SHA204 Library Examples for AVR 8-Bit Target

1.2.0

Generated by Doxygen 1.8.2

Fri Sep 28 2012 18:16:00

Contents

Chapter 1

Building The Projects

1.1 Work Space and Project Structure

```
The source files for the SHA204 library are contained in a single folder "src".
```

1.1.1 Hardware Independent Modules

sha204_example_main.c

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

swi_phys.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:

- bitbang_phys.c: Physical implementation as single wire interface (SWI) using GPIO (includes delay_x.h).
- 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^2C).

1.1.3 Projects

Three project files are supplied for the AVR Studio 4 IDE (.aps). One solution file (.sln) is supplied for the Atmel Studio 6.0 IDE that in turn contains three projects (.cproj). AVR Studio 4 project files and Atmel Studio 6 solution files and folders are located in the SHA204_90USB1287 folder. Choose the project that fits the communication interface you like to use.

If you don't use one of the IDE's mentioned above 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^2C.

- **SWI Using UART**

  sha204_example_main.c

  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_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

  delay_x.h

  bitbang_phys.c

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

- **I²C**

  sha204_example_main.c

  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 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 I²C add sha204_i2c.*. You might have to also modify sha204_i2c.c, especially for 32-bit CPUs, since their I²C peripherals implement such functionality in hardware. For instance, they might not support the generation of individual Start and Stop conditions.

- Supply communication interface hardware dependent modules. If you do not use an AVR CPU, you have to implement the functions in these modules. For SWI using UART add uart_phys.c, for SWI using GPIO add bitbang_phys.c, and for I²C add i2c_phys.*. Be aware that uart_phys.c includes avr_compatible.h and bitbang_phys.c includes delay_x.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). These functions are used to determine command response timeouts. They 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 compiler suite:

WinAVR 20100110

http://winavr.sourceforge.net/helpme.html

1.2.2 IDE's:

AVR Studio 4.18, Build 716

http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725

Atmel Studio 6.0.1843

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

1.3 Doxygen Generated Documentation

Important comments (functions, type and macro definitions, etc.) follow a syntax that the Doxygen document generator for source code can parse.

Chapter 2

File Index

2.1 File List

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

Chapter 3

File Documentation

3.1 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)

3.1.1 Detailed Description

AVR USART Register Compatibility Definitions.

Author

Atmel Crypto Products

Date

October 21, 2010

3.2 bitbang_config.h File Reference

Definitions for Hardware Dependent Part of SHA204 Physical Layer Using GPIO for Communication.

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

Macros

- #define swi_enable_interrupts sei

  < interrupt definitions
- #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 PORT_ACCESS_TIME (630)

  time it takes to toggle the pin at CPU clock of 16 MHz (ns)
- #define START_PULSE_WIDTH (4340)

  width of start pulse (ns)
- #define BIT_DELAY_1 _delay_ns(START_PULSE_WIDTH - PORT_ACCESS_TIME)

  delay macro for width of one pulse (start pulse or zero pulse, in ns)
- #define BIT_DELAY_5 _delay_ns(6 ∗ START_PULSE_WIDTH - PORT_ACCESS_TIME)

  time to keep pin high for five pulses plus stop bit (used to bit-bang CryptoAuth 'zero' bit, in ns)
- #define BIT_DELAY_7 _delay_ns(7 ∗ START_PULSE_WIDTH - PORT_ACCESS_TIME)

  time to keep pin high for seven bits plus stop bit (used to bit-bang CryptoAuth 'one' bit)
- #define RX_TX_DELAY _delay_us(15)

  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.

3.2.1 Detailed Description

Definitions for Hardware Dependent Part of SHA204 Physical Layer Using GPIO for Communication.

Author

Atmel Crypto Products

Date

May 6, 2011

3.2.2 Macro Definition Documentation

3.2.2.1 #define swi_enable_interrupts sei

< interrupt definitions

< GPIO definitionsenable interrupts

3.3 bitbang_phys.c File Reference

Functions of Hardware Dependent Part of SHA204 Physical Layer Using GPIO For Communication.

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

Functions

- void swi_set_device_id (uint8_t id)

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

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

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

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

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

  This GPIO function receives bytes from an SWI device.

3.3.1 Detailed Description

Functions of Hardware Dependent Part of SHA204 Physical Layer Using GPIO For Communication.

Author

Atmel Crypto Products

Date

June 14, 2011

3.3.2 Function Documentation

3.3.2.1 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

3.3.2.2 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

3.3.2.3 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

3.3.2.4 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

3.3.2.5 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.

3.4 delay_x.h File Reference

CPU Dependent Delay Functions for AVR Micro-Controllers.

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

Macros

- #define _delay_ns(__ns) _delay_cycles((double)(F_CPU)*((double)__ns)/1.0e9 + 0.5)

3.4.1 Detailed Description

CPU Dependent Delay Functions for AVR Micro-Controllers.

Author

Hans-Juergen Heinrichs

Date

2005

3.4.2 Macro Definition Documentation

3.4.2.1 #define _delay_ns(__ns) _delay_cycles((double)(F_CPU)*((double)__ns)/1.0e9 + 0.5)

Note

Note

delay_x.h

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

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

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

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

3.5 i2c_phys.c File Reference

Implementation of I2C Hardware Dependent Functions in the Physical Layer of Crypto Libraries.

```
#include <avr\io.h>
#include <util\twi.h>
#include <avr\power.h>
#include "i2c_phys.h"
```

Functions

- void i2c_enable (void)

  This function initializes and enables the I2C peripheral.
- void i2c_disable (void)

  This function disables the I2C peripheral.
- uint8_t i2c_send_start (void)

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

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

  This function sends bytes to an I2C device.
- uint8_t i2c_receive_byte (uint8_t *data)

  This function receives one byte from an I2C device.
- uint8_t i2c_receive_bytes (uint8_t count, uint8_t *data)

  This function receives bytes from an I2C device and sends a Stop.

3.5.1 Detailed Description

Implementation of I2C Hardware Dependent Functions in the Physical Layer of Crypto Libraries.

Author

Atmel Crypto Products

Date

February 2, 2011

3.5.2 Function Documentation

3.5.2.1 uint8_t i2c_receive_byte (uint8_t * *data*)

This function receives one byte from an I2C device.

Parameters

out	*data*	pointer to received byte

Returns

status of the operation

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

This function receives bytes from an I2C device and sends a Stop.

Parameters

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

Returns

status of the operation

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

This function sends bytes to an I2C device.

Parameters

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

Returns

status of the operation

3.5.2.4 uint8_t i2c_send_start (void)

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

Returns

status of the operation

3.5.2.5 uint8_t i2c_send_stop (void)

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

Returns

status of the operation

3.6 i2c_phys.h File Reference

Definitions and Prototypes of I2C Hardware Dependent Functions in the Physical Layer of Crypto Libraries.

```
#include <stdint.h>
```

Macros

- #define I2C_CLOCK (400000.0)

  I2C clock.
- #define I2C_PULLUP

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

  number of polling iterations for TWINT bit in TWSR after creating a Start condition in i2c_send_start()
- #define I2C_BYTE_TIMEOUT ((uint8_t) 100)

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

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

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

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

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

  TWI nack.

Functions

- void i2c_enable (void)

  This function initializes and enables the I2C peripheral.
- void i2c_disable (void)

  This function disables the I2C peripheral.
- uint8_t i2c_send_start (void)

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

- uint8_t i2c_send_stop (void)

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

- uint8_t i2c_send_bytes (uint8_t count, uint8_t *data)

  This function sends bytes to an I2C device.

- uint8_t i2c_receive_byte (uint8_t *data)

  This function receives one byte from an I2C device.

- uint8_t i2c_receive_bytes (uint8_t count, uint8_t *data)

  This function receives bytes from an I2C device and sends a Stop.

3.6.1 Detailed Description

Definitions and Prototypes of I2C Hardware Dependent Functions in the Physical Layer of Crypto Libraries.

Author

Atmel Crypto Products

Date

February 2, 2011

3.6.2 Macro Definition Documentation

3.6.2.1 #define I2C_BYTE_TIMEOUT ((uint8_t) 100)

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.

3.6.2.2 #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.

3.6.2.3 #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.

3.6.3 Function Documentation

3.6.3.1 uint8_t i2c_receive_byte (uint8_t * *data*)

This function receives one byte from an I2C device.

Parameters

out	*data*	pointer to received byte

Returns

status of the operation

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

This function receives bytes from an I2C device and sends a Stop.

Parameters

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

Returns

status of the operation

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

This function sends bytes to an I2C device.

Parameters

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

Returns

status of the operation

3.6.3.4 uint8_t i2c_send_start (void)

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

Returns

status of the operation

3.6.3.5 uint8_t i2c_send_stop (void)

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

Returns

status of the operation

3.7 sha204_comm.c File Reference

Communication Layer of SHA204 Library.

```
#include "sha204_comm.h"
#include "timer_utilities.h"
#include "sha204_lib_return_codes.h"
```

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.
- 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.

3.7.1 Detailed Description

Communication Layer of SHA204 Library.

Author

Atmel Crypto Products

Date

October 21, 2010

3.7.2 Function Documentation

3.7.2.1 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

3.7.2.2 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

3.7.2.3 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

3.7.2.4 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 got "nacked" (TWI), this function requests re-sending the response. If the response contains an error status, this function resends the command.

Parameters

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

Returns

status of the operation

3.7.2.5 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

3.8 sha204_comm.h File Reference

Definitions and Prototypes for Communication Layer of SHA204 Library.

```
#include <stddef.h>
#include "sha204_physical.h"
```

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: Append CRC to tx buffer, send command, delay, and verify response after receiving it.

3.8.1 Detailed Description

Definitions and Prototypes for Communication Layer of SHA204 Library.

Author

Atmel Crypto Products

Date

October 20, 2010

3.8.2 Function Documentation

3.8.2.1 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

3.8.2.2 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 got "nacked" (TWI), this function requests re-sending the response. If the response contains an error status, this function resends the command.

Parameters

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

in	*execution_timeout*	polling timeout in ms

Returns

status of the operation

3.8.2.3 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

3.9 sha204_comm_marshaling.c File Reference

Command Marshaling Layer of SHA204 Library.

```
#include <string.h>
#include "sha204_lib_return_codes.h"
#include "sha204_comm_marshaling.h"
```

Functions

- 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.

3.9.1 Detailed Description

Command Marshaling Layer of SHA204 Library.

Author

Atmel Crypto Products

Date

May 17, 2012

3.9.2 Function Documentation

3.9.2.1 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

3.9.2.2 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

3.9.2.3 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

3.9.2.4 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

3.9.2.5 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

3.9.2.6 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*	
in	*key_id*	slot index of key

Returns

status of the operation

3.9.2.7 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

3.9.2.8 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

3.9.2.9 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

3.9.2.10 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 pause.

Returns

status of the operation

3.9.2.11 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

3.9.2.12 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

3.9.2.13 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

3.9.2.14 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

3.10 sha204_comm_marshaling.h File Reference

Definitions and Prototypes for Command Marshaling Layer of SHA204 Library.

```
#include "sha204_comm.h"
```

Macros

- #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.
- #define SHA204_RSP_SIZE_VAL ((uint8_t) 7)
  size of response packet containing four bytes of data
- #define SHA204_KEY_ID_MAX ((uint8_t) 15)

maximum value for key id

- #define SHA204_OTP_BLOCK_MAX ((uint8_t) 1)

  maximum value for OTP block
- #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 second parameter
- #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.
- #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_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 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.
- #define DERIVE_KEY_MAC_SIZE (32)

  DeriveKey MAC size.
- #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.
- #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.
- #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.
- #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.
- #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_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.
- #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)
- #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.
- #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.
- #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.

- #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.
- #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.
- #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 TEMP_SENSE_RSP_SIZE SHA204_RSP_SIZE_VAL

  response size of TempSense command returns 4 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
- #define CHECKMAC_DELAY ((uint8_t) (12.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE - 0.5))

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

  DeriveKey minimum command delay.
- #define DEVREV_DELAY ((uint8_t) (0.4 ∗ CPU_CLOCK_DEVIATION_NEGATIVE - 0.5))

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

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

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

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

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

  Nonce minimum command delay.
- #define PAUSE_DELAY ((uint8_t) (0.4 ∗ CPU_CLOCK_DEVIATION_NEGATIVE - 0.5))

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

  Random minimum command delay.
- #define READ_DELAY ((uint8_t) (0.4 ∗ CPU_CLOCK_DEVIATION_NEGATIVE - 0.5))

  Read minimum command delay.
- #define TEMP_SENSE_DELAY ((uint8_t) (4.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE - 0.5))

  TempSense minimum command delay.
- #define UPDATE_DELAY ((uint8_t) (4.0 ∗ CPU_CLOCK_DEVIATION_NEGATIVE - 0.5))

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

  Write minimum command delay.
- #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 TEMP_SENSE_EXEC_MAX ((uint8_t) (11.0 * CPU_CLOCK_DEVIATION_POSITIVE + 0.5))

  TempSense maximum execution time.
- #define UPDATE_EXEC_MAX ((uint8_t) (6.0 * 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_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.

3.10.1 Detailed Description

Definitions and Prototypes for Command Marshaling Layer of SHA204 Library.

Author

Atmel Crypto Products

Date

September 14, 2011

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

Table 3.34: Command Packet Structure

3.10.2 Function Documentation

3.10.2.1 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

3.11 sha204_config.h File Reference

Definitions for Configurable Values of the SHA204 Library.

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

Macros

- #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

3.11.1 Detailed Description

```
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

February 2, 2011

3.11.2 Macro Definition Documentation

3.11.2.1 #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.

3.12 sha204_example_main.c File Reference

Example of an Application That Uses the SHA204 Library.

```
#include <stddef.h>
#include "sha204_lib_return_codes.h"
#include "sha204_comm_marshaling.h"
```

Functions

- void evaluate_ret_code (uint8_t ret_code)

  This function evaluates a function return code and puts the device to sleep if the return code indicates that the device is awake.
- int main (void)

  This function serves as an example for the SHA204 MAC command.

3.12.1 Detailed Description

Example of an Application That Uses the SHA204 Library.

Author

Atmel Crypto Products

Date

October 7, 2010

3.12.2 Function Documentation

3.12.2.1 void evaluate_ret_code (uint8_t *ret_code*)

This function evaluates a function return code and puts the device to sleep if the return code indicates that the device is awake.

Parameters

in	*ret_code*	return code of the last call to a SHA204 library function

3.12.2.2 int main (void)

This function serves as an example for the SHA204 MAC command.

```
In an infinite loop, it issues the same command
sequence using the Command Marshaling layer of
the SHA204 library.
```

Returns

exit status of application

3.13 sha204_i2c.c File Reference

Functions for I2C Physical Hardware Independent Layer of SHA204 Library.

```
#include "i2c_phys.h"
#include "sha204_physical.h"
#include "sha204_lib_return_codes.h"
#include "timer_utilities.h"
```

Macros

- #define SHA204_I2C_DEFAULT_ADDRESS ((uint8_t) 0xC8)

  TWI address used at SHA204 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 I2C read or write addressing.

Functions

- void sha204p_set_device_id (uint8_t id)

  This I2C function sets the I2C address. Communication functions will use this address.
- void sha204p_init (void)

  This I2C function initializes the hardware.
- uint8_t sha204p_wakeup (void)

  This I2C function generates a Wake-up pulse and delays.
- uint8_t sha204p_send_command (uint8_t count, uint8_t *command)

  This I2C function sends a command to the device.
- uint8_t sha204p_idle (void)

  This I2C function puts the SHA204 device into idle state.
- uint8_t sha204p_sleep (void)

  This I2C function puts the SHA204 device into low-power state.
- uint8_t sha204p_reset_io (void)

  This I2C function resets the I/O buffer of the SHA204 device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t *response)

  This TWI function receives a response from the SHA204 device.
- uint8_t sha204p_resync (uint8_t size, uint8_t *response)

  This I2C function resynchronizes communication.

3.13.1 Detailed Description

Functions for I2C Physical Hardware Independent Layer of SHA204 Library.

Author

Atmel Crypto Products

Date

February 2, 2011

3.13.2 Enumeration Type Documentation

3.13.2.1 enum i2c_read_write_flag

This enumeration lists flags for I2C read or write addressing.

Enumerator:

I2C_WRITE write command flag

I2C_READ read command flag

3.13.2.2 enum i2c_word_address

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

The following byte stream is sent to a SHA204 TWI device: {I2C start} {I2C address} {word address} [{data}] {I2C 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.

3.13.3 Function Documentation

3.13.3.1 uint8_t sha204p_idle (void)

This I2C function puts the SHA204 device into idle state.

Returns

status of the operation

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

This TWI function receives a response from the SHA204 device.

Parameters

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

Returns

status of the operation

3.13.3.3 uint8_t sha204p_reset_io (void)

This I2C function resets the I/O buffer of the SHA204 device.

Returns

status of the operation

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

This I2C function resynchronizes communication.

Parameters are not used for I2C.

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

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

This I2C function sends a command to the device.

Parameters

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

Returns

status of the operation

3.13.3.6 void sha204p_set_device_id (uint8_t *id*)

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

Parameters

in	*id*	I2C address

3.13.3.7 uint8_t sha204p_sleep (void)

This I2C function puts the SHA204 device into low-power state.

Returns

status of the operation

3.13.3.8 uint8_t sha204p_wakeup (void)

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

Returns

status of the operation

3.14 sha204_lib_return_codes.h File Reference

SHA204 Library Return Code Definitions.

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

Macros

- #define SHA204_SUCCESS ((uint8_t) 0x00)
  Function succeeded.
- #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.

3.14.1 Detailed Description

SHA204 Library Return Code Definitions.

Author

Atmel Crypto Products

Date

September 27, 2010

3.14.2 Macro Definition Documentation

3.14.2.1 #define SHA204_SUCCESS ((uint8_t) 0x00)

Function succeeded.

3.15 sha204_physical.h File Reference

Definitions and Prototypes for Physical Layer Interface of SHA204 Library.

```
#include <stdint.h>
#include "sha204_config.h"
```

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 I2C function sends a command to the device.
- uint8_t sha204p_receive_response (uint8_t size, uint8_t ∗response)

  This TWI function receives a response from the SHA204 device.
- void sha204p_init (void)

  This I2C function initializes the hardware.
- void sha204p_set_device_id (uint8_t id)

  This I2C function sets the I2C address. Communication functions will use this address.
- uint8_t sha204p_wakeup (void)

  This I2C function generates a Wake-up pulse and delays.
- uint8_t sha204p_idle (void)

  This I2C function puts the SHA204 device into idle state.
- uint8_t sha204p_sleep (void)

  This I2C function puts the SHA204 device into low-power state.
- uint8_t sha204p_reset_io (void)

  This I2C function resets the I/O buffer of the SHA204 device.
- uint8_t sha204p_resync (uint8_t size, uint8_t ∗response)

  This I2C function resynchronizes communication.

3.15.1 Detailed Description

Definitions and Prototypes for Physical Layer Interface of SHA204 Library.

Author

Atmel Crypto Products

Date

September 30, 2010

3.15.2 Function Documentation

3.15.2.1 uint8_t sha204p_idle (void)

This I2C function puts the SHA204 device into idle state.

Returns

status of the operation

This I2C function puts the SHA204 device into idle state.

Returns

status of the operation

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

This TWI function receives a response from the SHA204 device.

Parameters

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

Returns

status of the operation

This TWI function receives a response from the SHA204 device.

Parameters

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

Returns

status of the operation

3.15.2.3 uint8_t sha204p_reset_io (void)

This I2C function resets the I/O buffer of the SHA204 device.

Returns

status of the operation

This I2C function resets the I/O buffer of the SHA204 device.

Returns

success

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

This I2C function resynchronizes communication.

Parameters are not used for I2C.

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

Re-sychronizing 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

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

This I2C function sends a command to the device.

Parameters

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

Returns

status of the operation

This I2C function sends a command to the device.

Parameters

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

Returns

status of the operation

3.15.2.6 void sha204p_set_device_id (uint8_t *id*)

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

Parameters

in	*id*	I2C address

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

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

Parameters

in	*id*	index into array of pins

3.15.2.7 uint8_t sha204p_sleep (void)

This I2C function puts the SHA204 device into low-power state.

Returns

status of the operation

This I2C function puts the SHA204 device into low-power state.

Returns

status of the operation

3.15.2.8 uint8_t sha204p_wakeup (void)

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

Returns

status of the operation

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

Returns

success

3.16 sha204_swi.c File Reference

Functions for Single Wire, Hardware Independent Physical Layer of SHA204 Library.

```
#include "swi_phys.h"
#include "sha204_physical.h"
#include "sha204_lib_return_codes.h"
#include "timer_utilities.h"
```

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

  This SWI function selects the GPIO pin used for communication.
- uint8_t sha204p_send_command (uint8_t count, uint8_t *command)

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

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

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

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

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

  This SWI 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.

3.16.1 Detailed Description

Functions for Single Wire, Hardware Independent Physical Layer of SHA204 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 14, 2011

3.16.2 Function Documentation

3.16.2.1 uint8_t sha204p_idle (void)

This SWI function puts the device into idle state.

This I2C function puts the SHA204 device into idle state.

Returns

status of the operation

3.16.2.2 void sha204p_init (void)

This SWI function initializes the hardware.

This I2C function initializes the hardware.

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

This SWI function receives a response from the device.

This TWI function receives a response from the SHA204 device.

Parameters

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

Returns

status of the operation

3.16.2.4 uint8_t sha204p_reset_io (void)

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

This I2C function resets the I/O buffer of the SHA204 device.

Returns

success

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

This function re-synchronizes communication.

This I2C function resynchronizes communication.

Re-sychronizing 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

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

This SWI function sends a command to the device.

This I2C function sends a command to the device.

Parameters

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

Returns

status of the operation

3.16.2.7 void sha204p_set_device_id (uint8_t *id*)

This SWI function selects the GPIO pin used for communication.

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

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

Parameters

in	*id*	index into array of pins

3.16.2.8 uint8_t sha204p_sleep (void)

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

This I2C function puts the SHA204 device into low-power state.

Returns

status of the operation

3.16.2.9 uint8_t sha204p_wakeup (void)

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

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

Returns

success

3.17 swi_phys.h File Reference

Definitions and Prototypes for SWI Hardware Dependent Physical Layer of SHA204 Library.

```
#include <stdint.h>
```

Macros

- #define SWI_FUNCTION_RETCODE_SUCCESS ((uint8_t) 0x00)

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

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

  Communication failed after at least one byte was received.

Functions

- void swi_enable (void)

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

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

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

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

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

  This GPIO function receives bytes from an SWI device.

3.17.1 Detailed Description

Definitions and Prototypes for SWI Hardware Dependent Physical Layer of SHA204 Library.

Author

Atmel Crypto Products

Date

September 29, 2010

3.17.2 Function Documentation

3.17.2.1 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 rx buffer

Returns

status of the operation

3.17.2.2 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

3.17.2.3 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 tx buffer

Returns

status of the operation

3.17.2.4 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

3.17.2.5 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.\n
Another way to generate a Wake-up pulse is using the UART
at half the communication baud rate and sending a 0.
Keeping the baudrate at 230400 baud 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 baudrate and sending
a 0 produces a long enough Wake-up pulse of 69.4 us.\n
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.\n
Implementing a Wake-up pulse generation using the UART
would introduce a slight design flaw since this module
would now "know" something about the width of the Wake-up pulse.
We could add a function that sets the baudrate 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

3.18 timer_utilities.c File Reference

Timer Utility Functions.

```
#include <stdint.h>
```

Macros

- #define TIME_UTILS_US_CALIBRATION

  < data type definitions
- #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.

3.18.1 Detailed Description

Timer Utility Functions.

Author

Atmel Crypto Products

Date

February, 2011

3.18.2 Macro Definition Documentation

3.18.2.1 #define TIME_UTILS_US_CALIBRATION

< data type definitions

Fill the inner loop of delay_10us() with these CPU instructions to achieve 10 us per iteration.

3.18.3 Function Documentation

3.18.3.1 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

3.18.3.2 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

3.19 timer_utilities.h File Reference

Timer Utility Declarations.

```
#include <stdint.h>
```

Functions

- void delay_10us (uint8_t delay)

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

  This function delays for a number of milliseconds.

3.19.1 Detailed Description

Timer Utility Declarations.

Author

Atmel Crypto Products

Date

August 25, 2010

3.19.2 Function Documentation

3.19.2.1 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

3.19.2.2 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

3.20 uart_config.h File Reference

Definitions for Hardware Dependent Part of SHA204 Physical Layer Using a UART for Communication.

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

Macros

- #define BAUD_RATE (230400UL)

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

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

number of polling iterations over UART register before timing out

- #define RX_TX_DELAY ((uint8_t) (15.0 * TIME_PER_LOOP_ITERATION))

  Delay for this many loop iterations before sending.

- #define UART_GPIO_DDR DDRD

  direction register when using UART pin for Wake-up

- #define UART_GPIO_OUT PORTD

  output register when using UART pin for Wake-up

- #define UART_GPIO_PIN_RX _BV(PD2)

  bit position when using UART rx pin for Wake-up

- #define UART_GPIO_PIN_TX _BV(PD3)

  bit position when using UART tx pin for Wake-up

- #define DEBUG_LOW

  undefine debugging macro

- #define DEBUG_HIGH

  undefine debugging macro

3.20.1 Detailed Description

Definitions for Hardware Dependent Part of SHA204 Physical Layer Using a UART for Communication.

Author

Atmel Crypto Products

Date

October 19, 2010

3.20.2 Macro Definition Documentation

3.20.2.1 #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.

3.21 uart_phys.c File Reference

Physical Layer Functions of SHA204 Library When Using UART.

```
#include "swi_phys.h"
#include "uart_config.h"
#include "avr_compatible.h"
```

Functions

- void swi_set_device_id (uint8_t id)

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

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

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

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

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

  This UART function receives bytes from an SWI device.

3.21.1 Detailed Description

Physical Layer Functions of SHA204 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

September 27, 2012

3.21.2 Function Documentation

3.21.2.1 void swi_enable (void)

This UART function initializes the hardware.

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

3.21.2.2 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 rx buffer

Returns

status of the operation

3.21.2.3 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

3.21.2.4 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 tx buffer

Returns

status of the operation

3.21.2.5 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

3.21.2.6 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.\n
Another way to generate a Wake-up pulse is using the UART
at half the communication baud rate and sending a 0.
```

```
Keeping the baudrate at 230400 baud 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 baudrate and sending
a 0 produces a long enough Wake-up pulse of 69.4 us.\n
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.\n
Implementing a Wake-up pulse generation using the UART
would introduce a slight design flaw since this module
would now "know" something about the width of the Wake-up pulse.
We could add a function that sets the baudrate 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

Index