SHA204 Library Examples for AVR 8-Bit Target

2.0.0

Generated by Doxygen 1.8.2

Tue Jan 22 2013 22:39:29

Contents

1 Building The Projects **1**

1.1 Work Space and Project Structure . . . 1

1.1.1 Hardware Independent Modules . . . 1

1.1.2 Hardware Dependent Modules . . . 1

1.1.3 Projects . . . 2

1.2 Tools . . . 4

1.2.1 Integrated Development Environment . . . 4

1.3 Doxygen Generated Documentation . . . 4

2 Module Index **5**

2.1 Modules . . . 5

3 Data Structure Index **7**

3.1 Data Structures . . . 7

4 File Index **9**

4.1 File List . . . 9

5 Module Documentation **11**

5.1 Module 11: Main Application . . . 11

5.1.1 Detailed Description . . . 11

5.1.2 Function Documentation . . . 11

5.1.2.1 main . . . 11

5.2 Module 12: Example Functions . . . 12

5.2.1 Detailed Description . . . 12

5.2.2 Function Documentation . . . 13

5.2.2.1 sha204e_change_i2c_address . . . 13

5.2.2.2 sha204e_check_response_status . . . 14

5.2.2.3 sha204e_checkmac_derived_key . . . 14

5.2.2.4 sha204e_checkmac_device . . . 15

5.2.2.5 sha204e_checkmac_diversified_key . . . 15
5.2.2.6 sha204e_checkmac_firmware . . . 16
5.2.2.7 sha204e_configure_derive_key . . . 16
5.2.2.8 sha204e_configure_diversify_key . . . 17
5.2.2.9 sha204e_configure_key . . . 17
5.2.2.10 sha204e_lock_config_zone . . . 17
5.2.2.11 sha204e_read_config_zone . . . 17
5.2.2.12 sha204e_read_serial_number . . . 20
5.2.2.13 sha204e_sleep . . . 20
5.2.2.14 sha204e_wakeup_device . . . 20
5.3 Module 01: Command Marshaling . . . 22
5.3.1 Detailed Description . . . 32
5.3.2 Function Documentation . . . 32
5.3.2.1 sha204m_check_mac . . . 32
5.3.2.2 sha204m_check_parameters . . . 32
5.3.2.3 sha204m_derive_key . . . 33
5.3.2.4 sha204m_dev_rev . . . 33
5.3.2.5 sha204m_execute . . . 33
5.3.2.6 sha204m_gen_dig . . . 34
5.3.2.7 sha204m_hmac . . . 34
5.3.2.8 sha204m_lock . . . 34
5.3.2.9 sha204m_mac . . . 35
5.3.2.10 sha204m_nonce . . . 35
5.3.2.11 sha204m_pause . . . 35
5.3.2.12 sha204m_random . . . 36
5.3.2.13 sha204m_read . . . 36
5.3.2.14 sha204m_update_extra . . . 36
5.3.2.15 sha204m_write . . . 37
5.4 Module 02: Communication . . . 38
5.4.1 Detailed Description . . . 39
5.4.2 Function Documentation . . . 39
5.4.2.1 sha204c_calculate_crc . . . 39
5.4.2.2 sha204c_check_crc . . . 39
5.4.2.3 sha204c_resync . . . 39
5.4.2.4 sha204c_send_and_receive . . . 40
5.4.2.5 sha204c_wakeup . . . 40
5.5 Module 03: Header File for Interface Abstraction Modules . . . 41

5.5.1 Detailed Description 41
5.5.2 Function Documentation 42
5.5.2.1 sha204p_idle 42
5.5.2.2 sha204p_receive_response 42
5.5.2.3 sha204p_reset_io 42
5.5.2.4 sha204p_resync 42
5.5.2.5 sha204p_send_command 44
5.5.2.6 sha204p_set_device_id 44
5.5.2.7 sha204p_sleep 44
5.5.2.8 sha204p_wakeup 44
5.6 Module 05: I2C Abstraction Module 46
5.6.1 Detailed Description 46
5.6.2 Enumeration Type Documentation 46
5.6.2.1 i2c_read_write_flag 46
5.6.2.2 i2c_word_address 47
5.6.3 Function Documentation 47
5.6.3.1 sha204p_idle 47
5.6.3.2 sha204p_receive_response 47
5.6.3.3 sha204p_reset_io 47
5.6.3.4 sha204p_resync 48
5.6.3.5 sha204p_send_command 48
5.6.3.6 sha204p_set_device_id 49
5.6.3.7 sha204p_sleep 49
5.6.3.8 sha204p_wakeup 49
5.7 Module 04: SWI Abstraction Module 50
5.7.1 Detailed Description 50
5.7.2 Macro Definition Documentation 51
5.7.2.1 SWI_FUNCTION_RETCODE_SUCCESS 51
5.7.3 Function Documentation 51
5.7.3.1 sha204p_idle 51
5.7.3.2 sha204p_receive_response 51
5.7.3.3 sha204p_reset_io 51
5.7.3.4 sha204p_resync 51
5.7.3.5 sha204p_send_command 52
5.7.3.6 sha204p_set_device_id 52
5.7.3.7 sha204p_sleep 52
5.7.3.8 sha204p_wakeup 53

5.8 Module 06: Helper Functions . . . 54
5.8.1 Detailed Description . . . 59
5.8.2 Function Documentation . . . 59
5.8.2.1 sha204h_calculate_crc_chain . . . 59
5.8.2.2 sha204h_calculate_sha256 . . . 59
5.8.2.3 sha204h_check_mac . . . 59
5.8.2.4 sha204h_decrypt . . . 60
5.8.2.5 sha204h_derive_key . . . 60
5.8.2.6 sha204h_derive_key_mac . . . 61
5.8.2.7 sha204h_encrypt . . . 61
5.8.2.8 sha204h_gen_dig . . . 61
5.8.2.9 sha204h_get_library_version . . . 62
5.8.2.10 sha204h_hmac . . . 62
5.8.2.11 sha204h_include_data . . . 62
5.8.2.12 sha204h_mac . . . 63
5.8.2.13 sha204h_nonce . . . 63
5.9 Module 07: Configuration Definitions . . . 64
5.9.1 Detailed Description . . . 64
5.9.2 Macro Definition Documentation . . . 65
5.9.2.1 SHA204_RETRY_COUNT . . . 65
5.10 Module 08: Library Return Codes . . . 66
5.10.1 Detailed Description . . . 66
5.11 Module 09: Timers . . . 67
5.11.1 Detailed Description . . . 67
5.11.2 Function Documentation . . . 67
5.11.2.1 delay_10us . . . 67
5.11.2.2 delay_ms . . . 67
5.12 Module 18: I2C Interface . . . 68
5.13 Module 17: SWI Configuration - GPIO . . . 69
5.14 Module 16: GPIO Interface . . . 70
5.15 Module 14: SWI Configuration - UART . . . 71
5.16 Module 13: UART Interface . . . 72
5.17 Module 15: AVR UART Definitions . . . 73

6 Data Structure Documentation 75
6.1 sha204h_calculate_sha256_in_out Struct Reference . . . 75
6.1.1 Detailed Description . . . 75

6.2 sha204h_check_mac_in_out Struct Reference 75
6.2.1 Detailed Description 76
6.3 sha204h_decrypt_in_out Struct Reference 76
6.3.1 Detailed Description 76
6.4 sha204h_derive_key_in_out Struct Reference 76
6.4.1 Detailed Description 77
6.5 sha204h_derive_key_mac_in_out Struct Reference 77
6.5.1 Detailed Description 77
6.6 sha204h_encrypt_in_out Struct Reference 78
6.6.1 Detailed Description 78
6.7 sha204h_gen_dig_in_out Struct Reference 78
6.7.1 Detailed Description 79
6.8 sha204h_hmac_in_out Struct Reference 79
6.8.1 Detailed Description 79
6.9 sha204h_include_data_in_out Struct Reference 79
6.9.1 Detailed Description 80
6.10 sha204h_mac_in_out Struct Reference 80
6.10.1 Detailed Description 80
6.11 sha204h_nonce_in_out Struct Reference 81
6.11.1 Detailed Description 81
6.12 sha204h_temp_key Struct Reference 81
6.12.1 Detailed Description 82

7 File Documentation 83
7.1 avr_compatible.h File Reference 83
7.1.1 Detailed Description 84
7.2 bitbang_config.h File Reference 85
7.2.1 Detailed Description 86
7.3 bitbang_phys.c File Reference 87
7.3.1 Detailed Description 87
7.3.2 Function Documentation 88
7.3.2.1 swi_receive_bytes 88
7.3.2.2 swi_send_byte 88
7.3.2.3 swi_send_bytes 88
7.3.2.4 swi_set_device_id 89
7.3.2.5 swi_set_signal_pin 89
7.4 i2c_phys.c File Reference 89

7.4.1 Detailed Description 90
7.4.2 Function Documentation 90
7.4.2.1 i2c_receive_byte 90
7.4.2.2 i2c_receive_bytes 91
7.4.2.3 i2c_send_bytes 91
7.4.2.4 i2c_send_start 91
7.4.2.5 i2c_send_stop 91
7.5 i2c_phys.h File Reference 92
7.5.1 Detailed Description 93
7.5.2 Macro Definition Documentation 93
7.5.2.1 I2C_BYTE_TIMEOUT 93
7.5.2.2 I2C_START_TIMEOUT 94
7.5.2.3 I2C_STOP_TIMEOUT 94
7.5.3 Function Documentation 94
7.5.3.1 i2c_receive_byte 94
7.5.3.2 i2c_receive_bytes 94
7.5.3.3 i2c_send_bytes 94
7.5.3.4 i2c_send_start 95
7.5.3.5 i2c_send_stop 95
7.6 sha204_comm.c File Reference 95
7.6.1 Detailed Description 95
7.7 sha204_comm.h File Reference 96
7.7.1 Detailed Description 97
7.8 sha204_comm_marshaling.c File Reference 98
7.8.1 Detailed Description 99
7.9 sha204_comm_marshaling.h File Reference 100
7.9.1 Detailed Description 109
7.10 sha204_config.h File Reference 110
7.10.1 Detailed Description 111
7.11 sha204_example_main.c File Reference 112
7.11.1 Detailed Description 112
7.12 sha204_examples.c File Reference 113
7.12.1 Detailed Description 114
7.13 sha204_examples.h File Reference 115
7.13.1 Detailed Description 116
7.13.2 Macro Definition Documentation 117
7.13.2.1 SHA204_CLIENT_ADDRESS 117

7.13.2.2 SHA204_EXAMPLE . . . 117
7.13.2.3 SHA204_EXAMPLE_CHANGE_I2C_ADDRESS . . . 117
7.13.2.4 SHA204_EXAMPLE_CONFIG_WITH_LOCK . . . 117
7.13.2.5 SHA204_EXAMPLE_READ_CONFIG_ZONE . . . 117
7.13.2.6 SHA204_KEY_ID . . . 117
7.14 sha204_helper.c File Reference . . . 118
7.14.1 Detailed Description . . . 118
7.15 sha204_helper.h File Reference . . . 119
7.15.1 Detailed Description . . . 121
7.16 sha204_i2c.c File Reference . . . 122
7.16.1 Detailed Description . . . 123
7.17 sha204_lib_return_codes.h File Reference . . . 124
7.17.1 Detailed Description . . . 125
7.18 sha204_physical.h File Reference . . . 126
7.18.1 Detailed Description . . . 127
7.19 sha204_swi.c File Reference . . . 128
7.19.1 Detailed Description . . . 128
7.20 swi_phys.h File Reference . . . 129
7.20.1 Detailed Description . . . 130
7.20.2 Function Documentation . . . 131
7.20.2.1 swi_receive_bytes . . . 131
7.20.2.2 swi_send_byte . . . 131
7.20.2.3 swi_send_bytes . . . 132
7.20.2.4 swi_set_device_id . . . 132
7.20.2.5 swi_set_signal_pin . . . 132
7.21 timer_utilities.c File Reference . . . 133
7.21.1 Detailed Description . . . 134
7.22 timer_utilities.h File Reference . . . 134
7.22.1 Detailed Description . . . 135
7.23 uart_config.h File Reference . . . 136
7.23.1 Detailed Description . . . 136
7.23.2 Macro Definition Documentation . . . 137
7.23.2.1 BIT_TIMEOUT . . . 137
7.24 uart_phys.c File Reference . . . 137
7.24.1 Detailed Description . . . 138
7.24.2 Function Documentation . . . 138
7.24.2.1 swi_enable . . . 138

7.24.2.2 swi_receive_bytes 139
7.24.2.3 swi_send_byte 139
7.24.2.4 swi_send_bytes 139
7.24.2.5 swi_set_device_id 139
7.24.2.6 swi_set_signal_pin 140

Index 140

Chapter 1

Building The Projects

1.1 Work Space and Project Structure

The source files for the ATSHA204 library are contained in a single folder "src".

1.1.1 Hardware Independent Modules

sha204_example_main.c

sha204_examples.c

sha204_examples.h

sha204_examples.c

sha204_helper.c

sha204_helper.h

sha204_comm_marshaling.c

sha204_comm_marshaling.h

sha204_comm.c

sha204_comm.h

sha204_i2c.c

sha204_swi.c

sha204_lib_return_codes.h

sha204_config.h

sha204_physical.h

timer_utilities.c

timer_utilities.h

1.1.2 Hardware Dependent Modules

Hardware dependent modules are provided that support 8-bit AVR micro-controllers. If you are not using an AVR CPU, either implement the functions listed in sha204_physical.h or choose the appropriate module for the physical implementation of the communication with the device from one of the communication related modules:

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

- bitbang_phys.c: Physical implementation as single wire interface (SWI) using GPIO.
- uart_phys.c: Physical implementation as single wire interface (SWI) using a UART (includes avr_compatible.h).
- i2c_phys.c: Physical implementation as two wire interface (I^2 C).

1.1.3 Projects

A solution file (.sln) is supplied for the Atmel Studio 6.1 IDE that contains three projects (.cproj). This solution file and folders (src, output, etc.) are located in the SHA204_90USB1287 folder. Choose the project that fits the communication interface you like to use.

If you don't use Atmel Studio you can easily create a project under the IDE you are using. You need the following modules and compilation switch depending on the interface and its implementation, SWI using UART, SWI using GPIO, or I^2 C.

- **SWI Using UART**

  sha204_example_main.c

  sha204_examples.c

  sha204_examples.h

  sha204_examples.c

  sha204_helper.c

  sha204_helper.h

  sha204_comm_marshaling.c

  sha204_comm_marshaling.h

  sha204_comm.c

  sha204_comm.h

  sha204_swi.c

  sha204_lib_return_codes.h

  sha204_config.h

  sha204_physical.h

  swi_phys.h

  avr_compatible.h

  uart_phys.c

  timer_utilities.c

  timer_utilities.h

  Compilation switches: SHA204_SWI, SHA204_SWI_UART, F_CPU=[your CPU clock in Hz]

- **SWI Using GPIO**

  sha204_example_main.c

  sha204_examples.c

  sha204_examples.h

  sha204_examples.c

sha204_helper.c

sha204_helper.h

sha204_comm_marshaling.c

sha204_comm_marshaling.h

sha204_comm.c

sha204_comm.h

sha204_swi.c

sha204_lib_return_codes.h

sha204_config.h

sha204_physical.h

timer_utilities.c

timer_utilities.h

swi_phys.h

bitbang_phys.c

Compilation switches: SHA204_SWI, SHA204_SWI_BITBANG, F_CPU=[your CPU clock in Hz]

- **I^2 C**

  sha204_example_main.c

  sha204_examples.c

  sha204_examples.h

  sha204_examples.c

  sha204_helper.c

  sha204_helper.h

  sha204_comm_marshaling.c

  sha204_comm_marshaling.h

  sha204_comm.c

  sha204_comm.h

  sha204_i2c.c

  sha204_lib_return_codes.h

  sha204_config.h

  sha204_physical.h

  i2c_phys.c

  timer_utilities.c

  timer_utilities.h

  Compilation switches: SHA204_I2C, F_CPU=[your CPU clock in Hz]

Follow the few steps listed below to build a SHA204 project.

- Supply communication interface independent modules by adding sha204_example_main.c, sha204_examples.∗, sha204_helper.∗, and sha204_comm∗ to the project. Be aware that all hardware independent modules include sha204_lib_return_codes.h and sha204_physical.h
- Supply communication interface hardware independent modules. For SWI add sha204_swi.∗, for I^2 C add sha204_i2c.∗. You might have to also modify sha204_i2c.c, especially for 32-bit CPUs, since their I^2 C peripherals implement such functionality in hardware. For instance, they might not support the generation of individual Start and Stop conditions.

- Supply communication interface hardware dependent modules. If you do not use an AVR CPU, you have to implement the functions in these modules. For SWI using UART add uart_phys.c, for SWI using GPIO add bitbang_phys.c, and for I^2 C add i2c_phys.∗. Be aware that uart_phys.c includes avr_compatible.h. Also, both SWI modules include swi_phys.h.
- Supply a timer utility module. You can either use the provided timer_utilities.∗ files or supply your own. The SHA204 library uses two delay functions, delay_ms(uint8_t) and delay_10us(uint8_t). The delay_ms function is used to determine command response timeouts. The delay_10us function is used to create a wake-up pulse and wake-up delay. The timer functions do not use hardware timers but loop counters. The supplied module is tuned for an AT90USB1287 CPU running at 16 MHz, but you can easily tune it for other micro-controllers as long as one loop iteration (decrement, compare, and jump) does not take longer than 10 us.

1.2 Tools

1.2.1 Integrated Development Environment

Atmel Studio 6.0.1996 - Service Pack 2

AVRGCC - 3.4.1.95, AVR Toolchain 8 Bit, Version: 3.4.1.830 - GCC 4.6.2

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

1.3 Doxygen Generated Documentation

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

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Module 01: Command Marshaling 22
Module 02: Communication 38
Module 03: Header File for Interface Abstraction Modules 41
Module 04: SWI Abstraction Module 50
Module 05: I2C Abstraction Module 46
Module 06: Helper Functions 54
Module 07: Configuration Definitions 64
Module 08: Library Return Codes 66
Module 09: Timers 67
Module 11: Main Application 11
Module 12: Example Functions 12
Module 13: UART Interface 72
Module 14: SWI Configuration - UART 71
Module 15: AVR UART Definitions 73
Module 16: GPIO Interface 70
Module 17: SWI Configuration - GPIO 69
Module 18: I2C Interface 68

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

sha204h_calculate_sha256_in_out
Input/output parameters for function sha204h_nonce() 75
sha204h_check_mac_in_out
Input/output parameters for function sha204h_check_mac() 75
sha204h_decrypt_in_out
Input/output parameters for function sha204h_decrypt() 76
sha204h_derive_key_in_out
Input/output parameters for function sha204h_derive_key() 76
sha204h_derive_key_mac_in_out
Input/output parameters for function sha204h_derive_key_mac() 77
sha204h_encrypt_in_out
Input/output parameters for function sha204h_encrypt() 78
sha204h_gen_dig_in_out
Input/output parameters for function sha204h_gen_dig() 78
sha204h_hmac_in_out
Input/output parameters for function sha204h_hmac() 79
sha204h_include_data_in_out
Input / output parameters for function sha204h_include_data() 79
sha204h_mac_in_out
Input/output parameters for function sha204h_mac() 80
sha204h_nonce_in_out
Input/output parameters for function sha204h_nonce() 81
sha204h_temp_key
Structure to hold TempKey fields 81

Chapter 4

File Index

4.1 File List

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

avr_compatible.h
AVR USART Register Compatibility Definitions 83
bitbang_config.h
Definitions for Hardware Dependent Part of ATSHA204 Physical Layer Using GPIO for Communication 85
bitbang_phys.c
Functions of Hardware Dependent Part of ATSHA204 Physical Layer Using GPIO For Communication 87
i2c_phys.c
Functions of Hardware Dependent Part of ATSHA204 Physical Layer Using I^2 C For Communication 89
i2c_phys.h
Definitions for Hardware Dependent Part of ATSHA204 Physical Layer Using I^2 C for Communication 92
sha204_comm.c
Communication Layer of ATSHA204 Library 95
sha204_comm.h
Definitions and Prototypes for Communication Layer of ATSHA204 Library 96
sha204_comm_marshaling.c
Command Marshaling Layer of ATSHA204 Library 98
sha204_comm_marshaling.h
Definitions and Prototypes for Command Marshaling Layer of ATSHA204 Library 100
sha204_config.h
Definitions for Configurable Values of the ATSHA204 Library 110
sha204_example_main.c
Main Function for Application Examples that Use the ATSHA204 Library 112
sha204_examples.c
Application examples that Use the ATSHA204 Library 113
sha204_examples.h
Application Examples That Use the ATSHA204 Library 115
sha204_helper.c
ATSHA204 Helper Functions 118
sha204_helper.h
Definitions and Prototypes for ATSHA204 Helper Functions 119
sha204_i2c.c
Functions for I^2 C Physical Hardware Independent Layer of ATSHA204 Library 122

sha204_lib_return_codes.h
Definitions for ATSHA204 Library Return Codes . . . 124
sha204_physical.h
Definitions and Prototypes for Physical Layer Interface of ATSHA204 Library . . . 126
sha204_swi.c
Functions for Single Wire, Hardware Independent Physical Layer of ATSHA204 Library . . . 128
swi_phys.h
Definitions and Prototypes for SWI Hardware Dependent Physical Layer of CryptoAuth Library . . . 129
timer_utilities.c
Timer Utility Functions . . . 133
timer_utilities.h
Timer Utility Declarations . . . 134
uart_config.h
Definitions for Hardware Dependent Part of the Physical Layer of the ATSHA204 Library Using a UART . . . 136
uart_phys.c
Physical Layer Functions of ATSHA204 Library When Using UART . . . 137

Chapter 5

Module Documentation

5.1 Module 11: Main Application

Functions

- int main (void)

  This application calls one example function that can be selected with a compilation switch defined in sha204_examples.h.

5.1.1 Detailed Description

Example functions are given that demonstrate the device. They can be selected via compilation switches (SHA204_EXAMPLE_...) found in sha204_examples.h.

Please refer to sha204_examples.c for a detailed description of those examples. Most examples implement an authentication scheme. Compiling them will give you a quick and rough overlook of RAM and flash resources. An authentication with low security (E.g. replay attacks are possible.) requires the least resources, followed by command sequences with higher security. An implementation where the expected MAC is calculated in firmware (soft SHA-256) needs the biggest resources.

The best example to start with is the SHA204_EXAMPLE_READ_CONFIG_ZONE example. Building and running it verifies that your hardware is set up correctly and communication is working. This example does not depend on any personalization of the device and does not make any modifications to the device. It only reads from the configuration zone which is always readable, independent of the lock status of the device.

5.1.2 Function Documentation

5.1.2.1 int main (void)

This application calls one example function that can be selected with a compilation switch defined in sha204_examples.h.

The example functions for SHA204_EXAMPLE_CHECKMAC_DEVICE, SHA204_EXAMPLE_CHECKMAC_FIRMWARE, and SHA204_EXAMPLE_DIVERSIFY_KEY do not return since they are running in an endless loop.

Returns

exit status of application

5.2 Module 12: Example Functions

Functions

- void sha204e_sleep ()

  This function wraps sha204p_sleep().
- uint8_t sha204e_wakeup_device (uint8_t device_id)

  This function wakes up two I^2 C devices and puts one back to sleep, effectively waking up only one device among two that share the bus.
- uint8_t sha204e_check_response_status (uint8_t ret_code, uint8_t ∗response)

  This function checks the response status byte and puts the device to sleep if there was an error.
- uint8_t sha204e_read_serial_number (uint8_t ∗tx_buffer, uint8_t ∗sn)

  This function reads the serial number from the device.
- uint8_t sha204e_lock_config_zone (uint8_t device_id)

  This function locks the configuration zone.
- uint8_t sha204e_configure_key ()

  This function configures a child and parent key for derived key scenarios.
- uint8_t sha204e_configure_derive_key ()

  This function configures the client for the derived key and diversified key example.
- uint8_t sha204e_configure_diversify_key (void)

  This function configures a client device for the diversified key example.
- uint8_t sha204e_checkmac_device (void)

  This function serves as an authentication example using the SHA204 MAC and CheckMac commands.
- uint8_t sha204e_checkmac_firmware (void)

  This function serves as an authentication example using the SHA204 Nonce, GenDig, and MAC commands.
- uint8_t sha204e_checkmac_derived_key (void)

  This function serves as an authentication example using the SHA204 Nonce, DeriveKey, and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.
- uint8_t sha204e_checkmac_diversified_key (void)

  This function serves as an authentication example using the ATSHA204 Read and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.
- uint8_t sha204e_change_i2c_address (void)

  This function changes the I^2 C address of a device.
- uint8_t sha204e_read_config_zone (uint8_t device_id, uint8_t ∗config_data)

  This function reads all 88 bytes from the configuration zone.

Variables

- const uint8_t sha204_default_key [16][SHA204_KEY_SIZE]

  key values at time of shipping

5.2.1 Detailed Description

- sha204e_checkmac_device:

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

- sha204e_checkmac_firmware:

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

- sha204e_checkmac_derive_key:

  Demonstrates high security in a host / client scenario using a DeriveKey / MAC command sequence on one device (client) and a GenDig / CheckMac sequence on another device (host). No sha256 implementation in firmware is needed. All keys are only stored on the devices and never revealed. When using I^2 C you have to change the address of one of the devices first. Connect only one device to your CPU and use sha204e_change_i2c_address to change it.

  This example needs modifications introducing the Pause command when using the SWI UART interface.

- sha204e_checkmac_diversify_key:

  Demonstrates high security in a host / client scenario using a Read / MAC command sequence on one device (client) and a GenDig / CheckMac sequence on another device (host). The MAC command uses a key id for a key that was diversified from the serial number of the client. No sha256 implementation in firmware is needed. All keys are only stored on the devices and never revealed. When using I^2 C you have to change the address of one of the devices first. Connect only one device to your CPU and use sha204e_change_i2c_address to change it.

  This example needs modifications introducing the Pause command when using the SWI UART interface.

- sha204e_change_i2c_address:

  This is a utility function that changes the I^2 C address of a device so that you can run the sha204e_checkmac-_derived_key example when using I^2 C. Make sure that you don't have more than one device with the same address sitting on the bus.

- sha204e_read_config_zone:

  This function reads all 88 bytes from the configuration zone. Since it does not depend on how the device is personalized or the lock status of the device, it is a good starting point to work with the library.

The example functions for SHA204_EXAMPLE_CHECKMAC_DEVICE and SHA204_EXAMPLE_CHECKMAC_FIRM-WARE use the sha204m_execute function that can be used to send any ATSHA204 command. The other example functions use sha204m_... command wrapper functions. Using only the sha204m_execute function in your application might compile into smaller code size compared to using the command wrapper functions. You can use any approach, but if you use the wrapper functions make sure you allow the compiler and linker to garbage collect functions or remove unused functions manually to keep code size to a minimum.

Examples that use an ATSHA204 as host you can run conveniently on an AT88CK109STK3 ("Microbase" with 3-pin "Javan" kit, SWI). When using I^2 C, you can use the AT88CK109STK8 version ("Microbase" with 8-pin "Javan" kit), but you have to change the default I^2 address of one of the two devices first.

CAUTION WHEN DEBUGGING: Be aware of the timeout feature of the device. The device will go to sleep between 0.7 and 1.7 seconds after a Wakeup. This timeout cannot be re-started by any means. It only starts after a Wakeup pulse while the device is in Idle or Sleep mode. When hitting a break point, this timeout will kick in and the device has gone to sleep before you continue debugging. Therefore, after you have examined variables you have to restart your debug session if the device was awake at that point.

5.2.2 Function Documentation

5.2.2.1 uint8_t sha204e_change_i2c_address (void)

This function changes the I^2 C address of a device.

Running it will access the device with I^2 C address SHA204_CLIENT_ADDRESS and change it to SHA204_HOST_-ADDRESS as long as the configuration zone is not locked (byte at address 87 = 0x55). Be aware that bit 3 of the I^2 C address is also used as a TTL enable bit. So make sure you give it a value that agrees with your system (see data sheet).

Returns

status of the operation

5.2.2.2 uint8_t sha204e_check_response_status (uint8_t *ret_code,* uint8_t * *response*)

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

Parameters

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

Returns

status of the operation

5.2.2.3 uint8_t sha204e_checkmac_derived_key (void)

This function serves as an authentication example using the SHA204 Nonce, DeriveKey, and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.

Creating a child key on the client allows a host device to check a MAC in a highly secure fashion. No replay attacks are possible when using a random number generated by the host device as the challenge, SHA256 calculation in firmware is not needed, and keys are only stored on the secure device.

A brief explanation for this command sequence: The client generates a child key (DeriveKey command) derived from a parent key that it shares with the host device, using a random nonce (commands Random and Nonce). It then stores it in one of its key slots. The host generates the same key and stores it in its TempKey using the same nonce. Now, when the client receives a MAC command with the child key id, a CheckMac command on the host using the TempKey will succeed.

To run this command sequence successfully the devices have to be configured first: The child key has to point to the parent, and the parent key in the host device has to be flagged as CheckOnly.

Because every time this command sequence is executed the slot for the child key is being written, this sequence does not run in a loop to prevent wearing out the flash.

Command sequence when using a derived key:

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

6. MCU to host device: CheckMac -> sha256(child key / TempKey, challenge / fixed nonce, MAC command, 3 bytes of SN)

As you can see, the sha256 input values for the MAC and the CheckMac commands are the same (child key, fixed nonce, MAC command, the three constant bytes of SN).

Returns

status of the operation

5.2.2.4 uint8_t sha204e_checkmac_device (void)

This function serves as an authentication example using the SHA204 MAC and CheckMac commands.

```
In an infinite loop, it issues the same command
sequence using the sha204m_execute command of the
Command Marshaling layer of the ATSHA204 library.
```

The command sequence wakes up the device, issues a MAC command in mode 0 using the Command Marshaling layer, puts the device to sleep, and verifies the MAC (fixed challenge / response). Then it wakes up the same (SHA204_C-LIENT_ADDRESS == SHA204_HOST_ADDRESS) or a second device, issues a CheckMac command supplying data obtained from the previous MAC command, verifies the response status byte, and puts the device to sleep.

Returns

status of the operation

5.2.2.5 uint8_t sha204e_checkmac_diversified_key (void)

This function serves as an authentication example using the ATSHA204 Read and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.

Creating a diversified key on the client using its serial number allows a host device to check a MAC using a root key on devices with different diversified keys. The host device can calculate the diversified key by using a root key and the serial number of the client.

Brief explanation for this command sequence:

During personalization, a key is derived from a root key residing in the host, and the serial number of the client. The host reads the serial number of the client, pads it with zeros, and stores it in its TempKey. It then executes a GenDig command that hashes the root key and the TempKey, a.o. Now, when the client receives a MAC command with the child key id, a CheckMac command on the host using the TempKey will succeed.

To run this command sequence successfully the host device has to be configured first: The parent key has to be flagged as CheckOnly and the child key has to point to the parent key.

Use the following sequence for secure authentication using the default configuration for the host device and modifying the default configuration for the client. (This function does this for you by calling sha204e_configure_diversify_key.)

- Point slot 10 (child key) to key id 13 (parent key) by changing the default from 0x7A (parent key = 10, roll key operation) to 0x7D (parent key = 13).
- Reset the CheckOnly flag in key 13 by changing the default from 0xDD to 0xCD.

Command sequence when using a diversified key:

1. MCU to client device: Read serial number (Read command, zone = config, address = 0).
2. MCU to host device: Get random number (Random command).
3. MCU to host device: Pad serial number with zeros and store it in TempKey (Nonce command, mode = pass-through).
4. MCU to host device: GenDig -> Host TempKey now holds child key (GenDig command, other data = DeriveKey command).
5. MCU to client device: MAC -> response = sha256(child key, challenge = random, MAC command, 3 bytes of SN)
6. MCU to host device: CheckMac -> sha256(TempKey = child key, challenge = random = provided, MAC command, 3 bytes of SN)

Returns

status of the operation

5.2.2.6 uint8_t sha204e_checkmac_firmware (void)

This function serves as an authentication example using the SHA204 Nonce, GenDig, and MAC commands.

```
In an infinite loop, it issues the same command
sequence using the Command Marshaling layer of
the ATSHA204 library.
```

The following command sequence wakes up the device, issues a Nonce, a GenDig, and a MAC command using the Command Marshaling layer, and puts the device to sleep. In parallel, it calculates in firmware the TempKey and the MAC using helper functions located in sha204_helper.c and compares the MAC command response with the calculated result.

Returns

status of the operation

5.2.2.7 uint8_t sha204e_configure_derive_key ( )

This function configures the client for the derived key and diversified key example.

```
Creating a derived key allows a host device to check a MAC
in a highly secure fashion. No replay attacks are possible
and SHA256 calculation in firmware is not needed.
```

Returns

status of the operation

5.2.2.8 uint8_t sha204e_configure_diversify_key (void)

This function configures a client device for the diversified key example.

```
After configuration is done, the diversified key is programmed with the following
command sequence:
- Read 9-byte serial number from configuration zone and pad it with 23 zeros.
- Send the zero padded serial number with a Nonce command (mode = pass-through).
- Send a DeriveKey command with the child identifier as the target.
```

Returns

status of the operation

5.2.2.9 uint8_t sha204e_configure_key ( )

This function configures a child and parent key for derived key scenarios.

```
To run this scenario successfully the client device has
to be configured first: We use a key slot in the client device that is already
configured for this purpose, but we need to point to a parent whose
CheckOnly flag is set on the host device. On the client device we have
to reset this bit, otherwise the DeriveKey command would return an error.
Key id 10 is chosen for the child key because only its parent key needs to be changed
from its default configuration. Key id 13 is chosen for the parent key because only
its CheckOnly flag has to be reset compared to its default configuration.
```

Returns

status of the operation

5.2.2.10 uint8_t sha204e_lock_config_zone (uint8_t *device_id*)

This function locks the configuration zone.

It first reads it and calculates the CRC of its content. It then sends a Lock command to the device.

This function is disabled by default with the SHA204_EXAMPLE_CONFIG_WITH_LOCK switch.

Once the configuration zone is locked, the Random command returns a number from its high quality random number generator instead of a 0xFFFF0000FFFF0000... sequence.

Parameters

in	*device_id*	which device to lock

Returns

status of the operation

5.2.2.11 uint8_t sha204e_read_config_zone (uint8_t *device_id*, uint8_t * *config_data*)

This function reads all 88 bytes from the configuration zone.

Obtain the data by putting a breakpoint after every read and inspecting "response".

Factory Defaults of Configuration Zone

01 23 76 ab 00 04 05 00 0c 8f b7 bd ee 55 01 00 c8 00 55 00 8f 80 80 a1 82 e0 a3 60 94 40 a0 85

86 40 87 07 0f 00 89 f2 8a 7a 0b 8b 0c 4c dd 4d c2 42 af 8f ff 00 ff 00 ff 00 1f 00 ff 00 1f 00

ff 00 ff 00 1f ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff 00 00 55 55

Slot Summary

Slot 1 is parent key, and slot 1 is child key (DeriveKey-Roll).

Slot 2 is parent key, and slot 0 is child key (DeriveKey-Roll).

Slot 3 is parent key, and child key has to be given in Param2 (DeriveKey-Roll).

Slots 4, 13, and 14 are CheckOnly.

Slots 5 and 15 are single use.

Slot 8 is plain text.

Slot 10 is parent key and slot 10 is child key (DeriveKey-Create).

Slot 12 is not allowed as target.

Slot Details

Byte #	Name	Value	Description
0 - 3	SN[0-3]	012376ab	part of the serial number
4 - 7	RevNum	00040500	device revision (= 4)
8 - 12	SN[4-8]	0c8fb7bdee	part of the serial number
13	Reserved	55	set by Atmel (55: First 16 bytes are unlocked / special case.)
14	I2C_Enable	01	SWI / I2C (1: I2C)
15	Reserved	00	set by Atmel
16	I2C_Address	c8	default I2C address
17	RFU	00	reserved for future use; must be 0
18	OTPmode	55	55: consumption mode, not supported at this time
19	SelectorMode	00	00: Selector can always be written with UpdateExtra command.
20	slot 0, read	8f	8: Secret. f: Does not matter.
21	slot 0, write	80	8: Never write. 0: Does not matter.
22	slot 1, read	80	8: Secret. 0: CheckMac copy
23	slot 1, write	a1	a: MAC required (roll). 1: key id
24	slot 2, read	82	8: Secret. 2: Does not matter.
25	slot 2, write	e0	e: MAC required (roll) and write encrypted. 0: key id
26	slot 3, read	a3	a: Single use. 3: Does not matter.
27	slot 3, write	60	6: Encrypt, MAC not required (roll). 0: Does not matter.
28	slot 4, read	94	9: CheckOnly. 4: Does not matter.
29	slot 4, write	40	4: Encrypt. 0: key id
30	slot 5, read	a0	a: Single use. 0: key id

31	slot 5, write	85	8: Never write. 5: Does not matter.
32	slot 6, read	86	8: Secret. 6: Does not matter.
33	slot 6, write	40	4: Encrypt. 0: key id
34	slot 7, read	87	8: Secret. 7: Does not matter.
35	slot 7, write	07	0: Write. 7: Does not matter.
36	slot 8, read	0f	0: Read. f: Does not matter.
37	slot 8, write	00	0: Write. 0: Does not matter.
38	slot 9, read	89	8: Secret. 9: Does not matter.
39	slot 9, write	f2	f: Encrypt, MAC required (create). 2: key id
40	slot 10, read	8a	8: Secret. a: Does not matter.
41	slot 10, write	7a	7: Encrypt, MAC not required (create). a: key id
42	slot 11, read	0b	0: Read. b: Does not matter.
43	slot 11, write	8b	8: Never Write. b: Does not matter.
44	slot 12, read	0c	0: Read. c: Does not matter.
45	slot 12, write	4c	4: Encrypt, not allowed as target. c: key id
46	slot 13, read	dd	d: CheckOnly. d: key id
47	slot 13, write	4d	4: Encrypt, not allowed as target. d: key id
48	slot 14, read	c2	c: CheckOnly. 2: key id
49	slot 14, write	42	4: Encrypt. 2: key id
50	slot 15, read	af	a: Single use. f: Does not matter.
51	slot 15, write	8f	8: Never write. f: Does not matter.
52	UseFlag 0	ff	8 uses
53	UpdateCount 0	00	count = 0
54	UseFlag 1	ff	8 uses
55	UpdateCount 1	00	count = 0
56	UseFlag 2	ff	8 uses
57	UpdateCount 2	00	count = 0
58	UseFlag 3	1f	5 uses
59	UpdateCount 3	00	count = 0
60	UseFlag 4	ff	8 uses
61	UpdateCount 4	00	count = 0
62	UseFlag 5	1f	5 uses
63	UpdateCount 5	00	count = 0
64	UseFlag 6	ff	8 uses
65	UpdateCount 6	00	count = 0
66	UseFlag 7	ff	8 uses
67	UpdateCount 7	00	count = 0

68 - 83	LastKeyUse	1fffffffffffffffffffffffffffffff	
84	UserExtra		
85	Selector	00	Pause command with chip id 0 leaves this device active.
86	LockValue	55	OTP and Data zones are not locked.
87	LockConfig	55	Configuration zone is not locked.

Parameters

in	*device_id*	host or client device
out	*config_data*	pointer to all 88 bytes in configuration zone. Not used if NULL.

Returns

status of the operation

5.2.2.12 uint8_t sha204e_read_serial_number (uint8_t * *tx_buffer*, uint8_t * *sn*)

This function reads the serial number from the device.

```
The serial number is stored in bytes 0 to 3 and 8 to 12
of the configuration zone.
```

Parameters

in	*tx_buffer*	pointer to transmit buffer.
out	*sn*	pointer to nine-byte serial number

Returns

status of the operation

5.2.2.13 void sha204e_sleep ( )

This function wraps sha204p_sleep().

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

5.2.2.14 uint8_t sha204e_wakeup_device (uint8_t *device_id*)

This function wakes up two I^2 C devices and puts one back to sleep, effectively waking up only one device among two that share the bus.

Parameters

in	*device_id*	which device to wake up

Returns

status of the operation

5.3 Module 01: Command Marshaling

A function is provided for every ATSHA204 command. 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 sha204m_check_parameters (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 checks the parameters for sha204m_execute().
- uint8_t sha204m_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 sha204m_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 sha204m_dev_rev (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer)

  This function sends a DevRev command to the device.
- uint8_t sha204m_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 sha204m_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 sha204m_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 sha204m_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 sha204m_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 sha204m_pause (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t selector)

  This function sends a Pause command to the device.
- uint8_t sha204m_random (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t mode)

  This function sends a Random command to the device.
- uint8_t sha204m_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 sha204m_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 sha204m_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 sha204m_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 ATSHA204 Commands

- #define SHA204_CHECKMAC ((uint8_t) 0x28)

  CheckMac command op-code.
- #define SHA204_DERIVE_KEY ((uint8_t) 0x1C)

  DeriveKey command op-code.
- #define SHA204_DEVREV ((uint8_t) 0x30)

  DevRev command op-code.
- #define SHA204_GENDIG ((uint8_t) 0x15)

  GenDig command op-code.
- #define SHA204_HMAC ((uint8_t) 0x11)

  HMAC command op-code.
- #define SHA204_LOCK ((uint8_t) 0x17)

  Lock command op-code.
- #define SHA204_MAC ((uint8_t) 0x08)

  MAC command op-code.
- #define SHA204_NONCE ((uint8_t) 0x16)

  Nonce command op-code.
- #define SHA204_PAUSE ((uint8_t) 0x01)

  Pause command op-code.
- #define SHA204_RANDOM ((uint8_t) 0x1B)

  Random command op-code.
- #define SHA204_READ ((uint8_t) 0x02)

  Read command op-code.
- #define SHA204_UPDATE_EXTRA ((uint8_t) 0x20)

  UpdateExtra command op-code.
- #define SHA204_WRITE ((uint8_t) 0x12)

  Write command op-code.

Definitions of Data and Packet Sizes

- #define SHA204_RSP_SIZE_VAL ((uint8_t) 7)

  size of response packet containing four bytes of data
- #define SHA204_KEY_SIZE (32)

  size of key
- #define SHA204_KEY_COUNT (16)

  number of keys
- #define SHA204_CONFIG_SIZE (88)

  size of configuration zone
- #define SHA204_OTP_SIZE (64)

  size of OTP zone
- #define SHA204_DATA_SIZE (SHA204_KEY_COUNT ∗ SHA204_KEY_SIZE)

  size of data zone

Definitions for Command Parameter Ranges

- #define SHA204_KEY_ID_MAX (SHA204_KEY_COUNT - 1)

    maximum value for key id
- #define SHA204_OTP_BLOCK_MAX (1)

    maximum value for OTP block

Definitions for Indexes Common to All Commands

- #define SHA204_COUNT_IDX (0)

    command packet index for count
- #define SHA204_OPCODE_IDX (1)

    command packet index for op-code
- #define SHA204_PARAM1_IDX (2)

    command packet index for first parameter
- #define SHA204_PARAM2_IDX (3)

    command packet index for second parameter
- #define SHA204_DATA_IDX (5)

    command packet index for data load

Definitions for Zone and Address Parameters

- #define SHA204_ZONE_CONFIG ((uint8_t) 0x00)

    Configuration zone.
- #define SHA204_ZONE_OTP ((uint8_t) 0x01)

    OTP (One Time Programming) zone.
- #define SHA204_ZONE_DATA ((uint8_t) 0x02)

    Data zone.
- #define SHA204_ZONE_MASK ((uint8_t) 0x03)

    Zone mask.
- #define SHA204_ZONE_COUNT_FLAG ((uint8_t) 0x80)

    Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.
- #define SHA204_ZONE_ACCESS_4 ((uint8_t) 4)

    Read or write 4 bytes.
- #define SHA204_ZONE_ACCESS_32 ((uint8_t) 32)

    Read or write 32 bytes.
- #define SHA204_ADDRESS_MASK_CONFIG (0x001F)

    Address bits 5 to 7 are 0 for Configuration zone.
- #define SHA204_ADDRESS_MASK_OTP (0x000F)

    Address bits 4 to 7 are 0 for OTP zone.
- #define SHA204_ADDRESS_MASK (0x007F)

    Address bit 7 to 15 are always 0.

Definitions for the CheckMac Command

- #define CHECKMAC_MODE_IDX SHA204_PARAM1_IDX

  CheckMAC command index for mode.
- #define CHECKMAC_KEYID_IDX SHA204_PARAM2_IDX

  CheckMAC command index for key identifier.
- #define CHECKMAC_CLIENT_CHALLENGE_IDX SHA204_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 SHA204_PARAM1_IDX

  DeriveKey command index for random bit.
- #define DERIVE_KEY_TARGETKEY_IDX SHA204_PARAM2_IDX

  DeriveKey command index for target slot.
- #define DERIVE_KEY_MAC_IDX SHA204_DATA_IDX

  DeriveKey command index for optional MAC.
- #define DERIVE_KEY_COUNT_SMALL SHA204_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 SHA204_PARAM1_IDX

  DevRev command index for 1. parameter (ignored)
- #define DEVREV_PARAM2_IDX SHA204_PARAM2_IDX

  DevRev command index for 2. parameter (ignored)
- #define DEVREV_COUNT SHA204_CMD_SIZE_MIN

  DevRev command packet size.

Definitions for the GenDig Command

- #define GENDIG_ZONE_IDX SHA204_PARAM1_IDX

  GenDig command index for zone.
- #define GENDIG_KEYID_IDX SHA204_PARAM2_IDX

  GenDig command index for key id.
- #define GENDIG_DATA_IDX SHA204_DATA_IDX

  GenDig command index for optional data.
- #define GENDIG_COUNT SHA204_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 SHA204_PARAM1_IDX

  HMAC command index for mode.
- #define HMAC_KEYID_IDX SHA204_PARAM2_IDX

  HMAC command index for key id.
- #define HMAC_COUNT SHA204_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 SHA204_PARAM1_IDX

  Lock command index for zone.
- #define LOCK_SUMMARY_IDX SHA204_PARAM2_IDX

  Lock command index for summary.
- #define LOCK_COUNT SHA204_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 SHA204_PARAM1_IDX

  MAC command index for mode.
- #define MAC_KEYID_IDX SHA204_PARAM2_IDX

  MAC command index for key id.
- #define MAC_CHALLENGE_IDX SHA204_DATA_IDX

  MAC command index for optional challenge.
- #define MAC_COUNT_SHORT SHA204_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 SHA204_PARAM1_IDX
  Nonce command index for mode.
- #define NONCE_PARAM2_IDX SHA204_PARAM2_IDX
  Nonce command index for 2. parameter.
- #define NONCE_INPUT_IDX SHA204_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 SHA204_PARAM1_IDX
  Pause command index for Selector.
- #define PAUSE_PARAM2_IDX SHA204_PARAM2_IDX
  Pause command index for 2. parameter.
- #define PAUSE_COUNT SHA204_CMD_SIZE_MIN
  Pause command packet size.

Definitions for the Random Command

- #define RANDOM_MODE_IDX SHA204_PARAM1_IDX
  Random command index for mode.
- #define RANDOM_PARAM2_IDX SHA204_PARAM2_IDX
  Random command index for 2. parameter.
- #define RANDOM_COUNT SHA204_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 SHA204_PARAM1_IDX

  Read command index for zone.
- #define READ_ADDR_IDX SHA204_PARAM2_IDX

  Read command index for address.
- #define READ_COUNT SHA204_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 SHA204_PARAM1_IDX

  UpdateExtra command index for mode.
- #define UPDATE_VALUE_IDX SHA204_PARAM2_IDX

  UpdateExtra command index for new value.
- #define UPDATE_COUNT SHA204_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 SHA204_PARAM1_IDX

  Write command index for zone.
- #define WRITE_ADDR_IDX SHA204_PARAM2_IDX

  Write command index for address.
- #define WRITE_VALUE_IDX SHA204_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 SHA204_RSP_SIZE_MIN

  response size of DeriveKey command
- #define DERIVE_KEY_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of DeriveKey command
- #define DEVREV_RSP_SIZE SHA204_RSP_SIZE_VAL

  response size of DevRev command returns 4 bytes
- #define GENDIG_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of GenDig command
- #define HMAC_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of HMAC command
- #define LOCK_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of Lock command
- #define MAC_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of MAC command
- #define NONCE_RSP_SIZE_SHORT SHA204_RSP_SIZE_MIN

  response size of Nonce command with mode[0:1] = 3
- #define NONCE_RSP_SIZE_LONG SHA204_RSP_SIZE_MAX

  response size of Nonce command
- #define PAUSE_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of Pause command
- #define RANDOM_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of Random command
- #define READ_4_RSP_SIZE SHA204_RSP_SIZE_VAL

  response size of Read command when reading 4 bytes
- #define READ_32_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of Read command when reading 32 bytes
- #define UPDATE_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of UpdateExtra command
- #define WRITE_RSP_SIZE SHA204_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 command typical execution time.
- #define DERIVE_KEY_DELAY ((uint8_t) (14.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  DeriveKey command typical execution time.
- #define DEVREV_DELAY ((uint8_t) (1))

  DevRev command typical execution time.

- #define GENDIG_DELAY ((uint8_t) (11.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

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

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

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

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

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

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

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

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

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

  Write command typical execution time.

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.

5.3.1 Detailed Description

A function is provided for every ATSHA204 command. 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, you can use instead the sha204m_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 sha204m_execute function, you can just delete the command wrapper functions. If you do use the command wrapper functions, you can respectively delete the sha204m_execute function.

5.3.2 Function Documentation

5.3.2.1 uint8_t sha204m_check_mac (uint8_t ∗ *tx_buffer*, uint8_t ∗ *rx_buffer*, uint8_t *mode*, uint8_t *key_id*, uint8_t ∗ *client_challenge*, uint8_t ∗ *client_response*, uint8_t ∗ *other_data*)

This function sends a CheckMAC command to the device.

Parameters

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

Returns

status of the operation

5.3.2.2 uint8_t sha204m_check_parameters (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 checks the parameters for sha204m_execute().

Parameters

in	*op_code*	command op-code
in	*param1*	first parameter
in	*param2*	second parameter
in	*datalen1*	number of bytes in first data block
in	*data1*	pointer to first data block
in	*datalen2*	number of bytes in second data block
in	*data2*	pointer to second data block
in	*datalen3*	number of bytes in third data block
in	*data3*	pointer to third data block
in	*tx_size*	size of tx buffer
in	*tx_buffer*	pointer to tx buffer
in	*rx_size*	size of rx buffer
out	*rx_buffer*	pointer to rx buffer

Returns

status of the operation

5.3.2.3 uint8_t sha204m_derive_key (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *random*, uint8_t *target_key*, uint8_t * *mac*)

This function sends a DeriveKey command to the device.

Parameters

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

Returns

status of the operation

5.3.2.4 uint8_t sha204m_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

5.3.2.5 uint8_t sha204m_execute (uint8_t *op_code*, uint8_t *param1*, uint16_t *param2*, uint8_t *datalen1*, uint8_t * *data1*, uint8_t *datalen2*, uint8_t * *data2*, uint8_t *datalen3*, uint8_t * *data3*, uint8_t *tx_size*, uint8_t * *tx_buffer*, uint8_t *rx_size*, uint8_t * *rx_buffer*)

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

Parameters

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

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

Returns

status of the operation

5.3.2.6 uint8_t sha204m_gen_dig (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *zone,* uint8_t *key_id,* uint8_t * *other_data*)

This function sends a GenDig command to the device.

Parameters

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

Returns

status of the operation

5.3.2.7 uint8_t sha204m_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*	selects the hash inputs
in	*key_id*	slot index of key

Returns

status of the operation

5.3.2.8 uint8_t sha204m_lock (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *zone,* uint16_t *summary*)

This function sends a Lock command to the device.

Parameters

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

Returns

status of the operation

5.3.2.9 uint8_t sha204m_mac (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *mode*, uint16_t *key_id*, uint8_t * *challenge*)

This function sends a MAC command to the device.

Parameters

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

Returns

status of the operation

5.3.2.10 uint8_t sha204m_nonce (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *mode*, uint8_t * *numin*)

This function sends a Nonce command to the device.

Parameters

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

Returns

status of the operation

5.3.2.11 uint8_t sha204m_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 go into Idle mode.

Returns

status of the operation

5.3.2.12 uint8_t sha204m_random (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *mode*)

This function sends a Random command to the device.

Parameters

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

Returns

status of the operation

5.3.2.13 uint8_t sha204m_read (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *zone,* uint16_t *address*)

This function sends a Read command to the device.

Parameters

in	*tx_buffer*	pointer to transmit buffer
out	*rx_buffer*	pointer to receive buffer
in	*zone*	0: Configuration; 1: OTP; 2: Data
in	*address*	address to read from

Returns

status of the operation

5.3.2.14 uint8_t sha204m_update_extra (uint8_t * *tx_buffer,* uint8_t * *rx_buffer,* uint8_t *mode,* uint8_t *new_value*)

This function sends an UpdateExtra command to the device.

Parameters

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

Returns

status of the operation

5.3.2.15 uint8_t sha204m_write (uint8_t * *tx_buffer*, uint8_t * *rx_buffer*, uint8_t *zone*, uint16_t *address*, uint8_t * *new_value*, uint8_t * *mac*)

This function sends a Write command to the device.

Parameters

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

Returns

status of the operation

5.4 Module 02: Communication

Macros

- #define SHA204_COMMAND_EXEC_MAX ((uint8_t) (69.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  maximum command delay
- #define SHA204_CMD_SIZE_MIN ((uint8_t) 7)

  minimum number of bytes in command (from count byte to second CRC byte)
- #define SHA204_CMD_SIZE_MAX ((uint8_t) 84)

  maximum size of command packet (CheckMac)
- #define SHA204_CRC_SIZE ((uint8_t) 2)

  number of CRC bytes
- #define SHA204_BUFFER_POS_STATUS (1)

  buffer index of status byte in status response
- #define SHA204_BUFFER_POS_DATA (1)

  buffer index of first data byte in data response
- #define SHA204_STATUS_BYTE_WAKEUP ((uint8_t) 0x11)

  status byte after wake-up
- #define SHA204_STATUS_BYTE_PARSE ((uint8_t) 0x03)

  command parse error
- #define SHA204_STATUS_BYTE_EXEC ((uint8_t) 0x0F)

  command execution error
- #define SHA204_STATUS_BYTE_COMM ((uint8_t) 0xFF)

  communication error

Functions

- uint8_t sha204c_check_crc (uint8_t ∗response)

  This function checks the consistency of a response.
- uint8_t sha204c_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:
- void sha204c_calculate_crc (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates CRC.
- uint8_t sha204c_wakeup (uint8_t ∗response)

  This function wakes up a SHA204 device and receives a response.
- uint8_t sha204c_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.

5.4.1 Detailed Description

This module implements communication with the device. It does not depend on the interface (SWI or I^2 C).

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.

5.4.2 Function Documentation

5.4.2.1 void sha204c_calculate_crc (uint8_t *length,* uint8_t * *data,* uint8_t * *crc*)

This function calculates CRC.

Parameters

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

5.4.2.2 uint8_t sha204c_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

5.4.2.3 uint8_t sha204c_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

5.4.2.4 uint8_t sha204c_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 did not get acknowledged (I^2), this function requests the device to resend the response. If the response contains an error status, this function resends the command.

Parameters

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

Returns

status of the operation

5.4.2.5 uint8_t sha204c_wakeup (uint8_t * *response*)

This function wakes up a SHA204 device and receives a response.

Parameters

out	*response*	pointer to four-byte response

Returns

status of the operation

5.5 Module 03: Header File for Interface Abstraction Modules

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

Macros

- #define SHA204_RSP_SIZE_MIN ((uint8_t) 4)

  minimum number of bytes in response
- #define SHA204_RSP_SIZE_MAX ((uint8_t) 35)

  maximum size of response packet
- #define SHA204_BUFFER_POS_COUNT (0)

  buffer index of count byte in command or response
- #define SHA204_BUFFER_POS_DATA (1)

  buffer index of data in response
- #define SHA204_WAKEUP_PULSE_WIDTH (uint8_t) (6.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

  width of Wakeup pulse in 10 us units
- #define SHA204_WAKEUP_DELAY (uint8_t) (3.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

  delay between Wakeup pulse and communication in ms

Functions

- uint8_t sha204p_send_command (uint8_t count, uint8_t ∗command)

  This function sends a command to the device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t ∗response)

  This function receives a response from the device.
- void sha204p_init (void)

  This function initializes the hardware.
- void sha204p_set_device_id (uint8_t id)

  This function sets the I^2 C address. Communication functions will use this address.
- uint8_t sha204p_wakeup (void)

  This function generates a Wake-up pulse and delays.
- uint8_t sha204p_idle (void)

  This function puts the device into idle state.
- uint8_t sha204p_sleep (void)

  This function puts the device into low-power state.
- uint8_t sha204p_reset_io (void)

  This function resets the I/O buffer of the device.
- uint8_t sha204p_resync (uint8_t size, uint8_t ∗response)

  This function resynchronizes communication.

5.5.1 Detailed Description

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

5.5.2 Function Documentation

5.5.2.1 uint8_t sha204p_idle (void)

This function puts the device into idle state.

Returns

status of the operation

5.5.2.2 uint8_t sha204p_receive_response (uint8_t *size,* uint8_t * *response*)

This 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

Parameters

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

Returns

status of the operation

5.5.2.3 uint8_t sha204p_reset_io (void)

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

Returns

status of the operation

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

Returns

success

5.5.2.4 uint8_t sha204p_resync (uint8_t *size,* uint8_t * *response*)

This function resynchronizes communication.

Parameters are not used for I^2 C.

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 I^2 C and SWI, they are implemented in the communication layer (sha204c_resync).

1. To ensure an IO channel reset, the system should send the standard I2C software reset sequence, as follows:
   - a Start condition
   - nine cycles of SCL, with SDA held high
   - another Start condition
   - a Stop condition

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

   If the chip does acknowledge the device address, the system should reset the internal address counter to force the ATSHA204 to ignore any partial input command that may have been sent. This can be accomplished by sending a write sequence to word address 0x00 (Reset), followed by a Stop condition.
2. If the chip does NOT respond to the device address with an ACK, then it may be asleep. In this case, the system should send a complete Wake token and wait t_whi after the rising edge. The system may then send another read sequence and if synchronization has completed the chip will acknowledge the device address.
3. If the chip still does not respond to the device address with an acknowledge, then it may be busy executing a command. The system should wait the longest TEXEC and then send the read sequence, which will be acknowledged by the chip.

Parameters

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

Returns

status of the operation

This function resynchronizes communication.

Re-synchronizing communication is done in a maximum of five steps listed below. This function implements the first three steps. Since steps 4 and 5 (sending a Wake-up token and reading the response) are the same for TWI and SWI, they are implemented in the communication layer (sha204c_resync).

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

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

Parameters

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

Returns

status of the operation

5.5.2.5 uint8_t sha204p_send_command (uint8_t *count*, uint8_t * *command*)

This function sends a command to the device.

Parameters

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

Returns

status of the operation

5.5.2.6 void sha204p_set_device_id (uint8_t *id*)

This function sets the I^2 C address. Communication functions will use this address.

Parameters

in	*id*	I^2 C address

This function sets the I^2 C address. Communication functions will use this address.

Parameters

in	*id*	index into array of pins

5.5.2.7 uint8_t sha204p_sleep (void)

This function puts the device into low-power state.

Returns

status of the operation

5.5.2.8 uint8_t sha204p_wakeup (void)

This function generates a Wake-up pulse and delays.

Returns

status of the operation
success

5.6 Module 05: I2C Abstraction Module

Macros

- #define SHA204_I2C_DEFAULT_ADDRESS ((uint8_t) 0xC8)

  I^2 C address used at ATSHA204 library startup.

Enumerations

- enum i2c_word_address { SHA204_I2C_PACKET_FUNCTION_RESET, SHA204_I2C_PACKET_FUNCTION_-SLEEP, SHA204_I2C_PACKET_FUNCTION_IDLE, SHA204_I2C_PACKET_FUNCTION_NORMAL }

  This enumeration lists all packet types sent to a SHA204 device.
- enum i2c_read_write_flag { I2C_WRITE = (uint8_t) 0x00, I2C_READ = (uint8_t) 0x01 }

  This enumeration lists flags for I^2 C read or write addressing.

Functions

- void sha204p_set_device_id (uint8_t id)

  This function sets the I^2 C address. Communication functions will use this address.
- void sha204p_init (void)

  This function initializes the hardware.
- uint8_t sha204p_wakeup (void)

  This function generates a Wake-up pulse and delays.
- uint8_t sha204p_send_command (uint8_t count, uint8_t ∗command)

  This function sends a command to the device.
- uint8_t sha204p_idle (void)

  This function puts the device into idle state.
- uint8_t sha204p_sleep (void)

  This function puts the device into low-power state.
- uint8_t sha204p_reset_io (void)

  This function resets the I/O buffer of the device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t ∗response)

  This function receives a response from the device.
- uint8_t sha204p_resync (uint8_t size, uint8_t ∗response)

  This function resynchronizes communication.

5.6.1 Detailed Description

These functions and definitions abstract the I2C hardware. They implement the functions declared in sha204_physical.h.

5.6.2 Enumeration Type Documentation

5.6.2.1 enum i2c_read_write_flag

This enumeration lists flags for I^2 C read or write addressing.

Enumerator:

I2C_WRITE write command flag

I2C_READ read command flag

5.6.2.2 enum i2c_word_address

This enumeration lists all packet types sent to a SHA204 device.

The following byte stream is sent to a ATSHA204 I^2 C device: {I^2 C start} {I^2 C address} {word address} [{data}] {I^2 C stop}. Data are only sent after a word address of value SHA204_I2C_PACKET_FUNCTION_NORMAL.

Enumerator:

SHA204_I2C_PACKET_FUNCTION_RESET Reset device.

SHA204_I2C_PACKET_FUNCTION_SLEEP Put device into Sleep mode.

SHA204_I2C_PACKET_FUNCTION_IDLE Put device into Idle mode.

SHA204_I2C_PACKET_FUNCTION_NORMAL Write / evaluate data that follow this word address byte.

5.6.3 Function Documentation

5.6.3.1 uint8_t sha204p_idle (void)

This function puts the device into idle state.

Returns

status of the operation

5.6.3.2 uint8_t sha204p_receive_response (uint8_t *size,* uint8_t * *response*)

This 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

5.6.3.3 uint8_t sha204p_reset_io (void)

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

Returns

status of the operation

5.6.3.4 uint8_t sha204p_resync (uint8_t *size*, uint8_t ∗ *response*)

This function resynchronizes communication.

Parameters are not used for I^2 C.

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 I^2 C and SWI, they are implemented in the communication layer (sha204c_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

5.6.3.5 uint8_t sha204p_send_command (uint8_t *count*, uint8_t ∗ *command*)

This function sends a command to the device.

Parameters

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

Returns

status of the operation

5.6.3.6 void sha204p_set_device_id (uint8_t *id*)

This function sets the I^2 C address. Communication functions will use this address.

Parameters

in	*id*	I^2 C address

5.6.3.7 uint8_t sha204p_sleep (void)

This function puts the device into low-power state.

Returns

status of the operation

5.6.3.8 uint8_t sha204p_wakeup (void)

This function generates a Wake-up pulse and delays.

Returns

status of the operation

5.7 Module 04: SWI Abstraction Module

Macros

- #define SHA204_SWI_FLAG_CMD ((uint8_t) 0x77)

  flag preceding a command
- #define SHA204_SWI_FLAG_TX ((uint8_t) 0x88)

  flag requesting a response
- #define SHA204_SWI_FLAG_IDLE ((uint8_t) 0xBB)

  flag requesting to go into Idle mode
- #define SHA204_SWI_FLAG_SLEEP ((uint8_t) 0xCC)

  flag requesting to go into Sleep mode

Functions

- void sha204p_init (void)

  This function initializes the hardware.
- void sha204p_set_device_id (uint8_t id)

  This function selects the GPIO pin used for communication. It has no effect when using a UART.
- uint8_t sha204p_send_command (uint8_t count, uint8_t ∗command)

  This function sends a command to the device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t ∗response)

  This function receives a response from the device.
- uint8_t sha204p_wakeup (void)

  This function generates a Wake-up pulse and delays.
- uint8_t sha204p_idle ()

  This function puts the device into idle state.
- uint8_t sha204p_sleep ()

  This function puts the device into low-power state.
- uint8_t sha204p_reset_io (void)

  This function is only a dummy since the functionality does not exist for the SWI version of the SHA204 device.
- uint8_t sha204p_resync (uint8_t size, uint8_t ∗response)

  This function re-synchronizes communication.

- #define SWI_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

  Communication with device succeeded.
- #define SWI_FUNCTION_RETCODE_TIMEOUT ((uint8_t) 0xF1)

  Communication timed out.
- #define SWI_FUNCTION_RETCODE_RX_FAIL ((uint8_t) 0xF9)

  Communication failed after at least one byte was received.

5.7.1 Detailed Description

These functions and definitions abstract the SWI hardware. They implement the functions declared in sha204_physical.-h.

5.7.2 Macro Definition Documentation

5.7.2.1 #define SWI_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

Communication with device succeeded.

error codes for hardware dependent module Codes in the range 0x00 to 0xF7 are shared between physical interfaces (SWI, I^2). Codes in the range 0xF8 to 0xFF are special for the particular interface.

5.7.3 Function Documentation

5.7.3.1 uint8_t sha204p_idle (void)

This function puts the device into idle state.

Returns

status of the operation

5.7.3.2 uint8_t sha204p_receive_response (uint8_t *size,* uint8_t * *response*)

This 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

5.7.3.3 uint8_t sha204p_reset_io (void)

This function is only a dummy since the functionality does not exist for the SWI version of the SHA204 device.

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

Returns

success

5.7.3.4 uint8_t sha204p_resync (uint8_t *size,* uint8_t * *response*)

This function re-synchronizes communication.

This function resynchronizes communication.

Re-synchronizing communication is done in a maximum of five steps listed below. This function implements the first three steps. Since steps 4 and 5 (sending a Wake-up token and reading the response) are the same for TWI and SWI, they are implemented in the communication layer (sha204c_resync).

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

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

Parameters

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

Returns

status of the operation

5.7.3.5 uint8_t sha204p_send_command (uint8_t *count*, uint8_t * *command*)

This function sends a command to the device.

Parameters

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

Returns

status of the operation

5.7.3.6 void sha204p_set_device_id (uint8_t *id*)

This function selects the GPIO pin used for communication. It has no effect when using a UART.

This function sets the I^2 C address. Communication functions will use this address.

Parameters

`in`	*id*	index into array of pins

5.7.3.7 uint8_t sha204p_sleep (void)

This function puts the device into low-power state.

Returns

status of the operation

5.7.3.8 uint8_t sha204p_wakeup (void)

This function generates a Wake-up pulse and delays.

Returns

success

5.8 Module 06: Helper Functions

Use these functions if your system does not use an ATSHA204 as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATSHA204 client device. They are intended to accompany the ATSHA204 library functions. They can be called directly from an application, or integrated into an API.

Data Structures

- struct sha204h_temp_key

  Structure to hold TempKey fields.
- struct sha204h_include_data_in_out

  Input / output parameters for function sha204h_include_data().
- struct sha204h_calculate_sha256_in_out

  Input/output parameters for function sha204h_nonce().
- struct sha204h_nonce_in_out

  Input/output parameters for function sha204h_nonce().
- struct sha204h_mac_in_out

  Input/output parameters for function sha204h_mac().
- struct sha204h_hmac_in_out

  Input/output parameters for function sha204h_hmac().
- struct sha204h_gen_dig_in_out

  Input/output parameters for function sha204h_gen_dig().
- struct sha204h_derive_key_in_out

  Input/output parameters for function sha204h_derive_key().
- struct sha204h_derive_key_mac_in_out

  Input/output parameters for function sha204h_derive_key_mac().
- struct sha204h_encrypt_in_out

  Input/output parameters for function sha204h_encrypt().
- struct sha204h_decrypt_in_out

  Input/output parameters for function sha204h_decrypt().
- struct sha204h_check_mac_in_out

  Input/output parameters for function sha204h_check_mac().

Functions

- char ∗ sha204h_get_library_version (void)

  This function returns the library version. The version consists of three bytes. For a released version, the last byte is 0.
- uint8_t sha204h_nonce (struct sha204h_nonce_in_out ∗param)

  This function calculates a 32-byte nonce based on a 20-byte input value (param->num_in) and 32-byte random number (param->rand_out).
- uint8_t sha204h_mac (struct sha204h_mac_in_out ∗param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- uint8_t sha204h_check_mac (struct sha204h_check_mac_in_out ∗param)

  This function calculates a SHA-256 digest (MAC) of a password and other information, to be verified using the CheckMac device command.
- uint8_t sha204h_hmac (struct sha204h_hmac_in_out ∗param)

  This function generates an HMAC / SHA-256 hash of a key and other information.

- uint8_t sha204h_gen_dig (struct sha204h_gen_dig_in_out ∗param)

    This function combines the current TempKey with a stored value.
- uint8_t sha204h_derive_key (struct sha204h_derive_key_in_out ∗param)

    This function combines a key with the TempKey.
- uint8_t sha204h_derive_key_mac (struct sha204h_derive_key_mac_in_out ∗param)

    This function calculates the input MAC for a DeriveKey command.
- uint8_t sha204h_encrypt (struct sha204h_encrypt_in_out ∗param)

    This function encrypts 32-byte plain text data to be written using Write opcode, and optionally calculates input MAC.
- uint8_t sha204h_decrypt (struct sha204h_decrypt_in_out ∗param)

    This function decrypts 32-byte encrypted data received with the Read command.
- void sha204h_calculate_crc_chain (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

    This function calculates the packet CRC.
- void sha204h_calculate_sha256 (int32_t len, uint8_t ∗message, uint8_t ∗digest)

    This function creates a SHA256 digest on a little-endian system.
- uint8_t ∗ sha204h_include_data (struct sha204h_include_data_in_out ∗param)

    This function copies otp and sn data into a command buffer.

Variables

- uint8_t value [SHA204_KEY_SIZE]

    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.
- uint8_t ∗ p_temp

    [out] pointer to output buffer
- uint8_t ∗ otp

    [in] pointer to one-time-programming data
- uint8_t ∗ sn

    [out] pointer to serial number data
- 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 sha204h_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 sha204h_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 sha204h_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 sha204h_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 sha204h_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 ∗ crypto_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 sha204h_temp_key ∗ temp_key

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

  [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.
- struct sha204h_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 sha204h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

Definitions for SHA204 Message Sizes to Calculate a SHA256 Hash

"||" is the concatenation operator. The number in braces is the length of the hash input value in bytes.

- #define SHA204_MSG_SIZE_NONCE (55)

  RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.
- #define SHA204_MSG_SIZE_MAC (88)

  (Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8-_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}
- #define SHA204_MSG_SIZE_HMAC_INNER (152)

  HMAC = sha(HMAC outer || HMAC inner) HMAC inner = sha((zero-padded key ^ ipad) || message) = sha256((Key{32} || 0x36{32}) || 0{32} || Key{32} || OpCode{1} || Mode{1} || KeyId{2} || OTP0_7{8} || OTP8_10{3} || SN8{1} || SN4_7{4} || SN0_1{2} || SN2_3{2}){32}.
- #define SHA204_MSG_SIZE_HMAC (96)

  HMAC = sha(HMAC outer || HMAC inner) = sha256((Key{32} || 0x5C{32}) || HMAC inner{32})
- #define SHA204_MSG_SIZE_GEN_DIG (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define SHA204_MSG_SIZE_DERIVE_KEY (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define SHA204_MSG_SIZE_DERIVE_KEY_MAC (39)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.
- #define SHA204_MSG_SIZE_ENCRYPT_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define **SHA204_COMMAND_HEADER_SIZE** (4)
- #define **SHA204_GENDIG_ZEROS_SIZE** (25)
- #define **SHA204_DERIVE_KEY_ZEROS_SIZE** (25)
- #define **SHA204_OTP_SIZE_8** (8)
- #define **SHA204_OTP_SIZE_3** (3)
- #define **SHA204_SN_SIZE_4** (4)
- #define **SHA204_SN_SIZE_2** (2)
- #define **SHA204_OTHER_DATA_SIZE_2** (2)
- #define **SHA204_OTHER_DATA_SIZE_3** (3)
- #define **SHA204_OTHER_DATA_SIZE_4** (4)
- #define **HMAC_BLOCK_SIZE** (64)
- #define **SHA204_PACKET_OVERHEAD** (3)

Fixed Byte Values of Serial Number (SN[0:1] and SN[8])

- #define **SHA204_SN_0** (0x01)
- #define **SHA204_SN_1** (0x23)
- #define **SHA204_SN_8** (0xEE)

Definition for TempKey Mode

- #define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t) 0x03)

  mode mask for MAC command when using TempKey

5.8.1 Detailed Description

Use these functions if your system does not use an ATSHA204 as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATSHA204 client device. They are intended to accompany the ATSHA204 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 ATSHA204 as a host. Or, if you don't, delete the functions you do not use.

5.8.2 Function Documentation

5.8.2.1 void sha204h_calculate_crc_chain (uint8_t *length,* uint8_t ∗ *data,* uint8_t ∗ *crc*)

This function calculates the packet CRC.

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

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

5.8.2.2 void sha204h_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 CryptoAuth device in mind. It will therefore only work for length values of len % 64 $<$ 62.

Parameters

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

5.8.2.3 uint8_t sha204h_check_mac (struct sha204h_check_mac_in_out ∗ *param*)

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

This password checking operation is described in "Section 3.3.6 Password Checking" of "Atmel ATSHA204 [DATASH-EET]" (8740C-CRYPTO-7/11). Before performing password checking operation, TempKey should contain a randomly generated nonce. The TempKey in the device has to match the one in the application. A user enters the password to be verified by an application. The application passes this password to the CheckMac calculation function, along with 13 bytes of OtherData, a 32-byte target key, and optionally 11 bytes of OTP. The function calculates a 32-byte ClientResp, returns it to Application. The function also replaces the current TempKey value with the target key. The application passes the calculated ClientResp along with OtherData inside a CheckMac command to the device. The device validates ClientResp, and copies the target slot to its TempKey.

If the password is stored in an odd numbered slot, the target slot is the password slot itself, so the target_key parameter should point to the password being checked. If the password is stored in an even numbered slot, the target slot is the next odd numbered slot (KeyID + 1), so the target_key parameter should point to a key that is equal to the target slot in the device.

Note that the function does not check the result of the password checking operation. Regardless of whether the CheckMac command returns success or not, the TempKey variable of the application will hold the value of the target key. Therefore the application has to make sure that password checking operation succeeds before using the TempKey for subsequent operations.

Parameters

`in,out`	*param*	pointer to parameter structure

Returns

status of the operation

5.8.2.4 uint8_t sha204h_decrypt (struct **sha204h_decrypt_in_out** * *param*)

This function decrypts 32-byte encrypted data received with the Read command.

To use this function, first the nonce must be valid and synchronized between device and application. The application sends a GenDig command to the Device, using a key specified by SlotConfig.ReadKey. The device updates its TempKey. The application then updates its own TempKey using the GenDig calculation function, using the same key. The application sends a Read command to the device for a user zone configured with EncryptRead. The device encrypts 32-byte zone content, and outputs it to the host. The application passes these encrypted data to this decryption function. The function decrypts the data and returns them. TempKey must be updated by GenDig using a ParentKey as specified by SlotConfig.ReadKey before executing this function. The decryption function does not check whether the TempKey has been generated by a correct ParentKey for the corresponding zone. Therefore to get a correct result, the application has to make sure that prior GenDig calculation was done using correct ParentKey.

Parameters

`in,out`	*param*	pointer to parameter structure

Returns

status of the operation

5.8.2.5 uint8_t sha204h_derive_key (struct **sha204h_derive_key_in_out** * *param*)

This function combines a key with the TempKey.

Used in conjunction with DeriveKey command, the key derived by this function will match the key in the device. Two kinds of operation are supported:

- Roll Key operation: target_key and parent_key parameters should be set to point to the same location (TargetKey).
- Create Key operation: target_key should be set to point to TargetKey, parent_key should be set to point to ParentKey.

After executing this function, the 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*	pointer to parameter structure

Returns

status of the operation

5.8.2.6 uint8_t sha204h_derive_key_mac (struct sha204h_derive_key_mac_in_out * *param*)

This function calculates the input MAC for a DeriveKey command.

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

Parameters

`in,out`	*param*	pointer to parameter structure

Returns

status of the operation

5.8.2.7 uint8_t sha204h_encrypt (struct sha204h_encrypt_in_out * *param*)

This function encrypts 32-byte plain text 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. The application sends a GenDig command to the device, using a parent key. If the Data zone has been locked, this is specified by SlotConfig.WriteKey. The device updates its TempKey when executing the command. The application then updates its own TempKey using the GenDig calculation function, using the same key. The application passes the plain text data to the encryption function.

If input MAC is needed the application must pass a valid pointer to buffer in the "mac" command parameter. If input MAC is not needed the application can pass a NULL pointer in the "mac" command parameter. The function encrypts the data and optionally calculates the input MAC, and returns it to the application. Using these encrypted data and the input MAC, the application sends a Write command to the Device. The device validates the MAC, then decrypts and writes the data.

The encryption function does not check whether the TempKey has been generated by the correct ParentKey for the corresponding zone. Therefore, to get a correct result after the Data and OTP zones have been locked, the application has to make sure that prior GenDig calculation was done using the correct ParentKey.

Parameters

`in,out`	*param*	pointer to parameter structure

Returns

status of the operation

5.8.2.8 uint8_t sha204h_gen_dig (struct sha204h_gen_dig_in_out * *param*)

This function combines the 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 when executing a GenDig command. The TempKey should be valid (temp_key.valid = 1) before executing this function. To use this function, an application first sends a GenDig command with a chosen stored value to the device. This stored value must be known by the application and is passed to this GenDig calculation function. The function calculates a new TempKey and returns it.

Parameters

`in,out`	*param*	pointer to parameter structure

Returns

status of the operation

5.8.2.9 char∗ sha204h_get_library_version (void)

This function returns the library version. The version consists of three bytes. For a released version, the last byte is 0.

Returns

pointer to the version string

5.8.2.10 uint8_t sha204h_hmac (struct sha204h_hmac_in_out ∗ *param*)

This function generates an HMAC / SHA-256 hash of a key and other information.

The resulting hash will match with the one generated in the device by an HMAC command. The TempKey has to be valid (temp_key.valid = 1) before executing this function.

Parameters

`in,out`	*param*	pointer to parameter structure

Returns

status of the operation

5.8.2.11 uint8_t∗ sha204h_include_data (struct sha204h_include_data_in_out ∗ *param*)

This function copies otp and sn data into a command buffer.

Parameters

`in,out`	*param*	pointer to parameter structure

Returns

pointer to command buffer byte that was copied last

5.8.2.12 uint8_t sha204h_mac (struct sha204h_mac_in_out * *param*)

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

The resulting digest will match with the one generated by the device when executing a MAC command. The TempKey (if used) should be valid (temp_key.valid = 1) before executing this function.

Parameters

in,out	*param*	pointer to parameter structure

Returns

status of the operation

5.8.2.13 uint8_t sha204h_nonce (struct sha204h_nonce_in_out * *param*)

This function calculates a 32-byte nonce based on a 20-byte input value (param->num_in) and 32-byte random number (param->rand_out).

This nonce will match with the nonce generated in the device when executing a Nonce command. To use this function, an application first sends a Nonce command with a chosen param->num_in to the device. Nonce 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 the random number param->rand_out it used in the hash calculation to the host. The values of param->rand_out and param->num_in are passed to this nonce calculation function. The function calculates the nonce and returns it. This function can also be used to fill in the nonce directly to TempKey (pass-through mode). The flags will automatically be set according to the mode used.

Parameters

in,out	*param*	pointer to parameter structure

Returns

status of the operation

5.9 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 SHA204_RETRY_COUNT (1)

  number of command / response retries

Available Definitions for Interfaces

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

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

- #define DOXYGEN_DUMMY 0

  Dummy macro that allow Doxygen to parse this group.

Configuration Definitions for SWI (UART) Interface

- #define SWI_RECEIVE_TIME_OUT ((uint16_t) 153)

  receive timeout in us instead of loop counts
- #define SWI_US_PER_BYTE ((uint16_t) 313)

  It takes 312.5 us to send a byte (9 single-wire bits / 230400 Baud ∗ 8 flag bits).
- #define SHA204_RESPONSE_TIMEOUT ((uint16_t) SWI_RECEIVE_TIME_OUT + SWI_US_PER_BYTE)

  SWI response timeout is the sum of receive timeout and the time it takes to send the TX flag.

Configuration Definitions for SWI Interface, Common to GPIO and UART

- #define SHA204_SYNC_TIMEOUT ((uint8_t) 85)

  delay before sending a transmit flag in the synchronization routine

5.9.1 Detailed Description

Tune the values of these timing definitions for your system. Always include this file no matter whether you use SWI or I^2 C. Please refer to the actual file because Doxygen cannot parse nested macros with the same name.

5.9.2 Macro Definition Documentation

5.9.2.1 #define SHA204_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 SHA204_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 SHA204_COMMAND_EXEC_MAX.

5.10 Module 08: Library Return Codes

Macros

- #define SHA204_SUCCESS ((uint8_t) 0x00)

  Function succeeded.
- #define SHA204_CHECKMAC_FAILED ((uint8_t) 0xD1)

  response status byte indicates CheckMac failure
- #define SHA204_PARSE_ERROR ((uint8_t) 0xD2)

  response status byte indicates parsing error
- #define SHA204_CMD_FAIL ((uint8_t) 0xD3)

  response status byte indicates command execution error
- #define SHA204_STATUS_CRC ((uint8_t) 0xD4)

  response status byte indicates CRC error
- #define SHA204_STATUS_UNKNOWN ((uint8_t) 0xD5)

  response status byte is unknown
- #define SHA204_FUNC_FAIL ((uint8_t) 0xE0)

  Function could not execute due to incorrect condition / state.
- #define SHA204_GEN_FAIL ((uint8_t) 0xE1)

  unspecified error
- #define SHA204_BAD_PARAM ((uint8_t) 0xE2)

  bad argument (out of range, null pointer, etc.)
- #define SHA204_INVALID_ID ((uint8_t) 0xE3)

  invalid device id, id not set
- #define SHA204_INVALID_SIZE ((uint8_t) 0xE4)

  Count value is out of range or greater than buffer size.
- #define SHA204_BAD_CRC ((uint8_t) 0xE5)

  incorrect CRC received
- #define SHA204_RX_FAIL ((uint8_t) 0xE6)

  Timed out while waiting for response. Number of bytes received is $>$ 0.
- #define SHA204_RX_NO_RESPONSE ((uint8_t) 0xE7)

  Not an error while the Command layer is polling for a command response.
- #define SHA204_RESYNC_WITH_WAKEUP ((uint8_t) 0xE8)

  Re-synchronization succeeded, but only after generating a Wake-up.
- #define SHA204_COMM_FAIL ((uint8_t) 0xF0)

  Communication with device failed. Same as in hardware dependent modules.
- #define SHA204_TIMEOUT ((uint8_t) 0xF1)

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

5.10.1 Detailed Description

5.11 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.

5.11.1 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.

5.11.2 Function Documentation

5.11.2.1 void delay_10us (uint8_t *delay*)

This function delays for a number of tens of microseconds.

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

Parameters

in	*delay*	number of 0.01 milliseconds to delay

5.11.2.2 void delay_ms (uint8_t *delay*)

This function delays for a number of milliseconds.

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

Parameters

in	*delay*	number of milliseconds to delay

5.12 Module 18: I2C Interface

Definitions are supplied for various I^2 C configuration values such as clock, timeouts, and error codes.

5.13 Module 17: SWI Configuration - GPIO

Two definition blocks are supplied:

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

5.14 Module 16: GPIO Interface

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

5.15 Module 14: SWI Configuration - UART

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

5.16 Module 13: UART Interface

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

5.17 Module 15: AVR UART Definitions

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

Chapter 6

Data Structure Documentation

6.1 sha204h_calculate_sha256_in_out Struct Reference

Input/output parameters for function sha204h_nonce().

```
#include <sha204_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 sha204h_nonce().

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

- sha204_helper.h

6.2 sha204h_check_mac_in_out Struct Reference

Input/output parameters for function sha204h_check_mac().

```
#include <sha204_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 sha204h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

6.2.1 Detailed Description

Input/output parameters for function sha204h_check_mac().

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

- sha204_helper.h

6.3 sha204h_decrypt_in_out Struct Reference

Input/output parameters for function sha204h_decrypt().

```
#include <sha204_helper.h>
```

Data Fields

- uint8_t ∗ crypto_data

  [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.
- struct sha204h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

6.3.1 Detailed Description

Input/output parameters for function sha204h_decrypt().

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

- sha204_helper.h

6.4 sha204h_derive_key_in_out Struct Reference

Input/output parameters for function sha204h_derive_key().

```
#include <sha204_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 sha204h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

6.4.1 Detailed Description

Input/output parameters for function sha204h_derive_key().

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

- sha204_helper.h

6.5 sha204h_derive_key_mac_in_out Struct Reference

Input/output parameters for function sha204h_derive_key_mac().

```
#include <sha204_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 sha204h_derive_key_mac().

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

- sha204_helper.h

6.6 sha204h_encrypt_in_out Struct Reference

Input/output parameters for function sha204h_encrypt().

`#include <sha204_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 * crypto_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 sha204h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

6.6.1 Detailed Description

Input/output parameters for function sha204h_encrypt().

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

- sha204_helper.h

6.7 sha204h_gen_dig_in_out Struct Reference

Input/output parameters for function sha204h_gen_dig().

`#include <sha204_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 sha204h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

6.7.1 Detailed Description

Input/output parameters for function sha204h_gen_dig().

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

- sha204_helper.h

6.8 sha204h_hmac_in_out Struct Reference

Input/output parameters for function sha204h_hmac().

`#include <sha204_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 sha204h_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

6.8.1 Detailed Description

Input/output parameters for function sha204h_hmac().

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

- sha204_helper.h

6.9 sha204h_include_data_in_out Struct Reference

Input / output parameters for function sha204h_include_data().

`#include <sha204_helper.h>`

Data Fields

- uint8_t ∗ p_temp

    [out] pointer to output buffer
- uint8_t ∗ otp

    [in] pointer to one-time-programming data
- uint8_t ∗ sn

    [out] pointer to serial number data

6.9.1 Detailed Description

Input / output parameters for function sha204h_include_data().

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

- sha204_helper.h

6.10 sha204h_mac_in_out Struct Reference

Input/output parameters for function sha204h_mac().

```
#include <sha204_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 sha204h_temp_key ∗ temp_key

    [in,out] Pointer to TempKey structure.

6.10.1 Detailed Description

Input/output parameters for function sha204h_mac().

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

- sha204_helper.h

6.11 sha204h_nonce_in_out Struct Reference

Input/output parameters for function sha204h_nonce().

`#include <sha204_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 sha204h_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

6.11.1 Detailed Description

Input/output parameters for function sha204h_nonce().

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

- sha204_helper.h

6.12 sha204h_temp_key Struct Reference

Structure to hold TempKey fields.

`#include <sha204_helper.h>`

Data Fields

- uint8_t value [SHA204_KEY_SIZE]

  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.

6.12.1 Detailed Description

Structure to hold TempKey fields.

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

- sha204_helper.h

Chapter 7

File Documentation

7.1 avr_compatible.h File Reference

AVR USART Register Compatibility Definitions.

Macros

- #define UCSRA UCSR1A

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

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

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

  UART data register.
- #define UBRRL UBRR1L

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

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

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

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

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

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

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

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

  UART double-speed (bit 1, register A)

- #define MPCM MPCM1

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

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

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

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

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

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

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

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

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

7.1.1 Detailed Description

AVR USART Register Compatibility Definitions.

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

ATSHA204 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.

1. 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.

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

7.2 bitbang_config.h File Reference

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

Macros

- #define swi_enable_interrupts sei

  enable interrupts
- #define swi_disable_interrupts cli

  disable interrupts
- #define SIG2_BIT (2)

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

  identifier for client
- #define PORT_DDR (DDRD)

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

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

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

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

  identifier for host
- #define DEBUG_LOW

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

Macros for Bit-Banged SWI Timing

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

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

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

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

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

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

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

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

7.2.1 Detailed Description

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

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

ATSHA204 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.

1. 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.

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

1. 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, INCLUDI-NG, 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 D-AMAGES (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.

End of ATSHA204 Library License

7.3 bitbang_phys.c File Reference

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

Functions

- void swi_set_device_id (uint8_t id)

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

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

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

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

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

    This GPIO function receives bytes from an SWI device.

7.3.1 Detailed Description

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

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

7.3.2 Function Documentation

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

This GPIO function receives bytes from an SWI device.

Parameters

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

Returns

status of the operation

7.3.2.2 uint8_t swi_send_byte (uint8_t *value*)

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

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

This GPIO function sends bytes to an SWI device.

Parameters

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

Returns

status of the operation

7.3.2.4 void swi_set_device_id (uint8_t *id*)

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

Parameters

in	*id*	client if zero, otherwise host

Returns

status of the operation

7.3.2.5 void swi_set_signal_pin (uint8_t *is_high*)

This GPIO function sets the signal pin low or high.

Parameters

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

7.4 i2c_phys.c File Reference

Functions of Hardware Dependent Part of ATSHA204 Physical Layer Using I^2 C For Communication.

Functions

- void i2c_enable (void)

  This function initializes and enables the I^2 C peripheral.
- void i2c_disable (void)

  This function disables the I^2 C peripheral.
- uint8_t i2c_send_start (void)

  This function creates a Start condition (SDA low, then SCL low).
- uint8_t i2c_send_stop (void)

  This function creates a Stop condition (SCL high, then SDA high).
- uint8_t i2c_send_bytes (uint8_t count, uint8_t *data)

  This function sends bytes to an I^2 C device.
- uint8_t i2c_receive_byte (uint8_t *data)

  This function receives one byte from an I^2 C device.
- uint8_t i2c_receive_bytes (uint8_t count, uint8_t *data)

  This function receives bytes from an I^2 C device and sends a Stop.

7.4.1 Detailed Description

Functions of Hardware Dependent Part of ATSHA204 Physical Layer Using I^2 C For Communication.

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

7.4.2 Function Documentation

7.4.2.1 uint8_t i2c_receive_byte (uint8_t * *data*)

This function receives one byte from an I^2 C device.

Parameters

out	*data*	pointer to received byte

Returns

status of the operation

7.4.2.2 uint8_t i2c_receive_bytes (uint8_t *count*, uint8_t * *data*)

This function receives bytes from an I^2 C device and sends a Stop.

Parameters

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

Returns

status of the operation

7.4.2.3 uint8_t i2c_send_bytes (uint8_t *count*, uint8_t * *data*)

This function sends bytes to an I^2 C device.

Parameters

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

Returns

status of the operation

7.4.2.4 uint8_t i2c_send_start (void)

This function creates a Start condition (SDA low, then SCL low).

Returns

status of the operation

7.4.2.5 uint8_t i2c_send_stop (void)

This function creates a Stop condition (SCL high, then SDA high).

Returns

status of the operation

7.5 i2c_phys.h File Reference

Definitions for Hardware Dependent Part of ATSHA204 Physical Layer Using I^2 C for Communication.

Macros

- #define I2C_CLOCK (400000.0)

  I2C clock.
- #define I2C_PULLUP

  Use pull-up resistors.
- #define I2C_START_TIMEOUT ((uint8_t) 250)

  number of polling iterations for TWINT bit in TWSR after creating a Start condition in i2c_send_start()
- #define I2C_BYTE_TIMEOUT ((uint8_t) 200)

  number of polling iterations for TWINT bit in TWSR after sending or receiving a byte.
- #define I2C_STOP_TIMEOUT ((uint8_t) 250)

  number of polling iterations for TWSTO bit in TWSR after creating a Stop condition in i2c_send_stop().
- #define I2C_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

  Communication with device succeeded.
- #define I2C_FUNCTION_RETCODE_COMM_FAIL ((uint8_t) 0xF0)

  Communication with device failed.
- #define I2C_FUNCTION_RETCODE_TIMEOUT ((uint8_t) 0xF1)

  Communication timed out.
- #define I2C_FUNCTION_RETCODE_NACK ((uint8_t) 0xF8)

  TWI nack.

Functions

- void i2c_enable (void)

  This function initializes and enables the I^2 C peripheral.
- void i2c_disable (void)

  This function disables the I^2 C peripheral.
- uint8_t i2c_send_start (void)

  This function creates a Start condition (SDA low, then SCL low).
- uint8_t i2c_send_stop (void)

  This function creates a Stop condition (SCL high, then SDA high).
- uint8_t i2c_send_bytes (uint8_t count, uint8_t ∗data)

  This function sends bytes to an I^2 C device.
- uint8_t i2c_receive_byte (uint8_t ∗data)

  This function receives one byte from an I^2 C device.
- uint8_t i2c_receive_bytes (uint8_t count, uint8_t ∗data)

  This function receives bytes from an I^2 C device and sends a Stop.

7.5.1 Detailed Description

Definitions for Hardware Dependent Part of ATSHA204 Physical Layer Using I^2 C for Communication.

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

7.5.2 Macro Definition Documentation

7.5.2.1 #define I2C_BYTE_TIMEOUT ((uint8_t) 200)

number of polling iterations for TWINT bit in TWSR after sending or receiving a byte.

Adjust this value considering how long it takes to check a status bit in the TWI status register, decrement the timeout counter, compare its value with 0, branch, and to send or receive one byte.

7.5.2.2 #define I2C_START_TIMEOUT ((uint8_t) 250)

number of polling iterations for TWINT bit in TWSR after creating a Start condition in i2c_send_start()

Adjust this value considering how long it takes to check a status bit in the TWI status register, decrement the timeout counter, compare its value with 0, and branch.

7.5.2.3 #define I2C_STOP_TIMEOUT ((uint8_t) 250)

number of polling iterations for TWSTO bit in TWSR after creating a Stop condition in i2c_send_stop().

Adjust this value considering how long it takes to check a status bit in the TWI control register, decrement the timeout counter, compare its value with 0, and branch.

7.5.3 Function Documentation

7.5.3.1 uint8_t i2c_receive_byte (uint8_t * *data*)

This function receives one byte from an I^2 C device.

Parameters

`out`	*data*	pointer to received byte

Returns

status of the operation

7.5.3.2 uint8_t i2c_receive_bytes (uint8_t *count,* uint8_t * *data*)

This function receives bytes from an I^2 C device and sends a Stop.

Parameters

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

Returns

status of the operation

7.5.3.3 uint8_t i2c_send_bytes (uint8_t *count,* uint8_t * *data*)

This function sends bytes to an I^2 C device.

Parameters

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

Returns

status of the operation

7.5.3.4 uint8_t i2c_send_start (void)

This function creates a Start condition (SDA low, then SCL low).

Returns

status of the operation

7.5.3.5 uint8_t i2c_send_stop (void)

This function creates a Stop condition (SCL high, then SDA high).

Returns

status of the operation

7.6 sha204_comm.c File Reference

Communication Layer of ATSHA204 Library.

Functions

- void sha204c_calculate_crc (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates CRC.
- uint8_t sha204c_check_crc (uint8_t ∗response)

  This function checks the consistency of a response.
- uint8_t sha204c_wakeup (uint8_t ∗response)

  This function wakes up a SHA204 device and receives a response.
- uint8_t sha204c_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:
- uint8_t sha204c_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.

7.6.1 Detailed Description

Communication Layer of ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 15, 2013

Copyright

ATSHA204 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.

1. 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.

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

1. 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, INCLUDI-NG, 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 D-AMAGES (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.

End of ATSHA204 Library License

7.7 sha204_comm.h File Reference

Definitions and Prototypes for Communication Layer of ATSHA204 Library.

Macros

- #define SHA204_COMMAND_EXEC_MAX ((uint8_t) (69.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  maximum command delay
- #define SHA204_CMD_SIZE_MIN ((uint8_t) 7)

  minimum number of bytes in command (from count byte to second CRC byte)
- #define SHA204_CMD_SIZE_MAX ((uint8_t) 84)

  maximum size of command packet (CheckMac)

- #define SHA204_CRC_SIZE ((uint8_t) 2)

  number of CRC bytes
- #define SHA204_BUFFER_POS_STATUS (1)

  buffer index of status byte in status response
- #define SHA204_BUFFER_POS_DATA (1)

  buffer index of first data byte in data response
- #define SHA204_STATUS_BYTE_WAKEUP ((uint8_t) 0x11)

  status byte after wake-up
- #define SHA204_STATUS_BYTE_PARSE ((uint8_t) 0x03)

  command parse error
- #define SHA204_STATUS_BYTE_EXEC ((uint8_t) 0x0F)

  command execution error
- #define SHA204_STATUS_BYTE_COMM ((uint8_t) 0xFF)

  communication error

Functions

- void sha204c_calculate_crc (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates CRC.
- uint8_t sha204c_wakeup (uint8_t ∗response)

  This function wakes up a SHA204 device and receives a response.
- uint8_t sha204c_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.

7.7.1 Detailed Description

Definitions and Prototypes for Communication Layer of ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 15, 2013

Copyright

ATSHA204 Library License:

7.8 sha204_comm_marshaling.c File Reference

Command Marshaling Layer of ATSHA204 Library.

Functions

- uint8_t sha204m_check_parameters (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 checks the parameters for sha204m_execute().
- uint8_t sha204m_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 sha204m_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 sha204m_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 sha204m_dev_rev (uint8_t *tx_buffer, uint8_t *rx_buffer)

  This function sends a DevRev command to the device.
- uint8_t sha204m_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 sha204m_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 sha204m_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 sha204m_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 sha204m_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 sha204m_pause (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t selector)

    This function sends a Pause command to the device.
- uint8_t sha204m_random (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t mode)

    This function sends a Random command to the device.
- uint8_t sha204m_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 sha204m_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 sha204m_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.

7.8.1 Detailed Description

Command Marshaling Layer of ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 9, 2013

Copyright

ATSHA204 Library License:

7.9 sha204_comm_marshaling.h File Reference

Definitions and Prototypes for Command Marshaling Layer of ATSHA204 Library.

Macros

Codes for ATSHA204 Commands

- #define SHA204_CHECKMAC ((uint8_t) 0x28)
  CheckMac command op-code.
- #define SHA204_DERIVE_KEY ((uint8_t) 0x1C)
  DeriveKey command op-code.
- #define SHA204_DEVREV ((uint8_t) 0x30)
  DevRev command op-code.
- #define SHA204_GENDIG ((uint8_t) 0x15)
  GenDig command op-code.
- #define SHA204_HMAC ((uint8_t) 0x11)
  HMAC command op-code.
- #define SHA204_LOCK ((uint8_t) 0x17)
  Lock command op-code.
- #define SHA204_MAC ((uint8_t) 0x08)
  MAC command op-code.
- #define SHA204_NONCE ((uint8_t) 0x16)
  Nonce command op-code.
- #define SHA204_PAUSE ((uint8_t) 0x01)
  Pause command op-code.
- #define SHA204_RANDOM ((uint8_t) 0x1B)
  Random command op-code.
- #define SHA204_READ ((uint8_t) 0x02)
  Read command op-code.
- #define SHA204_UPDATE_EXTRA ((uint8_t) 0x20)
  UpdateExtra command op-code.
- #define SHA204_WRITE ((uint8_t) 0x12)
  Write command op-code.

Definitions of Data and Packet Sizes

- #define SHA204_RSP_SIZE_VAL ((uint8_t) 7)
  size of response packet containing four bytes of data

- #define SHA204_KEY_SIZE (32)
  size of key
- #define SHA204_KEY_COUNT (16)
  number of keys
- #define SHA204_CONFIG_SIZE (88)
  size of configuration zone
- #define SHA204_OTP_SIZE (64)
  size of OTP zone
- #define SHA204_DATA_SIZE (SHA204_KEY_COUNT * SHA204_KEY_SIZE)
  size of data zone

Definitions for Command Parameter Ranges

- #define SHA204_KEY_ID_MAX (SHA204_KEY_COUNT - 1)
  maximum value for key id
- #define SHA204_OTP_BLOCK_MAX (1)
  maximum value for OTP block

Definitions for Indexes Common to All Commands

- #define SHA204_COUNT_IDX (0)
  command packet index for count
- #define SHA204_OPCODE_IDX (1)
  command packet index for op-code
- #define SHA204_PARAM1_IDX (2)
  command packet index for first parameter
- #define SHA204_PARAM2_IDX (3)
  command packet index for second parameter
- #define SHA204_DATA_IDX (5)
  command packet index for data load

Definitions for Zone and Address Parameters

- #define SHA204_ZONE_CONFIG ((uint8_t) 0x00)
  Configuration zone.
- #define SHA204_ZONE_OTP ((uint8_t) 0x01)
  OTP (One Time Programming) zone.
- #define SHA204_ZONE_DATA ((uint8_t) 0x02)
  Data zone.
- #define SHA204_ZONE_MASK ((uint8_t) 0x03)
  Zone mask.
- #define SHA204_ZONE_COUNT_FLAG ((uint8_t) 0x80)
  Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.
- #define SHA204_ZONE_ACCESS_4 ((uint8_t) 4)
  Read or write 4 bytes.
- #define SHA204_ZONE_ACCESS_32 ((uint8_t) 32)
  Read or write 32 bytes.
- #define SHA204_ADDRESS_MASK_CONFIG (0x001F)
  Address bits 5 to 7 are 0 for Configuration zone.
- #define SHA204_ADDRESS_MASK_OTP (0x000F)
  Address bits 4 to 7 are 0 for OTP zone.
- #define SHA204_ADDRESS_MASK (0x007F)

Address bit 7 to 15 are always 0.

Definitions for the CheckMac Command

- #define CHECKMAC_MODE_IDX SHA204_PARAM1_IDX

  CheckMAC command index for mode.
- #define CHECKMAC_KEYID_IDX SHA204_PARAM2_IDX

  CheckMAC command index for key identifier.
- #define CHECKMAC_CLIENT_CHALLENGE_IDX SHA204_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 SHA204_PARAM1_IDX

  DeriveKey command index for random bit.
- #define DERIVE_KEY_TARGETKEY_IDX SHA204_PARAM2_IDX

  DeriveKey command index for target slot.
- #define DERIVE_KEY_MAC_IDX SHA204_DATA_IDX

  DeriveKey command index for optional MAC.
- #define DERIVE_KEY_COUNT_SMALL SHA204_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 SHA204_PARAM1_IDX
  DevRev command index for 1. parameter (ignored)
- #define DEVREV_PARAM2_IDX SHA204_PARAM2_IDX
  DevRev command index for 2. parameter (ignored)
- #define DEVREV_COUNT SHA204_CMD_SIZE_MIN
  DevRev command packet size.

Definitions for the GenDig Command

- #define GENDIG_ZONE_IDX SHA204_PARAM1_IDX
  GenDig command index for zone.
- #define GENDIG_KEYID_IDX SHA204_PARAM2_IDX
  GenDig command index for key id.
- #define GENDIG_DATA_IDX SHA204_DATA_IDX
  GenDig command index for optional data.
- #define GENDIG_COUNT SHA204_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 SHA204_PARAM1_IDX
  HMAC command index for mode.
- #define HMAC_KEYID_IDX SHA204_PARAM2_IDX
  HMAC command index for key id.
- #define HMAC_COUNT SHA204_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 SHA204_PARAM1_IDX
  Lock command index for zone.
- #define LOCK_SUMMARY_IDX SHA204_PARAM2_IDX
  Lock command index for summary.
- #define LOCK_COUNT SHA204_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 SHA204_PARAM1_IDX

  MAC command index for mode.
- #define MAC_KEYID_IDX SHA204_PARAM2_IDX

  MAC command index for key id.
- #define MAC_CHALLENGE_IDX SHA204_DATA_IDX

  MAC command index for optional challenge.
- #define MAC_COUNT_SHORT SHA204_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 SHA204_PARAM1_IDX

  Nonce command index for mode.
- #define NONCE_PARAM2_IDX SHA204_PARAM2_IDX

  Nonce command index for 2. parameter.
- #define NONCE_INPUT_IDX SHA204_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 SHA204_PARAM1_IDX

  Pause command index for Selector.
- #define PAUSE_PARAM2_IDX SHA204_PARAM2_IDX

  Pause command index for 2. parameter.
- #define PAUSE_COUNT SHA204_CMD_SIZE_MIN

  Pause command packet size.

Definitions for the Random Command

- #define RANDOM_MODE_IDX SHA204_PARAM1_IDX

  Random command index for mode.
- #define RANDOM_PARAM2_IDX SHA204_PARAM2_IDX

  Random command index for 2. parameter.
- #define RANDOM_COUNT SHA204_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 SHA204_PARAM1_IDX

  Read command index for zone.
- #define READ_ADDR_IDX SHA204_PARAM2_IDX

  Read command index for address.
- #define READ_COUNT SHA204_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 SHA204_PARAM1_IDX

UpdateExtra command index for mode.
- #define UPDATE_VALUE_IDX SHA204_PARAM2_IDX

  UpdateExtra command index for new value.
- #define UPDATE_COUNT SHA204_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 SHA204_PARAM1_IDX

  Write command index for zone.
- #define WRITE_ADDR_IDX SHA204_PARAM2_IDX

  Write command index for address.
- #define WRITE_VALUE_IDX SHA204_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 SHA204_RSP_SIZE_MIN

  response size of DeriveKey command
- #define DERIVE_KEY_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of DeriveKey command
- #define DEVREV_RSP_SIZE SHA204_RSP_SIZE_VAL

  response size of DevRev command returns 4 bytes
- #define GENDIG_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of GenDig command
- #define HMAC_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of HMAC command
- #define LOCK_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of Lock command
- #define MAC_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of MAC command
- #define NONCE_RSP_SIZE_SHORT SHA204_RSP_SIZE_MIN

response size of Nonce command with mode[0:1] = 3
- #define NONCE_RSP_SIZE_LONG SHA204_RSP_SIZE_MAX

  response size of Nonce command
- #define PAUSE_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of Pause command
- #define RANDOM_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of Random command
- #define READ_4_RSP_SIZE SHA204_RSP_SIZE_VAL

  response size of Read command when reading 4 bytes
- #define READ_32_RSP_SIZE SHA204_RSP_SIZE_MAX

  response size of Read command when reading 32 bytes
- #define UPDATE_RSP_SIZE SHA204_RSP_SIZE_MIN

  response size of UpdateExtra command
- #define WRITE_RSP_SIZE SHA204_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 command typical execution time.
- #define DERIVE_KEY_DELAY ((uint8_t) (14.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE + 0.5))

  DeriveKey command typical execution time.
- #define DEVREV_DELAY ((uint8_t) (1))

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

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

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

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

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

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

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

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

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

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

  Write command typical execution time.

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 sha204m_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 sha204m_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 sha204m_dev_rev (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer)

  This function sends a DevRev command to the device.
- uint8_t sha204m_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 sha204m_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 sha204m_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 sha204m_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 sha204m_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 sha204m_pause (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t selector)

  This function sends a Pause command to the device.
- uint8_t sha204m_random (uint8_t ∗tx_buffer, uint8_t ∗rx_buffer, uint8_t mode)

  This function sends a Random command to the device.
- uint8_t sha204m_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 sha204m_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 sha204m_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 sha204m_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.

7.9.1 Detailed Description

Definitions and Prototypes for Command Marshaling Layer of ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 9, 2013

Copyright

Byte #	Name	Meaning
0	Count	Number of bytes in the packet, includes the count byte, body and the checksum
1	Op-Code	Indicates type of command
2	Parameter 1	mode, zone, etc.
3 and 4	Parameter 2	key id, address, etc.
5 to n	data (not for every command)	challenge, pass-through, etc.
n+1 to n+2	Checksum	Checksum of the command packet

Table 7.12: Command Packet Structure

Byte #	Name	Meaning
0	Count	Number of bytes in the packet, includes the count byte, body and the checksum
1	Status / Data	Status or first data byte
2 to n	More data bytes	random, challenge response, read data, etc.
n+1 to n+2	Checksum	Checksum of the command packet

Table 7.13: Response Packet Structure

7.10 sha204_config.h File Reference

Definitions for Configurable Values of the ATSHA204 Library.

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 SHA204_RETRY_COUNT (1)

  number of command / response retries

Available Definitions for Interfaces

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

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

- #define DOXYGEN_DUMMY 0

  Dummy macro that allow Doxygen to parse this group.

Configuration Definitions for SWI (UART) Interface

- #define SWI_RECEIVE_TIME_OUT ((uint16_t) 153) *receive timeout in us instead of loop counts*

- #define SWI_US_PER_BYTE ((uint16_t) 313)

    It takes 312.5 us to send a byte (9 single-wire bits / 230400 Baud ∗ 8 flag bits).

- #define SHA204_RESPONSE_TIMEOUT ((uint16_t) SWI_RECEIVE_TIME_OUT + SWI_US_PER_BYTE)

    SWI response timeout is the sum of receive timeout and the time it takes to send the TX flag.

Configuration Definitions for SWI Interface, Common to GPIO and UART

- #define SHA204_SYNC_TIMEOUT ((uint8_t) 85)

    delay before sending a transmit flag in the synchronization routine

7.10.1 Detailed Description

Definitions for Configurable Values of the ATSHA204 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

January 9, 2013

Copyright

7.11 sha204_example_main.c File Reference

Main Function for Application Examples that Use the ATSHA204 Library.

Functions

- int main (void)

  This application calls one example function that can be selected with a compilation switch defined in sha204_examples.h.

7.11.1 Detailed Description

Main Function for Application Examples that Use the ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 15, 2013

Copyright

7.12 sha204_examples.c File Reference

Application examples that Use the ATSHA204 Library.

Functions

- void sha204e_sleep ()

  This function wraps sha204p_sleep().
- uint8_t sha204e_wakeup_device (uint8_t device_id)

  This function wakes up two I^2 C devices and puts one back to sleep, effectively waking up only one device among two that share the bus.
- uint8_t sha204e_check_response_status (uint8_t ret_code, uint8_t ∗response)

  This function checks the response status byte and puts the device to sleep if there was an error.
- uint8_t sha204e_read_serial_number (uint8_t ∗tx_buffer, uint8_t ∗sn)

  This function reads the serial number from the device.
- uint8_t sha204e_lock_config_zone (uint8_t device_id)

  This function locks the configuration zone.
- uint8_t sha204e_configure_key ()

  This function configures a child and parent key for derived key scenarios.
- uint8_t sha204e_configure_derive_key ()

  This function configures the client for the derived key and diversified key example.
- uint8_t sha204e_configure_diversify_key (void)

  This function configures a client device for the diversified key example.
- uint8_t sha204e_checkmac_device (void)

  This function serves as an authentication example using the SHA204 MAC and CheckMac commands.
- uint8_t sha204e_checkmac_firmware (void)

  This function serves as an authentication example using the SHA204 Nonce, GenDig, and MAC commands.
- uint8_t sha204e_checkmac_derived_key (void)

  This function serves as an authentication example using the SHA204 Nonce, DeriveKey, and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.
- uint8_t sha204e_checkmac_diversified_key (void)

This function serves as an authentication example using the ATSHA204 Read and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.

- uint8_t sha204e_change_i2c_address (void)

  This function changes the I^2 C address of a device.

- uint8_t sha204e_read_config_zone (uint8_t device_id, uint8_t ∗config_data)

  This function reads all 88 bytes from the configuration zone.

Variables

- const uint8_t sha204_default_key [16][SHA204_KEY_SIZE]

  key values at time of shipping

7.12.1 Detailed Description

Application examples that Use the ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 15, 2013

Copyright

ATSHA204 Library License:

7.13 sha204_examples.h File Reference

Application Examples That Use the ATSHA204 Library.

Macros

- #define SHA204_EXAMPLE_CHECKMAC_DEVICE 1

  This definition selects a simple MAC / CheckMac example using an ATSHA204 as the host (key storage and SHA-256 calculation).
- #define SHA204_EXAMPLE_CHECKMAC_FIRMWARE 2

  This definition selects a simple MAC / CheckMac example using firmware as the host (key storage and SHA-256 calculation).
- #define SHA204_EXAMPLE_DERIVE_KEY 3

  This definition selects an advanced MAC / CheckMac example using a derived key. This example runs only with two devices.
- #define SHA204_EXAMPLE_DIVERSIFY_KEY 4

  This definition selects an advanced MAC / CheckMac example using a diversified key. This example runs only with two devices.
- #define SHA204_EXAMPLE_CHANGE_I2C_ADDRESS 5

  This definition selects a utility that changes the I2C default address of the device to SHA204_HOST_ADDRESS.
- #define SHA204_EXAMPLE_READ_CONFIG_ZONE 6

  This definition selects a utility that reads all 88 bytes from the configuration zone.
- #define SHA204_EXAMPLE SHA204_EXAMPLE_CHECKMAC_DEVICE
- #define SHA204_EXAMPLE_CONFIG_WITH_LOCK 0

  Use this definition if you like to lock the configuration zone of the host during personalization.
- #define SHA204_CLIENT_ADDRESS (0x00)
- #define SHA204_KEY_ID (0)

Functions

- uint8_t sha204e_checkmac_device (void)

  This function serves as an authentication example using the SHA204 MAC and CheckMac commands.
- uint8_t sha204e_checkmac_firmware (void)

  This function serves as an authentication example using the SHA204 Nonce, GenDig, and MAC commands.
- uint8_t sha204e_checkmac_derived_key (void)

  This function serves as an authentication example using the SHA204 Nonce, DeriveKey, and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.

- uint8_t sha204e_checkmac_diversified_key (void)

  This function serves as an authentication example using the ATSHA204 Read and MAC commands for a client, and the Nonce, GenDig, and CheckMac commands for a host device.
- uint8_t sha204e_change_i2c_address (void)

  This function changes the I^2 C address of a device.
- uint8_t sha204e_read_config_zone (uint8_t device_id, uint8_t ∗config_data)

  This function reads all 88 bytes from the configuration zone.

7.13.1 Detailed Description

Application Examples That Use the ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 9, 2013

Copyright

ATSHA204 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.

1. 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.

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

1. 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, INCLUDI-NG, 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 D-AMAGES (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.

End of ATSHA204 Library License

Example functions are given that demonstrate the device. The examples demonstrate client / host scenarios with a random challenge. Using a random challenge makes replay attacks impossible. Examples that need two devices (advanced examples) run only with I^2 C devices or SWI devices using GPIO. When running the advanced examples with SWI devices, their SDA cannot be shared. Therefore, these examples run only in the bit-banged and not in the UART implementation of SWI. It is possible for SWI devices to share SDA, but then the Pause command has to be used to idle all devices except one to communicate with. In such a system, the Selector byte of every device has to be unique and not 0 which is the default when shipped.

7.13.2 Macro Definition Documentation

7.13.2.1 #define SHA204_CLIENT_ADDRESS (0x00)

These settings have an effect only when using bit-banging where the SDA of every device is connected to its own GPIO pin. When using only one UART the SDA of both devices is connected to the same GPIO pin. In that case you have create a version of sha204p_set_device_id that would use a Pause command. (Refer to data sheet about the Pause command.)

7.13.2.2 #define SHA204_EXAMPLE SHA204_EXAMPLE_CHECKMAC_DEVICE

------------------— Define an example. -------------------------—

7.13.2.3 #define SHA204_EXAMPLE_CHANGE_I2C_ADDRESS 5

This definition selects a utility that changes the I2C default address of the device to SHA204_HOST_ADDRESS.

You need to change the address on one device from its default in order to run the advanced MAC / CheckMac examples.

7.13.2.4 #define SHA204_EXAMPLE_CONFIG_WITH_LOCK 0

Use this definition if you like to lock the configuration zone of the host during personalization.

Once the configuration zone is locked you cannot modify the configuration zone anymore, but the ATSHA204 device will then generate true random numbers instead of a 0xFFFF0000FFFF0000... sequence. The example assumes that the data line of the host is much less accessible by an adversary than the data line of the client. Therefore, the example requests a random number from the host and not the client, since an adversary could take over the data line and inject a number of her choice.

7.13.2.5 #define SHA204_EXAMPLE_READ_CONFIG_ZONE 6

This definition selects a utility that reads all 88 bytes from the configuration zone.

This gives you easy access to the device configuration (e.g. serial number, lock status, configuration of keys).

7.13.2.6 #define SHA204_KEY_ID (0)

Do not change these key identifiers since related values (configuration addresses) are hard-coded in associated functions.

7.14 sha204_helper.c File Reference

ATSHA204 Helper Functions.

Functions

- char ∗ sha204h_get_library_version (void)

  This function returns the library version. The version consists of three bytes. For a released version, the last byte is 0.
- uint8_t ∗ sha204h_include_data (struct sha204h_include_data_in_out ∗param)

  This function copies otp and sn data into a command buffer.
- uint8_t sha204h_nonce (struct sha204h_nonce_in_out ∗param)

  This function calculates a 32-byte nonce based on a 20-byte input value (param->num_in) and 32-byte random number (param->rand_out).
- uint8_t sha204h_mac (struct sha204h_mac_in_out ∗param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- uint8_t sha204h_check_mac (struct sha204h_check_mac_in_out ∗param)

  This function calculates a SHA-256 digest (MAC) of a password and other information, to be verified using the CheckMac device command.
- uint8_t sha204h_hmac (struct sha204h_hmac_in_out ∗param)

  This function generates an HMAC / SHA-256 hash of a key and other information.
- uint8_t sha204h_gen_dig (struct sha204h_gen_dig_in_out ∗param)

  This function combines the current TempKey with a stored value.
- uint8_t sha204h_derive_key (struct sha204h_derive_key_in_out ∗param)

  This function combines a key with the TempKey.
- uint8_t sha204h_derive_key_mac (struct sha204h_derive_key_mac_in_out ∗param)

  This function calculates the input MAC for a DeriveKey command.
- uint8_t sha204h_encrypt (struct sha204h_encrypt_in_out ∗param)

  This function encrypts 32-byte plain text data to be written using Write opcode, and optionally calculates input MAC.
- uint8_t sha204h_decrypt (struct sha204h_decrypt_in_out ∗param)

  This function decrypts 32-byte encrypted data received with the Read command.
- void sha204h_calculate_crc_chain (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates the packet CRC.
- void sha204h_calculate_sha256 (int32_t len, uint8_t ∗message, uint8_t ∗digest)

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

7.14.1 Detailed Description

ATSHA204 Helper Functions.

Author

Atmel Crypto Products

Date

January 15, 2013

7.15 sha204_helper.h File Reference

Definitions and Prototypes for ATSHA204 Helper Functions.

Data Structures

- struct sha204h_temp_key

  Structure to hold TempKey fields.
- struct sha204h_include_data_in_out

  Input / output parameters for function sha204h_include_data().
- struct sha204h_calculate_sha256_in_out

  Input/output parameters for function sha204h_nonce().
- struct sha204h_nonce_in_out

  Input/output parameters for function sha204h_nonce().
- struct sha204h_mac_in_out

  Input/output parameters for function sha204h_mac().

- struct sha204h_hmac_in_out

  Input/output parameters for function sha204h_hmac().
- struct sha204h_gen_dig_in_out

  Input/output parameters for function sha204h_gen_dig().
- struct sha204h_derive_key_in_out

  Input/output parameters for function sha204h_derive_key().
- struct sha204h_derive_key_mac_in_out

  Input/output parameters for function sha204h_derive_key_mac().
- struct sha204h_encrypt_in_out

  Input/output parameters for function sha204h_encrypt().
- struct sha204h_decrypt_in_out

  Input/output parameters for function sha204h_decrypt().
- struct sha204h_check_mac_in_out

  Input/output parameters for function sha204h_check_mac().

Macros

Definitions for SHA204 Message Sizes to Calculate a SHA256 Hash

"||" is the concatenation operator. The number in braces is the length of the hash input value in bytes.

- #define SHA204_MSG_SIZE_NONCE (55)

  RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.
- #define SHA204_MSG_SIZE_MAC (88)

  (Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}
- #define SHA204_MSG_SIZE_HMAC_INNER (152)

  HMAC = sha(HMAC outer || HMAC inner) HMAC inner = sha((zero-padded key ^ ipad) || message) = sha256((Key{32} || 0x36{32}) || 0{32} || Key{32} || OpCode{1} || Mode{1} || KeyId{2} || OTP0_7{8} || OTP8_10{3} || SN8{1} || SN4_7{4} || SN0_1{2} || SN2_3{2}){32}.
- #define SHA204_MSG_SIZE_HMAC (96)

  HMAC = sha(HMAC outer || HMAC inner) = sha256((Key{32} || 0x5C{32}) || HMAC inner{32})
- #define SHA204_MSG_SIZE_GEN_DIG (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define SHA204_MSG_SIZE_DERIVE_KEY (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define SHA204_MSG_SIZE_DERIVE_KEY_MAC (39)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.
- #define SHA204_MSG_SIZE_ENCRYPT_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define **SHA204_COMMAND_HEADER_SIZE** (4)
- #define **SHA204_GENDIG_ZEROS_SIZE** (25)
- #define **SHA204_DERIVE_KEY_ZEROS_SIZE** (25)
- #define **SHA204_OTP_SIZE_8** (8)
- #define **SHA204_OTP_SIZE_3** (3)
- #define **SHA204_SN_SIZE_4** (4)
- #define **SHA204_SN_SIZE_2** (2)
- #define **SHA204_OTHER_DATA_SIZE_2** (2)
- #define **SHA204_OTHER_DATA_SIZE_3** (3)
- #define **SHA204_OTHER_DATA_SIZE_4** (4)
- #define **HMAC_BLOCK_SIZE** (64)
- #define **SHA204_PACKET_OVERHEAD** (3)

Fixed Byte Values of Serial Number (SN[0:1] and SN[8])

- #define **SHA204_SN_0** (0x01)
- #define **SHA204_SN_1** (0x23)
- #define **SHA204_SN_8** (0xEE)

Definition for TempKey Mode

- #define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t) 0x03)

  mode mask for MAC command when using TempKey

Functions

- char ∗ sha204h_get_library_version (void)

  This function returns the library version. The version consists of three bytes. For a released version, the last byte is 0.
- uint8_t sha204h_nonce (struct sha204h_nonce_in_out ∗param)

  This function calculates a 32-byte nonce based on a 20-byte input value (param->num_in) and 32-byte random number (param->rand_out).
- uint8_t sha204h_mac (struct sha204h_mac_in_out ∗param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- uint8_t sha204h_check_mac (struct sha204h_check_mac_in_out ∗param)

  This function calculates a SHA-256 digest (MAC) of a password and other information, to be verified using the CheckMac device command.
- uint8_t sha204h_hmac (struct sha204h_hmac_in_out ∗param)

  This function generates an HMAC / SHA-256 hash of a key and other information.
- uint8_t sha204h_gen_dig (struct sha204h_gen_dig_in_out ∗param)

  This function combines the current TempKey with a stored value.
- uint8_t sha204h_derive_key (struct sha204h_derive_key_in_out ∗param)

  This function combines a key with the TempKey.
- uint8_t sha204h_derive_key_mac (struct sha204h_derive_key_mac_in_out ∗param)

  This function calculates the input MAC for a DeriveKey command.
- uint8_t sha204h_encrypt (struct sha204h_encrypt_in_out ∗param)

  This function encrypts 32-byte plain text data to be written using Write opcode, and optionally calculates input MAC.
- uint8_t sha204h_decrypt (struct sha204h_decrypt_in_out ∗param)

  This function decrypts 32-byte encrypted data received with the Read command.
- void sha204h_calculate_crc_chain (uint8_t length, uint8_t ∗data, uint8_t ∗crc)

  This function calculates the packet CRC.
- void sha204h_calculate_sha256 (int32_t len, uint8_t ∗message, uint8_t ∗digest)

  This function creates a SHA256 digest on a little-endian system.
- uint8_t ∗ sha204h_include_data (struct sha204h_include_data_in_out ∗param)

  This function copies otp and sn data into a command buffer.

7.15.1 Detailed Description

Definitions and Prototypes for ATSHA204 Helper Functions.

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

ATSHA204 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.

1. 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.

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

1. 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, INCLUDI-NG, 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 D-AMAGES (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.

End of ATSHA204 Library License

7.16 sha204_i2c.c File Reference

Functions for I^2 C Physical Hardware Independent Layer of ATSHA204 Library.

Macros

- #define SHA204_I2C_DEFAULT_ADDRESS ((uint8_t) 0xC8)

  I^2 C address used at ATSHA204 library startup.

Enumerations

- enum i2c_word_address { SHA204_I2C_PACKET_FUNCTION_RESET, SHA204_I2C_PACKET_FUNCTION_SLEEP, SHA204_I2C_PACKET_FUNCTION_IDLE, SHA204_I2C_PACKET_FUNCTION_NORMAL }

    This enumeration lists all packet types sent to a SHA204 device.
- enum i2c_read_write_flag { I2C_WRITE = (uint8_t) 0x00, I2C_READ = (uint8_t) 0x01 }

    This enumeration lists flags for I^2 C read or write addressing.

Functions

- void sha204p_set_device_id (uint8_t id)

    This function sets the I^2 C address. Communication functions will use this address.
- void sha204p_init (void)

    This function initializes the hardware.
- uint8_t sha204p_wakeup (void)

    This function generates a Wake-up pulse and delays.
- uint8_t sha204p_send_command (uint8_t count, uint8_t *command)

    This function sends a command to the device.
- uint8_t sha204p_idle (void)

    This function puts the device into idle state.
- uint8_t sha204p_sleep (void)

    This function puts the device into low-power state.
- uint8_t sha204p_reset_io (void)

    This function resets the I/O buffer of the device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t *response)

    This function receives a response from the device.
- uint8_t sha204p_resync (uint8_t size, uint8_t *response)

    This function resynchronizes communication.

7.16.1 Detailed Description

Functions for I^2 C Physical Hardware Independent Layer of ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

ATSHA204 Library License:

7.17 sha204_lib_return_codes.h File Reference

Definitions for ATSHA204 Library Return Codes.

Macros

- #define SHA204_SUCCESS ((uint8_t) 0x00)

  Function succeeded.
- #define SHA204_CHECKMAC_FAILED ((uint8_t) 0xD1)

  response status byte indicates CheckMac failure
- #define SHA204_PARSE_ERROR ((uint8_t) 0xD2)

  response status byte indicates parsing error
- #define SHA204_CMD_FAIL ((uint8_t) 0xD3)

  response status byte indicates command execution error
- #define SHA204_STATUS_CRC ((uint8_t) 0xD4)

  response status byte indicates CRC error
- #define SHA204_STATUS_UNKNOWN ((uint8_t) 0xD5)

  response status byte is unknown
- #define SHA204_FUNC_FAIL ((uint8_t) 0xE0)

  Function could not execute due to incorrect condition / state.
- #define SHA204_GEN_FAIL ((uint8_t) 0xE1)

  unspecified error

- #define SHA204_BAD_PARAM ((uint8_t) 0xE2)

  bad argument (out of range, null pointer, etc.)
- #define SHA204_INVALID_ID ((uint8_t) 0xE3)

  invalid device id, id not set
- #define SHA204_INVALID_SIZE ((uint8_t) 0xE4)

  Count value is out of range or greater than buffer size.
- #define SHA204_BAD_CRC ((uint8_t) 0xE5)

  incorrect CRC received
- #define SHA204_RX_FAIL ((uint8_t) 0xE6)

  Timed out while waiting for response. Number of bytes received is $>$ 0.
- #define SHA204_RX_NO_RESPONSE ((uint8_t) 0xE7)

  Not an error while the Command layer is polling for a command response.
- #define SHA204_RESYNC_WITH_WAKEUP ((uint8_t) 0xE8)

  Re-synchronization succeeded, but only after generating a Wake-up.
- #define SHA204_COMM_FAIL ((uint8_t) 0xF0)

  Communication with device failed. Same as in hardware dependent modules.
- #define SHA204_TIMEOUT ((uint8_t) 0xF1)

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

7.17.1 Detailed Description

Definitions for ATSHA204 Library Return Codes.

Author

Atmel Crypto Products

Date

January 15, 2013

Copyright

ATSHA204 Library License:

7.18 sha204_physical.h File Reference

Definitions and Prototypes for Physical Layer Interface of ATSHA204 Library.

Macros

- #define SHA204_RSP_SIZE_MIN ((uint8_t) 4)

  minimum number of bytes in response
- #define SHA204_RSP_SIZE_MAX ((uint8_t) 35)

  maximum size of response packet
- #define SHA204_BUFFER_POS_COUNT (0)

  buffer index of count byte in command or response
- #define SHA204_BUFFER_POS_DATA (1)

  buffer index of data in response
- #define SHA204_WAKEUP_PULSE_WIDTH (uint8_t) (6.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

  width of Wakeup pulse in 10 us units
- #define SHA204_WAKEUP_DELAY (uint8_t) (3.0 ∗ CPU_CLOCK_DEVIATION_POSITIVE + 0.5)

  delay between Wakeup pulse and communication in ms

Functions

- uint8_t sha204p_send_command (uint8_t count, uint8_t ∗command)

  This function sends a command to the device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t ∗response)

  This function receives a response from the device.
- void sha204p_init (void)

  This function initializes the hardware.
- void sha204p_set_device_id (uint8_t id)

  This function sets the I^2 C address. Communication functions will use this address.
- uint8_t sha204p_wakeup (void)

  This function generates a Wake-up pulse and delays.
- uint8_t sha204p_idle (void)

  This function puts the device into idle state.

- uint8_t sha204p_sleep (void)

    This function puts the device into low-power state.
- uint8_t sha204p_reset_io (void)

    This function resets the I/O buffer of the device.
- uint8_t sha204p_resync (uint8_t size, uint8_t ∗response)

    This function resynchronizes communication.

7.18.1 Detailed Description

Definitions and Prototypes for Physical Layer Interface of ATSHA204 Library.

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

ATSHA204 Library License:

End of ATSHA204 Library License

7.19 sha204_swi.c File Reference

Functions for Single Wire, Hardware Independent Physical Layer of ATSHA204 Library.

Macros

- #define SHA204_SWI_FLAG_CMD ((uint8_t) 0x77)

    flag preceding a command
- #define SHA204_SWI_FLAG_TX ((uint8_t) 0x88)

    flag requesting a response
- #define SHA204_SWI_FLAG_IDLE ((uint8_t) 0xBB)

    flag requesting to go into Idle mode
- #define SHA204_SWI_FLAG_SLEEP ((uint8_t) 0xCC)

    flag requesting to go into Sleep mode

Functions

- void sha204p_init (void)

    This function initializes the hardware.
- void sha204p_set_device_id (uint8_t id)

    This function selects the GPIO pin used for communication. It has no effect when using a UART.
- uint8_t sha204p_send_command (uint8_t count, uint8_t ∗command)

    This function sends a command to the device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t ∗response)

    This function receives a response from the device.
- uint8_t sha204p_wakeup (void)

    This function generates a Wake-up pulse and delays.
- uint8_t sha204p_idle ()

    This function puts the device into idle state.
- uint8_t sha204p_sleep ()

    This function puts the device into low-power state.
- uint8_t sha204p_reset_io (void)

    This function is only a dummy since the functionality does not exist for the SWI version of the SHA204 device.
- uint8_t sha204p_resync (uint8_t size, uint8_t ∗response)

    This function re-synchronizes communication.

7.19.1 Detailed Description

Functions for Single Wire, Hardware Independent Physical Layer of ATSHA204 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. #sha204p_idle, #sha204p_sleep).
```

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

ATSHA204 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.

1. 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.

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

1. 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.

End of ATSHA204 Library License

7.20 swi_phys.h File Reference

Definitions and Prototypes for SWI Hardware Dependent Physical Layer of CryptoAuth Library.

Macros

- #define SWI_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

  Communication with device succeeded.
- #define SWI_FUNCTION_RETCODE_TIMEOUT ((uint8_t) 0xF1)

  Communication timed out.
- #define SWI_FUNCTION_RETCODE_RX_FAIL ((uint8_t) 0xF9)

  Communication failed after at least one byte was received.

Functions

- void swi_enable (void)

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

  This GPIO function sets the signal pin. Communication functions will use this signal pin.
- void swi_set_signal_pin (uint8_t end)

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

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

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

  This GPIO function receives bytes from an SWI device.

7.20.1 Detailed Description

Definitions and Prototypes for SWI Hardware Dependent Physical Layer of CryptoAuth Library.

Author

Atmel Crypto Products

Date

January 11, 2013

ATSHA204 Library License:

7.20.2 Function Documentation

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

This GPIO function receives bytes from an SWI device.

Parameters

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

Returns

status of the operation

This GPIO function receives bytes from an SWI device.

Parameters

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

Returns

status of the operation

7.20.2.2 uint8_t swi_send_byte (uint8_t *value*)

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

7.20.2.3 uint8_t swi_send_bytes (uint8_t *count*, uint8_t * *buffer*)

This GPIO function sends bytes to an SWI device.

Parameters

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

Returns

status of the operation

This GPIO function sends bytes to an SWI device.

Parameters

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

Returns

status of the operation

7.20.2.4 void swi_set_device_id (uint8_t *id*)

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

Parameters

in	*id*	client if zero, otherwise host

Returns

status of the operation

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

Parameters

in	*id*	not used in this UART module, only used in SWI bit-banging module To be able to talk to two devices (client or host) sending a Pause flag is required. Please refer to the data sheet.

7.20.2.5 void swi_set_signal_pin (uint8_t *is_high*)

This GPIO function sets the signal pin low or high.

Parameters

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

This GPIO function sets the signal pin low or high.

```
It is used to generate a Wake-up pulse.<BR>
Another way to generate a Wake-up pulse is using the UART
at half the communication baud rate and sending a 0.
Keeping the baud rate at 230400 would only produce
the signal wire going low for 34.7 us
when sending a data byte of 0 that causes the signal wire
being low for eight bits (start bit and seven data bits).
Configuring the UART for half the baud rate and sending
a 0 produces a long enough Wake-up pulse of 69.4 us.<BR>
The fact that a hardware independent Physical layer above
this hardware dependent layer delays for Wake-pulse width
after calling this function would only add this delay to the
much longer delay of 3 ms after the Wake-up pulse.
With other words, by not using GPIO for the generation of
a Wake-up pulse, we add only 69.4 us to the delay of
3000 us after the Wake-up pulse.<BR>
Implementing a Wake-up pulse generation using the UART
would introduce a slight design flaw though since this module
would now "know" something about the width of the Wake-up pulse.
We could add a function that sets the baud rate and
sends a 0, but that would add at least 150 bytes of code.
```

Parameters

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

7.21 timer_utilities.c File Reference

Timer Utility Functions.

Macros

- #define TIME_UTILS_US_CALIBRATION

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

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

    The delay_ms function calls delay_10us with this parameter.

Functions

- void delay_10us (uint8_t delay)

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

    This function delays for a number of milliseconds.

7.21.1 Detailed Description

Timer Utility Functions.

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

7.22 timer_utilities.h File Reference

Timer Utility Declarations.

Functions

- void delay_10us (uint8_t delay)

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

  This function delays for a number of milliseconds.

7.22.1 Detailed Description

Timer Utility Declarations.

Author

Atmel Crypto Products

Date

January 11, 2013

Copyright

7.23 uart_config.h File Reference

Definitions for Hardware Dependent Part of the Physical Layer of the ATSHA204 Library Using a UART.

Macros

- #define BAUD_RATE (230400UL)

  baud rate for SHA204 device in single-wire mode
- #define TIME_PER_LOOP_ITERATION (0.8)

  time in us it takes for decrementing a uint8_t and branching
- #define BIT_TIMEOUT ((uint8_t) (250.0 / TIME_PER_LOOP_ITERATION))

  number of polling iterations over UART register before timing out
- #define RX_TX_DELAY ((uint8_t) (15.0 / TIME_PER_LOOP_ITERATION))

  Delay for this many loop iterations before sending.
- #define UART_GPIO_DDR DDRD

  direction register when using UART pin for Wake-up
- #define UART_GPIO_OUT PORTD

  output register when using UART pin for Wake-up
- #define UART_GPIO_PIN_RX _BV(PD2)

  bit position when using UART rx pin for Wake-up
- #define UART_GPIO_PIN_TX _BV(PD3)

  bit position when using UART tx pin for Wake-up
- #define DEBUG_LOW

  undefine debugging macro
- #define DEBUG_HIGH

  undefine debugging macro

7.23.1 Detailed Description

Definitions for Hardware Dependent Part of the Physical Layer of the ATSHA204 Library Using a UART.

Author

Atmel Crypto Products

Date

January 15, 2013

Copyright

7.23.2 Macro Definition Documentation

7.23.2.1 #define BIT_TIMEOUT ((uint8_t) (250.0 / TIME_PER_LOOP_ITERATION))

number of polling iterations over UART register before timing out

The polling iteration takes about 0.8 us. For tx, we would need to wait bit time = 39 us. For rx, we need at least wait for tx / rx turn-around time + bit time = 95 us + 39 us = 134 us. Let's make the timeout larger to be safe.

7.24 uart_phys.c File Reference

Physical Layer Functions of ATSHA204 Library When Using UART.

Functions

- void swi_set_device_id (uint8_t id)

    This UART function is a dummy to satisfy the SWI module interface.
- void swi_enable (void)

    This UART function initializes the hardware.
- void swi_set_signal_pin (uint8_t is_high)

    This UART function sets the signal pin using GPIO.
- uint8_t swi_send_bytes (uint8_t count, uint8_t ∗buffer)

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

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

    This UART function receives bytes from an SWI device.

7.24.1 Detailed Description

Physical Layer Functions of ATSHA204 Library When Using UART.

```
This module supports most of ATmega and all ATXmega AVR microcontrollers.
http://www.atmel.com/dyn/products/param_table.asp?family_id=607&OrderBy=part_no&Direction=ASC
```

Author

Atmel Crypto Products

Date

January 14, 2013

Copyright

7.24.2 Function Documentation

7.24.2.1 void swi_enable (void)

This UART function initializes the hardware.

This GPIO function sets the bit position of the signal pin to its default.

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

This UART function receives bytes from an SWI device.

This GPIO function receives bytes from an SWI device.

Parameters

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

Returns

status of the operation

7.24.2.3 uint8_t swi_send_byte (uint8_t *value*)

This UART function sends one byte to an SWI device.

This GPIO function sends one byte to an SWI device.

Parameters

in	*value*	byte to send

Returns

status of the operation

7.24.2.4 uint8_t swi_send_bytes (uint8_t *count*, uint8_t * *buffer*)

This UART function sends bytes to an SWI device.

This GPIO function sends bytes to an SWI device.

Parameters

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

Returns

status of the operation

7.24.2.5 void swi_set_device_id (uint8_t *id*)

This UART function is a dummy to satisfy the SWI module interface.

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

Parameters

in	*id*	not used in this UART module, only used in SWI bit-banging module To be able to talk to two devices (client or host) sending a Pause flag is required. Please refer to the data sheet.

7.24.2.6 void swi_set_signal_pin (uint8_t *is_high*)

This UART function sets the signal pin using GPIO.

This GPIO function sets the signal pin low or high.

```
It is used to generate a Wake-up pulse.<BR>
Another way to generate a Wake-up pulse is using the UART
at half the communication baud rate and sending a 0.
Keeping the baud rate at 230400 would only produce
the signal wire going low for 34.7 us
when sending a data byte of 0 that causes the signal wire
being low for eight bits (start bit and seven data bits).
Configuring the UART for half the baud rate and sending
a 0 produces a long enough Wake-up pulse of 69.4 us.<BR>
The fact that a hardware independent Physical layer above
this hardware dependent layer delays for Wake-pulse width
after calling this function would only add this delay to the
much longer delay of 3 ms after the Wake-up pulse.
With other words, by not using GPIO for the generation of
a Wake-up pulse, we add only 69.4 us to the delay of
3000 us after the Wake-up pulse.<BR>
Implementing a Wake-up pulse generation using the UART
would introduce a slight design flaw though since this module
would now "know" something about the width of the Wake-up pulse.
We could add a function that sets the baud rate and
sends a 0, but that would add at least 150 bytes of code.
```

Parameters

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