Crypto Hash Library

Microchip Libraries for Applications (MLA)

Table of Contents

Crypto Hash Library

1 Crypto Hash Library

1.1 Introduction

This library provides convenient C language implementations of several cryptographic hash algorithms for the Microchip family of microcontrollers.

Description

This library provides convenient C language implementations of several cryptographic hash algorithms for the Microchip family of microcontrollers.

Hash functions are designed to map an arbitrary amount of data to a fixed size value. For example, the SHA-1 hashing algorithm will accept a variable-length message, perform some calculation on it, and produce a 32-byte output. Hashes have a wide variety of applications in computer programming.

Cryptographic hash functions are hash functions used for security purposes, like data authentication or data integrity verification. They are specifically designed to make it difficult to find multiple message texts that could produce the same hash value.

Implementations of the MD5, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 cryptographic hash algorithms are included with this library. Some of the hashing algorithms are similar enough that they can be grouped together. The SHA-224 and SHA-256 hashes are both implemented in the SHA-256 sub-module, and the SHA-384 and SHA-512 hashes are implemented in the SHA-512 sub-module.

1.2 Legal Information

This software distribution is controlled by the Legal Information at www.microchip.com/mla_license

1.3 Release Notes

Release notes for the current version of the Crypto Hash Library.

Description

Crypto Hash Library Version : 1.00

This is the first release of the library.

Tested with MPLAB XC16 v1.11.

1.4 Using the Library

This topic describes the basic architecture of the Crypto Hash Library and provides information and examples on how to use it.

Description

This topic describes the basic architecture of the Crypto Hash Library and provides information and examples on how to use it.

Interface Header File: crypto_hash.h

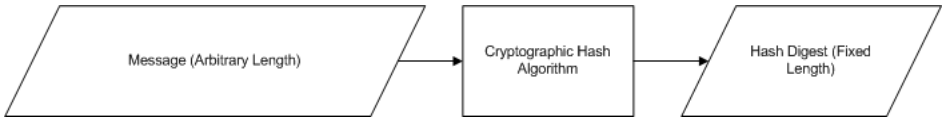
The interface to the Crypto Hash library is defined in the "crypto_hash.h" header file. Any C language source (.c) file that uses the Crypto Hash library should include "crypto_hash.h".

1.4.1 Abstraction Model

This library provides the low-level abstraction of the Crypto Hash module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Cryptographic Hash Software Abstraction Block Diagram



In the crypto hash module, the user will perform a cryptographic hash algorithm on a message of arbitrary length to produce a fixed-length hash digest.

1.4.2 Library Overview

Provides an overview of library functionality.

Description

The library interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Crypto Hash module.

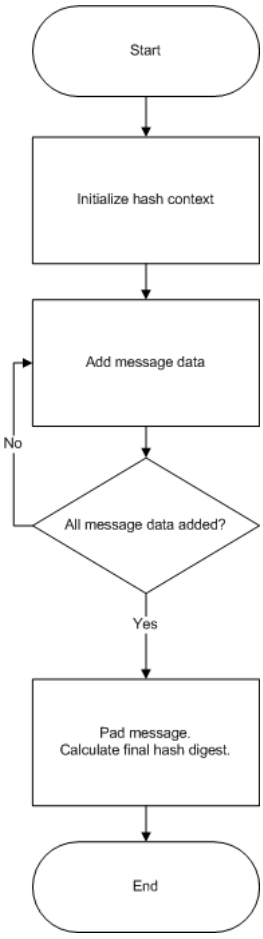
Library Interface Section	Description
MD5	Provides an implementation of the MD5 cryptographic hash function. Note that MD5 **is not recommended** for new designs.
SHA-1	Provides an implementation of the SHA-1 cryptographic hash function. Note that SHA-1 **is not recommended** for new designs.
SHA-256	Provides implementations of the SHA-224 and SHA-256 cryptographic hash functions.
SHA-512	Provides implementations of the SHA-384 and SHA-512 cryptographic hash functions.

1.4.3 How the Library Works

Provides a description of how the library works.

Description

Each algorithm implemented by the crypto hash library operates in the same way. First, the user will initialize a hash "context," which will store temporary hash values and information about the hash being calculated. The user will then provide some message data to the hash module. The cryptographic hash algorithms implemented by this library will parse the message data into blocks and run the hash algorithm on each of those blocks. When the user has passed all of the message to the hash module, he or she will instruct the module to finish calculating the hash. The module will add any necessary padding to the message, finish calculating the hash, and return a message digest to the user.



1.5 Configuring the Library

Configuration information for the Crypto Hash Library.

Macros

Name	Description
CRYPTO_HASH_CONFIG_SHA_SMALL_RAM	Allows the user to select an alternate implementation for the working buffers used by the SHA hash algorithm family.

Description

The configuration of the Crypto Hash module is based on the file sha_config.h. A copy of this file should be included in the system_config.h header file used to set system configuration.

Based on the selections made, the Cypto Hash module will support or not support selected features. These configuration settings will apply to all instances of this module.

The sha_config and system_config headers can be placed anywhere; however, the path of these headers needs to be present in the include search path for a successful build. Refer to the Application Overview section for more details.

1.5.1 CRYPTO_HASH_CONFIG_SHA_SMALL_RAM Macro

Allows the user to select an alternate implementation for the working buffers used by the SHA hash algorithm family.

File

sha_config_template.h

Syntax

```
#define CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
```

Description

The CRYPTO_HASH_CONFIG_SHA_SMALL_RAM option will decrease the required size of the working buffers for the SHA hash family. Enabling this configuration option will reduce the required RAM buffer size to 16 words. The word size depends on the algorithm: for SHA-1, SHA-224, and SHA-256, the word size is 32 bits. For SHA-384 and SHA-512, the word size is 64 bits. Enabling this option will slightly decrease performance.

1.6 Library Interface

This section describes the Application Programming Interface (API) functions of the Crypto Hash module.

Refer to each sub-section for a detailed description.

Modules

Name	Description
MD5	This section describes the types and functions used for the MD5 crypto hash module. Note that MD5 **is not recommended** for new designs.
SHA-1	This section describes the types and functions used for the SHA-1 crypto hash module. Note that SHA-1 **is not recommended** for new designs.
SHA-256	This section describes the types and functions used for the SHA-256 crypto hash module.
SHA-512	This section describes the types and functions used for the SHA-512 crypto hash module.

Description

1.6.1 MD5

This section describes the types and functions used for the MD5 crypto hash module.

Note that MD5 **is not recommended** for new designs.

Functions

	Name	Description
	MD5_Initialize	Initializes an MD5 context to perform an MD5 hash.
	MD5_DataAdd	Adds data to a hash being calculated.
	MD5_Calculate	Finishes calculating a hash.

Structures

Name	Description
MD5_CONTEXT	Context storage for a hash operation

Description

1.6.1.1 MD5_CONTEXT Structure

File

md5.h

Syntax

```
typedef struct {
  uint32_t h0;
```

```
  uint32_t h1;
  uint32_t h2;
  uint32_t h3;
  uint32_t bytesSoFar;
  uint8_t partialBlock[64];
} MD5_CONTEXT;
```

Members

Members	Description
uint32_t h0;	Hash state h0
uint32_t h1;	Hash state h1
uint32_t h2;	Hash state h2
uint32_t h3;	Hash state h3
uint32_t bytesSoFar;	Total number of bytes hashed so far
uint8_t partialBlock[64];	Beginning of next 64 byte block

Module

MD5

Description

Context storage for a hash operation

1.6.1.2 MD5_Initialize Function

Initializes an MD5 context to perform an MD5 hash.

File

md5.h

Syntax

```
void MD5_Initialize(MD5_CONTEXT* context);
```

Module

MD5

Returns

None.

Description

This routine initializes a hash context for the MD5 hash.

Remarks

You must initialize a context before calculating an MD5 hash.

Preconditions

None.

Example

```
MD5_CONTEXT context;
MD5_Initialize (&context);
```

Parameters

Parameters	Description
MD5_CONTEXT* context	The context to initialize.

Function

void MD5_Initialize(MD5_CONTEXT* context);

1.6.1.3 MD5_DataAdd Function

Adds data to a hash being calculated.

File

md5.h

Syntax

```
void MD5_DataAdd(MD5_CONTEXT* context, uint8_t* data, uint16_t len);
```

Module

MD5

Returns

None.

Description

This routine adds data to an MD5 hash being calculated. When the data length reaches a block size (64 bytes), this function will calculate the hash over that block and store the current hash value in the hash context.

Remarks

None.

Preconditions

The hash context must be initialized with MD5_Initialize.

Example

```
uint8_t data[] = "Hello.";
MD5_CONTEXT context;

MD5_Initialize (&context);
MD5_DataAdd (&context, data, 6);
```

Parameters

Parameters	Description
MD5_CONTEXT* context	The context of the hash being calculated.
uint8_t* data	The data being added.
uint16_t len	The length of the data being added.

Function

void MD5_DataAdd(MD5_CONTEXT* context, uint8_t* data, uint16_t len);

1.6.1.4 MD5_Calculate Function

Finishes calculating a hash.

File

md5.h

Syntax

```
void MD5_Calculate(MD5_CONTEXT* context, uint8_t* result);
```

Module

MD5

Returns

None.

Description

This routine finishes calculating an MD5 hash. It will automatically add the padding required by the hashing algorithm and return the hash digest.

Remarks

None.

Preconditions

The hash context must be initialized with MD5_Initialize.

Example

```
uint8_t data[] = "Hello.";
MD5_CONTEXT context;
uint8_t digest[16];

MD5_Initialize (&context);
MD5_DataAdd (&context, data, 6);
MD5_Calculate (&context, digest);
```

Parameters

Parameters	Description
MD5_CONTEXT* context	The context of the hash being calculated.
uint8_t* result	A 16-byte buffer to store the calculated hash digest.

Function

void MD5_Calculate(MD5_CONTEXT* context, uint8_t* result);

1.6.2 SHA-1

This section describes the types and functions used for the SHA-1 crypto hash module.

Note that SHA-1 **is not recommended** for new designs.

Functions

	Name	Description
	SHA1_Initialize	Initializes a SHA-1 context to perform a SHA-1 hash.
	SHA1_DataAdd	Adds data to a hash being calculated.
	SHA1_Calculate	Finishes calculating a hash.

Structures

Name	Description
SHA1_CONTEXT	Context storage for a hash operation

Description

1.6.2.1 SHA1_CONTEXT Structure

File

sha1.h

Syntax

```
typedef struct {
  uint32_t h0;
  uint32_t h1;
  uint32_t h2;
  uint32_t h3;
  uint32_t h4;
  uint32_t bytesSoFar;
  uint32_t * workingBuffer;
  uint8_t partialBlock[64];
} SHA1_CONTEXT;
```

Members

Members	Description
uint32_t h0;	Hash state h0
uint32_t h1;	Hash state h1
uint32_t h2;	Hash state h2
uint32_t h3;	Hash state h3
uint32_t h4;	Hash state h4
uint32_t bytesSoFar;	Total number of bytes hashed so far
uint32_t * workingBuffer;	Pointer to a working buffer for hash calculation
uint8_t partialBlock[64];	Beginning of next 64 byte block

Module

SHA-1

Description

Context storage for a hash operation

1.6.2.2 SHA1_Initialize Function

Initializes a SHA-1 context to perform a SHA-1 hash.

File

sha1.h

Syntax

```
void SHA1_Initialize(SHA1_CONTEXT* context, uint32_t * workingBuffer);
```

Module

SHA-1

Returns

None.

Description

This routine initializes a hash context for the SHA-1 hash.

Remarks

You must initialize a context before calculating a SHA-1 hash.

Preconditions

None.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint32_t buffer[16];
```

```
SHA1_CONTEXT context;
SHA1_Initialize (&context, buffer);
```

Parameters

Parameters	Description
SHA1_CONTEXT* context	The context to initialize.
uint32_t * workingBuffer	A working buffer used by the module to calculate the hash. If the CRYPTO_HASH_CONFIG_SHA_SMALL_RAM macro is defined in sha_config.h, this buffer must contain 16 uint32_t words. Otherwise, this buffer must contain 80 32-bit words, but performance will be slightly improved.

Function

void SHA1_Initialize(SHA1_CONTEXT* context, uint8_t * workingBuffer);

1.6.2.3 SHA1_DataAdd Function

Adds data to a hash being calculated.

File

sha1.h

Syntax

```
void SHA1_DataAdd(SHA1_CONTEXT* context, uint8_t * data, uint16_t len);
```

Module

SHA-1

Returns

None.

Description

This routine adds data to a SHA-1 hash being calculated. When the data length reaches a block size (64 bytes), this function will calculate the hash over that block and store the current hash value in the hash context.

Remarks

None.

Preconditions

The hash context must be initialized with SHA1_Initialize.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint8_t data[] = "Hello.";
uint32_t buffer[16];
SHA1_CONTEXT context;

SHA1_Initialize (&context, buffer);

SHA1_DataAdd (&context, data, 6);
```

Parameters

Parameters	Description
SHA1_CONTEXT* context	The context of the hash being calculated.
uint8_t * data	The data being added.
uint16_t len	The length of the data being added.

Function

void SHA1_DataAdd (SHA1_CONTEXT* context, uint8_t * data, uint16_t len);

1.6.2.4 SHA1_Calculate Function

Finishes calculating a hash.

File

sha1.h

Syntax

```
void SHA1_Calculate(SHA1_CONTEXT* context, uint8_t * result);
```

Module

SHA-1

Returns

None.

Description

This routine finishes calculating a SHA-1 hash. It will automatically add the padding required by the hashing algorithm and return the hash digest.

Remarks

None.

Preconditions

The hash context must be initialized with SHA1_Initialize.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint8_t data[] = "Hello.";
uint32_t buffer[16];
SHA1_CONTEXT context;
uint8_t digest[20];

SHA1_Initialize (&context, buffer);

SHA1_DataAdd (&context, data, 6);
SHA1_Calculate (&context, digest);
```

Parameters

Parameters	Description
SHA1_CONTEXT* context	The context of the hash being calculated.
uint8_t * result	A 20-byte buffer to store the calculated hash digest.

Function

void SHA1_Calculate(SHA1_CONTEXT* context, uint8_t * result);

1.6.3 SHA-256

This section describes the types and functions used for the SHA-256 crypto hash module.

Enumerations

Name	Description
SHA256_BIT_LENGTH	Enumeration for selecting output bit length

Functions

	Name	Description
	SHA256_Initialize	Initializes a SHA-256 context to perform a SHA-256 hash.
	SHA256_DataAdd	Adds data to a hash being calculated.
	SHA256_Calculate	Finishes calculating a hash.

Structures

Name	Description
SHA256_CONTEXT	Context storage for hash operation

Description

1.6.3.1 SHA256_BIT_LENGTH Enumeration

File

sha256.h

Syntax

```
typedef enum {
  SHA2_224,
  SHA2_256
} SHA256_BIT_LENGTH;
```

Members

Members	Description
SHA2_224	SHA-224 hash
SHA2_256	SHA-256 hash

Module

SHA-256

Description

Enumeration for selecting output bit length

1.6.3.2 SHA256_CONTEXT Structure

File

sha256.h

Syntax

```
typedef struct {
  uint32_t h[8];
  uint32_t totalBytes;
  uint8_t partialBlock[64];
  uint32_t * workingBuffer;
  SHA256_BIT_LENGTH length;
} SHA256_CONTEXT;
```

Members

Members	Description
uint32_t h[8];	Hash state
uint32_t totalBytes;	Total number of bytes hashed so far
uint8_t partialBlock[64];	Beginning of next 64 byte block

uint32_t * workingBuffer;	64 32-bit words for the working buffer
SHA256_BIT_LENGTH length;	Type of hash being calculated (SHA-224 or 256)

Module

SHA-256

Description

Context storage for hash operation

1.6.3.3 SHA256_Initialize Function

Initializes a SHA-256 context to perform a SHA-256 hash.

File

sha256.h

Syntax

```
void SHA256_Initialize(SHA256_CONTEXT * context, SHA256_BIT_LENGTH length, uint32_t *
workingBuffer);
```

Module

SHA-256

Returns

None.

Description

This routine initializes a hash context for the SHA-256 hash.

Remarks

You must initialize a context before calculating a SHA-256 hash.

Preconditions

None.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint32_t buffer[16];
SHA256_CONTEXT context;
SHA256_Initialize (&context, SHA2_256, buffer);
```

Parameters

Parameters	Description
SHA256_CONTEXT * context	The context to initialize.
SHA256_BIT_LENGTH length	Digest bit length to use with the SHA-256 algorithm. SHA2_224 or SHA2_256.
uint32_t * workingBuffer	A working buffer used by the module to calculate the hash. If the CRYPTO_HASH_CONFIG_SHA_SMALL_RAM macro is defined in sha_config.h, this buffer must contain 16 uint32_t words. Otherwise, this buffer must contain 64 32-bit words, but performance will be slightly improved.

Function

void SHA256_Initialize (SHA256_CONTEXT * context, SHA256_BIT_LENGTH length, uint32_t * workingBuffer);

1.6.3.4 SHA256_DataAdd Function

Adds data to a hash being calculated.

File

sha256.h

Syntax

```
void SHA256_DataAdd(SHA256_CONTEXT * context, uint8_t * data, uint32_t len);
```

Module

SHA-256

Returns

None.

Description

This routine adds data to a SHA-256 hash being calculated. When the data length reaches a block size (64 bytes), this function will calculate the hash over that block and store the current hash value in the hash context.

Remarks

None.

Preconditions

The hash context must be initialized with SHA256_Initialize.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint8_t data[] = "Hello.";
uint32_t buffer[16];
SHA256_CONTEXT context;

SHA256_Initialize (&context, SHA2_256, buffer);

SHA256_DataAdd (&context, data, 6);
```

Parameters

Parameters	Description
SHA256_CONTEXT * context	The context of the hash being calculated.
uint8_t * data	The data being added.
uint32_t len	The length of the data being added.

Function

void SHA256_DataAdd (SHA256_CONTEXT * context, uint8_t * data, uint32_t len);

1.6.3.5 SHA256_Calculate Function

Finishes calculating a hash.

File

sha256.h

Syntax

```
void SHA256_Calculate(SHA256_CONTEXT * context, uint8_t * result);
```

Module

SHA-256

Returns

None.

Description

This routine finishes calculating a SHA-256 hash. It will automatically add the padding required by the hashing algorithm and return the hash digest.

Remarks

None.

Preconditions

The hash context must be initialized with SHA256_Initialize.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint8_t data[] = "Hello.";
uint32_t buffer[16];
SHA256_CONTEXT context;
uint8_t digest[32];

SHA256_Initialize (&context, SHA2_256, buffer);

SHA256_DataAdd (&context, data, 6);
SHA256_Calculate (&context, digest);
```

Parameters

Parameters	Description
SHA256_CONTEXT * context	The context of the hash being calculated.
uint8_t * result	A buffer to store the calculated hash digest. 32 bytes for SHA-256, 28 bytes for SHA-224.

Function

void SHA256_Calculate (SHA256_CONTEXT * context, uint8_t * result);

1.6.4 SHA-512

This section describes the types and functions used for the SHA-512 crypto hash module.

Enumerations

Name	Description
SHA512_BIT_LENGTH	

Functions

	Name	Description
	SHA512_Initialize	Initializes a SHA-256 context to perform a SHA-512 hash.
	SHA512_DataAdd	Adds data to a hash being calculated.
	SHA512_Calculate	Finishes calculating a hash.

Structures

Name	Description
SHA512_CONTEXT	Context storage for hash operation

Description

1.6.4.1 SHA512_BIT_LENGTH Enumeration

File

sha512.h

Syntax

```
typedef enum {
  SHA2_384,
  SHA2_512
} SHA512_BIT_LENGTH;
```

Members

Members	Description
SHA2_384	SHA-384 hash
SHA2_512	SHA-512 hash

Module

SHA-512

Section

Data Types

****

Enumeration for selecting digest bit length

1.6.4.2 SHA512_CONTEXT Structure

File

sha512.h

Syntax

```
typedef struct {
  uint64_t h[8];
  uint32_t totalBytes;
  uint8_t partialBlock[128];
  uint64_t * workingBuffer;
  SHA512_BIT_LENGTH length;
} SHA512_CONTEXT;
```

Members

Members	Description
uint64_t h[8];	Hash state
uint32_t totalBytes;	Total number of bytes hashed so far
uint8_t partialBlock[128];	Beginning of the next 128 byte block
uint64_t * workingBuffer;	80 word uint64_t working buffer
SHA512_BIT_LENGTH length;	Type of hash being calculated (SHA-384 or SHA-512)

Module

SHA-512

Description

Context storage for hash operation

1.6.4.3 SHA512_Initialize Function

Initializes a SHA-256 context to perform a SHA-512 hash.

File

sha512.h

Syntax

```
void SHA512_Initialize(SHA512_CONTEXT * context, SHA512_BIT_LENGTH length, uint64_t *
workingBuffer);
```

Module

SHA-512

Returns

None.

Description

This routine initializes a hash context for the SHA-512 hash.

Remarks

You must initialize a context before calculating a SHA-512 hash.

Preconditions

None.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint64_t buffer[16];
SHA512_CONTEXT context;
SHA512_Initialize (&context, SHA2_512, buffer);
```

Parameters

Parameters	Description
SHA512_CONTEXT * context	The context to initialize.
SHA512_BIT_LENGTH length	Digest bit length to use with the SHA-512 algorithm. SHA2_384 or SHA2_512.
uint64_t * workingBuffer	A working buffer used by the module to calculate the hash. If the CRYPTO_HASH_CONFIG_SHA_SMALL_RAM macro is defined in sha_config.h, this buffer must contain 16 uint64_t words. Otherwise, this buffer must contain 80 64-bit words, but performance will be slightly improved.

Function

void SHA512_Initialize (SHA512_CONTEXT * context, SHA512_BIT_LENGTH length, uint32_t * workingBuffer);

1.6.4.4 SHA512_DataAdd Function

Adds data to a hash being calculated.

File

sha512.h

Syntax

```
void SHA512_DataAdd(SHA512_CONTEXT * context, uint8_t * data, uint32_t len);
```

Module

SHA-512

Returns

None.

Description

This routine adds data to a SHA-512 hash being calculated. When the data length reaches a block size (128 bytes), this function will calculate the hash over that block and store the current hash value in the hash context.

Remarks

None.

Preconditions

The hash context must be initialized with SHA512_Initialize.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint8_t data[] = "Hello.";
uint64_t buffer[16];
SHA512_CONTEXT context;

SHA512_Initialize (&context, SHA2_512, buffer);

SHA512_DataAdd (&context, data, 6);
```

Parameters

Parameters	Description
SHA512_CONTEXT * context	The context of the hash being calculated.
uint8_t * data	The data being added.
uint32_t len	The length of the data being added.

Function

void SHA512_DataAdd (SHA512_CONTEXT * context, uint8_t * data, uint32_t len);

1.6.4.5 SHA512_Calculate Function

Finishes calculating a hash.

File

sha512.h

Syntax

```
void SHA512_Calculate(SHA512_CONTEXT * context, uint8_t * result);
```

Module

SHA-512

Returns

None.

Description

This routine finishes calculating a SHA-512 hash. It will automatically add the padding required by the hashing algorithm and return the hash digest.

Remarks

None.

Preconditions

The hash context must be initialized with SHA512_Initialize.

Example

```
// Initialization for CRYPTO_HASH_CONFIG_SHA_SMALL_RAM
uint8_t data[] = "Hello.";
uint64_t buffer[16];
SHA512_CONTEXT context;
uint8_t digest[64];

SHA512_Initialize (&context, SHA2_512, buffer);

SHA512_DataAdd (&context, data, 6);
SHA512_Calculate (&context, digest);
```

Parameters

Parameters	Description
SHA512_CONTEXT * context	The context of the hash being calculated.
uint8_t * result	A buffer to store the calculated hash digest. 48 bytes for SHA-384, 64 bytes for SHA-512.

Function

void SHA512_Calculate (SHA512_CONTEXT * context, uint8_t * result);

Index