User's Guide

MSPM33C Security User's Guide

TEXAS INSTRUMENTS

ABSTRACT

This document serves as a guide for utilizing the security functionality built into the MSPM33C device family during application development. The functionality includes TrustZone® and several security accelerators. For more information on implementing secure booting or secure debugging, please refer to the M33 TRM.

Table of Contents

Trademarks

TrustZone®, Arm®, and Cortex® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

1 Overview of Security Functionality

The security functionality included in the MSPM33C device family create a framework for upholding the integrity and authenticity of any secure data deployed on the device.

Key security features that are included in the MSPM33C device family are:

- Secure Execution Environment
  - TrustZone®
    - Implementation Dependent Attribution Unit (IDAU)
    - Secure Attribution Unit (SAU)
  - Memory Protection Unit (MPU)
    - TrustZone® and MPU
  - Global Security Controller (GSC)
    - SRAM Protection Controller (SPC)
    - Flash Protection Controller (FPC)
    - Peripheral Protection Controller (PPC)
- Security Accelerators
  - Advanced Encryption Standard (AES)
  - Keystore
  - Secure Hash Algorithm (SHA)
  - Public Key Accelerator (PKA)
  - Post-Quantum Cryptography (PQC)

1.1 Terminology

Acronym/Terminology	Definition
GSC	Global Security Controller
PKA	Public Key Accelerator
SRAM	Synchronous Random Access Memory
MMR	Memory mapped register

2 Secure Execution Environment

The TrustZone® architecture is enabled through the Arm® Cortex®-M33. TrustZone® allows the programmer to partition sections of memory as secure and non-secure allowing the programmer to restrict sections of memory from unauthorized users. Additionally the MSPM33C devices include a global security controller (GSC) which provides additional security outside of the CPU subsystem (CPUSS). This prevents external peripherals, such as the DMA, from accessing secure regions of memory from an unsecure processor state.

2.1 TrustZone®

The Cortex®-M33 TrustZone® security allows the programmer to partition memory into three different types of regions: secure, non-secure, and non-secure callable. Secure regions of memory the user can access all regions of memory with no issues. non-secure regions of memory restrict the programmer to only access other non-secure or non-secure callable regions of memory. Non-secure callable regions of memory are unique where they are limited to only access non-secure regions of memory unless the user calls a SG instruction allowing the programmer to jump to a secure region of memory. for a more detailed description please see the TrustZone® technology for Armv8-M Architecture documentation.

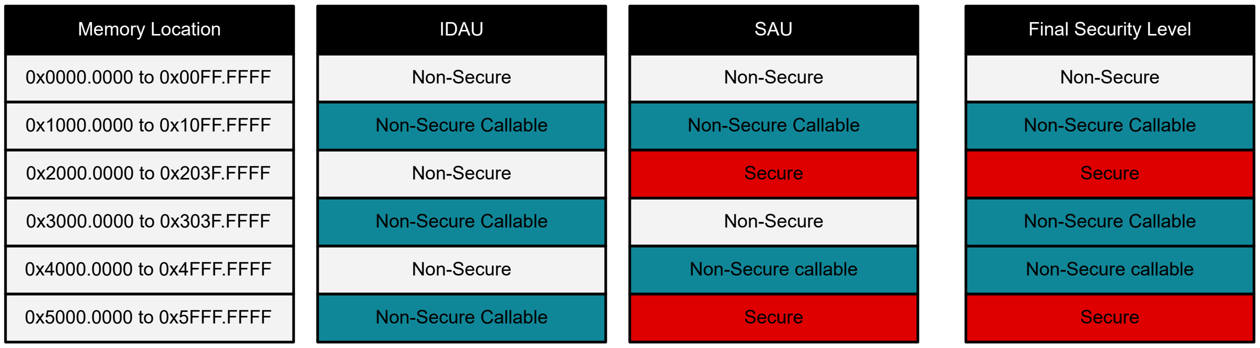
To define which regions of memory are secure and non-secure the Security Attribution unit (SAU) and Implementation Defined Attribution Unit (IDAU) are used. The IDAU is set at the RTL level and is configured by TI. For information on how the IDAU is configured please see the CPU section of the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual.

The SAU is used by the CPU subsystem (CPUSS) to define regions of memory as secure and non-secure. This peripheral allows the programmer to partition their code as secure and non-secure. For information on how the SAU and IDAU work together to define regions as secure or non-secure please see TrustZone® technology for Armv8-M Architecture documentation.

2.1.1 Implementation Defined Attribution Unit

The implementation defined attribution unit (IDAU) is defined by TI and is referenced in the CPU section of the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual. It is important to know how the IDAU is configured since the IDAU's configuration will be used with the SAU to configure the final security level of a region. For example in the MSPM33C32 devices the IDAU is configured such that when the 8th bit of the address is 1 the memory region is defined as non-secure callable and when it's 0 its only non-secure. For example the address 0x1000.0000 is defined as NSC.

The IDAU and SAU are both used to determine the final level of security. The highest security level out of the two is set as the final attributed level of security. For an example of this on the MSPM33C321A see Figure 2-1.



Memory Location	IDAU	SAU	Final Security Level
0x0000.0000 to 0x00FF.FFFF	Non-Secure	Non-Secure	Non-Secure
0x1000.0000 to 0x10FF.FFFF	Non-Secure Callable	Non-Secure Callable	Non-Secure Callable
0x2000.0000 to 0x203F.FFFF	Non-Secure	Secure	Secure
0x3000.0000 to 0x303F.FFFF	Non-Secure Callable	Non-Secure	Non-Secure Callable
0x4000.0000 to 0x4FFF.FFFF	Non-Secure	Non-Secure callable	Non-Secure callable
0x5000.0000 to 0x5FFF.FFFF	Non-Secure Callable	Secure	Secure

Figure 2-1. IDAU and SAU on MSPM33C321A configuration example

2.1.2 Security Attribution Unit

The security attribution unit (SAU) can be used by the programmer to define regions of memories security. The final security level uses both the SAU configuration and the IDAU configuration. For determining a section of memories security level take the higher security level between the SAU and IDAU. For example if the SAU defines a region as secure and the IDAU defines a region as non-secure the region is defined as secure.

On the MSPM33 devices the SAU is built to support defining 8 regions as Secure, non-secure, or non-secure callable.

Defining a region as secure

One major use case for the SAU is for is defining a region as secure or non-secure. To define a region the following steps are taken by writing to the SAU's registers

1. Selecting which region to configure through the SAU->RNR register
2. Selecting the regions base address using the SAU->RBAR register
3. Set the SAU->RLAR to configure the size of the region
4. Repeat steps 1 - 3 for all regions
5. Configure the SAU->CTL register to propagate all of the configured regions

2.1.3 TrustZone® software development

To integrate using the SAU in the programmers code development we recommend using the CMSE library. This library is designed by Arm® and allows the user to easily define functions as secure or non-secure. For more detailed information on the guidelines for this please see the ARMv8-M Secure software guidelines.

The ARMv8-M instruction uses BXNS (Branch and Exchange Non-secure) and SG (secure gateway) to transition between non-secure and secure code. The BXNS is used to transition from secure code to non-secure code and the SG is used to transition from non-secure to secure. The SG command utilizes the NSC regions to make this transition. To utilize these instructions easily into your programming environment, the CMSE library integrates these functions into the function calls.

Defining functions as secure and non-secure

One of the key features of the CMSE library is it allows the programmer to define functions as secure or non-secure. This determines the CPU state allowing the programmer to jump between secure and non-secure regions. To add the CMSE library into your program use the following header.

```
#include <arm_cmse.h>
```

To implement the SG command into your code you can give a function the cmse_nonsecure_entry attribute. This function definition calls a secure gateway veneer at the start of the code. This is useful for defining functions non-secure code can access that uses information in secure memory.

```
__attribute__((cmse_nonsecure_entry)) secure_fxn()
{

}
```

For calling non-secure functions form secure code you can use the BXNS function. To easily integrate this into your programming environment using the CMSE library you can give the function a the cmse_nonsecure_call attribute. See the below code block for an example.

```
__attribute__((cmse_nonsecure_call)) nonsecure_fxn()
{

}
```

2.2 Memory Protection Unit

In the MSPM33C devices an memory protection unit (MPU) is present in the CPUSS. The MPU allows the programmer to define regions of memory as privileged and unprivileged . This acts similar to how the SAU defines regions as secure and non-secure. The key difference is privileged mode changes based on if the device is in thread or handler mode.

Out of reset the device is in thread mode and goes to handler mode if an exception is issued by the processor or peripherals. Exceptions will always be in privilege mode and thread mode the MPU can be used to determine the privilege level. Please see MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual for more details.

2.2.1 TrustZone® and MPU

In the MSPM33C CPUSS the MPU, SAU, and IDAU are used to attribute regions of memory as a combination of different privilege and security levels. This allows the programmer to layer different levels of access by using privilege and security restrictions. An example of the combinations the programmer can use is seen in Figure 2-2. When the device is in Handler mode, the processor is always in a privileged state.

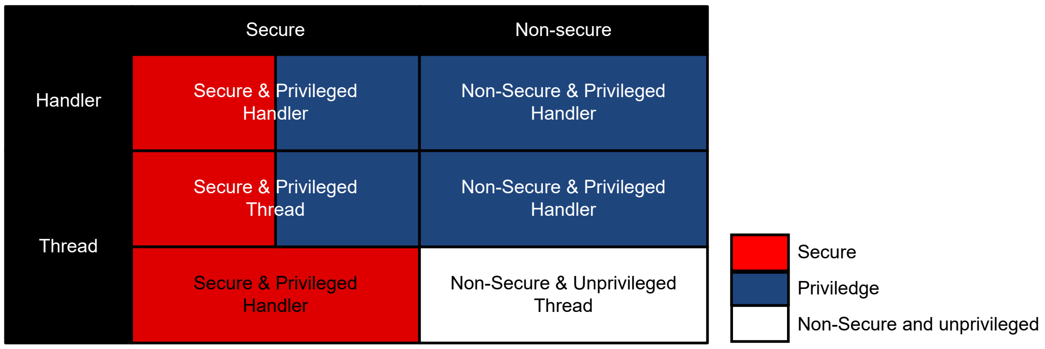

Secure
Non-secure
Handler
Thread
Secure & Privileged Handler
Non-Secure & Privileged Handler
Secure & Privileged Thread
Non-Secure & Privileged Handler
Secure & Privileged Handler
Non-Secure & Unprivileged Thread
Secure
Priviledge
Non-Secure and unprivileged

Figure 2-2. Security and Privileged memory attributes

2.3 Global Security Controller

The Global Security Controller (GSC) is a unique peripheral to MSP and adds an extra layer of security on top of the TrustZone® architecture. The GSC configures both secure and privilege attributes for peripherals, flash, and SRAM. This allows the programmer to use one IP to protect all of their resources and protects against non-secure peripherals accessing secure regions of memory. One example of this would be a non-secure DMA reading secure memory. The GSC allows the programmer to protect against these attacks by throwing faults when a non-secure DMA attempts to access a secure region of memory. For more detailed information on the GSC please see the global security controller chapter of the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual .

2.3.1 GSC Memory configuration

The GSC utilizes both the MPU and SAU to configure memory with security and privilege attributes. To configure this the GSC has three separate controllers: Peripheral Protection Controller (PPC), SRAM Protection Controller (SPC), and Flash Protection Controller (FPC). For details on how to configure and use the registers please see the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual

The GSC has similar properties to the SAU and MPU. This can lead confusion on which peripheral should be used for protecting memory. The SAU and MPU are limited on how many regions can configured or address granularity. For example the SAU for M33 devices can only attribute 8 regions of memory with the non-secure and secure attribute. Unconfigured regions of memory are left with the secure attribute. The GSC however attributes regions in a 2kB size allowing 255 different regions of security on a 1 MB device. Each one of these 2kB regions can be attributed either secure or privilege access.

2.3.1.1 Security Exceptions through the GSC & SAU

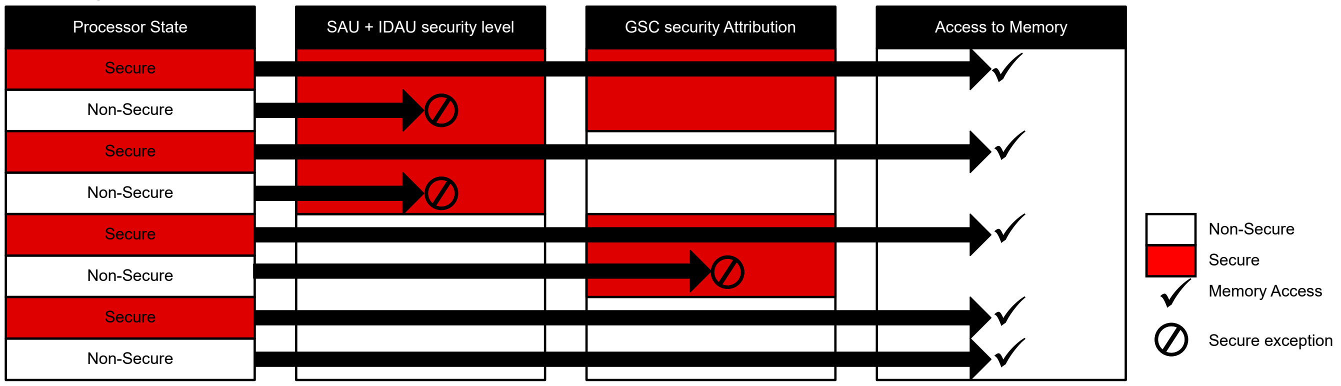
Both the GSC and SAU have capabilities to attribute regions of memory with security. One major difference between the two is the GSC will generate a secure NMI while the SAU generates a secure fault. A secure NMI is handled through the error aggregation module (EAM) which allows the programmer to see specific details about the error. Such as which peripheral caused the error and address the error occurred. The GSC also isn't limited to 8 regions instead the flash, SRAM, and Peripheral have their own granularity for security attributes. For more specifics please see MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual.

For understanding which error occurs when using the GSC and SAU to attribute regions of memory as secure or non-secure please see table Table 2-1. An image showing this can also be seen in Security exceptions with the GSC and SAU

Table 2-1. Security interrupt based on GSC and SAU security Attributions

Processor State	SAU + IDAU Security Attribution	GSC Security Attribution	Resulting Interrupt	Access violation
Secure	Secure	Secure	None	None
Non-secure	Secure	Secure	Security Fault	Yes blocked by CPU
Secure	Secure	Non-secure	None	None
Non-secure	Secure	Non-secure	Security Fault	Yes blocked by CPU
Secure	Non-secure	Secure	None	None
Non-secure	Non-secure	Secure	Security NMI	Yes blocked by GSC
Secure	Non-secure	Non-secure	None	None
Non-secure	Non-secure	Non-secure	None	None

Security exceptions with the GSC and SAU



2.3.1.2 Priviledge exceptions with GSC & MPU

The GSC and MPU can be used together to define regions of memory as priviledge or unpriviledge similar to how the GSC and SAU define regions of memory as secure or non-secure. This allows the programmer to add another layer of protection on regions of memory by using priviledge access. Due to the priviledge state also changing with exceptions occuring make sure priviledge access is expected when an interrupt occurs.

Use Table 2-2 to determine which fault will occur first when using the GSC and MPU to attribute regions of memory as priviledge or unpriviledge.

Table 2-2. Priviledge exception based on GSC and MPU priviledge attributions

Processor State	MPU Privilege Attribution	GSC Privilege Attribution	Resulting Interrupt	Access violation
Privilege	Privilege	Privilege	None	None
Unprivilege	Privilege	Privilege	MemMange fault	Yes blocked by CPU
Privilege	Privilege	Unprivilege	None	None
Unprivilege	Privilege	Unprivilege	MemMange fault	Yes blocked by CPU
Privilege	Unprivilege	Privilege	None	None
Unprivilege	Unprivilege	Privilege	Priviledge NMI	Yes blocked by GSC
Privilege	Unprivilege	Unprivilege	None	None
Unprivilege	Unprivilege	Unprivilege	None	None

3 Security Modules

The M33C device family include several modules that can be utilized to implement a secure execution environment for customers. These security modules include:

- Advanced Encryption Standard (AES)
- Keystore
- Secure Hash Algorithm (SHA)
- Public Key Accelerator (PKA)
- Post-Quantum Cryptography (PQC)
  - ML-DSA

3.1 AES

The AES accelerator module accelerates encryption and decryption operations in hardware based on the FIPS PUB 197 AES.

3.1.1 AES Overview

The AES accelerator module performs encryption and decryption of 128-bit data blocks with a 128-bit or 256-bit key in hardware. AES is a symmetric-key block cipher algorithm specified in FIPS PUB 197.

The AES accelerator features include:

- AES 128-bit block encryption and decryption
- Key scheduling in hardware
- Enc/decrypt only modes: CBC, CFB-1, CFB-8, CFB-128, OFB-128, CTR/ICM
- Authentication only modes: CBC-MAC, CMAC
- AES-CCM
- AES-GCM
- AES-CCM and AES-GCM modes support continuation with hold/resume of payload data
- 32-bit word access to provide key data, input data, and output data
- AES ready interrupt
- DMA triggers for input/output data
- Supported in RUN and SLEEP (see the *Operating Modes* section of the device technical reference manual)

A high level block diagram of the AES engine is shown in Figure 3-1. The AES engine consists of a processing core that performs both encryption/decryption as well as Galois field multiplication. The core is driven with configuration and data inputs that software will configure via memory mapped registers.

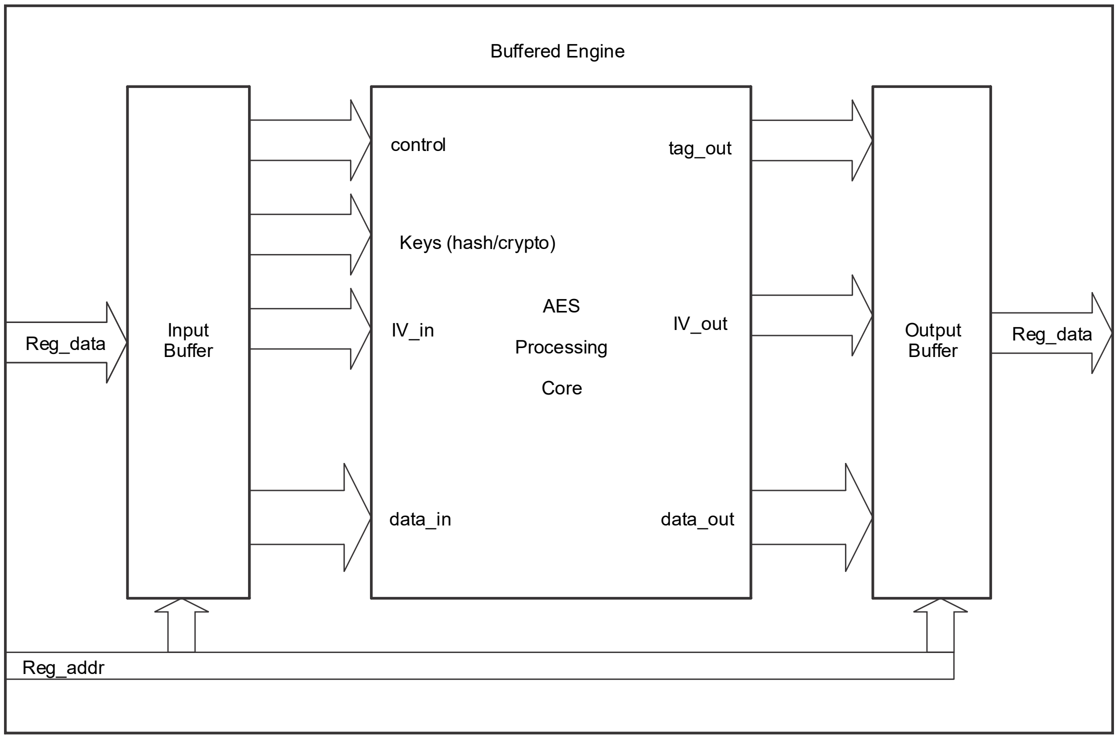


Figure 3-1. AES Block Diagram

3.1.2 AES Usage

Usage of the AES module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the AES DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder, located in the latest release of the MSPM33 SDK.

3.1.2.1 Configuration

Configuration of the AES module can be done via SysConfig. Under Basic Configuration users can:

- Configure aesadvhp key length (128/256-bit key length)
- Configure the operation type (Encryption / Decryption)
- Configure cipher mode

Check the data sheet of your selected device for more detailed information.

3.1.2.2 Setup

Standard setup for the AES module is performed in four parts:

1. Setup of AES state change monitoring
   a. The CPU can be notified of AES state changes via two methods (Triggered Interrupts and Register Polling). If AES Interrupts are used to monitor state changes, these interrupts need to be configured and enabled prior to initializing the AES module.
2. Setup of AES key
   a. The AES key must be set in the AES module before the module is initialized.
3. Setup of AES cipher mode
   a. The cipher mode selected in the SysConfig configuration, can be initialized after Steps 1 & 2
4. Wait for AES module to be ready
   a. The CPU can be notified that the AES module is ready via the state change monitoring methods selected in Step 1

3.1.2.3 Operation

After configuration and setup are complete, data can now be loaded into the AES module to be either encrypted or decrypted. Once the data is loaded into the module, the CPU must wait until the module has finished processing the data before reading the output encrypted/decrypted data. This wait can be done via the state monitoring method selected previously. If the operation type is changed at any point, the AES module must be re-initialized.

3.2 Keystore

3.2.1 Overview

The Keystore controller provides secure management of the Advanced Encryption Engine (AES) keys. The use model of the keystore controller is to securely deposit keys into it during the execution of customer secure code and to have the AES engine access them subsequently in a secure manner without leaking any key data to observers. Both 128 and 256-bit keys can be stored in the keystore key slots. The keystore and its interaction with the AES engine are designed for secure operation including thwarting partial key modification attacks.

3.2.2 Keystore Usage

Usage of the Keystore module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the Keystore DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder.

3.2.2.1 Configuration

Configuration of the Keystore module can be done via SysConfig. Under Basic Configuration users can:

- Configure key storage slot (0-3) or slots (for 256-bit keys)
- Configure key length (128/256-bit key length)
- Configure key crypto recipient (AES module)
- Configure key load location

Check the data sheet of your selected device for more detailed information.

3.2.2.2 Setup

Standard setup for the Keystore module is performed in two parts:

1. Setup of number of 256-bit keys
   a. The Keystore module must be notified of the number of 256-bit keys, so that the available key slots can be adjusted. All 256-bit keys are written to the lowest slot available.
2. Setup for key writing
   a. The Keystore module must have it's key write configuration performed before any keys can be transferred.

3.2.2.3 Operation

After configuration and setup are complete, the key transfer can be initiated by providing the Keystore module pointer and the key transfer configuration.

3.3 SHA2

The SHA2 (Secure Hash Algorithm-2) provides a set of cryptographic hash functions in accordance with NIST standards. This chapter describes the operations of the SHA module.

3.3.1 SHA Introduction

The SHA cryptographic peripheral supports a FIPS compliant secure hash algorithm (SHA-224, SHA-256) and the hash-based message authentication (HMAC) that can be used for message authentication applications.

3.3.1.1 SHA features

- Secure Hash Standard compliant implementation for FIPS PUB 180-2 and FIPS PUB 180-3

- HMAC support for all algorithms compliant with FIP PUB 198-1
- High performance hash with performance on one hash iteration (7.88 bits) per clock cycle for faster throughput
- Supports HMAC and basic hash operation for SHA-224 and SHA-256
- Hash and HMAC context switching
- Supports MAC key XOR and message padding
- Supports MAC key shorter, equal or larger than algorithm block size
- Supports automatic message data scheduling
- Supports message sizes up to 2^{64} bits in increments of 8-bits
- Autonomous support for DMA request generation based on message length to reduce CPU overhead
- Supports both constant and incremental address for input message parsing

3.3.2 SHA Performance

Table 3-1 provides performance metrics for multi-block images with the MSPM33C321A running at 32 MHz.

Table 3-1. SHA Hardware Accelerator Key Performance Metrics (Multi-block)

Size of Image	SHA 224 HASH	SHA 224 HMAC	SHA 256 HASH	SHA 256 HMAC
1 kb	0.016 ms	0.018 ms	0.018 ms	0.018 ms
32 kb	0.436 ms	0.436 ms	0.436 ms	0.436 ms
64 kb	0.867 ms	0.866 ms	0.866 ms	0.868 ms
128 kb	1.811 ms	1.811 ms	1.802 ms	1.809 ms

3.3.3 SHA Usage

Usage of the SHA module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the SHA DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder, located in the latest release of the MSPM33 SDK.

3.3.3.1 Configuration

Configuration of the SHA module can be done via SysConfig. Under the basic configuration users can:

- Configure SHA Mode (HASH/HMAC)
- Configure SHA Algorithm (224/256)

Check the data sheet of your selected device for more detailed information.

3.3.3.2 Setup

Standard setup for the SHA module is performed in four parts:

1. Setup of SHA state monitoring
   a. The CPU can be notified of SHA state changes via two methods (Triggered Interrupts and Register Polling). If SHA Interrupts are used to monitor state changes, these interrupts need to be configured and enabled prior to using the SHA module.
2. Setup SHA data length
   a. The SHA data length needs to be configured before using the SHA module
   b. The data length must be specified in bytes with the SHA mode includes
   c. The SHA module can then be notified that the mode and data length are available
3. Setup up of MAC key (Only for HMAC mode)
   a. If using the SHA HMAC mode, the module must be given a MAC key to use for encryption before using the SHA module
   b. The MAC key must be written to the SHA module in words, including the total key size
   c. The SHA module must then be notified that the key is available
   d. The SHA module must be notified that it can begin processing the key
   e. Finally, the CPU must wait until the SHA module is done processing
4. Setup of Direct Memory Access (DMA)

a. The SHA module operates on blocks of data and must be manually fed data for the module to digest. To perform this, the data could be fed via the CPU or via a DMA transfer. If using a DMA transfer, the DMA channel must be setup prior to using the SHA module.
b. The DMA source address must point to the start of the message to be hashed
c. The DMA destination address must point to the SHAW_DATA_FIXED register
d. The DMA message length must be set to the message length in words
e. The DMA source and destination width must be the size of a word
f. The DMA source must be configured in single increment mode and the destination must not change
g. The DMA trigger must be set to the SHAW_TRIGGER
h. Finally, the DMA channel can be enabled

3.3.3.3 Operation

After configuration and setup are complete, data can now be loaded into the SHA module by either DMA or the CPU. Once the data is loaded into the module, the CPU must wait until the module has finished processing the data before reading the output hashed data. This wait can be done via the state monitoring method selected previously. After the hashed data is read from the SHA module, it must be released to be used for the next hashing request.

3.4 PKA

3.4.1 PKA Introduction

The PKA is an integrated module for public key acceleration to offload the computation of intensive public cryptography operations.

3.4.1.1 PKA features

- The PKA engine provides the following basic operations
  - Large vector addition, subtraction and combined addition and subtraction
  - Large vector shift left or right
  - Large vector multiplication and division (with or without quotient)
  - Large vector compare and copy
- The PKA engine provides the following complex operations:
  - Large vector unsigned value modular exponentiation
  - Large vector unsigned value modular exponentiation using the CRT method with pre-calculated Q inverse vector
  - Modular inversion
  - ECC operations on two type of curve: Montgomery curves like Curve25519 and Curve448, and any curve of the form $y^2=x^3+ax+b$ (mod p).
  - ECC point addition/doubling on elliptic curve with affine or projective points as input/output
  - ECC point multiplication on elliptic curve
- Illegal state and timing attack detection

3.4.2 PKA Usage

Usage of the PKA module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the PKA DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder.

3.4.2.1 Configuration

Configuration of the PKA module can be done via SysConfig by simply adding the PKA module to the project.

Check the data sheet of your selected device for more detailed information.

3.4.2.2 Setup

Setup requirements for the PKA module are handled by SysConfig as the initialization steps are automatically generated. All that is required is to call the generated *SYSCFG_DL_init()* function.

3.4.2.3 Operation

After configuration and setup are complete, the PKA module can now be used to perform the cryptographic operations. It is good practice to start a PKA operation and then wait for the operation to finish by polling the PKA status. After the PKA status clears, the result of the PKA operation can then be collected.

3.5 PQC

Post-Quantum Cryptography (PQC) is a set of ecryption standards that are designed to uphold against threats from quantum computers and are defined by the National Institute of Standards and Technology (NIST). One of these standards is the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)

3.5.1 ML-DSA

3.5.1.1 ML-DSA Introduction

ML-DSA is one of the NIST defined PQC algorithms used to protect against threats from quantum computers. This standard can be used with Customer Secure Code (CSC) to ensure the authenticity of an embedded application.

3.5.1.2 ML-DSA Usage

Usage of the ML-DSA standard is conveniently wrapped in the Texas Instruments MSPM33 Crypto Library. Examples utilizing the ML-DSA standard along with MCUBoot can be found in EVM specific, non-RTOS, examples sub-folder, located in the latest release of the MSPM33 SDK.

3.5.1.2.1 Configuration

Configuring MCUBoot to utilize ML-DSA can be done via SysConfig. Under the basic configuration, users can:

- Enable authentication using ML-DSA
- Enable authentication using ECDSA and ML-DSA

Check the data sheet of your selected device for more detailed information.

3.5.1.2.2 Setup

Setup for ML-DSA usage in MCU Boot can then be finalized by storing the MLDSA key and key length in the global bootutil_keys structure that MCUBoot utilizes. An example of this is shown below:

```
const struct bootutil_key bootutil_keys[] = {
    {
        .key = mldsa_pub_key,
        .len = &mldsa_pub_key_len,
    },
};
const int bootutil_key_cnt = 1;
```

3.5.1.2.3 Operation

After configuration and setup are complete, MCUBoot will now utilize ML-DSA, and the previously setup public ML-DSA key, to validate the application image when *boot_go* is initiated.

4 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

DATE	REVISION	NOTES
December 2025	*	Initial Release