15); not used if random is 1
in	*mac*	pointer to optional MAC

Returns

status of the operation

5.1.2.3 uint8_t ecc108m_execute (uint8_t *op_code*, uint8_t *param1*, uint16_t *param2*, uint8_t *datalen1*, uint8_t * *data1*, uint8_t *datalen2*, uint8_t * *data2*, uint8_t *datalen3*, uint8_t * *data3*, uint8_t *tx_size*, uint8_t * *tx_buffer*, uint8_t *rx_size*, uint8_t * *rx_buffer*)

This function creates a command packet, sends it, and receives its response.

Parameters

in	*op_code*	command op-code
in	*param1*	first parameter
in	*param2*	second parameter
in	*datalen1*	number of bytes in first data block
in	*data1*	pointer to first data block
in	*datalen2*	number of bytes in second data block
in	*data2*	pointer to second data block
in	*datalen3*	number of bytes in third data block
in	*data3*	pointer to third data block

in	*tx_size*	size of tx buffer
in	*tx_buffer*	pointer to tx buffer
in	*rx_size*	size of rx buffer
out	*rx_buffer*	pointer to rx buffer

Returns

status of the operation

5.1.2.4 uint8_t ecc108m_gen_dig (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *zone*, uint8_t *key_id*, uint8_t * *other_data*)

This function sends a GenDig command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*zone*	0: config, zone 1: OTP zone, 2: data zone
in	*key_id*	zone 1: OTP block; zone 2: key id
in	*other_data*	pointer to 4 bytes of data when using CheckOnly key

Returns

status of the operation

5.1.2.5 uint8_t ecc108m_hmac (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *mode*, uint16_t *key_id*)

This function sends an HMAC command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*mode*	
in	*key_id*	slot index of key

Returns

status of the operation

5.1.2.6 uint8_t ecc108m_info (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *mode*, uint8_t *gpio_state*)

This function sends an Info command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer

out	*rx_buffer*	pointer to receive buffer
in	*mode*	what info to get
in	*gpio_state*	what GPIO state to get

Returns

status of the operation

5.1.2.7 uint8_t ecc108m_lock (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *zone*, uint16_t *summary*)

This function sends a Lock command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*zone*	zone id to lock
in	*summary*	zone digest

Returns

status of the operation

5.1.2.8 uint8_t ecc108m_mac (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *mode*, uint16_t *key_id*, uint8_t * *challenge*)

This function sends a MAC command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*mode*	selects message fields
in	*key_id*	slot index of key
in	*challenge*	pointer to challenge (not used if mode bit 0 is set)

Returns

status of the operation

5.1.2.9 uint8_t ecc108m_nonce (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *mode*, uint8_t * *numin*)

This function sends a Nonce command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer

in	*mode*	controls the mechanism of the internal random number generator and seed update
in	*numin*	pointer to system input (mode = 3: 32 bytes same as in TempKey; mode $<$ 2: 20 bytes mode == 2: not allowed)

Returns

status of the operation

5.1.2.10 uint8_t ecc108m_pause (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *selector*)

This function sends a Pause command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*selector*	Devices not matching this value will pause.

Returns

status of the operation

5.1.2.11 uint8_t ecc108m_random (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *mode*)

This function sends a Random command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*mode*	0: update seed; 1: no seed update

Returns

status of the operation

5.1.2.12 uint8_t ecc108m_read (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *zone,* uint16_t *address*)

This function sends a Read command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer

in	*zone*	0: Configuration; 1: OTP; 2: Data
in	*address*	address to read from

Returns

status of the operation

5.1.2.13 uint8_t ecc108m_update_extra (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *mode,* uint8_t *new_value*)

This function sends an UpdateExtra command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*mode*	0: update Configuration zone byte 85; 1: byte 86
in	*new_value*	byte to write

Returns

status of the operation

5.1.2.14 uint8_t ecc108m_write (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *zone,* uint16_t *address,* uint8_t * *new_value,* uint8_t * *mac*)

This function sends a Write command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*zone*	0: Configuration; 1: OTP; 2: Data
in	*address*	address to write to
in	*new_value*	pointer to 32 (zone bit 7 set) or 4 bytes of data
in	*mac*	pointer to MAC (ignored if zone is unlocked)

Returns

status of the operation

5.2 Module 02: Communication

Macros

- #define ECC108_COMMAND_EXEC_MAX ((uint8_t) (120.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))
  maximum command delay
- #define ECC108_CMD_SIZE_MIN ((uint8_t) 7)
  minimum number of bytes in command (from count byte to second CRC byte)
- #define ECC108_CMD_SIZE_MAX ((uint8_t) 4 ∗ 36 + 7)
  maximum size of command packet (Verify)
- #define ECC108_CRC_SIZE ((uint8_t) 2)
  number of CRC bytes
- #define ECC108_BUFFER_POS_STATUS (1)
  buffer index of status byte in status response
- #define ECC108_BUFFER_POS_DATA (1)
  buffer index of first data byte in data response
- #define ECC108_STATUS_BYTE_WAKEUP ((uint8_t) 0x11)
  status byte after wake-up
- #define ECC108_STATUS_BYTE_PARSE ((uint8_t) 0x03)
  command parse error
- #define ECC108_STATUS_BYTE_EXEC ((uint8_t) 0x0F)
  command execution error
- #define ECC108_STATUS_BYTE_COMM ((uint8_t) 0xFF)
  communication error

Functions

- void ecc108c_calculate_crc (uint8_t length, uint8_t ∗data, uint8_t ∗crc)
  This function calculates CRC.
- uint8_t ecc108c_wakeup (uint8_t ∗response)
  This function wakes up a ECC108 device and receives a response.
- uint8_t ecc108c_send_and_receive (uint8_t ∗tx_buffer, uint8_t rx_size, uint8_t ∗rx_buffer, uint8_t execution_delay, uint8_t execution_timeout)
  This function runs a communication sequence: Append CRC to tx buffer, send command, delay, and verify response after receiving it.

5.2.1 Detailed Description

This module implements communication with the device. It does not depend on the interface (SWI or I2C).

Basic communication flow:

- Calculate CRC of command packet and append.
- Send command and repeat if it failed.
- Delay for minimum command execution time.
- Poll for response until maximum execution time. Repeat if communication failed.

Retries are implemented including sending the command again depending on the type of failure. A retry might include waking up the device which will be indicated by an appropriate return status. The number of retries is defined with a macro and can be set to 0 at compile time.

5.2.2 Function Documentation

5.2.2.1 void ecc108c_calculate_crc (uint8_t *length,* uint8_t ∗ *data,* uint8_t ∗ *crc*)

This function calculates CRC.

Parameters

in	*length*	number of bytes in buffer
in	*data*	pointer to data for which CRC should be calculated
out	*crc*	pointer to 16-bit CRC

5.2.2.2 uint8_t ecc108c_send_and_receive (uint8_t ∗ *tx_buffer,* uint8_t *rx_size,* uint8_t ∗ *rx_buffer,* uint8_t *execution_delay,* uint8_t *execution_timeout*)

This function runs a communication sequence: Append CRC to tx buffer, send command, delay, and verify response after receiving it.

The first byte in tx buffer must be the byte count of the packet. If CRC or count of the response is incorrect, or a command byte got "nacked" (TWI), this function requests re-sending the response. If the response contains an error status, this function resends the command.

Parameters

in	*tx_buffer*	pointer to command
in	*rx_size*	size of response buffer
out	*rx_buffer*	pointer to response buffer
in	*execution_delay*	Start polling for a response after this many ms .
in	*execution_-timeout*	polling timeout in ms

Returns

status of the operation

5.2.2.3 uint8_t ecc108c_wakeup (uint8_t ∗ *response*)

This function wakes up a ECC108 device and receives a response.

Parameters

out	*response*	pointer to four-byte response

Returns

status of the operation

5.3 Module 03: Header File for Interface Abstraction Modules

This header file contains definitions and function prototypes for SWI and I^2 C. The prototypes are the same for both interfaces but are of course implemented differently. Always include this file no matter whether you use SWI or I2C.

Macros

- #define ECC108_RSP_SIZE_MIN ((uint8_t) 4)

    minimum number of bytes in response
- #define ECC108_RSP_SIZE_MAX ((uint8_t) (72 + 3))

    maximum size of response packet (GenKey and Verify command)
- #define ECC108_BUFFER_POS_COUNT (0)

    buffer index of count byte in command or response
- #define ECC108_BUFFER_POS_DATA (1)

    buffer index of data in response
- #define ECC108_WAKEUP_PULSE_WIDTH (uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)
- #define ECC108_WAKEUP_DELAY (uint8_t) (200.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

Functions

- uint8_t ecc108p_send_command (uint8_t count, uint8_t ∗command)

    This I2C function sends a command to the device.
- uint8_t ecc108p_receive_response (uint8_t size, uint8_t ∗response)

    This I2C function receives a response from the ECC108 device.
- void ecc108p_init (void)

    This I2C function initializes the hardware.
- void ecc108p_set_device_id (uint8_t id)

    This I2C function sets the I2C address. Communication functions will use this address.
- uint8_t ecc108p_wakeup (void)

    This I2C function generates a Wake-up pulse and delays.
- uint8_t ecc108p_idle (void)

    This I2C function puts the ECC108 device into idle state.
- uint8_t ecc108p_sleep (void)

    This I2C function puts the ECC108 device into low-power state.
- uint8_t ecc108p_reset_io (void)

    This I2C function resets the I/O buffer of the ECC108 device.
- uint8_t ecc108p_resync (uint8_t size, uint8_t ∗response)

    This I2C function resynchronizes communication.

5.3.1 Detailed Description

This header file contains definitions and function prototypes for SWI and I^2 C. The prototypes are the same for both interfaces but are of course implemented differently. Always include this file no matter whether you use SWI or I2C.

5.3.2 Macro Definition Documentation

5.3.2.1 #define ECC108_WAKEUP_DELAY (uint8_t) (200.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

delay between Wakeup pulse and communication in 10 us units Device versions <= 0x100 need a longer delay of 2 ms instead of 0.5 ms.

5.3.2.2 #define ECC108_WAKEUP_PULSE_WIDTH (uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

width of Wakeup pulse in 10 us units Device versions <= 0x100 need a longer pulse of 120 us instead of 60 us.

5.3.3 Function Documentation

5.3.3.1 uint8_t ecc108p_idle (void)

This I2C function puts the ECC108 device into idle state.

Returns

status of the operation

This I2C function puts the ECC108 device into idle state.

Returns

status of the operation

5.3.3.2 uint8_t ecc108p_receive_response (uint8_t *size*, uint8_t ∗ *response*)

This I2C function receives a response from the ECC108 device.

Parameters

in	*size*	size of rx buffer
out	*response*	pointer to rx buffer

Returns

status of the operation

This I2C function receives a response from the ECC108 device.

Parameters

in	*size*	number of bytes to receive
out	*response*	pointer to response buffer

Returns

status of the operation

5.3.3.3 uint8_t ecc108p_reset_io (void)

This I2C function resets the I/O buffer of the ECC108 device.

Returns

status of the operation

This I2C function resets the I/O buffer of the ECC108 device.

Returns

success

5.3.3.4 uint8_t ecc108p_resync (uint8_t *size*, uint8_t ∗ *response*)

This I2C function resynchronizes communication.

Parameters are not used for I2C.

Re-synchronizing communication is done in a maximum of three steps listed below. This function implements the first step. Since steps 2 and 3 (sending a Wake-up token and reading the response) are the same for I2C and SWI, they are implemented in the communication layer (ecc108c_resync).

1. To ensure an IO channel reset, the system should send the standard I2C software reset sequence, as follows:

    - a Start condition
    - nine cycles of SCL, with SDA held high
    - another Start condition
    - a Stop condition

    It should then be possible to send a read sequence and if synchronization has completed properly the ATSHA204 will acknowledge the device address. The chip may return data or may leave the bus floating (which the system will interpret as a data value of 0xFF) during the data periods.

    If the chip does acknowledge the device address, the system should reset the internal address counter to force the ATSHA204 to ignore any partial input command that may have been sent. This can be accomplished by sending a write sequence to word address 0x00 (Reset), followed by a Stop condition.

2. If the chip does NOT respond to the device address with an ACK, then it may be asleep. In this case, the system should send a complete Wake token and wait t_whi after the rising edge. The system may then send another read sequence and if synchronization has completed the chip will acknowledge the device address.

3. If the chip still does not respond to the device address with an acknowledge, then it may be busy executing a command. The system should wait the longest TEXEC and then send the read sequence, which will be acknowledged by the chip.

Parameters

in	*size*	size of rx buffer
out	*response*	pointer to response buffer

Returns

status of the operation

This I2C function resynchronizes communication.

Re-synchronizing communication is done in a maximum of five steps listed below. This function implements the first three steps. Since steps 4 and 5 (sending a Wake-up token and reading the response) are the same for TWI and SWI, they are implemented in the communication layer (ecc108c_resync).

If the chip is not busy when the system sends a transmit flag, the chip should respond within t_turnaround. If t_exec has not already passed, the chip may be busy and the system should poll or wait until the maximum tEXEC time has elapsed. If the chip still does not respond to a second transmit flag within t_turnaround, it may be out of synchronization. At this point the system may take the following steps to reestablish communication:

1. Wait t_timeout.
2. Send the transmit flag.
3. If the chip responds within t_turnaround, then the system may proceed with more commands.
4. Send a Wake token, wait t_whi, and send the transmit flag.
5. The chip should respond with a 0x11 return status within t_turnaround, after which the system may proceed with more commands.

Parameters

in	*size*	size of rx buffer
out	*response*	pointer to response buffer

Returns

status of the operation

5.3.3.5 uint8_t ecc108p_send_command (uint8_t *count*, uint8_t ∗ *command*)

This I2C function sends a command to the device.

Parameters

in	*count*	number of bytes to send
in	*command*	pointer to command buffer

Returns

status of the operation

This I2C function sends a command to the device.

Parameters

in	*count*	number of bytes to send

in	*command*	pointer to command buffer

Returns

status of the operation

5.3.3.6 void ecc108p_set_device_id (uint8_t *id*)

This I2C function sets the I2C address. Communication functions will use this address.

Parameters

in	*id*	I2C address

This I2C function sets the I2C address. Communication functions will use this address.

```
It has no effect when using a UART.
```

Parameters

in	*id*	index into array of pins

5.3.3.7 uint8_t ecc108p_sleep (void)

This I2C function puts the ECC108 device into low-power state.

Returns

status of the operation

This I2C function puts the ECC108 device into low-power state.

Returns

status of the operation

5.3.3.8 uint8_t ecc108p_wakeup (void)

This I2C function generates a Wake-up pulse and delays.

Returns

status of the operation

This I2C function generates a Wake-up pulse and delays.

Returns

success

5.4 Module 04: SWI Abstraction Module

< definitions for delay functions

Macros

- #define ECC108_SWI_FLAG_CMD ((uint8_t) 0x77)

    flag preceding a command
- #define ECC108_SWI_FLAG_TX ((uint8_t) 0x88)

    flag requesting a response
- #define ECC108_SWI_FLAG_IDLE ((uint8_t) 0xBB)

    flag requesting to go into Idle mode
- #define ECC108_SWI_FLAG_SLEEP ((uint8_t) 0xCC)

    flag requesting to go into Sleep mode

Functions

- void ecc108p_init (void)

    This SWI function initializes the hardware.
- void ecc108p_set_device_id (uint8_t id)

    This SWI function selects the GPIO pin used for communication.
- uint8_t ecc108p_send_command (uint8_t count, uint8_t *command)

    This SWI function sends a command to the device. Device versions <= 0x100 need the flag to last longer than 500 us. Therefore, we send a dummy flag of 0 before sending the command flag.
- uint8_t ecc108p_receive_response (uint8_t size, uint8_t *response)

    This SWI function receives a response from the device.
- uint8_t ecc108p_wakeup (void)

    This SWI function generates a Wake-up pulse and delays.
- uint8_t ecc108p_idle ()

    This SWI function puts the device into idle state.
- uint8_t ecc108p_sleep ()

    This SWI function puts the device into low-power state.
- uint8_t ecc108p_reset_io (void)

    This SWI function is only a dummy since the functionality does not exist for the SWI version of the ECC108 device.
- uint8_t ecc108p_resync (uint8_t size, uint8_t *response)

    This function re-synchronizes communication.

5.4.1 Detailed Description

< definitions for delay functions < hardware dependent declarations for SWI < declarations that are common to all interface implementations < declarations of function return codes

These functions and definitions abstract the SWI hardware. They implement the functions declared in ecc108_physical.h.

5.4.2 Function Documentation

5.4.2.1 uint8_t ecc108p_idle (void)

This SWI function puts the device into idle state.

This I2C function puts the ECC108 device into idle state.

Returns

status of the operation

5.4.2.2 void ecc108p_init (void)

This SWI function initializes the hardware.

This I2C function initializes the hardware.

5.4.2.3 uint8_t ecc108p_receive_response (uint8_t *size*, uint8_t * *response*)

This SWI function receives a response from the device.

This I2C function receives a response from the ECC108 device.

Parameters

in	*size*	number of bytes to receive
out	*response*	pointer to response buffer

Returns

status of the operation

5.4.2.4 uint8_t ecc108p_reset_io (void)

This SWI function is only a dummy since the functionality does not exist for the SWI version of the ECC108 device.

This I2C function resets the I/O buffer of the ECC108 device.

Returns

success

5.4.2.5 uint8_t ecc108p_resync (uint8_t *size*, uint8_t * *response*)

This function re-synchronizes communication.

This I2C function resynchronizes communication.

Re-synchronizing communication is done in a maximum of five steps listed below. This function implements the first three steps. Since steps 4 and 5 (sending a Wake-up token and reading the response) are the same for TWI and SWI, they are implemented in the communication layer (ecc108c_resync).

If the chip is not busy when the system sends a transmit flag, the chip should respond within t_turnaround. If t_exec has not already passed, the chip may be busy and the system should poll or wait until the maximum tEXEC time has elapsed. If the chip still does not respond to a second transmit flag within t_turnaround, it may be out of synchronization. At this point the system may take the following steps to reestablish communication:

1. Wait t_timeout.
2. Send the transmit flag.
3. If the chip responds within t_turnaround, then the system may proceed with more commands.
4. Send a Wake token, wait t_whi, and send the transmit flag.
5. The chip should respond with a 0x11 return status within t_turnaround, after which the system may proceed with more commands.

Parameters

in	*size*	size of rx buffer
out	*response*	pointer to response buffer

Returns

status of the operation

5.4.2.6 uint8_t ecc108p_send_command (uint8_t *count,* uint8_t ∗ *command*)

This SWI function sends a command to the device. Device versions <= 0x100 need the flag to last longer than 500 us. Therefore, we send a dummy flag of 0 before sending the command flag.

This I2C function sends a command to the device.

Parameters

in	*count*	number of bytes to send
in	*command*	pointer to command buffer

Returns

status of the operation

5.4.2.7 void ecc108p_set_device_id (uint8_t *id*)

This SWI function selects the GPIO pin used for communication.

This I2C function sets the I2C address. Communication functions will use this address.

```
It has no effect when using a UART.
```

Parameters

in	*id*	index into array of pins

5.4.2.8 uint8_t ecc108p_sleep (void)

This SWI function puts the device into low-power state.

This I2C function puts the ECC108 device into low-power state.

Returns

status of the operation

5.4.2.9 uint8_t ecc108p_wakeup (void)

This SWI function generates a Wake-up pulse and delays.

This I2C function generates a Wake-up pulse and delays.

Returns

success

5.5 Module 06: Helper Functions

Use these functions if your system does not use an ATECC108 as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATECC108 client device. They are intended to accompany the ATECC108 library functions. They can be called directly from an application, or integrated into an API.

Data Structures

- struct ecc108h_temp_key

  Structure to hold TempKey fields.
- struct ecc108h_calculate_sha256_in_out

  Input/output parameters for function ecc108h_nonce().
- struct ecc108h_nonce_in_out

  Input/output parameters for function ecc108h_nonce().
- struct ecc108h_mac_in_out

  Input/output parameters for function ecc108h_mac().
- struct ecc108h_hmac_in_out

  Input/output parameters for function ecc108h_hmac().
- struct ecc108h_gen_dig_in_out

  Input/output parameters for function ecc108h_gen_dig().
- struct ecc108h_derive_key_in_out

  Input/output parameters for function ecc108h_derive_key().
- struct ecc108h_derive_key_mac_in_out

  Input/output parameters for function ecc108h_derive_key_mac().
- struct ecc108h_encrypt_in_out

  Input/output parameters for function ecc108h_encrypt().
- struct ecc108h_decrypt_in_out

  Input/output parameters for function ecc108h_decrypt().
- struct ecc108h_check_mac_in_out

  Input/output parameters for function ecc108h_check_mac().

Functions

- uint8_t ecc108h_nonce (struct ecc108h_nonce_in_out ∗param)

  This function calculates a 32-byte nonce based on 20-byte input value (NumIn) and 32-byte random number (RandOut).
- uint8_t ecc108h_mac (struct ecc108h_mac_in_out ∗param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other informations.
- uint8_t ecc108h_check_mac (struct ecc108h_check_mac_in_out ∗param)

  This function calculates SHA-256 digest (MAC) of a password and other informations, to be verified using CheckMac command in the Device.
- uint8_t ecc108h_hmac (struct ecc108h_hmac_in_out ∗param)

  This function generates an HMAC/SHA-256 digest of a key and other informations.
- uint8_t ecc108h_gen_dig (struct ecc108h_gen_dig_in_out ∗param)

  This function combines current TempKey with a stored value.
- uint8_t ecc108h_derive_key (struct ecc108h_derive_key_in_out ∗param)

  This function combines current value of a key with the TempKey.
- uint8_t ecc108h_derive_key_mac (struct ecc108h_derive_key_mac_in_out ∗param)

This function calculates input MAC for DeriveKey opcode.

- uint8_t ecc108h_encrypt (struct ecc108h_encrypt_in_out ∗param)

  This function encrypts 32-byte cleartext data to be written using Write opcode, and optionally calculates input MAC.

- uint8_t ecc108h_decrypt (struct ecc108h_decrypt_in_out ∗param)

  This function decrypts 32-byte encrypted data (Contents) from Read opcode.

- void ecc108h_calculate_crc_chain (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates CRC.

- void ecc108h_calculate_sha256 (int32_t len, uint8_t ∗message, uint8_t ∗digest)

  This function creates a SHA256 digest on a little-endian system.

Variables

- uint8_t value [32]

  The value of TempKey. Nonce (from nonce command) or Digest (from GenDig command)

- unsigned int key_id:4

  If TempKey was generated by GenDig (see the GenData and CheckFlag bits), these bits indicate which key was used in its computation.

- unsigned int source_flag:1

  The source of the randomness in TempKey: 0=Rand, 1=Input.

- unsigned int gen_data:1

  Indicates if TempKey has been generated by GenDig using Data zone.

- unsigned int check_flag:1

  Not used in the library.

- unsigned int valid:1

  Indicates if the information in TempKey is valid.

- uint32_t length

  [in] Length of input message to be digested.

- uint8_t ∗ message

  [in] Pointer to input message.

- uint8_t ∗ digest

  [out] Pointer to 32-byte SHA256 digest of input message.

- uint8_t mode

  [in] Mode parameter used in Nonce command (Param1).

- uint8_t ∗ num_in

  [in] Pointer to 20-byte NumIn data used in Nonce command.

- uint8_t ∗ rand_out

  [in] Pointer to 32-byte RandOut data from Nonce command.

- struct ecc108h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

- uint8_t mode

  [in] Mode parameter used in MAC command (Param1).

- uint16_t key_id

  [in] KeyID parameter used in MAC command (Param2).

- uint8_t ∗ challenge

  [in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.

- uint8_t ∗ key

  [in] Pointer to 32-byte key used to generate MAC digest.

- uint8_t ∗ otp

  [in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.
- uint8_t ∗ sn

  [in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.
- uint8_t ∗ response

  [out] Pointer to 32-byte SHA-256 digest (MAC).
- struct ecc108h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t mode

  [in] Mode parameter used in HMAC command (Param1).
- uint16_t key_id

  [in] KeyID parameter used in HMAC command (Param2).
- uint8_t ∗ key

  [in] Pointer to 32-byte key used to generate HMAC digest.
- uint8_t ∗ otp

  [in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.
- uint8_t ∗ sn

  [in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.
- uint8_t ∗ response

  [out] Pointer to 32-byte SHA-256 HMAC digest.
- struct ecc108h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t zone

  [in] Zone parameter used in GenDig command (Param1).
- uint16_t key_id

  [in] KeyID parameter used in GenDig command (Param2).
- uint8_t ∗ stored_value

  [in] Pointer to 32-byte stored value, can be a data slot, OTP page, configuration zone, or hardware transport key.
- struct ecc108h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t random

  [in] Random parameter used in DeriveKey command (Param1).
- uint16_t target_key_id

  [in] KeyID to be derived, TargetKey parameter used in DeriveKey command (Param2).
- uint8_t ∗ parent_key

  [in] Pointer to 32-byte ParentKey. Set equal to target_key if Roll Key operation is intended.
- uint8_t ∗ target_key

  [out] Pointer to 32-byte TargetKey.
- struct ecc108h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t random

  [in] Random parameter used in DeriveKey command (Param1).
- uint16_t target_key_id

  [in] KeyID to be derived, TargetKey parameter used in DeriveKey command (Param2).
- uint8_t ∗ parent_key

  [in] Pointer to 32-byte ParentKey. ParentKey here is always SlotConfig[TargetKey].WriteKey, regardless whether the operation is Roll or Create.

- uint8_t * mac

  [out] Pointer to 32-byte Mac.
- uint8_t zone

  [in] Zone parameter used in Write (Param1).
- uint16_t address

  [in] Address parameter used in Write command (Param2).
- uint8_t * data

  [in,out] Pointer to 32-byte data. Input cleartext data, output encrypted data to Write command (Value field).
- uint8_t * mac

  [out] Pointer to 32-byte Mac. Can be set to NULL if input MAC is not required by the Write command (write to OTP, unlocked user zone).
- struct ecc108h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t * data

  [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.
- struct ecc108h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t mode

  [in] Mode parameter used in CheckMac command (Param1).
- uint8_t * password

  [in] Pointer to 32-byte password that will be verified against Key[KeyID] in the Device.
- uint8_t * other_data

  [in] Pointer to 13-byte OtherData that will be used in CheckMac command.
- uint8_t * otp

  [in] Pointer to 11-byte OTP. OTP[0:7] is included in the calculation if Mode bit 5 is one.
- uint8_t * target_key

  [in] Pointer to 32-byte TargetKey that will be copied to TempKey.
- uint8_t * client_resp

  [out] Pointer to 32-byte ClientResp to be used in CheckMac command.
- struct ecc108h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

5.5.1 Detailed Description

Use these functions if your system does not use an ATECC108 as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATECC108 client device. They are intended to accompany the ATECC108 library functions. They can be called directly from an application, or integrated into an API. Modern compilers can garbage-collect unused functions. If your compiler does not support this feature, you can just discard this module from your project if you do use an ATECC108 as a host. Or, if you don't, delete the functions you do not use.

5.5.2 Function Documentation

5.5.2.1 void ecc108h_calculate_crc_chain (uint8_t *length*, uint8_t * *data*, uint8_t * *crc*)

This function calculates CRC.

```
crc_register is initialized with *crc, so it can be chained to calculate CRC from large array of data.
For the first calculation or calculation without chaining, crc[0] and crc[1] values must be initialized to 0 
```

Parameters

in	*length*	number of bytes in buffer
in	*data*	pointer to data for which CRC should be calculated
out	*crc*	pointer to 16-bit CRC

5.5.2.2 void ecc108h_calculate_sha256 (int32_t *len,* uint8_t ∗ *message,* uint8_t ∗ *digest*)

This function creates a SHA256 digest on a little-endian system.

Limitations: This function was implemented with the ATSHA204 crypto device in mind. It will therefore only work for length values of len % 64 $<$ 62.

Parameters

in	*len*	byte length of message
in	*message*	pointer to message
out	*digest*	SHA256 of message

5.5.2.3 uint8_t ecc108h_check_mac (struct ecc108h_check_mac_in_out ∗ *param*)

This function calculates SHA-256 digest (MAC) of a password and other informations, to be verified using CheckMac command in the Device.

```
This password checking operation is described in "Section 3.3.6 Password Checking" of "Atmel ATSHA204 [DATASHI
Before performing password checking operation, TempKey should contain a randomly generated nonce. The TempKey
User enters the password to be verified to Application.
Application passes this password to CheckMac calculation function, along with 13-byte OtherData, 32-byte targ
The function calculates a 32-byte ClientResp, returns it to Application. The function also replaces the curre
Application passes the calculated ClientResp along with OtherData to the Device, and has it execute CheckMac
The Device validates ClientResp, and copies target slot to TempKey.

If the password is stored in odd numbered slot, the target slot is the password slot itself, so target_key pa
If the password is stored in even numbered slot, the target slot is next odd numbered slot (KeyID+1), so targ

Note that the function does not check the result of password checking operation.
Regardless of whether the CheckMac command returns success or not, TempKey in Application will hold the value
Therefore Application has to make sure that password checking operation succeeds before using the TempKey for
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_check_mac_in_out.

Returns

status of the operation.

5.5.2.4 uint8_t ecc108h_decrypt (struct ecc108h_decrypt_in_out ∗ *param*)

This function decrypts 32-byte encrypted data (Contents) from Read opcode.

```
To use this function, first the nonce must be valid and synchronized between Device and Application.
Application executes GenDig command in the Device, using key specified by SlotConfig.ReadKey. The Device upda
Application then updates its own TempKey using GenDig calculation function, using the same key.
Application executes Read command in the Device to a user zone configured with EncryptRead.
```

```
The Device encrypts 32-byte zone contents, and outputs it to the host.
Application passes this encrypted data to decryption function. The function decrypts the data, and returns it
TempKey must be updated by GenDig using a ParentKey as specified by SlotConfig.ReadKey before executing this
The decryption function does not check whether the TempKey has been generated by correct ParentKey for the co
Therefore to get a correct result, Application has to make sure that prior GenDig calculation was done using
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_decrypt_in_out.

Returns

status of the operation.

5.5.2.5 uint8_t ecc108h_derive_key (struct ecc108h_derive_key_in_out * *param*)

This function combines current value of a key with the TempKey.

```
Used in conjunction with DeriveKey command, the key derived by this function will match with the key in the De
Two kinds of operation are supported:
- Roll Key operation, target_key and parent_key parameters should be set to point to the same location (Target
- Create Key operation, target_key should be set to point to TargetKey, parent_key should be set to point to
After executing this function, initial value of target_key will be overwritten with the derived key.
The TempKey should be valid (temp_key.valid = 1) before executing this function.
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_derive_key_in_out.

Returns

status of the operation.

5.5.2.6 uint8_t ecc108h_derive_key_mac (struct ecc108h_derive_key_mac_in_out * *param*)

This function calculates input MAC for DeriveKey opcode.

```
DeriveKey command will need an input MAC if SlotConfig[TargetKey].Bit15 is set.
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_derive_key_mac_in_out.

Returns

status of the operation.

5.5.2.7 uint8_t ecc108h_encrypt (struct ecc108h_encrypt_in_out * *param*)

This function encrypts 32-byte cleartext data to be written using Write opcode, and optionally calculates input MAC.

```
To use this function, first the nonce must be valid and synchronized between Device and Application.
Application executes GenDig command in the Device, using parent key. If Data zone has been locked, this is sp
Application then updates its own TempKey using GenDig calculation function, using the same key.
Application passes the cleartext data to encryption function.
If input MAC is needed, application must pass a valid pointer to buffer in the "mac" parameter.
If input MAC is not needed, application can pass NULL pointer in "mac" parameter.
The function encrypts the data and optionally calculate input MAC, returns it to Application.
Using this encrypted data and input MAC, Application executes Write command in the Device. Device validates t
The encryption function does not check whether the TempKey has been generated by correct ParentKey for the co
Therefore to get a correct result, after Data/OTP locked, Application has to make sure that prior GenDig calc
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_encrypt_in_out.

Returns

status of the operation.

5.5.2.8 uint8_t ecc108h_gen_dig (struct ecc108h_gen_dig_in_out ∗ *param*)

This function combines current TempKey with a stored value.

```
The stored value can be a data slot, OTP page, configuration zone, or hardware transport key.
The TempKey generated by this function will match with the TempKey in the Device generated by GenDig opcode.
The TempKey should be valid (temp_key.valid = 1) before executing this function.
To use this function, Application first executes GenDig command in the Device, with a chosen stored value.
This stored value must be known by the Application, and is passed to GenDig calculation function.
The function calculates new TempKey, and returns it.
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_gen_dig_in_out.

Returns

status of the operation.

5.5.2.9 uint8_t ecc108h_hmac (struct ecc108h_hmac_in_out ∗ *param*)

This function generates an HMAC/SHA-256 digest of a key and other informations.

```
The resulting digest will match with those generated in the Device by HMAC opcode.
The TempKey should be valid (temp_key.valid = 1) before executing this function.
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_hmac_in_out.

Returns

status of the operation.

5.5.2.10 uint8_t ecc108h_mac (struct ecc108h_mac_in_out * *param*)

This function generates an SHA-256 digest (MAC) of a key, challenge, and other informations.

```
The resulting digest will match with those generated in the Device by MAC opcode.
The TempKey (if used) should be valid (temp_key.valid = 1) before executing this function.
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_mac_in_out.

Returns

status of the operation.

5.5.2.11 uint8_t ecc108h_nonce (struct ecc108h_nonce_in_out * *param*)

This function calculates a 32-byte nonce based on 20-byte input value (NumIn) and 32-byte random number (RandOut).

```
This nonce will match with the nonce generated in the Device by Nonce opcode.
To use this function, Application first executes Nonce command in the Device, with a chosen NumIn.
Nonce opcode Mode parameter must be set to use random nonce (mode 0 or 1).
The Device generates a nonce, stores it in its TempKey, and outputs random number RandOut to host.
This RandOut along with NumIn are passed to nonce calculation function. The function calculates the nonce, an
This function can also be used to fill in the nonce directly to TempKey (pass-through mode). The flags will a
```

Parameters

in,out	*param*	Structure for input/output parameters. Refer to ecc108h_nonce_in_out.

Returns

status of the operation.

5.6 Module 07: Configuration Definitions

Configuration Definitions Common to All Interfaces

- #define CPU_CLOCK_DEVIATION_POSITIVE (1.01)

  maximum CPU clock deviation to higher frequency (crystal etc.) This value is used to establish time related worst case numbers, for example to calculate execution delays and timeouts.
- #define CPU_CLOCK_DEVIATION_NEGATIVE (0.99)

  maximum CPU clock deviation to lower frequency (crystal etc.) This value is used to establish time related worst case numbers, for example to calculate execution delays and timeouts.
- #define ECC108_RETRY_COUNT (1)

  number of command / response retries

Available Definitions for Interfaces

Either un-comment one of the definitions or place it in your project settings. The definitions to choose from are:

- SHA204_SWI_BITBANG (SWI using GPIO peripheral)
- SHA204_SWI_UART (SWI using UART peripheral)
- SHA204_I2C (I^2 C using I^2 C peripheral)

- #define DOXYGEN_DUMMY 0

  Dummy macro that allow Doxygen to parse this group.

5.6.1 Detailed Description

Tune the values of these timing definitions for your system. Always include this file no matter whether you use SWI or I2C. Please refer to the actual file because Doxygen cannot parse nested macros with the same name.

5.6.2 Macro Definition Documentation

5.6.2.1 #define CPU_CLOCK_DEVIATION_POSITIVE (1.01)

maximum CPU clock deviation to higher frequency (crystal etc.) This value is used to establish time related worst case numbers, for example to calculate execution delays and timeouts.

5.6.2.2 #define ECC108_RETRY_COUNT (1)

number of command / response retries

If communication is lost, re-synchronization includes waiting for the longest possible execution time of a command. This adds a ECC108_COMMAND_EXEC_MAX delay to every retry. Every increment of the number of retries increases the time the library is spending in the retry loop by ECC108_COMMAND_EXEC_MAX.

5.7 Module 08: Library Return Codes

Macros

- #define ECC108_SUCCESS ((uint8_t) 0x00)

  Function succeeded.
- #define ECC108_CHECKMAC_FAILED ((uint8_t) 0xD1)

  response status byte indicates CheckMac failure
- #define ECC108_PARSE_ERROR ((uint8_t) 0xD2)

  response status byte indicates parsing error
- #define ECC108_CMD_FAIL ((uint8_t) 0xD3)

  response status byte indicates command execution error
- #define ECC108_STATUS_CRC ((uint8_t) 0xD4)

  response status byte indicates CRC error
- #define ECC108_STATUS_UNKNOWN ((uint8_t) 0xD5)

  response status byte is unknown
- #define ECC108_FUNC_FAIL ((uint8_t) 0xE0)

  Function could not execute due to incorrect condition / state.
- #define ECC108_GEN_FAIL ((uint8_t) 0xE1)

  unspecified error
- #define ECC108_BAD_PARAM ((uint8_t) 0xE2)

  bad argument (out of range, null pointer, etc.)
- #define ECC108_INVALID_ID ((uint8_t) 0xE3)

  invalid device id, id not set
- #define ECC108_INVALID_SIZE ((uint8_t) 0xE4)

  Count value is out of range or greater than buffer size.
- #define ECC108_BAD_CRC ((uint8_t) 0xE5)

  incorrect CRC received
- #define ECC108_RX_FAIL ((uint8_t) 0xE6)

  Timed out while waiting for response. Number of bytes received is $>$ 0.
- #define ECC108_RX_NO_RESPONSE ((uint8_t) 0xE7)

  Not an error while the Command layer is polling for a command response.
- #define ECC108_RESYNC_WITH_WAKEUP ((uint8_t) 0xE8)

  re-synchronization succeeded, but only after generating a Wake-up
- #define ECC108_COMM_FAIL ((uint8_t) 0xF0)

  Communication with device failed. Same as in hardware dependent modules.
- #define ECC108_TIMEOUT ((uint8_t) 0xF1)

  Timed out while waiting for response. Number of bytes received is 0.

5.7.1 Detailed Description

5.8 Module 09: Timers

Macros

- #define TIME_UTILS_US_CALIBRATION

  Fill the inner loop of delay_10us() with these CPU instructions to achieve 10 us per iteration.
- #define TIME_UTILS_LOOP_COUNT ((uint8_t) 28)

  Decrement the inner loop of delay_10us() this many times to achieve 10 us per iteration of the outer loop.
- #define TIME_UTILS_MS_CALIBRATION ((uint8_t) 104)

  The delay_ms function calls delay_10us with this parameter.

Functions

- void delay_10us (uint8_t delay)

  This function delays for a number of tens of microseconds.
- void delay_ms (uint8_t delay)

  This function delays for a number of milliseconds.

5.8.1 Detailed Description

This module implements timers used during communication. They are implemented using loop counters. But if you have hardware timers available, you can implement the functions using them.

5.8.2 Function Documentation

5.8.2.1 void delay_10us (uint8_t *delay*)

This function delays for a number of tens of microseconds.

This function will not time correctly, if one loop iteration plus the time it takes to enter this function takes more than 10 us.

Parameters

in	*delay*	number of 0.01 milliseconds to delay

5.8.2.2 void delay_ms (uint8_t *delay*)

This function delays for a number of milliseconds.

```
You can override this function if you like to do
something else in your system while delaying.
```

Parameters

in	*delay*	number of milliseconds to delay

5.9 Module 18: I2C Interface

Definitions are supplied for various I2C configuration values such as clock, timeouts, and error codes.

5.10 Module 17: SWI Configuration - GPIO

Two definition blocks are supplied:

- port definitions for various Atmel evaluation kits
- loop definitions that result in correct pulse widths for an AVR CPU running at 16 MHz

5.11 Module 16: GPIO Interface

This module implements functions defined in swi_phys.h. This implementation targets an eight-bit AVR CPU.

5.12 Module 14: SWI Configuration - UART

This module contains hardware configuration values for the UART implementation of the single-wire interface. It uses macro definitions from avr/io.h for an AT90USB1287 micro-controller.

5.13 Module 13: UART Interface

This module implements the single-wire interface using a UART micro-controller peripheral.

5.14 Module 15: AVR UART Definitions

This module contains mappings of UART port definitions for the AT90USB1287 micro-controller.

Chapter 6

Data Structure Documentation

6.1 ecc108h_calculate_sha256_in_out Struct Reference

Input/output parameters for function ecc108h_nonce().

```
#include <ecc108_helper.h>
```

Data Fields

- uint32_t length
  [in] Length of input message to be digested.
- uint8_t ∗ message
  [in] Pointer to input message.
- uint8_t ∗ digest
  [out] Pointer to 32-byte SHA256 digest of input message.

6.1.1 Detailed Description

Input/output parameters for function ecc108h_nonce().

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

- ecc108_helper.h

6.2 ecc108h_check_mac_in_out Struct Reference

Input/output parameters for function ecc108h_check_mac().

```
#include <ecc108_helper.h>
```

Data Fields

- uint8_t mode
  [in] Mode parameter used in CheckMac command (Param1).

- uint8_t ∗ password

    [in] Pointer to 32-byte password that will be verified against Key[KeyID] in the Device.
- uint8_t ∗ other_data

    [in] Pointer to 13-byte OtherData that will be used in CheckMac command.
- uint8_t ∗ otp

    [in] Pointer to 11-byte OTP. OTP[0:7] is included in the calculation if Mode bit 5 is one.
- uint8_t ∗ target_key

    [in] Pointer to 32-byte TargetKey that will be copied to TempKey.
- uint8_t ∗ client_resp

    [out] Pointer to 32-byte ClientResp to be used in CheckMac command.
- struct ecc108h_temp_key ∗ temp_key

    [in,out] Pointer to TempKey structure.

6.2.1 Detailed Description

Input/output parameters for function ecc108h_check_mac().

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

- ecc108_helper.h

6.3 ecc108h_decrypt_in_out Struct Reference

Input/output parameters for function ecc108h_decrypt().

```
#include <ecc108_helper.h>
```

Data Fields

- uint8_t ∗ data

    [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.
- struct ecc108h_temp_key ∗ temp_key

    [in,out] Pointer to TempKey structure.

6.3.1 Detailed Description

Input/output parameters for function ecc108h_decrypt().

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

- ecc108_helper.h

6.4 ecc108h_derive_key_in_out Struct Reference

Input/output parameters for function ecc108h_derive_key().

```
#include <ecc108_helper.h>
```

Data Fields

- uint8_t random

  [in] Random parameter used in DeriveKey command (Param1).
- uint16_t target_key_id

  [in] KeyID to be derived, TargetKey parameter used in DeriveKey command (Param2).
- uint8_t * parent_key

  [in] Pointer to 32-byte ParentKey. Set equal to target_key if Roll Key operation is intended.
- uint8_t * target_key

  [out] Pointer to 32-byte TargetKey.
- struct ecc108h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

6.4.1 Detailed Description

Input/output parameters for function ecc108h_derive_key().

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

- ecc108_helper.h

6.5 ecc108h_derive_key_mac_in_out Struct Reference

Input/output parameters for function ecc108h_derive_key_mac().

```
#include <ecc108_helper.h>
```

Data Fields

- uint8_t random

  [in] Random parameter used in DeriveKey command (Param1).
- uint16_t target_key_id

  [in] KeyID to be derived, TargetKey parameter used in DeriveKey command (Param2).
- uint8_t * parent_key

  [in] Pointer to 32-byte ParentKey. ParentKey here is always SlotConfig[TargetKey].WriteKey, regardless whether the operation is Roll or Create.
- uint8_t * mac

  [out] Pointer to 32-byte Mac.

6.5.1 Detailed Description

Input/output parameters for function ecc108h_derive_key_mac().

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

- ecc108_helper.h

6.6 ecc108h_encrypt_in_out Struct Reference

Input/output parameters for function ecc108h_encrypt().

`#include <ecc108_helper.h>`

Data Fields

- uint8_t zone

    [in] Zone parameter used in Write (Param1).

- uint16_t address

    [in] Address parameter used in Write command (Param2).

- uint8_t * data

    [in,out] Pointer to 32-byte data. Input cleartext data, output encrypted data to Write command (Value field).

- uint8_t * mac

    [out] Pointer to 32-byte Mac. Can be set to NULL if input MAC is not required by the Write command (write to OTP, unlocked user zone).

- struct ecc108h_temp_key * temp_key

    [in,out] Pointer to TempKey structure.

6.6.1 Detailed Description

Input/output parameters for function ecc108h_encrypt().

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

- ecc108_helper.h

6.7 ecc108h_gen_dig_in_out Struct Reference

Input/output parameters for function ecc108h_gen_dig().

`#include <ecc108_helper.h>`

Data Fields

- uint8_t zone

    [in] Zone parameter used in GenDig command (Param1).

- uint16_t key_id

    [in] KeyID parameter used in GenDig command (Param2).

- uint8_t * stored_value

    [in] Pointer to 32-byte stored value, can be a data slot, OTP page, configuration zone, or hardware transport key.

- struct ecc108h_temp_key * temp_key

    [in,out] Pointer to TempKey structure.

6.7.1 Detailed Description

Input/output parameters for function ecc108h_gen_dig().

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

- ecc108_helper.h

6.8 ecc108h_hmac_in_out Struct Reference

Input/output parameters for function ecc108h_hmac().

`#include <ecc108_helper.h>`

Data Fields

- uint8_t mode

  [in] Mode parameter used in HMAC command (Param1).
- uint16_t key_id

  [in] KeyID parameter used in HMAC command (Param2).
- uint8_t * key

  [in] Pointer to 32-byte key used to generate HMAC digest.
- uint8_t * otp

  [in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.
- uint8_t * sn

  [in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.
- uint8_t * response

  [out] Pointer to 32-byte SHA-256 HMAC digest.
- struct ecc108h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

6.8.1 Detailed Description

Input/output parameters for function ecc108h_hmac().

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

- ecc108_helper.h

6.9 ecc108h_mac_in_out Struct Reference

Input/output parameters for function ecc108h_mac().

`#include <ecc108_helper.h>`

Data Fields

- uint8_t mode

    [in] Mode parameter used in MAC command (Param1).
- uint16_t key_id

    [in] KeyID parameter used in MAC command (Param2).
- uint8_t ∗ challenge

    [in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.
- uint8_t ∗ key

    [in] Pointer to 32-byte key used to generate MAC digest.
- uint8_t ∗ otp

    [in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.
- uint8_t ∗ sn

    [in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.
- uint8_t ∗ response

    [out] Pointer to 32-byte SHA-256 digest (MAC).
- struct ecc108h_temp_key ∗ temp_key

    [in,out] Pointer to TempKey structure.

6.9.1 Detailed Description

Input/output parameters for function ecc108h_mac().

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

- ecc108_helper.h

6.10 ecc108h_nonce_in_out Struct Reference

Input/output parameters for function ecc108h_nonce().

`#include <ecc108_helper.h>`

Data Fields

- uint8_t mode

    [in] Mode parameter used in Nonce command (Param1).
- uint8_t ∗ num_in

    [in] Pointer to 20-byte NumIn data used in Nonce command.
- uint8_t ∗ rand_out

    [in] Pointer to 32-byte RandOut data from Nonce command.
- struct ecc108h_temp_key ∗ temp_key

    [in,out] Pointer to TempKey structure.

6.10.1 Detailed Description

Input/output parameters for function ecc108h_nonce().

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

- ecc108_helper.h

6.11 ecc108h_temp_key Struct Reference

Structure to hold TempKey fields.

```
#include <ecc108_helper.h>
```

Data Fields

- uint8_t value [32]

  The value of TempKey. Nonce (from nonce command) or Digest (from GenDig command)
- unsigned int key_id:4

  If TempKey was generated by GenDig (see the GenData and CheckFlag bits), these bits indicate which key was used in its computation.
- unsigned int source_flag:1

  The source of the randomness in TempKey: 0=Rand, 1=Input.
- unsigned int gen_data:1

  Indicates if TempKey has been generated by GenDig using Data zone.
- unsigned int check_flag:1

  Not used in the library.
- unsigned int valid:1

  Indicates if the information in TempKey is valid.

6.11.1 Detailed Description

Structure to hold TempKey fields.

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

- ecc108_helper.h

Chapter 7

File Documentation

7.1 avr_compatible.h File Reference

AVR USART Register Compatibility Definitions.

Macros

- #define UCSRA UCSR1A

    UART control and status register A.
- #define UCSRB UCSR1B

    UART control and status register B.
- #define UCSRC UCSR1C

    UART control and status register C.
- #define UDR UDR1

    UART data register.
- #define UBRRL UBRR1L

    UART baud rate register, low byte.
- #define UBRRH UBRR1H

    UART baud rate register, high byte.
- #define RXC RXC1

    UART receive-complete (bit 7, register A)
- #define TXC TXC1

    UART transmit-complete (bit 6, register A)
- #define UDRE UDRE1

    UART data-register-empty (bit 5, register A)
- #define FE FE1

    UART frame-error (bit 4, register A)
- #define DOR DOR1

    UART data-overrun (bit 3, register A)
- #define UPE UPE1

    UART parity-error (bit 2, register A)
- #define U2X U2X1

    UART double-speed (bit 1, register A)

- #define MPCM MPCM1

  UART multi-processor communication (bit 0, register A)
- #define RXCIE RXCIE1

  UART rx complete interrupt enable (bit 7, register B)
- #define TXCIE TXCIE1

  UART tx complete interrupt enable (bit 6, register B)
- #define UDRIE UDRIE1

  UART data register empty interrupt enable (bit 5, register B)
- #define RXEN RXEN1

  UART enable-receiver (bit 4, register B)
- #define TXEN TXEN1

  UART enable-transmitter (bit 3, register B)
- #define UCSZ_2 UCSZ12

  UART msb of number of data bits (bit 2, register B)
- #define RXB8 RXB81

  UART receive ninth data bit (bit 1, register B)
- #define TXB8 TXB81

  UART send ninth data bit (bit 0, register B)

7.1.1 Detailed Description

AVR USART Register Compatibility Definitions.

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

ATECC108 Library License:

7.2 bitbang_config.h File Reference

Definitions for Hardware Dependent Part of ATSHA204 Physical Layer Using GPIO for Communication.

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

Macros

- #define swi_enable_interrupts sei

  enable interrupts
- #define swi_disable_interrupts cli

  disable interrupts
- #define SIG2_BIT (2)

  bit position of port register for second device
- #define CLIENT_ID (0)

  identifier for client
- #define PORT_DDR (DDRD)

  direction register for device id 0
- #define PORT_OUT (PORTD)

  output port register for device id 0
- #define PORT_IN (PIND)

  input port register for device id 0
- #define SIG1_BIT (6)

  bit position of port register for first device
- #define HOST_ID (1)

  identifier for host
- #define DEBUG_LOW

  Debug pin that indicates pulse edge detection. This is only enabled if compilation switch DEBUG_BITBANG is used. To debug timing, disable host power (H1 and H2 on AT88CK109BK8 daughter board) and connect logic analyzer or storage oscilloscope to the H2 pin that is closer to the H1 header. The logic analyzer from Saleae (www.saleae.com) comes with a protocol analyzer for this Atmel SWI protocol.

Macros for Bit-Banged SWI Timing

Times to drive bits at 230.4 kbps. For a CPU clock of 16 MHz on an 8-bit AVR, the delay loops used take about 580 ns per iteration. Another 800 ns are needed to access the port.

- #define BIT_DELAY_1 {volatile uint8_t delay = 6; while (delay--);}

  delay macro for width of one pulse (start pulse or zero pulse, in ns)
- #define BIT_DELAY_5 {volatile uint8_t delay = 44; while (delay--);}

  time to keep pin high for five pulses plus stop bit (used to bit-bang CryptoAuth 'zero' bit, in ns)
- #define BIT_DELAY_7 {volatile uint8_t delay = 59; while (delay--);}

  time to keep pin high for seven bits plus stop bit (used to bit-bang CryptoAuth 'one' bit)
- #define RX_TX_DELAY {volatile uint8_t delay = 25; while (delay--);}

  turn around time when switching from receive to transmit
- #define START_PULSE_TIME_OUT (255)

  This value is decremented while waiting for the falling edge of a start pulse.
- #define ZERO_PULSE_TIME_OUT (26)

  This value is decremented while waiting for the falling edge of a zero pulse.

7.2.1 Detailed Description

Definitions for Hardware Dependent Part of ATSHA204 Physical Layer Using GPIO for Communication.

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

7.3 bitbang_phys.c File Reference

Functions of Hardware Dependent Part of ATSHA204 Physical Layer Using GPIO For Communication.

```
#include <stdint.h>
#include "swi_phys.h"
#include "bitbang_config.h"
```

Functions

- void swi_set_device_id (uint8_t id)

  This GPIO function sets the signal pin. Communication functions will use this signal pin.
- void swi_enable (void)

  This GPIO function sets the bit position of the signal pin to its default.
- void swi_set_signal_pin (uint8_t is_high)

  This GPIO function sets the signal pin low or high.
- uint8_t swi_send_bytes (uint8_t count, uint8_t ∗buffer)

  This GPIO function sends bytes to an SWI device.
- uint8_t swi_send_byte (uint8_t value)

  This GPIO function sends one byte to an SWI device.
- uint8_t swi_receive_bytes (uint8_t count, uint8_t ∗buffer)

  This GPIO function receives bytes from an SWI device.

7.3.1 Detailed Description

Functions of Hardware Dependent Part of ATSHA204 Physical Layer Using GPIO For Communication.

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

7.3.2 Function Documentation

7.3.2.1 uint8_t swi_receive_bytes (uint8_t *count*, uint8_t * *buffer*)

This GPIO function receives bytes from an SWI device.

Parameters

in	*count*	number of bytes to receive
out	*buffer*	pointer to rx buffer

Returns

status of the operation

7.3.2.2 uint8_t swi_send_byte (uint8_t *value*)

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

7.3.2.3 uint8_t swi_send_bytes (uint8_t *count*, uint8_t ∗ *buffer*)

This GPIO function sends bytes to an SWI device.

Parameters

in	*count*	number of bytes to send
in	*buffer*	pointer to tx buffer

Returns

status of the operation

7.3.2.4 void swi_set_device_id (uint8_t *id*)

This GPIO function sets the signal pin. Communication functions will use this signal pin.

Parameters

in	*id*	client if zero, otherwise host

Returns

status of the operation

7.3.2.5 void swi_set_signal_pin (uint8_t *is_high*)

This GPIO function sets the signal pin low or high.

Parameters

in	*is_high*	0: set signal low, otherwise high.

7.4 ecc108_comm.c File Reference

Communication Layer of ECC108 Library.

```
#include "ecc108_comm.h"
#include "timer_utilities.h"
#include "ecc108_lib_return_codes.h"
```

Functions

- void ecc108c_calculate_crc (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates CRC.
- uint8_t ecc108c_check_crc (uint8_t ∗response)

  This function checks the consistency of a response.
- uint8_t ecc108c_wakeup (uint8_t ∗response)

  This function wakes up a ECC108 device and receives a response.
- uint8_t ecc108c_resync (uint8_t size, uint8_t ∗response)

  This function re-synchronizes communication.
- uint8_t ecc108c_send_and_receive (uint8_t ∗tx_buffer, uint8_t rx_size, uint8_t ∗rx_buffer, uint8_t execution_-delay, uint8_t execution_timeout)

  This function runs a communication sequence: Append CRC to tx buffer, send command, delay, and verify response after receiving it.

7.4.1 Detailed Description

Communication Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.4.2 Function Documentation

7.4.2.1 uint8_t ecc108c_check_crc (uint8_t ∗ *response*)

This function checks the consistency of a response.

Parameters

in	*response*	pointer to response

Returns

status of the consistency check

7.4.2.2 uint8_t ecc108c_resync (uint8_t *size*, uint8_t ∗ *response*)

This function re-synchronizes communication.

Be aware that succeeding only after waking up the device could mean that it had gone to sleep and lost its TempKey in the process.

Re-synchronizing communication is done in a maximum of three steps:

1. Try to re-synchronize without sending a Wake token. This step is implemented in the Physical layer.
2. If the first step did not succeed send a Wake token.
3. Try to read the Wake response.

Parameters

in	*size*	size of response buffer
out	*response*	pointer to Wake-up response buffer

Returns

status of the operation

7.5 ecc108_comm.h File Reference

Definitions and Prototypes for Communication Layer of ECC108 Library.

```
#include <stddef.h>
#include "ecc108_physical.h"
```

Macros

- #define ECC108_COMMAND_EXEC_MAX ((uint8_t) (120.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

    maximum command delay
- #define ECC108_CMD_SIZE_MIN ((uint8_t) 7)

    minimum number of bytes in command (from count byte to second CRC byte)
- #define ECC108_CMD_SIZE_MAX ((uint8_t) 4 ∗ 36 + 7)

    maximum size of command packet (Verify)
- #define ECC108_CRC_SIZE ((uint8_t) 2)

    number of CRC bytes
- #define ECC108_BUFFER_POS_STATUS (1)

    buffer index of status byte in status response
- #define ECC108_BUFFER_POS_DATA (1)

    buffer index of first data byte in data response
- #define ECC108_STATUS_BYTE_WAKEUP ((uint8_t) 0x11)

    status byte after wake-up
- #define ECC108_STATUS_BYTE_PARSE ((uint8_t) 0x03)

    command parse error
- #define ECC108_STATUS_BYTE_EXEC ((uint8_t) 0x0F)

    command execution error
- #define ECC108_STATUS_BYTE_COMM ((uint8_t) 0xFF)

    communication error

Functions

- void ecc108c_calculate_crc (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

    This function calculates CRC.
- uint8_t ecc108c_wakeup (uint8_t ∗response)

    This function wakes up a ECC108 device and receives a response.
- uint8_t ecc108c_send_and_receive (uint8_t ∗tx_buffer, uint8_t rx_size, uint8_t ∗rx_buffer, uint8_t execution_-delay, uint8_t execution_timeout)

    This function runs a communication sequence: Append CRC to tx buffer, send command, delay, and verify response after receiving it.

7.5.1 Detailed Description

Definitions and Prototypes for Communication Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

7.6 ecc108_comm_marshaling.c File Reference

Command Marshaling Layer of ECC108 Library.

```
#include <string.h>
#include "ecc108_lib_return_codes.h"
#include "ecc108_comm_marshaling.h"
```

Functions

- uint8_t ecc108m_execute (uint8_t op_code, uint8_t param1, uint16_t param2, uint8_t datalen1, uint8_t *data1, uint8_t datalen2, uint8_t *data2, uint8_t datalen3, uint8_t *data3, uint8_t tx_size, uint8_t *tx_buffer, uint8_t rx_size, uint8_t *rx_buffer)

  This function creates a command packet, sends it, and receives its response.
- uint8_t ecc108m_check_mac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t key_id, uint8_t *client_challenge, uint8_t *client_response, uint8_t *other_data)

  This function sends a CheckMAC command to the device.
- uint8_t ecc108m_derive_key (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t random, uint8_t target_key, uint8_t *mac)

This function sends a DeriveKey command to the device.

- uint8_t ecc108m_info (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t gpio_state)

  This function sends an Info command to the device.
- uint8_t ecc108m_gen_dig (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint8_t key_id, uint8_t *other_data)

  This function sends a GenDig command to the device.
- uint8_t ecc108m_hmac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint16_t key_id)

  This function sends an HMAC command to the device.
- uint8_t ecc108m_lock (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t summary)

  This function sends a Lock command to the device.
- uint8_t ecc108m_mac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint16_t key_id, uint8_t *challenge)

  This function sends a MAC command to the device.
- uint8_t ecc108m_nonce (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t *numin)

  This function sends a Nonce command to the device.
- uint8_t ecc108m_pause (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t selector)

  This function sends a Pause command to the device.
- uint8_t ecc108m_random (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode)

  This function sends a Random command to the device.
- uint8_t ecc108m_read (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t address)

  This function sends a Read command to the device.
- uint8_t ecc108m_update_extra (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t new_value)

  This function sends an UpdateExtra command to the device.
- uint8_t ecc108m_write (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t address, uint8_t *new_value, uint8_t *mac)

  This function sends a Write command to the device.

7.6.1 Detailed Description

Command Marshaling Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.7 ecc108_comm_marshaling.h File Reference

Definitions and Prototypes for Command Marshaling Layer of ECC108 Library.

```
#include "ecc108_comm.h"
```

Macros

Codes for ATECC108 Commands

- #define ECC108_CHECKMAC ((uint8_t) 0x28)

  CheckMac command op-code.
- #define ECC108_DERIVE_KEY ((uint8_t) 0x1C)

  DeriveKey command op-code.
- #define ECC108_INFO ((uint8_t) 0x30)

  DevRev command op-code.
- #define ECC108_GENDIG ((uint8_t) 0x15)

  GenDig command op-code.
- #define ECC108_HMAC ((uint8_t) 0x11)

  HMAC command op-code.
- #define ECC108_LOCK ((uint8_t) 0x17)

  Lock command op-code.
- #define ECC108_MAC ((uint8_t) 0x08)

  MAC command op-code.
- #define ECC108_NONCE ((uint8_t) 0x16)

  Nonce command op-code.
- #define ECC108_PAUSE ((uint8_t) 0x01)

  Pause command op-code.
- #define ECC108_RANDOM ((uint8_t) 0x1B)

  Random command op-code.
- #define ECC108_READ ((uint8_t) 0x02)

Read command op-code.

- #define ECC108_UPDATE_EXTRA ((uint8_t) 0x20)

  UpdateExtra command op-code.
- #define ECC108_WRITE ((uint8_t) 0x12)

  Write command op-code.

Definitions of Data and Packet Sizes

- #define ECC108_RSP_SIZE_VAL ((uint8_t) 7)

  size of response packet containing four bytes of data
- #define ECC108_KEY_SIZE (32)

  size of key

Definitions for Command Parameter Ranges

- #define ECC108_KEY_ID_MAX ((uint8_t) 15)

  maximum value for key id
- #define ECC108_OTP_BLOCK_MAX ((uint8_t) 1)

  maximum value for OTP block

Definitions for Indexes Common to All Commands

- #define ECC108_COUNT_IDX (0)

  command packet index for count
- #define ECC108_OPCODE_IDX (1)

  command packet index for op-code
- #define ECC108_PARAM1_IDX (2)

  command packet index for first parameter
- #define ECC108_PARAM2_IDX (3)

  command packet index for second parameter
- #define ECC108_DATA_IDX (5)

  command packet index for second parameter

Definitions for Zone and Address Parameters

- #define ECC108_ZONE_CONFIG ((uint8_t) 0x00)

  Configuration zone.
- #define ECC108_ZONE_OTP ((uint8_t) 0x01)

  OTP (One Time Programming) zone.
- #define ECC108_ZONE_DATA ((uint8_t) 0x02)

  Data zone.
- #define ECC108_ZONE_MASK ((uint8_t) 0x03)

  Zone mask.
- #define ECC108_ZONE_COUNT_FLAG ((uint8_t) 0x80)

  Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.
- #define ECC108_ZONE_ACCESS_4 ((uint8_t) 4)

  Read or write 4 bytes.
- #define ECC108_ZONE_ACCESS_32 ((uint8_t) 32)

  Read or write 32 bytes.
- #define ECC108_ADDRESS_MASK_CONFIG (0x001F)

  Address bits 5 to 7 are 0 for Configuration zone.
- #define ECC108_ADDRESS_MASK_OTP (0x000F)

Address bits 4 to 7 are 0 for OTP zone.
- #define ECC108_ADDRESS_MASK (0x007F)

  Address bit 7 to 15 are always 0.

Definitions for the CheckMac Command

- #define CHECKMAC_MODE_IDX ECC108_PARAM1_IDX

  CheckMAC command index for mode.
- #define CHECKMAC_KEYID_IDX ECC108_PARAM2_IDX

  CheckMAC command index for key identifier.
- #define CHECKMAC_CLIENT_CHALLENGE_IDX ECC108_DATA_IDX

  CheckMAC command index for client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_IDX (37)

  CheckMAC command index for client response.
- #define CHECKMAC_DATA_IDX (69)

  CheckMAC command index for other data.
- #define CHECKMAC_COUNT (84)

  CheckMAC command packet size.
- #define CHECKMAC_MODE_CHALLENGE ((uint8_t) 0x00)

  CheckMAC mode 0: first SHA block from key id.
- #define CHECKMAC_MODE_BLOCK2_TEMPKEY ((uint8_t) 0x01)

  CheckMAC mode bit 0: second SHA block from TempKey.
- #define CHECKMAC_MODE_BLOCK1_TEMPKEY ((uint8_t) 0x02)

  CheckMAC mode bit 1: first SHA block from TempKey.
- #define CHECKMAC_MODE_SOURCE_FLAG_MATCH ((uint8_t) 0x04)

  CheckMAC mode bit 2: match TempKey.SourceFlag.
- #define CHECKMAC_MODE_INCLUDE_OTP_64 ((uint8_t) 0x20)

  CheckMAC mode bit 5: include first 64 OTP bits.
- #define CHECKMAC_MODE_MASK ((uint8_t) 0x27)

  CheckMAC mode bits 3, 4, 6, and 7 are 0.
- #define CHECKMAC_CLIENT_CHALLENGE_SIZE (32)

  CheckMAC size of client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_SIZE (32)

  CheckMAC size of client response.
- #define CHECKMAC_OTHER_DATA_SIZE (13)

  CheckMAC size of "other data".
- #define CHECKMAC_CLIENT_COMMAND_SIZE (4)

  CheckMAC size of client command header size inside "other data".

Definitions for the DeriveKey Command

- #define DERIVE_KEY_RANDOM_IDX ECC108_PARAM1_IDX

  DeriveKey command index for random bit.
- #define DERIVE_KEY_TARGETKEY_IDX ECC108_PARAM2_IDX

  DeriveKey command index for target slot.
- #define DERIVE_KEY_MAC_IDX ECC108_DATA_IDX

  DeriveKey command index for optional MAC.
- #define DERIVE_KEY_COUNT_SMALL ECC108_CMD_SIZE_MIN

  DeriveKey command packet size without MAC.
- #define DERIVE_KEY_COUNT_LARGE (39)

  DeriveKey command packet size with MAC.
- #define DERIVE_KEY_RANDOM_FLAG ((uint8_t) 4)

DeriveKey 1. parameter; has to match TempKey.SourceFlag.
- #define DERIVE_KEY_MAC_SIZE (32)

    DeriveKey MAC size.

Definitions for the GenDig Command

- #define GENDIG_ZONE_IDX ECC108_PARAM1_IDX

    GenDig command index for zone.
- #define GENDIG_KEYID_IDX ECC108_PARAM2_IDX

    GenDig command index for key id.
- #define GENDIG_DATA_IDX ECC108_DATA_IDX

    GenDig command index for optional data.
- #define GENDIG_COUNT ECC108_CMD_SIZE_MIN

    GenDig command packet size without "other data".
- #define GENDIG_COUNT_DATA (11)

    GenDig command packet size with "other data".
- #define GENDIG_OTHER_DATA_SIZE (4)

    GenDig size of "other data".
- #define GENDIG_ZONE_CONFIG ((uint8_t) 0)

    GenDig zone id config.
- #define GENDIG_ZONE_OTP ((uint8_t) 1)

    GenDig zone id OTP.
- #define GENDIG_ZONE_DATA ((uint8_t) 2)

    GenDig zone id data.

Definitions for the HMAC Command

- #define HMAC_MODE_IDX ECC108_PARAM1_IDX

    HMAC command index for mode.
- #define HMAC_KEYID_IDX ECC108_PARAM2_IDX

    HMAC command index for key id.
- #define HMAC_COUNT ECC108_CMD_SIZE_MIN

    HMAC command packet size.
- #define HMAC_MODE_MASK ((uint8_t) 0x74)

    HMAC mode bits 0, 1, 3, and 7 are 0.

Definitions for the Info Command

- #define INFO_PARAM1_IDX ECC108_PARAM1_IDX

    Info command index for 1. parameter.
- #define INFO_PARAM2_IDX ECC108_PARAM2_IDX

    Info command index for 2. parameter.
- #define INFO_COUNT ECC108_CMD_SIZE_MIN

    Info command packet size.
- #define INFO_MODE_REVISION ((uint8_t) 0x00)

    Info mode Revision.
- #define INFO_MODE_KEY_VALID ((uint8_t) 0x01)

    Info mode KeyValid.
- #define INFO_MODE_STATE ((uint8_t) 0x02)

    Info mode State.
- #define INFO_MODE_GPIO ((uint8_t) 0x03)

    Info mode GPIO.

- #define INFO_MODE_MAX ((uint8_t) 0x03)
  Info mode maximum value.
- #define INFO_NO_STATE ((uint8_t) 0x00)
  Info mode is not the state mode.
- #define INFO_OUTPUT_STATE_MASK ((uint8_t) 0x01)
  Info output state mask.
- #define INFO_DRIVER_STATE_MASK ((uint8_t) 0x02)
  Info driver state mask.
- #define INFO_PARAM2_MAX ((uint8_t) 0x03)
  Info param2 (state) maximum value.

Definitions for the Lock Command

- #define LOCK_ZONE_IDX ECC108_PARAM1_IDX
  Lock command index for zone.
- #define LOCK_SUMMARY_IDX ECC108_PARAM2_IDX
  Lock command index for summary.
- #define LOCK_COUNT ECC108_CMD_SIZE_MIN
  Lock command packet size.
- #define LOCK_ZONE_NO_CONFIG ((uint8_t) 0x01)
  Lock zone is OTP or Data.
- #define LOCK_ZONE_NO_CRC ((uint8_t) 0x80)
  Lock command: Ignore summary.
- #define LOCK_ZONE_MASK (0x81)
  Lock parameter 1 bits 2 to 6 are 0.

Definitions for the MAC Command

- #define MAC_MODE_IDX ECC108_PARAM1_IDX
  MAC command index for mode.
- #define MAC_KEYID_IDX ECC108_PARAM2_IDX
  MAC command index for key id.
- #define MAC_CHALLENGE_IDX ECC108_DATA_IDX
  MAC command index for optional challenge.
- #define MAC_COUNT_SHORT ECC108_CMD_SIZE_MIN
  MAC command packet size without challenge.
- #define MAC_COUNT_LONG (39)
  MAC command packet size with challenge.
- #define MAC_MODE_CHALLENGE ((uint8_t) 0x00)
  MAC mode 0: first SHA block from data slot.
- #define MAC_MODE_BLOCK2_TEMPKEY ((uint8_t) 0x01)
  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_BLOCK1_TEMPKEY ((uint8_t) 0x02)
  MAC mode bit 1: first SHA block from TempKey.
- #define MAC_MODE_SOURCE_FLAG_MATCH ((uint8_t) 0x04)
  MAC mode bit 2: match TempKey.SourceFlag.
- #define MAC_MODE_PASSTHROUGH ((uint8_t) 0x07)
  MAC mode bit 0-2: pass-through mode.
- #define MAC_MODE_INCLUDE_OTP_88 ((uint8_t) 0x10)
  MAC mode bit 4: include first 88 OTP bits.
- #define MAC_MODE_INCLUDE_OTP_64 ((uint8_t) 0x20)
  MAC mode bit 5: include first 64 OTP bits.

- #define MAC_MODE_INCLUDE_SN ((uint8_t) 0x40)

  MAC mode bit 6: include serial number.
- #define MAC_CHALLENGE_SIZE (32)

  MAC size of challenge.
- #define MAC_MODE_MASK ((uint8_t) 0x77)

  MAC mode bits 3 and 7 are 0.

Definitions for the Nonce Command

- #define NONCE_MODE_IDX ECC108_PARAM1_IDX

  Nonce command index for mode.
- #define NONCE_PARAM2_IDX ECC108_PARAM2_IDX

  Nonce command index for 2. parameter.
- #define NONCE_INPUT_IDX ECC108_DATA_IDX

  Nonce command index for input data.
- #define NONCE_COUNT_SHORT (27)

  Nonce command packet size for 20 bytes of data.
- #define NONCE_COUNT_LONG (39)

  Nonce command packet size for 32 bytes of data.
- #define NONCE_MODE_MASK ((uint8_t) 3)

  Nonce mode bits 2 to 7 are 0.
- #define NONCE_MODE_SEED_UPDATE ((uint8_t) 0x00)

  Nonce mode: update seed.
- #define NONCE_MODE_NO_SEED_UPDATE ((uint8_t) 0x01)

  Nonce mode: do not update seed.
- #define NONCE_MODE_INVALID ((uint8_t) 0x02)

  Nonce mode 2 is invalid.
- #define NONCE_MODE_PASSTHROUGH ((uint8_t) 0x03)

  Nonce mode: pass-through.
- #define NONCE_NUMIN_SIZE (20)

  Nonce data length.
- #define NONCE_NUMIN_SIZE_PASSTHROUGH (32)

  Nonce data length in pass-through mode (mode = 3)

Definitions for the Pause Command

- #define PAUSE_SELECT_IDX ECC108_PARAM1_IDX

  Pause command index for Selector.
- #define PAUSE_PARAM2_IDX ECC108_PARAM2_IDX

  Pause command index for 2. parameter.
- #define PAUSE_COUNT ECC108_CMD_SIZE_MIN

  Pause command packet size.

Definitions for the Random Command

- #define RANDOM_MODE_IDX ECC108_PARAM1_IDX

  Random command index for mode.
- #define RANDOM_PARAM2_IDX ECC108_PARAM2_IDX

  Random command index for 2. parameter.
- #define RANDOM_COUNT ECC108_CMD_SIZE_MIN

  Random command packet size.
- #define RANDOM_SEED_UPDATE ((uint8_t) 0x00)

Random mode for automatic seed update.
- #define RANDOM_NO_SEED_UPDATE ((uint8_t) 0x01)

    Random mode for no seed update.

Definitions for the Read Command

- #define READ_ZONE_IDX ECC108_PARAM1_IDX

    Read command index for zone.
- #define READ_ADDR_IDX ECC108_PARAM2_IDX

    Read command index for address.
- #define READ_COUNT ECC108_CMD_SIZE_MIN

    Read command packet size.
- #define READ_ZONE_MASK ((uint8_t) 0x83)

    Read zone bits 2 to 6 are 0.
- #define READ_ZONE_MODE_32_BYTES ((uint8_t) 0x80)

    Read mode: 32 bytes.

Definitions for the UpdateExtra Command

- #define UPDATE_MODE_IDX ECC108_PARAM1_IDX

    UpdateExtra command index for mode.
- #define UPDATE_VALUE_IDX ECC108_PARAM2_IDX

    UpdateExtra command index for new value.
- #define UPDATE_COUNT ECC108_CMD_SIZE_MIN

    UpdateExtra command packet size.
- #define UPDATE_CONFIG_BYTE_86 ((uint8_t) 0x01)

    UpdateExtra mode: update Config byte 86.

Definitions for the Write Command

- #define WRITE_ZONE_IDX ECC108_PARAM1_IDX

    Write command index for zone.
- #define WRITE_ADDR_IDX ECC108_PARAM2_IDX

    Write command index for address.
- #define WRITE_VALUE_IDX ECC108_DATA_IDX

    Write command index for data.
- #define WRITE_MAC_VS_IDX (9)

    Write command index for MAC following short data.
- #define WRITE_MAC_VL_IDX (37)

    Write command index for MAC following long data.
- #define WRITE_COUNT_SHORT (11)

    Write command packet size with short data and no MAC.
- #define WRITE_COUNT_LONG (39)

    Write command packet size with long data and no MAC.
- #define WRITE_COUNT_SHORT_MAC (43)

    Write command packet size with short data and MAC.
- #define WRITE_COUNT_LONG_MAC (71)

    Write command packet size with long data and MAC.
- #define WRITE_MAC_SIZE (32)

    Write MAC size.
- #define WRITE_ZONE_MASK ((uint8_t) 0xC3)

    Write zone bits 2 to 5 are 0.

- #define WRITE_ZONE_WITH_MAC ((uint8_t) 0x40)

  Write zone bit 6: write encrypted with MAC.

Response Size Definitions

- #define CHECKMAC_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of DeriveKey command
- #define DERIVE_KEY_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of DeriveKey command
- #define INFO_RSP_SIZE ECC108_RSP_SIZE_VAL

  response size of Info command returns 4 bytes
- #define GENDIG_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of GenDig command
- #define HMAC_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of HMAC command
- #define LOCK_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of Lock command
- #define MAC_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of MAC command
- #define NONCE_RSP_SIZE_SHORT ECC108_RSP_SIZE_MIN

  response size of Nonce command with mode[0:1] = 3
- #define NONCE_RSP_SIZE_LONG ECC108_RSP_SIZE_MAX

  response size of Nonce command
- #define PAUSE_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of Pause command
- #define RANDOM_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of Random command
- #define READ_4_RSP_SIZE ECC108_RSP_SIZE_VAL

  response size of Read command when reading 4 bytes
- #define READ_32_RSP_SIZE ECC108_RSP_SIZE_MAX

  response size of Read command when reading 32 bytes
- #define UPDATE_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of UpdateExtra command
- #define WRITE_RSP_SIZE ECC108_RSP_SIZE_MIN

  response size of Write command

Definitions of Typical Command Execution Times

The library starts polling the device for a response after these delays.

- #define CHECKMAC_DELAY ((uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  CheckMAC typical command delay.
- #define DERIVE_KEY_DELAY ((uint8_t) (14.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  DeriveKey typical command delay.
- #define INFO_DELAY ((uint8_t) (1))

  DevRev typical command delay.
- #define GENDIG_DELAY ((uint8_t) (11.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  GenDig typical command delay.
- #define HMAC_DELAY ((uint8_t) (27.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  HMAC typical command delay.
- #define LOCK_DELAY ((uint8_t) (5.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Lock typical command delay.
- #define MAC_DELAY ((uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

MAC typical command delay.
- #define NONCE_DELAY ((uint8_t) (22.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Nonce typical command delay.
- #define PAUSE_DELAY ((uint8_t) (1))

  Pause typical command delay.
- #define RANDOM_DELAY ((uint8_t) (11.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Random typical command delay.
- #define READ_DELAY ((uint8_t) (1))

  Read typical command delay.
- #define UPDATE_DELAY ((uint8_t) (8.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  UpdateExtra typical command delay.
- #define WRITE_DELAY ((uint8_t) (4.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  Write typical command delay.

Definitions of Maximum Command Execution Times

- #define CHECKMAC_EXEC_MAX ((uint8_t) (38.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  CheckMAC maximum execution time.
- #define DERIVE_KEY_EXEC_MAX ((uint8_t) (62.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  DeriveKey maximum execution time.
- #define INFO_EXEC_MAX ((uint8_t) (2.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  DevRev maximum execution time.
- #define GENDIG_EXEC_MAX ((uint8_t) (43.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  GenDig maximum execution time.
- #define HMAC_EXEC_MAX ((uint8_t) (69.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  HMAC maximum execution time.
- #define LOCK_EXEC_MAX ((uint8_t) (24.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Lock maximum execution time.
- #define MAC_EXEC_MAX ((uint8_t) (35.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  MAC maximum execution time.
- #define NONCE_EXEC_MAX ((uint8_t) (60.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Nonce maximum execution time.
- #define PAUSE_EXEC_MAX ((uint8_t) (2.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Pause maximum execution time.
- #define RANDOM_EXEC_MAX ((uint8_t) (50.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Random maximum execution time.
- #define READ_EXEC_MAX ((uint8_t) (4.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Read maximum execution time.
- #define UPDATE_EXEC_MAX ((uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  UpdateExtra maximum execution time.
- #define WRITE_EXEC_MAX ((uint8_t) (42.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  Write maximum execution time.

Functions

- uint8_t ecc108m_check_mac (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t mode, uint8_t key_id, uint8_t ∗client_challenge, uint8_t ∗client_response, uint8_t ∗other_data)

  This function sends a CheckMAC command to the device.
- uint8_t ecc108m_derive_key (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t random, uint8_t target_key, uint8_t ∗mac)

  This function sends a DeriveKey command to the device.

- uint8_t ecc108m_info (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t gpio_state)

    This function sends an Info command to the device.
- uint8_t ecc108m_gen_dig (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint8_t key_id, uint8_t *other_data)

    This function sends a GenDig command to the device.
- uint8_t ecc108m_hmac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint16_t key_id)

    This function sends an HMAC command to the device.
- uint8_t ecc108m_lock (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t summary)

    This function sends a Lock command to the device.
- uint8_t ecc108m_mac (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint16_t key_id, uint8_t *challenge)

    This function sends a MAC command to the device.
- uint8_t ecc108m_nonce (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t *numin)

    This function sends a Nonce command to the device.
- uint8_t ecc108m_pause (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t selector)

    This function sends a Pause command to the device.
- uint8_t ecc108m_random (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode)

    This function sends a Random command to the device.
- uint8_t ecc108m_read (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t address)

    This function sends a Read command to the device.
- uint8_t ecc108m_update_extra (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t mode, uint8_t new_value)

    This function sends an UpdateExtra command to the device.
- uint8_t ecc108m_write (uint8_t *tx_buffer, uint8_t *rx_buffer, uint8_t zone, uint16_t address, uint8_t *value, uint8_t *mac)

    This function sends a Write command to the device.
- uint8_t ecc108m_execute (uint8_t op_code, uint8_t param1, uint16_t param2, uint8_t datalen1, uint8_t *data1, uint8_t datalen2, uint8_t *data2, uint8_t datalen3, uint8_t *data3, uint8_t tx_size, uint8_t *tx_buffer, uint8_t rx_size, uint8_t *rx_buffer)

    This function creates a command packet, sends it, and receives its response.

7.7.1 Detailed Description

Definitions and Prototypes for Command Marshaling Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

Byte #	Name	Meaning
0	Count	Number of bytes in the packet, includes the count byte, body and the checksum
1	Ordinal	Command Opcode (Ordinal)
2 to n	Parameters	Parameters for specific command
n+1 to n+2	Checksum	Checksum of the command packet

Table 7.1: Command Packet Structure

7.8 ecc108_config.h File Reference

Definitions for Configurable Values of the ECC108 Library.

```
#include <stddef.h>
```

Macros

Configuration Definitions Common to All Interfaces

- #define CPU_CLOCK_DEVIATION_POSITIVE (1.01)

  maximum CPU clock deviation to higher frequency (crystal etc.) This value is used to establish time related worst case numbers, for example to calculate execution delays and timeouts.
- #define CPU_CLOCK_DEVIATION_NEGATIVE (0.99)

  maximum CPU clock deviation to lower frequency (crystal etc.) This value is used to establish time related worst case numbers, for example to calculate execution delays and timeouts.
- #define ECC108_RETRY_COUNT (1)

  number of command / response retries

Available Definitions for Interfaces

Either un-comment one of the definitions or place it in your project settings. The definitions to choose from are:

- *SHA204_SWI_BITBANG (SWI using GPIO peripheral)*
- *SHA204_SWI_UART (SWI using UART peripheral)*
- *SHA204_I2C (I^2 C using I^2 C peripheral)*

- #define DOXYGEN_DUMMY 0

  Dummy macro that allow Doxygen to parse this group.

7.8.1 Detailed Description

Definitions for Configurable Values of the ECC108 Library.

```
This file contains several library configuration sections
for the three interfaces the library supports
(SWI using GPIO or UART, and I2C) and one that is common
to all interfaces.
```

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.9 ecc108_example_main.c File Reference

Application Examples that Use the ECC108 Library.

```
#include <stddef.h>
#include "ecc108_examples.h"
#include "ecc108_physical.h"
```

Functions

- int main (void)

  This application calls one example function that can be selected with a compilation switch. The example functions for ECC108_EXAMPLE_SEND_INFO_COMMAND, ECC108_EXAMPLE_CHECKMAC_DEVICE and ECC108_EXAMPLE_-CHECKMAC_FIRMWARE do not return since they are running in an endless loop.

7.9.1 Detailed Description

Application Examples that Use the ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.9.2 Function Documentation

7.9.2.1 int main (void)

This application calls one example function that can be selected with a compilation switch. The example functions for ECC108_EXAMPLE_SEND_INFO_COMMAND, ECC108_EXAMPLE_CHECKMAC_DEVICE and ECC108_EXAMPLE_CHECKMAC_FIRMWARE do not return since they are running in an endless loop.

Returns

exit status of application

7.10 ecc108_examples.c File Reference

Application examples that Use the ECC108 Library.

```
#include <string.h>
#include <stdbool.h>
#include "ecc108_lib_return_codes.h"
#include "ecc108_comm_marshaling.h"
#include "ecc108_helper.h"
#include "ecc108_examples.h"
#include "timer_utilities.h"
#include "swi_phys.h"
```

Macros

- #define ecc108e_wakeup_sleep() {ecc108p_wakeup(); ecc108p_sleep();}

  This macro brings a device from Idle mode into Sleep mode by waking it up and sending a Sleep flag.

Functions

- void ecc108e_sleep ()

This function wraps ecc108p_sleep(). It puts both devices to sleep if two devices (client and host) are used. This function is also called when a Wakeup did not succeed. This would not make sense if a device did not wakeup and it is the only device on SDA, but if there are two devices (client and host) that share SDA, the device that is not selected might have woken up.

- uint8_t ecc108e_wakeup_device (uint8_t device_id)

  This function wakes up two I2C devices and puts one back to sleep, effectively waking up only one device among two that share SDA.
- uint8_t ecc108e_check_response_status (uint8_t ret_code, uint8_t ∗response)

  This function checks the response status byte and puts the device to sleep if there was an error.
- uint8_t ecc108e_send_info_command (void)

  This function is a simple example for how to use the library. It wakes up the device, sends a DevRev command, receives its response, and puts the device to sleep. It uses a total of four library functions from all three layers, physical, communication, and command marshaling layer.
- uint8_t ecc108e_configure (uint8_t parent_id)

  This function configures client and host device for the ChildKey example.
- uint8_t ecc108e_checkmac_device (void)

  This function serves as an example for the ECC108 Mac and CheckMac commands.
- uint8_t ecc108e_checkmac_firmware (void)

  This function serves as an example for the ECC108 Nonce, GenDig, and Mac commands.
- uint8_t ecc108e_derive_key (void)

  This function serves as an example for the ECC108 Nonce, DeriveKey, and Mac commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.
- uint8_t ecc108e_change_i2c_address (void)

  This function changes the I2C address of a device. Running it will access the device with I2C address ECC108_CLIENT_ADDRESS and change it to ECC108_HOST_ADDRESS as long as the configuration zone is not locked (byte under address 87 = 0x55). Be aware that bit 3 of the I2C address is also used as a TTL enable bit. So make sure you give it a value that agrees with your system (see data sheet).
- uint8_t ecc108e_read_config_zone (void)

  This function reads all 88 bytes from the configuration zone. Obtain the data by putting a breakpoint after every read and inspecting "response".

7.10.1 Detailed Description

Application examples that Use the ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ATECC108 Library License:

Example functions are given that demonstrate the device.

ecc108e_checkmac_device: Demonstrates communication using a Mac - CheckMac command sequence with relatively low security (mode 0: no nonce), but little code space usage.

ecc108e_checkmac_firmware: Demonstrates high security using a Nonce - GenDig - Mac command sequence and MAC verification in firmware. This requires more code space because a sha256 implementation in firmware is needed. Also, the firmware has to be able to access keys. Having a key stored outside the device poses a higher security risk.

ecc108e_checkmac_derive_key: Demonstrates high security in a host / client scenario using a DeriveKey / Mac command sequence on one device (client) and a GenDig / CheckMac sequence on another device (host). No sha256 implementation in firmware is needed. All keys are only stored on the devices and never revealed. When using I2C you have to change the address of one of the devices first. This example needs modifications introducing the Pause command when using the SWI UART interface.

ecc108e_change_i2c_address: This is just a utility that changes the I2C address of a device.

CAUTION WHEN DEBUGGING: Be aware of the timeout feature of the device. The device will go to sleep between 0.7 and 1.5 seconds after a Wakeup. When hitting a break point, this timeout will likely to kick in and the device has gone to sleep before you continue debugging. Therefore, after you have examined variables you might have to restart your debug session.

7.10.2 Function Documentation

7.10.2.1 uint8_t ecc108e_change_i2c_address (void)

This function changes the I2C address of a device. Running it will access the device with I2C address ECC108_CLIEN-T_ADDRESS and change it to ECC108_HOST_ADDRESS as long as the configuration zone is not locked (byte under address 87 = 0x55). Be aware that bit 3 of the I2C address is also used as a TTL enable bit. So make sure you give it a value that agrees with your system (see data sheet).

Returns

status of the operation

7.10.2.2 uint8_t ecc108e_check_response_status (uint8_t *ret_code*, uint8_t * *response*)

This function checks the response status byte and puts the device to sleep if there was an error.

Parameters

in	*ret_code*	return code of function
in	*response*	pointer to response buffer

Returns

status of the operation

7.10.2.3 uint8_t ecc108e_checkmac_device (void)

This function serves as an example for the ECC108 Mac and CheckMac commands.

```
In an infinite loop, it issues the same command
sequence using the Command Marshaling layer of
the ECC108 library.
```

Returns

status of the operation

7.10.2.4 uint8_t ecc108e_checkmac_firmware (void)

This function serves as an example for the ECC108 Nonce, GenDig, and Mac commands.

```
In an infinite loop, it issues the same command
sequence using the Command Marshaling layer of
the ECC108 library.
```

Returns

status of the operation

7.10.2.5 uint8_t ecc108e_configure (uint8_t *parent_id*)

This function configures client and host device for the ChildKey example.

```
Creating a child key allows a host device to check a MAC
in a highly secure fashion. No replay attacks are possible
and SHA256 calculation in firmware is not needed.
To run this command sequence successfully the devices have
to be configured first: We use a slot in the client device that is already
configured for this purpose, but we want to disable SingleUse. Since we
are re-configuring it anyway, we also allow encrypted read.
Only one device could be used for demonstration purpose, but since the
parent keys have to match, the parent key would have to be duplicated
in a different slot with the CheckMac flag set.
```

Parameters

in	*parent_id*	slot to re-configure

Returns

status of the operation

7.10.2.6 uint8_t ecc108e_derive_key (void)

This function serves as an example for the ECC108 Nonce, DeriveKey, and Mac commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.

```
Creating a child key on the client allows a host device to check a MAC
in a highly secure fashion. No replay attacks are possible,
SHA256 calculation in firmware is not needed, and keys are only stored
on the secure device.
A brief explanation for this command sequence:
The client generates a child key (DeriveKey command) derived from a
parent key that it shares with the host device, and stores it in one
of its key slots using a random nonce (commands Random and Nonce).
The host generates the same key and stores it in its TempKey using
the same nonce. Now, when the client receives a Mac command with the
child key id, a CheckMac command on the host using the TempKey will
succeed.
To run this command sequence successfully the devices have
to be configured first: The child key has to point to the parent,
and the parent key in the host device has to be flagged as CheckOnly.
This sequence could be run with one device, but since the
parent keys have to match the parent key would have to be duplicated
in a different slot.
Because every time this command sequence is executed the slot for the
child key is being written, this sequence does not run in loop to
prevent wearing out the flash.
```

Use the following sequence for secure authentication using the default configuration and setting ReadConfig of slot 3 (parent key) from the default A3 (single use) to C0 (read encrypted using slot 0). Use slot 8 as the child key for debugging since it is readable.

1. MCU to client device: fixed nonce -> TempKey
2. MCU to client device: DeriveKey -> child key in chosen slot (parent key in slot 3)
3. MCU to client device: fixed nonce -> TempKey
4. MCU to client device: Mac -> response = sha256(chosen slot / child key, fixed nonce / TempKey, command, 3 bytes of SN)
5. MCU to host device: GenDig -> TempKey = child key
6. MCU to host device: CheckMac -> sha256(child key / TempKey, challenge / fixed nonce, MAC command, 3 bytes of SN)

   Returns

   status of the operation

7.10.2.7 uint8_t ecc108e_read_config_zone (void)

This function reads all 88 bytes from the configuration zone. Obtain the data by putting a breakpoint after every read and inspecting "response".

factory defaults of configuration zone 01 23 76 ab 00 04 05 00 0c 8f b7 bd ee 55 01 00 c8 00 55 00 8f 80 80 a1 82 e0 a3 60 94 40 a0 85 86 40 87 07 0f 00 89 f2 8a 7a 0b 8b 0c 4c dd 4d c2 42 af 8f ff 00 ff 00 ff 00 1f 00 ff 00 1f 00 ff 00 ff 00 1f ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 00 00 55 55

Byte # Name Value Description 0 - 3 SN[0-3] 012376ab part of the serial number 4 - 7 RevNum 00040500 device revision (= 4) 8 - 12 SN[4-8] 0c8fb7bdee part of the serial number 13 Reserved 55 set by Atmel (55: First 16 bytes are unlocked / special case.) 14 I2C_Enable 01 SWI / I2C (1: I2C) 15 Reserved 00 set by Atmel 16 I2C_Address c8 default I2C address 17 RFU 00 reserved for future use; must be 0 18 OTPmode 55 55: consumption mode, not supported at this time 19 SelectorMode 00 00: Selector can always be written with UpdateExtra command. 20 slot 0, read 8f 8: Secret. f: Does not matter. 21 slot 0, write 80 8: Never write. 0: Does not matter. 22 slot 1, read 80 8: Secret. 0: CheckMac copy 23 slot 1, write a1 a: MAC required (roll). 1: key id 24 slot 2, read 82 8: Secret. 2: Does not matter. 25 slot 2, write e0 e: MAC required (roll) and write encrypted. 0: key id 26 slot 3, read a3 a: Single use. 3: Does not matter. 27 slot 3, write 60 6: Encrypt, MAC not required (roll). 0: Does not matter. 28 slot 4, read 94 9: CheckOnly. 4: Does not matter. 29 slot 4, write 40 4: Encrypt. 0: key id 30 slot 5, read a0 a: Single use. 0: key id 31 slot 5, write 85 8: Never write. 5: Does not matter. 32 slot 6, read 86 8: Secret. 6: Does not matter. 33 slot 6, write 40 4: Encrypt. 0: key id 34 slot 7, read 87 8: Secret. 7: Does not matter. 35 slot 7, write 07 0: Write. 7: Does not matter. 36 slot 8, read 0f 0: Read. f: Does not matter. 37 slot 8, write 00 0: Write. 0: Does not matter. 38 slot 9, read 89 8: Secret. 9: Does not matter. 39 slot 9, write f2 f: Encrypt, MAC required (create). 2: key id 40 slot 10, read 8a 8: Secret. a: Does not matter. 41 slot 10, write 7a 7: Encrypt, MAC not required (create). a: key id 42 slot 11, read 0b 0: Read. b: Does not matter. 43 slot 11, write 8b 8: Never Write. b: Does not matter. 44 slot 12, read 0c 0: Read. c: Does not matter. 45 slot 12, write 4c 4: Encrypt, not allowed as target. c: key id 46 slot 13, read dd d: CheckOnly. d: key id 47 slot 13, write 4d 4: Encrypt, not allowed as target. d: key id 48 slot 14, read c2 c: CheckOnly. 2: key id 49 slot 14, write 42 4: Encrypt. 2: key id 50 slot 15, read af a: Single use. f: Does not matter. 51 slot 15, write 8f 8: Never write. f: Does not matter. 52 UseFlag 0 ff 8 uses 53 UpdateCount 0 00 count = 0 54 UseFlag 1 ff 8 uses 55 UpdateCount 1 00 count = 0 56 UseFlag 2 ff 8 uses 57 UpdateCount 2 00 count = 0 58 UseFlag 3 1f 5 uses 59 UpdateCount 3 00 count = 0 60 UseFlag 4 ff 8 uses 61 UpdateCount 4 00 count = 0 62 UseFlag 5 1f 5 uses 63 UpdateCount 5 00 count = 0 64 UseFlag 6 ff 8 uses 65 UpdateCount 6 00 count = 0 66 UseFlag 7 ff 8 uses 67 UpdateCount 7 00 count = 0 68 - 83 LastKeyUse 1fffffffffffffffffffffffffffffff 84 UserExtra 85 Selector 00 Pause command with chip id 0 leaves this device active. 86 LockValue 55 OTP and Data zones are not locked. 87 LockConfig 55 Configuration zone is unlocked.

slot summary: Slot 1 is parent key and slot 1 is child key (DeriveKey-Roll). Slot 2 is parent key and slot 0 is child key (DeriveKey-Roll). Slot 3 is parent key and child key has to be given in Param2 (DeriveKey-Roll). Slots 4, 13, and 14 are CheckOnly. Slots 5 and 15 are single use. Slot 8 is plain text. Slot 10 is parent key and slot 10 is child key (DeriveKey-Create). Slot 12 is not allowed as target.

Returns

status of the operation

7.10.2.8 uint8_t ecc108e_send_info_command (void)

This function is a simple example for how to use the library. It wakes up the device, sends a DevRev command, receives its response, and puts the device to sleep. It uses a total of four library functions from all three layers, physical, communication, and command marshaling layer.

```
Use this example to familiarize yourself with the library
and device communication before proceeding to examples that
deal with the security features of the device.
```

Returns

status of the operation

7.10.2.9 uint8_t ecc108e_wakeup_device (uint8_t *device_id*)

This function wakes up two I2C devices and puts one back to sleep, effectively waking up only one device among two that share SDA.

Parameters

in	*device_id*	which device to wake up

Returns

status of the operation

7.11 ecc108_examples.h File Reference

Application examples that Use the ECC108 Library.

`#include <stdint.h>`

Functions

- uint8_t ecc108e_send_info_command (void)

  This function is a simple example for how to use the library. It wakes up the device, sends a DevRev command, receives its response, and puts the device to sleep. It uses a total of four library functions from all three layers, physical, communication, and command marshaling layer.
- uint8_t ecc108e_checkmac_device (void)

  This function serves as an example for the ECC108 Mac and CheckMac commands.
- uint8_t ecc108e_checkmac_firmware (void)

  This function serves as an example for the ECC108 Nonce, GenDig, and Mac commands.
- uint8_t ecc108e_derive_key (void)

  This function serves as an example for the ECC108 Nonce, DeriveKey, and Mac commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.
- uint8_t ecc108e_change_i2c_address (void)

  This function changes the I2C address of a device. Running it will access the device with I2C address ECC108_CLIENT_ADDRESS and change it to ECC108_HOST_ADDRESS as long as the configuration zone is not locked (byte under address 87 = 0x55). Be aware that bit 3 of the I2C address is also used as a TTL enable bit. So make sure you give it a value that agrees with your system (see data sheet).
- uint8_t ecc108e_read_config_zone (void)

  This function reads all 88 bytes from the configuration zone. Obtain the data by putting a breakpoint after every read and inspecting "response".

7.11.1 Detailed Description

Application examples that Use the ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

Three example functions are given that demonstrate the device.

7.11.2 Function Documentation

7.11.2.1 uint8_t ecc108e_change_i2c_address (void)

This function changes the I2C address of a device. Running it will access the device with I2C address ECC108_CLIENT_ADDRESS and change it to ECC108_HOST_ADDRESS as long as the configuration zone is not locked (byte under address 87 = 0x55). Be aware that bit 3 of the I2C address is also used as a TTL enable bit. So make sure you give it a value that agrees with your system (see data sheet).

Returns

status of the operation

7.11.2.2 uint8_t ecc108e_checkmac_device (void)

This function serves as an example for the ECC108 Mac and CheckMac commands.

```
In an infinite loop, it issues the same command
sequence using the Command Marshaling layer of
the ECC108 library.
```

Returns

status of the operation

7.11.2.3 uint8_t ecc108e_checkmac_firmware (void)

This function serves as an example for the ECC108 Nonce, GenDig, and Mac commands.

```
In an infinite loop, it issues the same command
sequence using the Command Marshaling layer of
the ECC108 library.
```

Returns

status of the operation

7.11.2.4 uint8_t ecc108e_derive_key (void)

This function serves as an example for the ECC108 Nonce, DeriveKey, and Mac commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.

```
Creating a child key on the client allows a host device to check a MAC
in a highly secure fashion. No replay attacks are possible,
SHA256 calculation in firmware is not needed, and keys are only stored
on the secure device.
A brief explanation for this command sequence:
The client generates a child key (DeriveKey command) derived from a
parent key that it shares with the host device, and stores it in one
of its key slots using a random nonce (commands Random and Nonce).
The host generates the same key and stores it in its TempKey using
the same nonce. Now, when the client receives a Mac command with the
child key id, a CheckMac command on the host using the TempKey will
succeed.
To run this command sequence successfully the devices have
to be configured first: The child key has to point to the parent,
and the parent key in the host device has to be flagged as CheckOnly.
This sequence could be run with one device, but since the
parent keys have to match the parent key would have to be duplicated
in a different slot.
Because every time this command sequence is executed the slot for the
child key is being written, this sequence does not run in loop to
prevent wearing out the flash.
```

Use the following sequence for secure authentication using the default configuration and setting ReadConfig of slot 3 (parent key) from the default A3 (single use) to C0 (read encrypted using slot 0). Use slot 8 as the child key for debugging since it is readable.

1. MCU to client device: fixed nonce -> TempKey
2. MCU to client device: DeriveKey -> child key in chosen slot (parent key in slot 3)
3. MCU to client device: fixed nonce -> TempKey
4. MCU to client device: Mac -> response = sha256(chosen slot / child key, fixed nonce / TempKey, command, 3 bytes of SN)
5. MCU to host device: GenDig -> TempKey = child key
6. MCU to host device: CheckMac -> sha256(child key / TempKey, challenge / fixed nonce, MAC command, 3 bytes of SN)

   Returns

   status of the operation

7.11.2.5 uint8_t ecc108e_read_config_zone (void)

This function reads all 88 bytes from the configuration zone. Obtain the data by putting a breakpoint after every read and inspecting "response".

factory defaults of configuration zone 01 23 76 ab 00 04 05 00 0c 8f b7 bd ee 55 01 00 c8 00 55 00 8f 80 80 a1 82 e0 a3 60 94 40 a0 85 86 40 87 07 0f 00 89 f2 8a 7a 0b 8b 0c 4c dd 4d c2 42 af 8f ff 00 ff 00 ff 00 1f 00 ff 00 1f 00 ff 00 ff 00 1f ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 00 00 55 55

Byte # Name Value Description 0 - 3 SN[0-3] 012376ab part of the serial number 4 - 7 RevNum 00040500 device revision (= 4) 8 - 12 SN[4-8] 0c8fb7bdee part of the serial number 13 Reserved 55 set by Atmel (55: First 16 bytes are unlocked / special case.) 14 I2C_Enable 01 SWI / I2C (1: I2C) 15 Reserved 00 set by Atmel 16 I2C_Address c8 default I2C address 17 RFU 00 reserved for future use; must be 0 18 OTPmode 55 55: consumption mode, not supported at this time 19 SelectorMode 00 00: Selector can always be written with UpdateExtra command. 20 slot 0, read 8f 8: Secret. f: Does not matter. 21 slot 0, write 80 8: Never write. 0: Does not matter. 22 slot 1, read 80 8: Secret. 0: CheckMac copy 23 slot 1, write a1 a: MAC required (roll). 1: key id 24 slot 2, read 82 8: Secret. 2: Does not matter. 25 slot 2, write e0 e: MAC required (roll) and write encrypted. 0: key id 26 slot 3, read a3 a: Single use. 3: Does not matter. 27 slot 3, write 60 6: Encrypt, MAC not required (roll). 0: Does not matter. 28 slot 4, read 94 9: CheckOnly. 4: Does not matter. 29 slot 4, write 40 4: Encrypt. 0: key id 30 slot 5, read a0 a: Single use. 0: key id 31 slot 5, write 85 8: Never write. 5: Does not matter. 32 slot 6, read 86 8: Secret. 6: Does not matter. 33 slot 6, write 40 4: Encrypt. 0: key id 34 slot 7, read 87 8: Secret. 7: Does not matter. 35 slot 7, write 07 0: Write. 7: Does not matter. 36 slot 8, read 0f 0: Read. f: Does not matter. 37 slot 8, write 00 0: Write. 0: Does not matter. 38 slot 9, read 89 8: Secret. 9: Does not matter. 39 slot 9, write f2 f: Encrypt, MAC required (create). 2: key id 40 slot 10, read 8a 8: Secret. a: Does not matter. 41 slot 10, write 7a 7: Encrypt, MAC not required (create). a: key id 42 slot 11, read 0b 0: Read. b: Does not matter. 43 slot 11, write 8b 8: Never Write. b: Does not matter. 44 slot 12, read 0c 0: Read. c: Does not matter. 45 slot 12, write 4c 4: Encrypt, not allowed as target. c: key id 46 slot 13, read dd d: CheckOnly. d: key id 47 slot 13, write 4d 4: Encrypt, not allowed as target. d: key id 48 slot 14, read c2 c: CheckOnly. 2: key id 49 slot 14, write 42 4: Encrypt. 2: key id 50 slot 15, read af a: Single use. f: Does not matter. 51 slot 15, write 8f 8: Never write. f: Does not matter. 52 UseFlag 0 ff 8 uses 53 UpdateCount 0 00 count = 0 54 UseFlag 1 ff 8 uses 55 UpdateCount 1 00 count = 0 56 UseFlag 2 ff 8 uses 57 UpdateCount 2 00 count = 0 58 UseFlag 3 1f 5 uses 59 UpdateCount 3 00 count = 0 60 UseFlag 4 ff 8 uses 61 UpdateCount 4 00 count = 0 62 UseFlag 5 1f 5 uses 63 UpdateCount 5 00 count = 0 64 UseFlag 6 ff 8 uses 65 UpdateCount 6 00 count = 0 66 UseFlag 7 ff 8 uses 67 UpdateCount 7 00 count = 0 68 - 83 LastKeyUse 1fffffffffffffffffffffffffffffff 84 UserExtra 85 Selector 00 Pause command with chip id 0 leaves this device active. 86 LockValue 55 OTP and Data zones are not locked. 87 LockConfig 55 Configuration zone is unlocked.

slot summary: Slot 1 is parent key and slot 1 is child key (DeriveKey-Roll). Slot 2 is parent key and slot 0 is child key (DeriveKey-Roll). Slot 3 is parent key and child key has to be given in Param2 (DeriveKey-Roll). Slots 4, 13, and 14 are CheckOnly. Slots 5 and 15 are single use. Slot 8 is plain text. Slot 10 is parent key and slot 10 is child key (DeriveKey-Create). Slot 12 is not allowed as target.

Returns

status of the operation

7.11.2.6 uint8_t ecc108e_send_info_command (void)

This function is a simple example for how to use the library. It wakes up the device, sends a DevRev command, receives its response, and puts the device to sleep. It uses a total of four library functions from all three layers, physical, communication, and command marshaling layer.

```
Use this example to familiarize yourself with the library
and device communication before proceeding to examples that
deal with the security features of the device.
```

Returns

status of the operation

7.12 ecc108_helper.c File Reference

ECC108 Helper Functions.

```
#include <string.h>
#include <stdint.h>
#include "ecc108_helper.h"
#include "ecc108_lib_return_codes.h"
#include "ecc108_comm_marshaling.h"
```

Functions

- uint8_t ecc108h_nonce (struct ecc108h_nonce_in_out *param)

  This function calculates a 32-byte nonce based on 20-byte input value (NumIn) and 32-byte random number (RandOut).
- uint8_t ecc108h_mac (struct ecc108h_mac_in_out *param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other informations.
- uint8_t ecc108h_check_mac (struct ecc108h_check_mac_in_out *param)

  This function calculates SHA-256 digest (MAC) of a password and other informations, to be verified using CheckMac command in the Device.
- uint8_t ecc108h_hmac (struct ecc108h_hmac_in_out *param)

  This function generates an HMAC/SHA-256 digest of a key and other informations.
- uint8_t ecc108h_gen_dig (struct ecc108h_gen_dig_in_out *param)

  This function combines current TempKey with a stored value.
- uint8_t ecc108h_derive_key (struct ecc108h_derive_key_in_out *param)

  This function combines current value of a key with the TempKey.
- uint8_t ecc108h_derive_key_mac (struct ecc108h_derive_key_mac_in_out *param)

  This function calculates input MAC for DeriveKey opcode.
- uint8_t ecc108h_encrypt (struct ecc108h_encrypt_in_out *param)

  This function encrypts 32-byte cleartext data to be written using Write opcode, and optionally calculates input MAC.
- uint8_t ecc108h_decrypt (struct ecc108h_decrypt_in_out *param)

This function decrypts 32-byte encrypted data (Contents) from Read opcode.

- void ecc108h_calculate_crc_chain (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates CRC.

- void ecc108h_calculate_sha256 (int32_t len, uint8_t ∗message, uint8_t ∗digest)

  This function creates a SHA256 digest on a little-endian system.

7.12.1 Detailed Description

ECC108 Helper Functions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.13 ecc108_helper.h File Reference

Declarations and Prototypes for ECC108 Helper Functions.

```
#include <stdint.h>
```

Data Structures

- struct ecc108h_temp_key

  Structure to hold TempKey fields.
- struct ecc108h_calculate_sha256_in_out

  Input/output parameters for function ecc108h_nonce().
- struct ecc108h_nonce_in_out

  Input/output parameters for function ecc108h_nonce().
- struct ecc108h_mac_in_out

  Input/output parameters for function ecc108h_mac().
- struct ecc108h_hmac_in_out

  Input/output parameters for function ecc108h_hmac().
- struct ecc108h_gen_dig_in_out

  Input/output parameters for function ecc108h_gen_dig().
- struct ecc108h_derive_key_in_out

  Input/output parameters for function ecc108h_derive_key().
- struct ecc108h_derive_key_mac_in_out

  Input/output parameters for function ecc108h_derive_key_mac().
- struct ecc108h_encrypt_in_out

  Input/output parameters for function ecc108h_encrypt().
- struct ecc108h_decrypt_in_out

  Input/output parameters for function ecc108h_decrypt().
- struct ecc108h_check_mac_in_out

  Input/output parameters for function ecc108h_check_mac().

Functions

- uint8_t ecc108h_nonce (struct ecc108h_nonce_in_out ∗param)

  This function calculates a 32-byte nonce based on 20-byte input value (NumIn) and 32-byte random number (RandOut).
- uint8_t ecc108h_mac (struct ecc108h_mac_in_out ∗param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other informations.
- uint8_t ecc108h_check_mac (struct ecc108h_check_mac_in_out ∗param)

  This function calculates SHA-256 digest (MAC) of a password and other informations, to be verified using CheckMac command in the Device.
- uint8_t ecc108h_hmac (struct ecc108h_hmac_in_out ∗param)

  This function generates an HMAC/SHA-256 digest of a key and other informations.
- uint8_t ecc108h_gen_dig (struct ecc108h_gen_dig_in_out ∗param)

  This function combines current TempKey with a stored value.
- uint8_t ecc108h_derive_key (struct ecc108h_derive_key_in_out ∗param)

  This function combines current value of a key with the TempKey.
- uint8_t ecc108h_derive_key_mac (struct ecc108h_derive_key_mac_in_out ∗param)

This function calculates input MAC for DeriveKey opcode.

- uint8_t ecc108h_encrypt (struct ecc108h_encrypt_in_out ∗param)

  This function encrypts 32-byte cleartext data to be written using Write opcode, and optionally calculates input MAC.
- uint8_t ecc108h_decrypt (struct ecc108h_decrypt_in_out ∗param)

  This function decrypts 32-byte encrypted data (Contents) from Read opcode.
- void ecc108h_calculate_crc_chain (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates CRC.
- void ecc108h_calculate_sha256 (int32_t len, uint8_t ∗message, uint8_t ∗digest)

  This function creates a SHA256 digest on a little-endian system.

7.13.1 Detailed Description

Declarations and Prototypes for ECC108 Helper Functions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.14 ecc108_i2c.c File Reference

Functions for I2C Physical Hardware Independent Layer of ECC108 Library.

```
#include <avr/io.h>
#include "i2c_phys.h"
#include "ecc108_physical.h"
#include "ecc108_lib_return_codes.h"
#include "timer_utilities.h"
```

Macros

- #define ECC108_GPIO_WAKEUP

  GPIO definitions.

Enumerations

- enum i2c_word_address { ECC108_I2C_PACKET_FUNCTION_RESET, ECC108_I2C_PACKET_FUNCTION_SLEEP, ECC108_I2C_PACKET_FUNCTION_IDLE, ECC108_I2C_PACKET_FUNCTION_NORMAL }

  This enumeration lists all packet types sent to a ECC108 device.
- enum i2c_read_write_flag { I2C_WRITE = (uint8_t) 0x00, I2C_READ = (uint8_t) 0x01 }

  This enumeration lists flags for I2C read or write addressing.

Functions

- void ecc108p_set_device_id (uint8_t id)

  This I2C function sets the I2C address. Communication functions will use this address.
- void ecc108p_init (void)

  This I2C function initializes the hardware.
- uint8_t ecc108p_wakeup (void)

  This I2C function generates a Wake-up pulse and delays.
- uint8_t ecc108p_send_command (uint8_t count, uint8_t *command)

  This I2C function sends a command to the device.
- uint8_t ecc108p_idle (void)

  This I2C function puts the ECC108 device into idle state.
- uint8_t ecc108p_sleep (void)

  This I2C function puts the ECC108 device into low-power state.
- uint8_t ecc108p_reset_io (void)

  This I2C function resets the I/O buffer of the ECC108 device.
- uint8_t ecc108p_receive_response (uint8_t size, uint8_t *response)

  This I2C function receives a response from the ECC108 device.
- uint8_t ecc108p_resync (uint8_t size, uint8_t *response)

  This I2C function resynchronizes communication.

7.14.1 Detailed Description

Functions for I2C Physical Hardware Independent Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ATECC108 Library License:

7.14.2 Enumeration Type Documentation

7.14.2.1 enum i2c_read_write_flag

This enumeration lists flags for I2C read or write addressing.

Enumerator

I2C_WRITE write command flag

I2C_READ read command flag

7.14.2.2 enum i2c_word_address

This enumeration lists all packet types sent to a ECC108 device.

The following byte stream is sent to a ECC108 I2C device: {I2C start} {I2C address} {word address} [{data}] {I2C stop}. Data are only sent after a word address of value ECC108_I2C_PACKET_FUNCTION_NORMAL.

Enumerator

ECC108_I2C_PACKET_FUNCTION_RESET Reset device.

ECC108_I2C_PACKET_FUNCTION_SLEEP Put device into Sleep mode.

ECC108_I2C_PACKET_FUNCTION_IDLE Put device into Idle mode.

ECC108_I2C_PACKET_FUNCTION_NORMAL Write / evaluate data that follow this word address byte.

7.15 ecc108_lib_return_codes.h File Reference

ECC108 Library Return Code Definitions.

```
#include <stddef.h>
```

Macros

- #define ECC108_SUCCESS ((uint8_t) 0x00)

  Function succeeded.
- #define ECC108_CHECKMAC_FAILED ((uint8_t) 0xD1)

  response status byte indicates CheckMac failure
- #define ECC108_PARSE_ERROR ((uint8_t) 0xD2)

  response status byte indicates parsing error
- #define ECC108_CMD_FAIL ((uint8_t) 0xD3)

  response status byte indicates command execution error
- #define ECC108_STATUS_CRC ((uint8_t) 0xD4)

  response status byte indicates CRC error
- #define ECC108_STATUS_UNKNOWN ((uint8_t) 0xD5)

  response status byte is unknown
- #define ECC108_FUNC_FAIL ((uint8_t) 0xE0)

  Function could not execute due to incorrect condition / state.
- #define ECC108_GEN_FAIL ((uint8_t) 0xE1)

  unspecified error
- #define ECC108_BAD_PARAM ((uint8_t) 0xE2)

  bad argument (out of range, null pointer, etc.)
- #define ECC108_INVALID_ID ((uint8_t) 0xE3)

  invalid device id, id not set
- #define ECC108_INVALID_SIZE ((uint8_t) 0xE4)

  Count value is out of range or greater than buffer size.
- #define ECC108_BAD_CRC ((uint8_t) 0xE5)

  incorrect CRC received
- #define ECC108_RX_FAIL ((uint8_t) 0xE6)

Timed out while waiting for response. Number of bytes received is > 0.

- #define ECC108_RX_NO_RESPONSE ((uint8_t) 0xE7)

  Not an error while the Command layer is polling for a command response.

- #define ECC108_RESYNC_WITH_WAKEUP ((uint8_t) 0xE8)

  re-synchronization succeeded, but only after generating a Wake-up

- #define ECC108_COMM_FAIL ((uint8_t) 0xF0)

  Communication with device failed. Same as in hardware dependent modules.

- #define ECC108_TIMEOUT ((uint8_t) 0xF1)

  Timed out while waiting for response. Number of bytes received is 0.

7.15.1 Detailed Description

ECC108 Library Return Code Definitions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.16 ecc108_physical.h File Reference

Definitions and Prototypes for Physical Layer Interface of ECC108 Library.

```
#include <stdint.h>
#include "ecc108_config.h"
```

Macros

- #define ECC108_RSP_SIZE_MIN ((uint8_t) 4)

    minimum number of bytes in response
- #define ECC108_RSP_SIZE_MAX ((uint8_t) (72 + 3))

    maximum size of response packet (GenKey and Verify command)
- #define ECC108_BUFFER_POS_COUNT (0)

    buffer index of count byte in command or response
- #define ECC108_BUFFER_POS_DATA (1)

    buffer index of data in response
- #define ECC108_WAKEUP_PULSE_WIDTH (uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)
- #define ECC108_WAKEUP_DELAY (uint8_t) (200.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

Functions

- uint8_t ecc108p_send_command (uint8_t count, uint8_t ∗command)

    This I2C function sends a command to the device.
- uint8_t ecc108p_receive_response (uint8_t size, uint8_t ∗response)

    This I2C function receives a response from the ECC108 device.
- void ecc108p_init (void)

    This I2C function initializes the hardware.
- void ecc108p_set_device_id (uint8_t id)

    This I2C function sets the I2C address. Communication functions will use this address.
- uint8_t ecc108p_wakeup (void)

    This I2C function generates a Wake-up pulse and delays.
- uint8_t ecc108p_idle (void)

    This I2C function puts the ECC108 device into idle state.
- uint8_t ecc108p_sleep (void)

    This I2C function puts the ECC108 device into low-power state.
- uint8_t ecc108p_reset_io (void)

    This I2C function resets the I/O buffer of the ECC108 device.
- uint8_t ecc108p_resync (uint8_t size, uint8_t ∗response)

    This I2C function resynchronizes communication.

7.16.1 Detailed Description

Definitions and Prototypes for Physical Layer Interface of ECC108 Library.

Author

Atmel Crypto Products

Date

June 21, 2013

Copyright

7.17 ecc108_swi.c File Reference

Functions for Single Wire, Hardware Independent Physical Layer of ECC108 Library.

```
#include "swi_phys.h"
#include "ecc108_physical.h"
#include "ecc108_lib_return_codes.h"
#include "timer_utilities.h"
```

Macros

- #define ECC108_SWI_FLAG_CMD ((uint8_t) 0x77)

  flag preceding a command
- #define ECC108_SWI_FLAG_TX ((uint8_t) 0x88)

  flag requesting a response
- #define ECC108_SWI_FLAG_IDLE ((uint8_t) 0xBB)

  flag requesting to go into Idle mode
- #define ECC108_SWI_FLAG_SLEEP ((uint8_t) 0xCC)

  flag requesting to go into Sleep mode

Functions

- void ecc108p_init (void)

  This SWI function initializes the hardware.
- void ecc108p_set_device_id (uint8_t id)

  This SWI function selects the GPIO pin used for communication.
- uint8_t ecc108p_send_command (uint8_t count, uint8_t ∗command)

  This SWI function sends a command to the device. Device versions <= 0x100 need the flag to last longer than 500 us. Therefore, we send a dummy flag of 0 before sending the command flag.
- uint8_t ecc108p_receive_response (uint8_t size, uint8_t ∗response)

  This SWI function receives a response from the device.
- uint8_t ecc108p_wakeup (void)

  This SWI function generates a Wake-up pulse and delays.
- uint8_t ecc108p_idle ()

  This SWI function puts the device into idle state.
- uint8_t ecc108p_sleep ()

  This SWI function puts the device into low-power state.
- uint8_t ecc108p_reset_io (void)

  This SWI function is only a dummy since the functionality does not exist for the SWI version of the ECC108 device.
- uint8_t ecc108p_resync (uint8_t size, uint8_t ∗response)

  This function re-synchronizes communication.

7.17.1 Detailed Description

Functions for Single Wire, Hardware Independent Physical Layer of ECC108 Library.

```
Possible return codes from send functions in the hardware dependent module
are SWI_FUNCTION_RETCODE_SUCCESS and SWI_FUNCTION_RETCODE_TIMEOUT. These
are the same values in swi_phys.h and sha204_lib_return_codes.h. No return code
translation is needed in these cases (e.g. #ecc108p_idle, #ecc108p_sleep).
```

Author

Atmel Crypto Products

Date

September 13, 2012

7.18 i2c_phys.c File Reference

Functions of Hardware Dependent Part of Physical Layer Using I2C For Communication.

```
#include <avr\io.h>
#include <util\twi.h>
#include <avr\power.h>
#include "i2c_phys.h"
```

Functions

- void i2c_enable (void)

  This function initializes and enables the I2C peripheral.
- void i2c_disable (void)

  This function disables the I2C peripheral.
- uint8_t i2c_send_start (void)

  This function creates a Start condition (SDA low, then SCL low).
- uint8_t i2c_send_stop (void)

  This function creates a Stop condition (SCL high, then SDA high).
- uint8_t i2c_send_bytes (uint8_t count, uint8_t ∗data)

  This function sends bytes to an I^2 C device.
- uint8_t i2c_receive_byte (uint8_t ∗data)

  This function receives one byte from an I^2 C device.
- uint8_t i2c_receive_bytes (uint8_t count, uint8_t ∗data)

  This function receives bytes from an I^2 C device and sends a Stop.

7.18.1 Detailed Description

Functions of Hardware Dependent Part of Physical Layer Using I2C For Communication.

Author

Atmel Crypto Products

Date

June 24, 2013

Copyright

ATECC108 Library License:

7.18.2 Function Documentation

7.18.2.1 uint8_t i2c_receive_byte (uint8_t ∗ *data*)

This function receives one byte from an I^2 C device.

Parameters

out	*data*	pointer to received byte

Returns

status of the operation

7.18.2.2 uint8_t i2c_receive_bytes (uint8_t *count,* uint8_t ∗ *data*)

This function receives bytes from an I^2 C device and sends a Stop.

Parameters

in	*count*	number of bytes to receive
out	*data*	pointer to rx buffer

Returns

status of the operation

7.18.2.3 uint8_t i2c_send_bytes (uint8_t *count,* uint8_t ∗ *data*)

This function sends bytes to an I^2 C device.

Parameters

in	*count*	number of bytes to send
in	*data*	pointer to tx buffer

Returns

status of the operation

7.18.2.4 uint8_t i2c_send_start (void)

This function creates a Start condition (SDA low, then SCL low).

Returns

status of the operation

7.18.2.5 uint8_t i2c_send_stop (void)

This function creates a Stop condition (SCL high, then SDA high).

Returns

status of the operation

7.19 i2c_phys.h File Reference

Definitions for Hardware Dependent Part of Physical Layer Using I2C for Communication.

```
#include <stdint.h>
```

Macros

- #define I2C_CLOCK (400000.0)

  I2C clock.
- #define I2C_PULLUP

  Use pull-up resistors.
- #define I2C_START_TIMEOUT ((uint8_t) 250)

  number of polling iterations for TWINT bit in TWSR after creating a Start condition in i2c_send_start()
- #define I2C_BYTE_TIMEOUT ((uint8_t) 200)

  number of polling iterations for TWINT bit in TWSR after sending or receiving a byte.
- #define I2C_STOP_TIMEOUT ((uint8_t) 250)

  number of polling iterations for TWSTO bit in TWSR after creating a Stop condition in i2c_send_stop().
- #define I2C_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

  Communication with device succeeded.
- #define I2C_FUNCTION_RETCODE_COMM_FAIL ((uint8_t) 0xF0)

  Communication with device failed.

- #define I2C_FUNCTION_RETCODE_TIMEOUT ((uint8_t) 0xF1)

  Communication timed out.
- #define I2C_FUNCTION_RETCODE_NACK ((uint8_t) 0xF8)

  TWI nack.

Functions

- void i2c_enable (void)

  This function initializes and enables the I2C peripheral.
- void i2c_disable (void)

  This function disables the I2C peripheral.
- uint8_t i2c_send_start (void)

  This function creates a Start condition (SDA low, then SCL low).
- uint8_t i2c_send_stop (void)

  This function creates a Stop condition (SCL high, then SDA high).
- uint8_t i2c_send_bytes (uint8_t count, uint8_t ∗data)

  This function sends bytes to an I^2 C device.
- uint8_t i2c_receive_byte (uint8_t ∗data)

  This function receives one byte from an I^2 C device.
- uint8_t i2c_receive_bytes (uint8_t count, uint8_t ∗data)

  This function receives bytes from an I^2 C device and sends a Stop.

7.19.1 Detailed Description

Definitions for Hardware Dependent Part of Physical Layer Using I2C for Communication.

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

ATECC108 Library License:

7.19.2 Macro Definition Documentation

7.19.2.1 #define I2C_BYTE_TIMEOUT ((uint8_t) 200)

number of polling iterations for TWINT bit in TWSR after sending or receiving a byte.

Adjust this value considering how long it takes to check a status bit in the TWI status register, decrement the timeout counter, compare its value with 0, branch, and to send or receive one byte.

7.19.2.2 #define I2C_START_TIMEOUT ((uint8_t) 250)

number of polling iterations for TWINT bit in TWSR after creating a Start condition in i2c_send_start()

Adjust this value considering how long it takes to check a status bit in the TWI status register, decrement the timeout counter, compare its value with 0, and branch.

7.19.2.3 #define I2C_STOP_TIMEOUT ((uint8_t) 250)

number of polling iterations for TWSTO bit in TWSR after creating a Stop condition in i2c_send_stop().

Adjust this value considering how long it takes to check a status bit in the TWI control register, decrement the timeout counter, compare its value with 0, and branch.

7.19.3 Function Documentation

7.19.3.1 uint8_t i2c_receive_byte (uint8_t ∗ *data*)

This function receives one byte from an I^2 C device.

Parameters

out	*data*	pointer to received byte

Returns

status of the operation

7.19.3.2 uint8_t i2c_receive_bytes (uint8_t *count*, uint8_t * *data*)

This function receives bytes from an I^2 C device and sends a Stop.

Parameters

in	*count*	number of bytes to receive
out	*data*	pointer to rx buffer

Returns

status of the operation

7.19.3.3 uint8_t i2c_send_bytes (uint8_t *count,* uint8_t ∗ *data*)

This function sends bytes to an I^2 C device.

Parameters

in	*count*	number of bytes to send
in	*data*	pointer to tx buffer

Returns

status of the operation

7.19.3.4 uint8_t i2c_send_start (void)

This function creates a Start condition (SDA low, then SCL low).

Returns

status of the operation

7.19.3.5 uint8_t i2c_send_stop (void)

This function creates a Stop condition (SCL high, then SDA high).

Returns

status of the operation

7.20 swi_phys.h File Reference

Definitions and Prototypes for SWI Hardware Dependent Physical Layer of CryptoAuth Library.

```
#include <stdint.h>
```

Macros

- #define SWI_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

  Communication with device succeeded.

- #define SWI_FUNCTION_RETCODE_TIMEOUT ((uint8_t) 0xF1)

  Communication timed out.
- #define SWI_FUNCTION_RETCODE_RX_FAIL ((uint8_t) 0xF9)

  Communication failed after at least one byte was received.

Functions

- void swi_enable (void)

  This GPIO function sets the bit position of the signal pin to its default.
- void swi_set_device_id (uint8_t id)

  This GPIO function sets the signal pin. Communication functions will use this signal pin.
- void swi_set_signal_pin (uint8_t end)

  This GPIO function sets the signal pin low or high.
- uint8_t swi_send_bytes (uint8_t count, uint8_t *buffer)

  This GPIO function sends bytes to an SWI device.
- uint8_t swi_send_byte (uint8_t value)

  This GPIO function sends one byte to an SWI device.
- uint8_t swi_receive_bytes (uint8_t count, uint8_t *buffer)

  This GPIO function receives bytes from an SWI device.

7.20.1 Detailed Description

Definitions and Prototypes for SWI Hardware Dependent Physical Layer of CryptoAuth Library.

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

4. This software may only be redistributed and used in connection with an Atmel integrated circuit.

7.20.2 Macro Definition Documentation

7.20.2.1 #define SWI_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

Communication with device succeeded.

error codes for hardware dependent module Codes in the range 0x00 to 0xF7 are shared between physical interfaces (SWI, I^2). Codes in the range 0xF8 to 0xFF are special for the particular interface.

7.20.3 Function Documentation

7.20.3.1 uint8_t swi_receive_bytes (uint8_t *count,* uint8_t * *buffer*)

This GPIO function receives bytes from an SWI device.

Parameters

in	*count*	number of bytes to receive
out	*buffer*	pointer to rx buffer

Returns

status of the operation

This GPIO function receives bytes from an SWI device.

Parameters

in	*count*	number of bytes to receive
out	*buffer*	pointer to receive buffer

Returns

status of the operation

7.20.3.2 uint8_t swi_send_byte (uint8_t *value*)

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

7.20.3.3 uint8_t swi_send_bytes (uint8_t *count*, uint8_t * *buffer*)

This GPIO function sends bytes to an SWI device.

Parameters

in	*count*	number of bytes to send
in	*buffer*	pointer to tx buffer

Returns

status of the operation

This GPIO function sends bytes to an SWI device.

Parameters

in	*count*	number of bytes to send
in	*buffer*	pointer to transmit buffer

Returns

status of the operation

7.20.3.4 void swi_set_device_id (uint8_t *id*)

This GPIO function sets the signal pin. Communication functions will use this signal pin.

Parameters

in	*id*	client if zero, otherwise host

Returns

status of the operation

This GPIO function sets the signal pin. Communication functions will use this signal pin.

Parameters

in	*id*	not used in this UART module, only used in SWI bit-banging module To be able to talk to two devices (client or host) sending a Pause flag is required. Please refer to the data sheet.

7.20.3.5 void swi_set_signal_pin (uint8_t *is_high*)

This GPIO function sets the signal pin low or high.

Parameters

in	*is_high*	0: set signal low, otherwise high.

This GPIO function sets the signal pin low or high.

```
It is used to generate a Wake-up pulse.<BR>
Another way to generate a Wake-up pulse is using the UART
at half the communication baud rate and sending a 0.
Keeping the baud rate at 230400 would only produce
the signal wire going low for 34.7 us
when sending a data byte of 0 that causes the signal wire
being low for eight bits (start bit and seven data bits).
Configuring the UART for half the baud rate and sending
a 0 produces a long enough Wake-up pulse of 69.4 us.<BR>
The fact that a hardware independent Physical layer above
this hardware dependent layer delays for Wake-pulse width
after calling this function would only add this delay to the
much longer delay of 3 ms after the Wake-up pulse.
With other words, by not using GPIO for the generation of
a Wake-up pulse, we add only 69.4 us to the delay of
3000 us after the Wake-up pulse.<BR>
Implementing a Wake-up pulse generation using the UART
would introduce a slight design flaw though since this module
would now "know" something about the width of the Wake-up pulse.
We could add a function that sets the baud rate and
sends a 0, but that would add at least 150 bytes of code.
```

Parameters

in	*is_high*	0: set signal low, otherwise set signal high

7.21 timer_utilities.c File Reference

Timer Utility Functions.

`#include <stdint.h>`

Macros

- #define TIME_UTILS_US_CALIBRATION

  Fill the inner loop of delay_10us() with these CPU instructions to achieve 10 us per iteration.
- #define TIME_UTILS_LOOP_COUNT ((uint8_t) 28)

  Decrement the inner loop of delay_10us() this many times to achieve 10 us per iteration of the outer loop.
- #define TIME_UTILS_MS_CALIBRATION ((uint8_t) 104)

  The delay_ms function calls delay_10us with this parameter.

Functions

- void delay_10us (uint8_t delay)

  This function delays for a number of tens of microseconds.
- void delay_ms (uint8_t delay)

  This function delays for a number of milliseconds.

7.21.1 Detailed Description

Timer Utility Functions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.22 timer_utilities.h File Reference

Timer Utility Declarations.

`#include <stdint.h>`

Functions

- void delay_10us (uint8_t delay)

  This function delays for a number of tens of microseconds.
- void delay_ms (uint8_t delay)

  This function delays for a number of milliseconds.

7.22.1 Detailed Description

Timer Utility Declarations.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

7.23 uart_config.h File Reference

Definitions for Hardware Dependent Part of the Physical Layer of the Crypto Device Library Using a UART.

`#include <avr/io.h>`

Macros

- #define BAUD_RATE (230400UL)

  baud rate for SHA204 device in single-wire mode
- #define TIME_PER_LOOP_ITERATION (0.8)

  time in us it takes for decrementing a uint8_t and branching
- #define BIT_TIMEOUT ((uint8_t) (250.0 / TIME_PER_LOOP_ITERATION))

  number of polling iterations over UART register before timing out
- #define RX_TX_DELAY ((uint8_t) (15.0 / TIME_PER_LOOP_ITERATION))

  Delay for this many loop iterations before sending.
- #define UART_GPIO_DDR DDRD

  direction register when using UART pin for Wake-up
- #define UART_GPIO_OUT PORTD

  output register when using UART pin for Wake-up
- #define UART_GPIO_PIN_RX _BV(PD2)

  bit position when using UART rx pin for Wake-up
- #define UART_GPIO_PIN_TX _BV(PD3)

  bit position when using UART tx pin for Wake-up
- #define DEBUG_LOW

  undefine debugging macro
- #define DEBUG_HIGH

  undefine debugging macro

7.23.1 Detailed Description

Definitions for Hardware Dependent Part of the Physical Layer of the Crypto Device Library Using a UART.

Author

Atmel Crypto Products

Date

January 15, 2013

Copyright

ATECC108 Library License:

7.23.2 Macro Definition Documentation

7.23.2.1 #define BIT_TIMEOUT ((uint8_t) (250.0 / TIME_PER_LOOP_ITERATION))

number of polling iterations over UART register before timing out

The polling iteration takes about 0.8 us. For tx, we would need to wait bit time = 39 us. For rx, we need at least wait for tx / rx turn-around time + bit time = 95 us + 39 us = 134 us. Let's make the timeout larger to be safe.

7.24 uart_phys.c File Reference

Physical Layer Functions of the Crypto Device Library When Using UART.

```
#include "swi_phys.h"
#include "uart_config.h"
#include "avr_compatible.h"
```

Functions

- void swi_set_device_id (uint8_t id)

  This UART function is a dummy to satisfy the SWI module interface.
- void swi_enable (void)

  This UART function initializes the hardware.
- void swi_set_signal_pin (uint8_t is_high)

  This UART function sets the signal pin using GPIO.
- uint8_t swi_send_bytes (uint8_t count, uint8_t *buffer)

  This UART function sends bytes to an SWI device.
- uint8_t swi_send_byte (uint8_t value)

  This UART function sends one byte to an SWI device.
- uint8_t swi_receive_bytes (uint8_t count, uint8_t *buffer)

  This UART function receives bytes from an SWI device.

7.24.1 Detailed Description

Physical Layer Functions of the Crypto Device Library When Using UART.

```
This module supports most of ATmega and all ATXmega AVR microcontrollers.
http://www.atmel.com/dyn/products/param_table.asp?family_id=607&OrderBy=part_no&Direction=ASC
```

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

ATECC108 Library License:

7.24.2 Function Documentation

7.24.2.1 void swi_enable (void)

This UART function initializes the hardware.

This GPIO function sets the bit position of the signal pin to its default.

7.24.2.2 uint8_t swi_receive_bytes (uint8_t *count,* uint8_t ∗ *buffer*)

This UART function receives bytes from an SWI device.

This GPIO function receives bytes from an SWI device.

Parameters

in	*count*	number of bytes to receive
out	*buffer*	pointer to receive buffer

Returns

status of the operation

7.24.2.3 uint8_t swi_send_byte (uint8_t *value*)

This UART function sends one byte to an SWI device.

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

7.24.2.4 uint8_t swi_send_bytes (uint8_t *count,* uint8_t ∗ *buffer*)

This UART function sends bytes to an SWI device.

This GPIO function sends bytes to an SWI device.

Parameters

in	*count*	number of bytes to send
in	*buffer*	pointer to transmit buffer

Returns

status of the operation

7.24.2.5 void swi_set_device_id (uint8_t *id*)

This UART function is a dummy to satisfy the SWI module interface.

This GPIO function sets the signal pin. Communication functions will use this signal pin.

Parameters

in	*id*	not used in this UART module, only used in SWI bit-banging module To be able to talk to two devices (client or host) sending a Pause flag is required. Please refer to the data sheet.

7.24.2.6 void swi_set_signal_pin (uint8_t *is_high*)

This UART function sets the signal pin using GPIO.

This GPIO function sets the signal pin low or high.

```
It is used to generate a Wake-up pulse.<BR>
Another way to generate a Wake-up pulse is using the UART
at half the communication baud rate and sending a 0.
Keeping the baud rate at 230400 would only produce
the signal wire going low for 34.7 us
when sending a data byte of 0 that causes the signal wire
being low for eight bits (start bit and seven data bits).
Configuring the UART for half the baud rate and sending
a 0 produces a long enough Wake-up pulse of 69.4 us.<BR>
The fact that a hardware independent Physical layer above
this hardware dependent layer delays for Wake-pulse width
after calling this function would only add this delay to the
much longer delay of 3 ms after the Wake-up pulse.
With other words, by not using GPIO for the generation of
a Wake-up pulse, we add only 69.4 us to the delay of
3000 us after the Wake-up pulse.<BR>
Implementing a Wake-up pulse generation using the UART
would introduce a slight design flaw though since this module
would now "know" something about the width of the Wake-up pulse.
We could add a function that sets the baud rate and
sends a 0, but that would add at least 150 bytes of code.
```

Parameters

in	*is_high*	0: set signal low, otherwise set signal high