ECC108 Library

0.0.1

Generated by Doxygen 1.8.4

Mon Jun 24 2013 10:50:59

Contents

Chapter 1

ATECC108 Library Source Code

1.1 Introduction

This library enables a user to more quickly implement an application that uses an Atmel CryptoAuthentication ATEC-C108 device.

The library is distributed as source code. It is licensed according to terms of the included license agreement, which the user agreed to during the installation process.

The code comes in three layers, Command Marshaling, Communication and Physical layer.

The Command Marshaling layer assembles command packets from marshaling function parameters, sends them to the device, and returns the response from the device. This library is derived from the library for the ATSHA204 device. Currently, it provides only the ATSHA204 subset of command wrapper functions.

The Communication layer handles communication sequences (wake up, send command, receive response, sleep). It retries such sequences in case of certain communication failures.

The Physical layer puts the command packets on the chosen interface bus, and reads responses from it.

The library does not yet support the Alternate Single Wire Interface (ROM commands).

1.2 Getting Started

The user should be able to use most of the library as is, simply adding the library modules into her C project. Functions in the Physical layers will have to be modified or re-written if the processor is not an eight-bit AVR. Starting values for timeout loop counters and timer routines have to be adjusted.

To start development, add the library files to your project, modify / implement the functions in the Physical layer modules, and supply values for the timeout loop counters that match the execution time of your CPU (and the I2C clock if you are using I2C).

1.3 ATECC108 Communication Interfaces

The ECC108 device can be configured to either communicate in SWI or I2C mode. If the device is configured for single wire communication you can use either a UART or GPIO peripheral:

- The chip will communicate with a UART (or USART) at 230.4 kBaud. No driver chip is required (as in RS-232 or RS-285), the chip will talk directly to the UART pins.

- The chip will communicate with a soft UART, or a "big-banged" pin, at the same speed.

  Be aware that the actual baud-rate of the ATECC108 is the baud-rate divided by 9 (1 start bit, 7 data bits, 1 stop bit). One UART byte is one bit of information read from or written to the device. Therefore, the actual data through-put is 230,400 baud / 9 = 25,600 baud.

- If the device is configured for I2C communication the device will communicate using the standard I2C protocol (also known as two-wire interface or TWI) at speeds of up to 1 MHz.

Early versions of the SWI device need the command flag to be longer than 500 μs. The library achieves this by sending a dummy flag of 0 before sending the command flag.

With the distribution of this library example projects are provided for all methods.

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_dev_rev (uint8_t *tx_buffer, uint8_t *rx_buffer)

  This function sends a DevRev 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_DEVREV ((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 DevRev Command

- #define DEVREV_PARAM1_IDX ECC108_PARAM1_IDX

  DevRev command index for 1. parameter (ignored)
- #define DEVREV_PARAM2_IDX ECC108_PARAM2_IDX

  DevRev command index for 2. parameter (ignored)
- #define DEVREV_COUNT ECC108_CMD_SIZE_MIN

  DevRev command packet 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 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 DEVREV_RSP_SIZE ECC108_RSP_SIZE_VAL

  response size of DevRev 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 DEVREV_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 DEVREV_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

References CHECKMAC_CLIENT_CHALLENGE_IDX, CHECKMAC_CLIENT_CHALLENGE_SIZE, CHECKMAC_CLIENT_RESPONSE_IDX, CHECKMAC_CLIENT_RESPONSE_SIZE, CHECKMAC_COUNT, CHECKMAC_DATA_IDX, CHECKMAC_DELAY, CHECKMAC_EXEC_MAX, CHECKMAC_KEYID_IDX, CHECKMAC_MODE_IDX, CHECKMAC_MODE_MASK, CHECKMAC_OTHER_DATA_SIZE, CHECKMAC_RSP_SIZE, ECC108_BAD_PARAM, ECC108_CHECKMAC, ECC108_COUNT_IDX, ECC108_KEY_ID_MAX, ECC108_OPCODE_IDX, and ecc108c_send_and_receive().

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

References DERIVE_KEY_COUNT_LARGE, DERIVE_KEY_COUNT_SMALL, DERIVE_KEY_DELAY, DERIVE_KEY_EXEC_MAX, DERIVE_KEY_MAC_IDX, DERIVE_KEY_MAC_SIZE, DERIVE_KEY_RANDOM_FLAG, DERIVE_KEY_RANDOM_IDX, DERIVE_KEY_RSP_SIZE, DERIVE_KEY_TARGETKEY_IDX, ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_DERIVE_KEY, ECC108_KEY_ID_MAX, ECC108_OPCODE_IDX, and ecc108c_send_and_receive().

5.1.2.3 uint8_t ecc108m_dev_rev (uint8_t * *tx_buffer,* uint8_t * *rx_buffer*)

This function sends a DevRev command to the device.

Parameters

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

Returns

status of the operation

References DEVREV_COUNT, DEVREV_DELAY, DEVREV_EXEC_MAX, DEVREV_PARAM1_IDX, DEVREV_PARAM2_IDX, DEVREV_RSP_SIZE, ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_DEVREV, ECC108_OPCODE_IDX, and ecc108c_send_and_receive().

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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_GENDIG, ECC108_KEY_ID_MAX, ECC108_OPCODE_IDX, ECC108_OTP_BLOCK_MAX, ecc108c_send_and_receive(), GENDIG_COUNT, GENDIG_COUNT_DATA, GENDIG_DATA_IDX, GENDIG_DELAY, GENDIG_EXEC_MAX, GENDIG_KEYID_IDX, GENDIG_OTHER_DATA_SIZE, GENDIG_RSP_SIZE, GENDIG_ZONE_DATA, GENDIG_ZONE_IDX, and GENDIG_ZONE_OTP.

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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_HMAC, ECC108_OPCODE_IDX, ecc108c_send_and_receive(), HMAC_COUNT, HMAC_DELAY, HMAC_EXEC_MAX, HMAC_KEYID_IDX, HMAC_MODE_IDX, HMAC_MODE_MASK, and HMAC_RSP_SIZE.

5.1.2.6 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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_LOCK, ECC108_OPCODE_IDX, ecc108c_send_and_receive(), LOCK_COUNT, LOCK_DELAY, LOCK_EXEC_MAX, LOCK_RSP_SIZE, LOCK_SUMMARY_IDX, LOCK_ZONE_IDX, and LOCK_ZONE_MASK.

5.1.2.7 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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_MAC, ECC108_OPCODE_IDX, ecc108c_send_and_receive(), MAC_CHALLENGE_IDX, MAC_CHALLENGE_SIZE, MAC_COUNT_LONG, MAC_COUNT_SHORT, MAC_DELAY, MAC_EXEC_MAX, MAC_KEYID_IDX, MAC_MODE_IDX, MAC_MODE_MASK, and MAC_RSP_SIZE.

5.1.2.8 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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_NONCE, ECC108_OPCODE_IDX, ecc108c_send_and_receive(), NONCE_COUNT_LONG, NONCE_COUNT_SHORT, NONCE_DELAY, NONCE_EXEC_MAX, NONCE_INPUT_IDX, NONCE_MODE_IDX, NONCE_MODE_INVALID, NONCE_MODE_PASSTHROUGH, NONCE_NUMIN_SIZE, NONCE_NUMIN_SIZE_PASSTHROUGH, NONCE_PARAM2_IDX, NONCE_RSP_SIZE_LONG, and NONCE_RSP_SIZE_SHORT.

5.1.2.9 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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_OPCODE_IDX, ECC108_PAUSE, ecc108c_send_and_receive(), PAUSE_COUNT, PAUSE_DELAY, PAUSE_EXEC_MAX, PAUSE_PARAM2_IDX, PAUSE_RSP_SIZE, and PAUSE_SELECT_IDX.

5.1.2.10 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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_OPCODE_IDX, ECC108_RANDOM, ecc108c_send_and_receive(), RANDOM_COUNT, RANDOM_DELAY, RANDOM_EXEC_MAX, RANDOM_MODE_IDX, RANDOM_NO_SEED_UPDATE, RANDOM_PARAM2_IDX, RANDOM_RSP_SIZE, and RANDOM_SEED_UPDATE.

5.1.2.11 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

References ECC108_ADDRESS_MASK, ECC108_ADDRESS_MASK_CONFIG, ECC108_ADDRESS_MASK_OTP, ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_OPCODE_IDX, ECC108_READ, ECC108_ZONE_CONFIG, ECC108_ZONE_COUNT_FLAG, ECC108_ZONE_DATA, ECC108_ZONE_MASK, ECC108_ZONE_OTP, ecc108c_send_and_receive(), READ_32_RSP_SIZE, READ_4_RSP_SIZE, READ_ADDR_IDX, READ_COUNT, READ_DELAY, READ_EXEC_MAX, READ_ZONE_IDX, and READ_ZONE_MASK.

5.1.2.12 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

References ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_OPCODE_IDX, ECC108_UPDATE_EXTRA, ecc108c_send_and_receive(), UPDATE_CONFIG_BYTE_86, UPDATE_COUNT, UPDATE_DELAY, UPDATE_EXEC_MAX, UPDATE_MODE_IDX, UPDATE_RSP_SIZE, and UPDATE_VALUE_IDX.

5.1.2.13 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

References ECC108_ADDRESS_MASK, ECC108_ADDRESS_MASK_CONFIG, ECC108_ADDRESS_MASK_OTP, ECC108_BAD_PARAM, ECC108_COUNT_IDX, ECC108_CRC_SIZE, ECC108_OPCODE_IDX, ECC108_WRITE, ECC108_ZONE_ACCESS_32, ECC108_ZONE_ACCESS_4, ECC108_ZONE_CONFIG, ECC108_ZONE_COUNT_FLAG, ECC108_ZONE_DATA, ECC108_ZONE_MASK, ECC108_ZONE_OTP, ecc108c_send_and_receive(), WRITE_DELAY, WRITE_EXEC_MAX, WRITE_MAC_SIZE, WRITE_RSP_SIZE, and WRITE_ZONE_MASK.

5.1.2.14 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

References CHECKMAC_DELAY, CHECKMAC_EXEC_MAX, CHECKMAC_RSP_SIZE, DERIVE_KEY_DELAY, DERIVE_KEY_EXEC_MAX, DERIVE_KEY_RSP_SIZE, DEVREV_DELAY, DEVREV_EXEC_MAX, DEVREV_RSP_SIZE, ECC108_CHECKMAC, ECC108_CMD_SIZE_MIN, ECC108_COMMAND_EXEC_MAX, ECC108_CRC_SIZE, ECC108_DERIVE_KEY, ECC108_DEVREV, ECC108_GENDIG, ECC108_HMAC, ECC108_LOCK, ECC108_MAC, ECC108_NONCE, ECC108_PAUSE, ECC108_RANDOM, ECC108_READ, ECC108_SUCCESS, ECC108_UPDATE_EXTRA, ECC108_WRITE, ECC108_ZONE_COUNT_FLAG, ecc108c_calculate_crc(), ecc108c_send_and_receive(), GENDIG_DELAY, GENDIG_EXEC_MAX, GENDIG_RSP_SIZE, HMAC_DELAY, HMAC_EXEC_MAX, HMAC_RSP_SIZE, LOCK_DELAY, LOCK_EXEC_MAX, LOCK_RSP_SIZE, MAC_DELAY, MAC_EXEC_MAX, MAC_RSP_SIZE, NONCE_DELAY, NONCE_EXEC_MAX, NONCE_MODE_PASSTHROUGH, NONCE_RSP_SIZE_LONG, NONCE_RSP_SIZE_SHORT, PAUSE_DELAY, PAUSE_EXEC_MAX, PAUSE_RSP_SIZE, RANDOM_DELAY, RANDOM_EXEC_MAX, RANDOM_RSP_SIZE, READ_32_RSP_SIZE, READ_4_RSP_SIZE, READ_DELAY, READ_EXEC_MAX, UPDATE_DELAY, UPDATE_EXEC_MAX, UPDATE_RSP_SIZE, WRITE_DELAY, WRITE_EXEC_MAX, and WRITE_RSP_SIZE.

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

Referenced by ecc108c_check_crc(), ecc108c_send_and_receive(), and ecc108m_execute().

5.2.2.2 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

References delay_ms(), ECC108_BAD_CRC, ECC108_BUFFER_POS_COUNT, ECC108_BUFFER_POS_STATUS, ECC108_COMM_FAIL, ECC108_COMMAND_EXEC_MAX, ECC108_CRC_SIZE, ECC108_INVALID_SIZE, ECC108_RSP_SIZE_MIN, ECC108_STATUS_BYTE_WAKEUP, ECC108_SUCCESS, ecc108p_receive_response(), and ecc108p_wakeup().

Referenced by ecc108c_resync().

5.2.2.3 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

References delay_ms(), ECC108_BUFFER_POS_COUNT, ECC108_BUFFER_POS_STATUS, ECC108_CMD_FAIL, ECC108_CRC_SIZE, ECC108_FUNC_FAIL, ECC108_INVALID_SIZE, ECC108_PARSE_ERROR, ECC108_RESYNC_WITH_WAKEUP, ECC108_RETRY_COUNT, ECC108_RSP_SIZE_MIN, ECC108_RX_NO_RESPONSE, ECC108_STATUS_BYTE_COMM, ECC108_STATUS_BYTE_EXEC, ECC108_STATUS_BYTE_PARSE, ECC108_STATUS_CRC, ECC108_SUCCESS, ecc108c_calculate_crc(), ecc108c_check_crc(), ecc108c_resync(), ecc108p_receive_response(), and ecc108p_send_command().

Referenced by ecc108m_check_mac(), ecc108m_derive_key(), ecc108m_dev_rev(), ecc108m_execute(), ecc108m_gen_dig(), ecc108m_hmac(), ecc108m_lock(), ecc108m_mac(), ecc108m_nonce(), ecc108m_pause(), ecc108m_random(), ecc108m_read(), ecc108m_update_extra(), and ecc108m_write().

5.3 Module 03: Header File for Interface Abstraction Modules

This header file contains definitions and function prototypes for SWI and I^2C. 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)

  width of Wakeup pulse in 10 us units
- #define ECC108_WAKEUP_DELAY (uint8_t) (100.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

  delay between Wakeup pulse and communication in 10 us units

Functions

- uint8_t ecc108p_send_command (uint8_t count, uint8_t ∗command)

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

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

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

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

  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.

5.3.1 Detailed Description

This header file contains definitions and function prototypes for SWI and I^2C. 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 Function Documentation

5.3.2.1 uint8_t ecc108p_send_command (uint8_t *count*, uint8_t * *command*)

This SWI 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 SWI 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

References ECC108_COMM_FAIL, ECC108_I2C_PACKET_FUNCTION_NORMAL, and ECC108_SWI_FLAG_CMD.

Referenced by ecc108c_send_and_receive().

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

This SWI function receives a response from the device.

Parameters

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

Returns

status of the operation

This SWI function receives a response from the device.

Parameters

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

Returns

status of the operation

References ECC108_BUFFER_POS_COUNT, ECC108_BUFFER_POS_DATA, ECC108_COMM_FAIL, ECC108_INVALID_SIZE, ECC108_RSP_SIZE_MIN, ECC108_RX_FAIL, ECC108_RX_NO_RESPONSE, ECC108_SUCCESS, ECC108_SWI_FLAG_TX, and I2C_READ.

Referenced by ecc108c_send_and_receive(), ecc108c_wakeup(), and ecc108p_resync().

5.3.2.3 void ecc108p_set_device_id (uint8_t *id*)

This SWI function selects the GPIO pin used for communication.

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

Parameters

in	*id*	index into array of pins

This SWI function selects the GPIO pin used for communication.

Parameters

in	*id*	I2C address

5.3.2.4 uint8_t ecc108p_wakeup (void)

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

Returns

success

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

Returns

status of the operation

References delay_10us(), ECC108_COMM_FAIL, ECC108_SUCCESS, ECC108_WAKEUP_DELAY, and ECC108_WAKEUP_PULSE_WIDTH.

Referenced by ecc108c_wakeup().

5.3.2.5 uint8_t ecc108p_idle (void)

This SWI function puts the device into idle state.

Returns

status of the operation

This SWI function puts the device into idle state.

Returns

status of the operation

References ECC108_I2C_PACKET_FUNCTION_IDLE, and ECC108_SWI_FLAG_IDLE.

5.3.2.6 uint8_t ecc108p_sleep (void)

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

Returns

status of the operation

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

Returns

status of the operation

References ECC108_I2C_PACKET_FUNCTION_SLEEP, and ECC108_SWI_FLAG_SLEEP.

Referenced by ecc108c_resync().

5.3.2.7 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.

Returns

success

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

Returns

status of the operation

References ECC108_I2C_PACKET_FUNCTION_RESET, and ECC108_SUCCESS.

Referenced by ecc108p_resync().

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

This function re-synchronizes 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

This function re-synchronizes 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

References delay_ms(), ECC108_COMM_FAIL, ecc108p_receive_response(), ecc108p_reset_io(), and I2C_READ.

Referenced by ecc108c_resync().

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.
- 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 void ecc108p_set_device_id (uint8_t *id*)

This SWI function selects the GPIO pin used for communication.

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

Parameters

in	*id*	index into array of pins

5.4.2.2 uint8_t ecc108p_send_command (uint8_t *count,* uint8_t * *command*)

This SWI 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

References ECC108_COMM_FAIL, and ECC108_SWI_FLAG_CMD.

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

This SWI function receives a response from the device.

Parameters

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

Returns

status of the operation

References ECC108_BUFFER_POS_COUNT, ECC108_INVALID_SIZE, ECC108_RSP_SIZE_MIN, ECC108_RX_FAIL, ECC108_RX_NO_RESPONSE, ECC108_SUCCESS, and ECC108_SWI_FLAG_TX.

5.4.2.4 uint8_t ecc108p_wakeup (void)

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

Returns

success

References delay_10us(), ECC108_SUCCESS, ECC108_WAKEUP_DELAY, and ECC108_WAKEUP_PULSE_WIDTH.

5.4.2.5 uint8_t ecc108p_idle (void)

This SWI function puts the device into idle state.

Returns

status of the operation

References ECC108_SWI_FLAG_IDLE.

5.4.2.6 uint8_t ecc108p_sleep (void)

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

Returns

status of the operation

References ECC108_SWI_FLAG_SLEEP.

5.4.2.7 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.

Returns

success

References ECC108_SUCCESS.

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

This function re-synchronizes 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

References delay_ms(), and ecc108p_receive_response().

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 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.

References ecc108h_temp_key::check_flag, ECC108_BAD_PARAM, ECC108_NONCE, ECC108_SUCCESS, ecc108h_calculate_sha256(), ecc108h_temp_key::gen_data, ecc108h_temp_key::key_id, ecc108h_nonce_in_out::mode, NONCE_MODE_INVALID, NONCE_MODE_NO_SEED_UPDATE, NONCE_MODE_PASSTHROUGH, NONCE_MODE_SEED_UPDATE, ecc108h_nonce_in_out::num_in, ecc108h_nonce_in_out::rand_out, ecc108h_temp_key::source_flag, ecc108h_nonce_in_out::temp_key, ecc108h_temp_key::valid, and ecc108h_temp_key::value.

5.5.2.2 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.

References ecc108h_mac_in_out::challenge, ecc108h_temp_key::check_flag, ECC108_BAD_PARAM, ECC108_CMD_FAIL, ECC108_MAC, ECC108_SUCCESS, ecc108h_calculate_sha256(), ecc108h_mac_in_out::key, ecc108h_mac_in_out::key_id, MAC_MODE_BLOCK1_TEMPKEY, MAC_MODE_BLOCK2_TEMPKEY, MAC_MODE_INCLUDE_OTP_64, MAC_MODE_INCLUDE_OTP_88, MAC_MODE_INCLUDE_SN, MAC_MODE_MASK, MAC_MODE_SOURCE_FLAG_MATCH, ecc108h_mac_in_out::mode, ecc108h_mac_in_out::otp, ecc108h_mac_in_out::response, ecc108h_mac_in_out::sn, ecc108h_temp_key::source_flag, ecc108h_mac_in_out::temp_key, ecc108h_temp_key::valid, and ecc108h_temp_key::value.

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 [DATASH
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.

References ecc108h_temp_key::check_flag, ecc108h_check_mac_in_out::client_resp, ECC108_BAD_PARAM, ECC108_CMD_FAIL, ECC108_SUCCESS, ecc108h_calculate_sha256(), ecc108h_temp_key::gen_data, MAC_MODE_BLOCK2_TEMPKEY, MAC_MODE_INCLUDE_OTP_64, ecc108h_check_mac_in_out::mode, ecc108h_check_mac_in_out::other_data, ecc108h_check_mac_in_out::otp, ecc108h_check_mac_in_out::password, ecc108h_temp_key::source_flag, ecc108h_check_mac_in_out::target_key, ecc108h_check_mac_in_out::temp_key, ecc108h_temp_key::valid, and ecc108h_temp_key::value.

5.5.2.4 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.

References ecc108h_temp_key::check_flag, ECC108_BAD_PARAM, ECC108_CMD_FAIL, ECC108_HMAC, ECC108_SUCCESS, ecc108h_calculate_sha256(), HMAC_MODE_MASK, ecc108h_hmac_in_out::key, ecc108h_hmac_in_out::key_id, MAC_MODE_INCLUDE_OTP_64, MAC_MODE_INCLUDE_OTP_88, MAC_MODE_INCLUDE_SN, MAC_MODE_SOURCE_FLAG_MATCH, ecc108h_hmac_in_out::mode, ecc108h_hmac_in_out::otp, ecc108h_hmac_in_out::response, ecc108h_hmac_in_out::sn, ecc108h_temp_key::source_flag, ecc108h_hmac_in_out::temp_key, ecc108h_temp_key::valid, and ecc108h_temp_key::value.

5.5.2.5 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.

References ecc108h_temp_key::check_flag, ECC108_BAD_PARAM, ECC108_CMD_FAIL, ECC108_GENDIG, ECC108_SUCCESS, ecc108h_calculate_sha256(), ecc108h_temp_key::gen_data, GENDIG_ZONE_CONFIG, GENDIG_ZONE_DATA, GENDIG_ZONE_OTP, ecc108h_temp_key::key_id, ecc108h_gen_dig_in_out::key_id, ecc108h_gen_dig_in_out::stored_value, ecc108h_gen_dig_in_out::temp_key, ecc108h_temp_key::valid, ecc108h_temp_key::value, and ecc108h_gen_dig_in_out::zone.

5.5.2.6 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 D
Two kinds of operation are supported:
- Roll Key operation, target_key and parent_key parameters should be set to point to the same location (Targe
- 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.

References ecc108h_temp_key::check_flag, DERIVE_KEY_RANDOM_FLAG, ECC108_BAD_PARAM, ECC108_CMD_FAIL, ECC108_DERIVE_KEY, ECC108_KEY_ID_MAX, ECC108_SUCCESS, ecc108h_calculate_sha256(), ecc108h_derive_key_in_out::parent_key, ecc108h_derive_key_in_out::random, ecc108h_temp_key::source_flag, ecc108h_derive_key_in_out::target_key, ecc108h_derive_key_in_out::target_key_id, ecc108h_derive_key_in_out::temp_key, ecc108h_temp_key::valid, and ecc108h_temp_key::value.

5.5.2.7 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.

References DERIVE_KEY_RANDOM_FLAG, ECC108_BAD_PARAM, ECC108_DERIVE_KEY, ECC108_KEY_ID_MAX, ECC108_SUCCESS, ecc108h_calculate_sha256(), ecc108h_derive_key_mac_in_out::mac, ecc108h_derive_key_mac_in_out::parent_key, ecc108h_derive_key_mac_in_out::random, and ecc108h_derive_key_mac_in_out::target_key_id.

5.5.2.8 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.

References ecc108h_encrypt_in_out::address, ecc108h_temp_key::check_flag, ecc108h_encrypt_in_out::data, ECC108_ADDRESS_MASK, ECC108_BAD_PARAM, ECC108_CMD_FAIL, ECC108_SUCCESS, ECC108_WRITE, ecc108h_calculate_sha256(), ecc108h_temp_key::gen_data, ecc108h_encrypt_in_out::mac, ecc108h_temp_key::source_flag, ecc108h_encrypt_in_out::temp_key, ecc108h_temp_key::valid, ecc108h_temp_key::value, WRITE_ZONE_MASK, and ecc108h_encrypt_in_out::zone.

5.5.2.9 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.

References ecc108h_temp_key::check_flag, ecc108h_decrypt_in_out::data, ECC108_BAD_PARAM, ECC108_CMD_FAIL, ECC108_SUCCESS, ecc108h_temp_key::gen_data, ecc108h_temp_key::source_flag, ecc108h_decrypt_in_out::temp_key, ecc108h_temp_key::valid, and ecc108h_temp_key::value.

5.5.2.10 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 b
```

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

5.5.2.11 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

Referenced by ecc108h_check_mac(), ecc108h_derive_key(), ecc108h_derive_key_mac(), ecc108h_encrypt(), ecc108h_gen_dig(), ecc108h_hmac(), ecc108h_mac(), and ecc108h_nonce().

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^2C using I^2C 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.

Referenced by ecc108c_send_and_receive().

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

References delay_10us(), TIME_UTILS_LOOP_COUNT, and TIME_UTILS_US_CALIBRATION.

Referenced by delay_10us(), delay_ms(), and ecc108p_wakeup().

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

References delay_10us(), and TIME_UTILS_MS_CALIBRATION.

Referenced by ecc108c_send_and_receive(), ecc108c_wakeup(), and ecc108p_resync().

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 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.

7.1.1 Detailed Description

Communication Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

"atmel_crypto_device__library_license_stop=**End of ECC108 Library License**

7.1.2 Function Documentation

7.1.2.1 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

References ECC108_BAD_CRC, ECC108_BUFFER_POS_COUNT, ECC108_CRC_SIZE, ECC108_SUCCESS, and ecc108c_calculate_crc().

Referenced by ecc108c_send_and_receive().

7.1.2.2 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

References ECC108_RESYNC_WITH_WAKEUP, ECC108_SUCCESS, ecc108c_wakeup(), ecc108p_resync(), and ecc108p_sleep().

Referenced by ecc108c_send_and_receive().

7.2 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.

7.2.1 Detailed Description

Definitions and Prototypes for Communication Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

4. This software may only be redistributed and used in connection with an Atmel integrated circuit.

"atmel_crypto_device__library_license_stop=**End of ECC108 Library License**

7.3 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_dev_rev (uint8_t *tx_buffer, uint8_t *rx_buffer)

  This function sends a DevRev 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.

7.3.1 Detailed Description

Command Marshaling Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

7.4 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_DEVREV ((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 DevRev Command

- #define DEVREV_PARAM1_IDX ECC108_PARAM1_IDX

  DevRev command index for 1. parameter (ignored)
- #define DEVREV_PARAM2_IDX ECC108_PARAM2_IDX

  DevRev command index for 2. parameter (ignored)
- #define DEVREV_COUNT ECC108_CMD_SIZE_MIN

  DevRev command packet 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 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 DEVREV_RSP_SIZE ECC108_RSP_SIZE_VAL
  response size of DevRev 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 DEVREV_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 DEVREV_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_dev_rev (uint8_t *tx_buffer, uint8_t *rx_buffer)

  This function sends a DevRev 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.

7.4.1 Detailed Description

Definitions and Prototypes for Command Marshaling Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

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

7.5 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^2C using I^2C peripheral)*
- #define DOXYGEN_DUMMY 0

  Dummy macro that allow Doxygen to parse this group.

7.5.1 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

ECC108 Library License:

3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.

4. This software may only be redistributed and used in connection with an Atmel integrated circuit.

"atmel_crypto_device__library_license_stop=**End of ECC108 Library License**

7.6 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.

7.6.1 Detailed Description

ECC108 Helper Functions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

7.7 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.

7.7.1 Detailed Description

Declarations and Prototypes for ECC108 Helper Functions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

7.8 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.

7.8.1 Detailed Description

Functions for I2C Physical Hardware Independent Layer of ECC108 Library.

Author

Atmel Crypto Products

Date

June 20, 2013

7.8.2 Enumeration Type Documentation

7.8.2.1 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.

7.8.2.2 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

7.9 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.

7.9.1 Detailed Description

ECC108 Library Return Code Definitions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

7.10 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)

  width of Wakeup pulse in 10 us units
- #define ECC108_WAKEUP_DELAY (uint8_t) (100.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

  delay between Wakeup pulse and communication in 10 us units

Functions

- uint8_t ecc108p_send_command (uint8_t count, uint8_t ∗command)

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

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

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

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

  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.

7.10.1 Detailed Description

Definitions and Prototypes for Physical Layer Interface of ECC108 Library.

Author

Atmel Crypto Products

Date

June 21, 2013

7.11 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.
- 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.

7.11.1 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

7.12 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.

7.12.1 Detailed Description

Timer Utility Functions.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

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

"atmel_crypto_device__library_license_stop=**End of ECC108 Library License**

7.13 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.

7.13.1 Detailed Description

Timer Utility Declarations.

Author

Atmel Crypto Products

Date

June 20, 2013

Copyright

ECC108 Library License:

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

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

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

3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.

4. This software may only be redistributed and used in connection with an Atmel integrated circuit.

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

"atmel_crypto_device__library_license_stop=**End of ECC108 Library License**