ColdFire/ColdFire+ CAU and Kinetis mmCAU Software Library

User Guide

Rev. 2.3

Freescale Semiconductor Inc.
Microcontroller Solutions Group

Contents

Revision History

Revision	Author	Date	Changes
1.0	JPW	Feb-07	Launch release for CAU software library
2.0	PA	Apr-11	Updated for mmCAU software library
2.1	PA	May-11	Minor fixes
2.2	LS	May-11	API minor fixes
2.3	JC, DS, JPW	Aug-11	Document rewording

1 Purpose

The following document shows details on how to use the CAU and mmCAU software library in any application that needs to integrate a cryptographic algorithm or hashing function supported by the software library. Freescale products supported by the software library are ColdFire, ColdFire+ and Kinetis MCU/MPUs. Check the specific Freescale product for CAU and mmCAU availability.

2 Overview

The ColdFire/ColdFire+ CAU (cryptographic acceleration unit) software library is a set of low-level cryptographic functions implemented using CAU co-processor instructions. The Kinetis mmCAU (memory mapped cryptographic unit) software library uses the mmCAU co-processor that is connected to the Kinetis ARM Cortex-M4 Private Peripheral Bus (PPB). During the rest of the document, CAU refers to both CAU and mmCAU unless explicitly noted. Also, both software libraries share the same application programming interface (API); both interfaces are referenced as a single definition throughout the document unless explicitly noted.

2.1 Library Features

The library is as compact and generic as possible to simplify integration with existing cryptographic software. The library has a standard header file with ANSI C prototypes for all functions. This software library is thread safe only if CAU registers are saved on a context switch. The ColdFire/ColdFire+ CAU software library adheres to ColdFire C-language compiler conventions (ABI) using standard parameter passing (stack passing). The Kinetis mmCAU software library is also compatible to ARM C compiler conventions (EABI). Only ColdFire and Kinetis devices with the CAU or mmCAU hardware coprocessors are supported. For best performance, input and output data blocks must be aligned on 0-modulo-4 byte addresses.

The CAU library supports the following encryption/decryption algorithms and hashing functions:

- AES128
- AES192
- AES256
- DES
- 3DES
- MD5
- SHA1
- SHA256

Note:

3DES crypto algorithms are supported by calling the corresponding DES crypto function three times.

Hardware support for SHA256 is only present in CAU version 2. Go to the appropriate MCU/MPU reference manual for details of availability. Additionally, the cau_sha256_initialize_output() function checks the hardware revision and returns a (-1) value if the CAU lacks SHA256 support.

3 Software library details

3.1 CAU and mmCAU software library overview

Table 1 shows the crypto algorithms and hashing functions included in the software library:

Crypto Algorithms	AES128 AES192 AES256	cau_aes_set_key
		cau_aes_encrypt
		cau_aes_decrypt
	DES/3DES	cau_des_chk_parity
		cau_des_encrypt
		cau_des_decrypt
Hashing Functions	MD5	cau_md5_initialize_output
		cau_md5_hash_n
		cau_md5_update
		cau_md5_hash
	SHA1	cau_sha1_initialize_output
		cau_sha1_hash_n
		cau_sha1_update
		cau_sha1_hash
	SHA256	cau_sha256_initialize_output
		cau_sha256_hash_n
		cau_sha256_update
		cau_sha256_hash

Table 1: Library Overview

3.2 CAU and mmCAU software library usage

The software library contains the following files:

File	Description
cau_api.h	CAU and mmCAU header file
cau_lib.a	CAU library: ColdFire/ColdFire+
mmcau_lib.a	mmCAU library: Kinetis

Table 2: File Description

The header file must be always included in the project. Only cau_lib.a or mmcau_lib.a must be included in the project depending on CAU or mmCAU hardware presence.

The header file is shared between CAU and mmCAU software library, but only one of the following C-language macros must be defined for the project to include CAU or mmCAU respectively.

Macro	Description
FREESCALE_CAU	Using CAU software library
FREESCALE_MMCAU	Using mmCAU software library

Table 3: C-language Macros Description

Example:

```
#define FREESCALE_CAU   //CAU functions are implemented in cau_lib.a library
```

When using FREESCALE_CAU on ColdFire/ColdFire+ products, standard memory-based parameter passing is assumed. This feature assumes arguments are provided to the library functions using the stack and NOT by using CPU registers (register parameter passing). This argument passing format is required by Freescale CodeWarrior compiler. If using another compiler, parameter passing must be properly selected for argument stack usage.

Note:

For CodeWarrior, by adding the compiler directive ***__declspec (standard_abi)***, a function is forced to use standard parameter passing:

```
__declspec (standard_abi)
void
cau_aes_set_key (const unsigned char *key, const int key_size, unsigned char
*key_sch);
```

3.3 CAU and mmCAU software library application programming interface (API)

The library includes the following functions which are valid for CAU and mmCAU software libraries:

3.3.1 cau_aes_set_key

AES: Performs an AES key expansion.

Prototype:

```
void
cau_aes_set_key(const unsigned char *key,
                const int           key_size,
                unsigned char       *key_sch
                )
```

Parameters:

[in]	*key	pointer to input key (128, 192, 256 bits in length)
[in]	key_size	key size in bits (128, 192, 256)
[out]	*key_sch	pointer to key schedule output (44, 52, 60 longwords)

Returns:

None

Notes:

Table 4 shows the requirements for the cau_aes_set_key() function when using AES128, AES192 or AES256.

AES cau_aes_set_key()	
[in] Block Size (bits)	**[out]** Key Schedule Size (32-bit data values)
128	44
192	52
256	60

Table 4: AES set_key Requirements

3.3.2 cau_aes_encrypt

AES: Encrypts a single 16-byte block for AES128, AES192 or AES256

Prototype:

```
void
cau_aes_encrypt (const unsigned char *in,
                 const unsigned char *key_sch,
                 const int nr,
                 unsigned char *out
                )
```

Parameters:

[in]	*in	pointer to 16-byte block of input plaintext
[in]	*key_sch	pointer to key schedule (44, 52, 60 longwords)
[in]	nr	number of AES rounds (10, 12, 14 = f(key_schedule))
[out]	*out	pointer to 16-byte block of output ciphertext

Returns:

None

Notes:

Input and output blocks may overlap.
Table 5 shows the requirements for the cau_aes_encrypt()/cau_aes_decrypt() function when using AES128, AES192 or AES256.

AES cau_aes_encrypt()/cau_aes_decrypt()		
Block Cipher	**[in]** Key Schedule Size (longwords)	**[in]** Number of AES rounds
AES128	44	10
AES192	52	12
AES256	60	14

Table 5: AES Encryption Requirements

3.3.3 cau_aes_decrypt

AES: Decrypts a single 16-byte block for AES128, AES192 and AES256

Prototype:

```
void
cau_aes_decrypt (const unsigned char *in,
                 const unsigned char *key_sch,
                 const int nr,
                 unsigned char *out
                )
```

Parameters:

[in]	*in	pointer to 16-byte block of input ciphertext
[in]	*key_sch	pointer to key schedule (44, 52, 60 longwords)
[in]	nr	number of AES rounds (10, 12, 14 = f(key_schedule))
[out]	*out	pointer to 16-byte block of output plaintext

Returns:

None

Notes:

Input and output blocks may overlap.
Table 6 shows the requirements for the cau_aes_encrypt()/cau_aes_decrypt() function when using AES128, AES192 or AES256.

AES cau_aes_encrypt()/cau_aes_decrypt()		
Block Cypher	**[in]** Key Schedule Size (longwords)	**[in]** Number of AES rounds
AES128	44	10
AES192	52	12
AES256	60	14

Table 6: AES Decryption Requirements

3.3.4 cau_des_chk_parity

DES: Checks key parity

Prototype:

```
int
cau_des_chk_parity (const unsigned char *key
                    )
```

Parameters:

[in]	*key	pointer to 64-bit DES key with parity bits

Returns:

0	no error
-1	parity error

Notes:

None

3.3.5 cau_des_encrypt

DES: Encrypts a single 8-byte block

Prototype:

```
void
cau_des_encrypt (const unsigned char *in,
                 const unsigned char *key,
                 unsigned char *out
                 )
```

Parameters:

[in]	*in	pointer to 8-byte block of input plaintext
[in]	*key	pointer to 64-bit DES key with parity bits
[out]	*out	pointer to 8-byte block of output ciphertext

Returns:

None

Notes:

Input and output blocks may overlap.

3.3.6 cau_des_decrypt

DES: Decrypts a single 8-byte block

Prototype:

```
void
cau_des_decrypt (const unsigned char *in,
                 const unsigned char *key,
                 unsigned char *out
                 )
```

Parameters:

[in]	*in	pointer to 8-byte block of input ciphertext
[in]	*key	pointer to 64-bit DES key with parity bits
[out]	*out	pointer to 8-byte block of output plaintext

Returns:

None

Notes:

Input and output blocks may overlap.

3.3.7 cau_md5_initialize_output

MD5: Initializes the MD5 state variables

Prototype:

```
void
cau_md5_initialize_output (const unsigned char *md5_state
                           )
```

Parameters:

[out]	*md5_state	pointer to 120-bit block of md5 state variables: a,b,c,d

Returns:

None

Notes:

None

3.3.8 cau_md5_hash_n

MD5: Updates MD5 state variables for one or more input message blocks

Prototype:

```
void
cau_md5_hash_n (const unsigned char *msg_data,
                const int num_blks,
                unsigned char *md5_state
                )
```

Parameters:

[in]	*msg_data	pointer to start of input message data
[in]	num_blks	number of 512-bit blocks to process
[in,out]	*md5_state	pointer to 128-bit block of MD5 state variables: a,b,c,d

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_md5_initialize_output() function must be called first.
Useful when handling non-contiguous input message blocks.

3.3.9 cau_md5_update

MD5: Updates MD5 state variables for one or more input message blocks

Prototype:

```
void
cau_md5_update (const unsigned char *msg_data,
                const int num_blks,
                unsigned char *md5_state
                )
```

Parameters:

[in]	*msg_data	pointer to start of input message data
[in]	num_blks	number of 512-bit blocks to process
[in,out]	*md5_state	pointer to 128-bit block of MD5 state variables: a,b,c,d

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_md5_initialize_output() function is not required.
All input message blocks must be contiguous.

3.3.10 cau_md5_hash

MD5: Performs MD5 hash algorithm for a single input message block

Prototype:

```
void
cau_md5_hash (const unsigned char *msg_data,
              unsigned char *md5_state
              )
```

Parameters:

[in]	*msg_data	pointer to start of input message data
[in,out]	*md5_state	pointer to 128-bit block of MD5 state variables: a,b,c,d

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_md5_initialize_output() function must be called first.
Only works for a single input message block.

3.3.11 cau_sha1_initialize_output

SHA1: Initializes the SHA1 state variables

Prototype:

```
void
cau_sha1_initialize_output (const unsigned int *sha1_state
                            )
```

Parameters:

[out]	*sha1_state	pointer to 160-bit block of SHA1 state variables: a,b,c,d,e

Returns:

None

Notes:

None

3.3.12 cau_sha1_hash_n

SHA1: Perform the hash and generate SHA1 state variables for one or more input message blocks

Prototype:

```
void
cau_sha1_hash_n (const unsigned char *msg_data,
                 const int num_blks,
                 unsigned int *sha1_state
                 )
```

Parameters:

[in]	*msg_data	pointer to start of input message data
[in]	num_blks	number of 512-bit blocks to process
[in,out]	*sha1_state	pointer to 160-bit block of SHA1 state variables: a,b,c,d,e

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_sha1_initialize_output() function must be called first.
Useful when handling non-contiguous input message blocks.

3.3.13 cau_sha1_update

SHA1: Updates SHA1 state variables for one or more input message blocks arguments

Prototype:

```
void
cau_sha1_update (const unsigned char *msg_data,
                 const int num_blks,
                 unsigned int *sha1_state
                 )
```

Parameters:

[in]	*msg_data	pointer to start of input message data
[in]	num_blks	number of 512-bit blocks to process
[out]	*sha1_state	pointer to 160-bit block of SHA1 state variables: a,b,c,d,e

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_sha1_initialize_output() function is not required.
All input message blocks must be contiguous.

3.3.14 cau_sha1_hash

SHA1: Performs SHA1 hash algorithm on a single input message block

Prototype:

```
void
cau_sha1_hash (const unsigned char *msg_data,
               unsigned int *sha1_state
               )
```

Parameters:

[in]	*msg_data	pointer to start of input message data
[in,out]	*sha1_state	pointer to 160-bit block of SHA1 state variables: a,b,c,d,e

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_sha1_initialize_output() function must be called first.
Only works for a single input message block.

3.3.15 cau_sha256_initialize_output

SHA256: Initializes the hash output and checks the CAU hardware revision

Prototype:

```
int
cau_sha256_initialize_output (const unsigned int *output
                              )
```

Parameters:

[out]	*output	pointer to 256-bit message digest output

Returns:

0	No error. CAU2 hardware present. SHA256 HW support available
-1	Error. CAU2 hardware not present. No SHA256 HW support available

Notes:

None

3.3.16 cau_sha256_hash_n

SHA256: Updates SHA256 digest output for one or more message block arguments

Prototype:

```
void
cau_sha256_hash_n (const unsigned char *input,
                   const int num_blks,
                   unsigned int *output
                   )
```

Parameters:

[in]	*input	pointer to start of input message
[in]	num_blks	number of 512-bit blocks to process
[in,out]	*output	pointer to 256-bit message digest output

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_sha256_initialize_output() function must be called first.
Useful when handling non-contiguous input message blocks.

3.3.17 cau_sha256_update

SHA256: Updates SHA256 state variables for one or more input message blocks arguments

Prototype:

```
void
cau_sha256_update (const unsigned char *input,
                   const int num_blks,
                   unsigned int *output
                   )
```

Parameters:

[in]	*input	pointer to start of input message data
[in]	num_blks	number of 512-bit blocks to process
[out]	*output	pointer to 256-bit message digest output

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_sha256_initialize_output() function is not required.
All input message blocks must be contiguous.

3.3.18 cau_sha256_hash

SHA256: Performs SHA256 hash algorithm for a single input message block

Prototype:

```
void
cau_sha256_hash (const unsigned char *input,
                 unsigned int *output
                 )
```

Parameters:

[in]	*input	pointer to start of input message data
[in,out]	*output	pointer to 256-bit message digest output

Returns:

None

Notes:

Input message and digest output blocks must not overlap.
The cau_sha256_initialize_output() function must be called first.
Only works for a single input message block.

4 Considerations

4.1 Problem reporting instructions

Issues and suggestions about this document and software should be provided through the support web page at www.freescale.com/support.

4.2 Considerations and references

- CAU and mmCAU software libraries require the presence in hardware of CAU or mmCAU respectively.
- Review your target MCU/MPU for CAU or mmCAU availability.
- For examples on how to use CAU and mmCAU software libraries, review AN4307
- Freescale NanoSSL and NanoSSH low level crypto/hashing drivers use CAU and mmCAU software library. Details at www.freescale.com/nanossl and www.freescale.com/nanossh.
- Freescale Semiconductor, Inc. operates in accordance with United States export control laws and regulations and such other U.S. and other national laws as may apply. Download CAU and mmCAU libraries (CAU_MMCAU_SW.zip) from www.freescale.com by filling export control online form.