

C2000 SHA384 Library

USER'S GUIDE



Table of Contents

1. Introduction
 2. SHA384 APIs
 3. Example
 4. Performance and memory details
-

1. Introduction

This library contains the implementation of SHA384 hashing algorithm.

The library contains two implementations of the algorithm:

1. **SHA384 Bytewise** : Implementation that accepts the input message to be hashed as bytes.
2. **SHA384 Wordwise**: Implementation that accepts the input message to be hashed as 32-bit words.
Note: In SHA384 Wordwise implementation, length (in bytes) of the input message to be hashed must be a multiple of 4

Example projects *sha384_ex12_bytewise* and *sha384_ex13_wordwise* are provided along with the library as a reference to the users.

2. SHA384 APIs

Data structures of the type *SHA384_ObjectByteWise* (bytewise implementation) or *SHA384_ObjectWordWise* (wordwise implementation) must be initialized and passed as arguments to SHA384 APIs as described below.

2.1 SHA384_hashByteWise()

Prototype

```
uint16_t SHA384_hashByteWise(SHA384_HandleByteWise handle, uint16_t *data,
                             size_t length, uint64_t digest[6])
```

Description

Perform a complete hash operation when input is supplied byte-wise, producing a final digest for the data.

- This function wraps *SHA384_startByteWise()*, *SHA384_addDataByteWise()*, and *SHA384_finalizeByteWise()*.
- There is no need to call *SHA384_startByteWise()* prior to calling this function.
- The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.)

Parameters

handle A 'SHA384_HandleByteWise' handle

data 16-bit pointer pointing to the memory location of the starting byte the data to be hashed.

length Length of the data (in number of bytes) to be hashed.

digest Output location for the final digest

Returns

SHA384_STATUS_SUCCESS The hash operation succeeded.

SHA384_STATUS_LENGTH_TOO_LARGE The requested length of data to hash is more than the implementation supports

SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

2.2 SHA384_hashWordWise()

Prototype

```
uint16_t SHA384_hashWordWise(SHA384_HandleWordWise handle, uint32_t *data,
                             size_t length, uint64_t digest[6])
```

Description

Perform a complete hash operation when input is supplied byte-wise, producing a final digest for the data.

- This function wraps SHA384_startWordWise(), SHA384_addDataWordWise(), and SHA384_finalizeWordWise().
- There is no need to call SHA384_startWordWise() prior to calling this function.
- The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.)

Parameters

handle A 'SHA384_HandleWordWise' handle

data 32-bit pointer pointing to the memory location of the starting byte the data to be hashed.

length Length of the data (in number of bytes) to be hashed. Note that for word-wise input, length of the data should be a multiple of 4

digest Output location for the final digest

Returns

SHA384_STATUS_SUCCESS The hash operation succeeded.

SHA384_STATUS_LENGTH_TOO_LARGE The requested length of data to hash is more than the implementation supports

SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

SHA384_STATUS_NOT_ALIGNED – Length is not a multiple of 4

2.3

Prototype

```
uint16_t SHA384_startByteWise(SHA384_HandleByteWise handle)
```

Description

Initialize a 'SHA384SW_HandleByteWise' handle, preparing for hashing data

Parameters

handle A 'SHA384_HandleByteWise' handle

Returns

SHA384_STATUS_SUCCESS The operation succeeded.

SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

2.4

Prototype

```
uint16_t SHA384_startWordWise(SHA384_HandleWordWise handle)
```

Description

Initialize a 'SHA384SW_HandleWordWise' handle, preparing for hashing data

Parameters

handle A 'SHA384_HandleWordWise' handle

Returns

SHA384_STATUS_SUCCESS The operation succeeded.
SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

2.5

Prototype

```
uint16_t SHA384_addDataByteWise(SHA384_HandleByteWise handle, uint16_t *data,  
                                size_t length)
```

Description

Add data to SHA384 operation when inputs are supplied byte-wise

- Adds data to a hash operation. The @c handle must have been initialized by a call to SHA384_startByteWise first.
- The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.).
- After passing in all data to be hashed, call #SHA384_finalizeWordWise() to obtain the final digest.

Parameters

handle A 'SHA384_HandleByteWise' handle

data 16-bit pointer pointing to the memory location of the starting byte the data to be hashed.

length Length of the data (in number of bytes) to be hashed.

Returns

SHA384_STATUS_SUCCESS The hash operation succeeded.
SHA384_STATUS_LENGTH_TOO_LARGE The requested length of data to hash is more than the implementation supports
SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

2.6

Prototype

```
uint16_t SHA384_addDataWordWise(SHA384_HandleWordWise handle, uint32_t *data,
                                size_t length)
```

Description

Add data to SHA384 operation when inputs are supplied as 32-bit words

- Adds data to a hash operation. The @c handle must have been initialized by a call to SHA384_startWordWise first.
- The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.).
- After passing in all data to be hashed, call #SHA384_finalizeWordWise() to obtain the final digest.

Parameters

handle A 'SHA384_HandleWordWise' handle

data 32-bit pointer pointing to the memory location of the starting byte the data to be hashed.

length Length of the data (in number of bytes) to be hashed. Note that for word-wise input, length of the data should be a multiple of 4

digest Output location for the final digest

Returns

SHA384_STATUS_SUCCESS The hash operation succeeded.

SHA384_STATUS_LENGTH_TOO_LARGE The requested length of data to hash is more than the implementation supports

SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

SHA384_STATUS_NOT_ALIGNED – Length is not a multiple of 4

2.7

Prototype

```
uint16_t SHA384_finalizeByteWise(SHA384_HandleByteWise handle,
                                  uint64_t digest[6])
```

Description

Finalize the SHA384 operation when input is supplied byte-wise, creating the final digest.

- After calling this function, @c handle should not be used again until it has been reinitialized via a call to SHA384_startByteWise().

Parameters

handle A 'SHA384_HandleByteWise' handle

digest Output location for the final digest

Returns

SHA384_STATUS_SUCCESS The hash operation succeeded.
 SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

2.8

Prototype

```
uint16_t SHA384_finalizeWordWise(SHA384_HandleWordWise handle,
                                  uint64_t digest[6])
```

Description

Finalize the SHA384 operation when input is supplied as 32-bit words, creating the final digest.

- After calling this function, @c handle should not be used again until it has been reinitialized via a call to SHA384_startWordWise().

Parameters

handle A 'SHA384_HandleWordWise' handle

digest Output location for the final digest

Returns

SHA384_STATUS_SUCCESS The hash operation succeeded.
 SHA384_STATUS_NULL_INPUT One or more of the pointer inputs is NULL

2.9

Prototype

```
void SHA384_cancelByteWise(SHA384_HandleByteWise handle)
```

Description

Cancels SHA384 operation by clearing intermediate data stored in the 'SHA384_HandleByteWise' handle.

- The handle will not be ready for a new operation until after SHA384_startByteWise() is called on the handle.

Parameters

handle A 'SHA384_HandleByteWise' handle

Returns

None

2.10

Prototype

```
void SHA384_cancelWordWise(SHA384_HandleWordWise handle)
```

Description

Cancels SHA384 operation by clearing intermediate data stored in the 'SHA384_HandleWordWise' handle.

- The handle will not be ready for a new operation until after SHA384_startWordWise() is called on the handle.

Parameters

handle A 'SHA384_HandleWordWise' handle

Returns

None

2.11

Prototype

```
void SHA384_processBlockByteWise(uint64_t digest[8], uint16_t Ws[128])
```

Description

Performs core SHA2-256 algorithm on one 512 bit block of data when input is supplied byte-wise.

- object->digest will contain the updated digest resulting from hashing the block.

Parameters

digest pre-loaded with previous block's digest (or initial digest if call is to process first block). On function return digest will hold the updated digest.

Ws loaded with data block, including padding as specified by the SHA384 standard.

Returns

None

2.12

Prototype

```
void SHA384_processBlockWordWise(uint64_t digest[8], uint64_t Ws[16])
```

Description

Performs core SHA2-256 algorithm on one 512 bit block of data when input is supplied as 32-bit words.

- object->digest will contain the updated digest resulting from hashing the block.

Parameters

digest pre-loaded with previous block's digest (or initial digest if call is to process first block). On function return digest will hold the updated digest.

Ws loaded with data block, including padding as specified by the SHA384 standard.

Returns

None

3. Example

Example projects are provided along with the SHA384 library and acts as a reference to the users on how we use the SHA384 APIs to hash a given input message to generate the message digest.

Example projects include:

- 1.) *sha384_ex12_bytewise*: Example project using Bytewise APIs
- 2.) *sha384_ex13_wordwise*: Example project using Wordwise APIs

4. Performance and memory details

The table below lists the performance numbers of the library when run in an F28004x device at 100MHz.

	SHA384 HASHING ALGORITHM (speed)	
	Execution from FLASH	Execution from RAM
SHA384 Bytewise	0.60 Mb/s	0.92 Mb/s
SHA384 Wordwise	1.09 Mb/s	1.64 Mb/s

Note: The above measurements were made at an `--opt_for_speed=5, -opt_level = 2` setting

The table below lists the code size of the files that are included in the library.

File	Code size in 16-bit words	Data size in 16-bit words
sha384.c (Bytewise implementation)	1290	320
sha384.c (Wordwise implementation)	986	320

Note: The above code sizes are at an `--opt_for_speed=5, --opt_level=2` setting