Application Note

评估 MSPM33 MCU 上的客户安全代码

Yuhao Zhao

摘要

本文档介绍了如何在 HS-FS（高安全性工厂状态）生命周期状态的 MSPM33C32 器件（本文档中称为 C32）上评估客户安全代码。

内容

1 简介.....2
1.1 关键概念.....2
2 客户安全代码概述.....3
2.1 引导和启动序列.....3
2.2 CSC 流程.....4
2.3 FLASH 存储器映射.....7
2.4 存储体交换.....8
2.5 CSC 特性.....8
2.6 CSC 上的保护.....10
2.7 CSC 性能.....11
3 评估基于 CSC 的安全引导.....13
3.1 环境设置.....13
3.2 对 CSC 进行编程.....13
3.3 编程应用程序.....16
3.4 运行 CSC 应用程序.....21
4 问答.....24
4.1 如何修改应用程序映像版本.....24
4.2 如何修改应用程序映像地址.....24
4.3 如何修改插槽大小.....26
4.4 如何修改应用程序映像大小.....27
4.5 对应用程序映像或 CSC 执行更多自定义操作.....27
5 参考资料.....28
6 修订历史记录.....28

商标

所有商标均为其各自所有者的财产。

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

1 简介

客户安全代码 (CSC) 是针对 MSP 系列中具有高级安全功能（例如器件系列 MSPM33C321x）的器件公开可用的 MCUboot 实现，旨在让客户在开发和部署中拥有更多且更高级的安全引导功能。与应用程序映像搭配使用时，这为器件上的新映像更新和验证提供了完整的解决方案。但是，CSC 仅指映像验证和配置安全设置，而不是将映像加载到器件中。根据所使用的具体器件和器件上存在的功能，完整的功能集和执行流程会有所不同。

该 CSC 具有以下特性：

- 使用含 NIST P-256 曲线的椭圆曲线数字签名算法 (ECDSA) 和哈希算法 SHA-256 验证一个或多个应用程序映像
- 使用 ML-DSA 87 和哈希算法 SHA-256 验证一个或多个应用程序映像
- 开源 MCUboot 经修改可符合终端开发人员的需求。此外，CSC 具有一组安全功能，并且可以使用头文件或 SysConfig 轻松配置，以根据用户的最终需求定制设计
- 使用全局安全控制器 (GSC) 在器件上设置安全功能，例如写入、读取和 IP 保护防火墙
- 设置 KEYSTORE 和存储体交换策略，并根据用户需要动态管理和撤销密钥
- 回滚保护
- 能够使其不可更改，并在引导期间充当信任根 (RoT) 链的一部分。
- 通过 SysConfig 工具轻松配置功能，以便用户快速构建定制的 CSC

1.1 关键概念

NONMAIN	一个专用闪存区域，可配置器件引导相关参数。请参阅 *MSPM33 NONMAIN 闪存存储器配置指南* 了解 NONMAIN 配置。
安全启动	作为执行的先决条件，确认和验证可更新固件和软件元件的完整性和真实性的过程。
INITDONE	INITDONE 是 MSPM33 器件中的一个寄存器，用于隔离 CSC 和应用程序。INITDONE 在 CSC 结束时触发，而在 INITDONE 期间，CSC 中配置的所有非静态安全策略都将生效
客户安全代码 (CSC)	MSPM33 SDK 中针对具有 INITDONE 机制的器件提供的安全引导设计。CSC 是 RoT 的一部分，可在生产后保持不可更改，并实现应用程序映像完整性和真实性验证以及其他安全策略配置。CSC 还可以表示 MSPM33 硬件特性，这意味着 MSPM33 器件支持 INITDONE 机制。
信任根 (RoT)	特别是指不可更改的信任根，这是设备上最受信任的安全组件。RoT 本质上是可信的，因为 RoT 不能在制造后进行修改。不存在任何更深层次的软件可以验证 RoT 是否真实且未经修改。在 CSC 选项中包括 ROM 引导代码及具有静态写保护的 CSC。
密钥库	AES 密钥的安全存储。只有 CSC 可以将密钥配置到密钥库中，主应用程序可以将加密引擎 (AES) 配置为使用存储的密钥之一，但永远无法访问任何存储的密钥。
防火墙	一种针对特定闪存存储器区域的动态保护机制，由全局安全控制器 (GSC) 实现
存储体交换	在 MSPM33 双组器件上配置闪存存储体地址映射的机制。存储体交换在 CSC 中配置，并在 INITDONE 之后生效。
静态写保护	由 NONMAIN 配置启用的静态写保护机制。除非更改了 NONMAIN 配置以再次启用写入，否则 ROM 引导代码完成后无法修改受保护区域。
SHA2-256	哈希算法采用整个消息并且将消息压缩成一个固定长度（256 位）摘要。此算法用来验证消息完整性。一些 MSPM33 器件为 SHA 提供硬件加速器。
ECDSA P256	椭圆曲线数字签名算法是一种用于验证消息真实性的非对称算法。一些 MSPM33 器件为 ECDSA 提供硬件加速器。
MLDSA-87	基于模格的数字签名算法是一种用于验证消息真实性的非对称算法。
AES	高级加密标准。一些 MSPM33 器件为 AES 提供硬件加速器。
TRNG	真随机数生成器。一些 MSPM33 器件为 TRNG 提供硬件加速器。

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

2 客户安全代码概述

本节概述了 CSC 流程以及用户指定的策略，可对这些策略进行设置以满足各种不同的使用场景。

2.1 引导和启动序列

图 2-1 展示了 CSC 引导和启动序列。在 BOOTRST 时，TI ROM 代码开始执行。成功引导后，引导代码会发出 BOOTDONE。此时，SYSCTL 向器件发出 SYSRST 以触发从 MAIN 闪存存储器执行。对于 HSFS 状态器件，MAIN 闪存程序总是在引导代码从物理地址 0x0004 向量（复位处理程序）执行完毕后启动。根据 NONMAIN 闪存 BCR 中的 CSCEXISTS 配置，BOOTDONE 之后有两个执行流程：

- CSCEXISTS 设置：CSC 引导序列启用，并且 MAIN 闪存程序以 INITDONE 在清除状态下启动。在这种情况下，用户需要将 CSC 固件放入 MAIN 闪存 0x0000 地址。用户可以在 NONMAIN BCR 配置中设置静态写保护策略以保护 CSC 固件。
- CSCEXISTS 清除：不允许 CSC 引导序列，MAIN 闪存程序在置位状态下以 INITDONE 启动。任何与安全相关的策略都是不可配置的，在这种情况下，用户需要将应用程序固件放入 MAIN 闪存 0x0000 地址中。

备注

由于在 BOOTRST 期间会复位存储体交换策略，因此 MAIN 闪存程序始终在 BOOTDONE 之后启动而不会进行存储体交换。仅当在 NONMAIN 配置中同时启用 CSCEXISTS 和 FLASHBANKSWAPPOLICY 时，存储体交换才会在 INITDONE 之后生效。

对于 CSC 现有的情况，CSC 负责确定执行存储体、将安全密钥初始化到 KEYSTORE 中、获取应用程序完整性和真实性验证等。器件在特权状态下工作，具有配置这些安全策略的权限。在 CSC 结束时会发出 INITDONE（通过写入 SYSCTL.SECCFG.INITDONE，请参阅特定于器件的技术参考手册以了解寄存器定义），然后 SYSCTL 会发出第二个 SYSRST，下面列出的所有安全策略在 INITDONE 期间生效，并且在下一个 BOOTRST 之前不修改：

- 存储体交换策略
- 密钥库保护

在 CSC 期间，可以针对安全策略配置全局安全控制器 (GSC)。GSC 中的矢量表偏移寄存器 (VTOR) 也会更新主应用程序的矢量表地址。

INITDONE 之后，SYSRST 被触发，器件从 VTOR 中的地址开始执行，并直接跳转到主应用程序。

有关引导和启动序列的更多详细信息，请参阅 *MSPM33 C3 系列 160MHz 微控制器* 技术参考手册的 *安全* 一章。

Copyright © 2026 Texas Instruments Incorporated

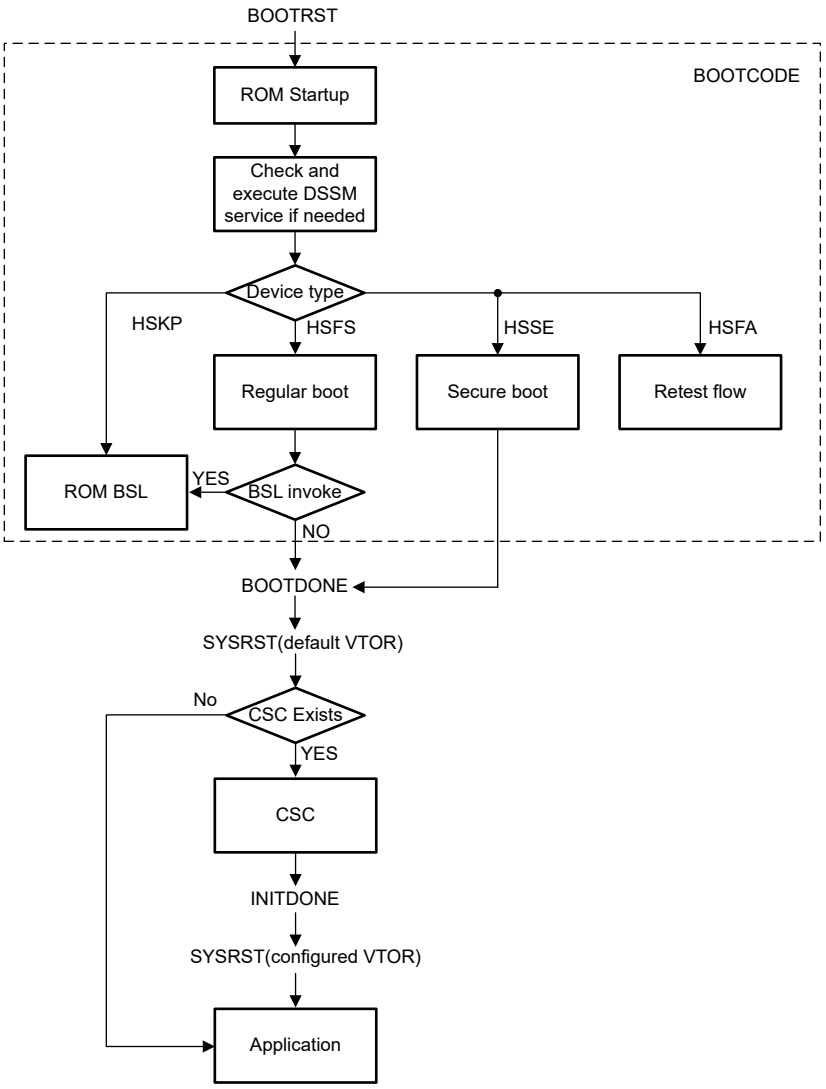


图 2-1. 引导和启动序列

2.2 CSC 流程

本节介绍了基于 MSPM33 SDK CSC 示例 (MSPM33 SDK 1.04.00.00) 的 CSC 设计中的详细引导流程，如图 2-2 所示。整个执行流程与图 2-1 中所示的流程图基本兼容。

完成 ROM 引导代码执行后，程序进入 CSC 固件，并且 INITDONE (SYSCTL.SECCFG.SECSTATUS.INITDONE) 处于清除状态。CSC 从两个闪存存储体中搜索最高版本的映像，检查版本回滚，然后通过非对称方法（SHA256+ECDSA_p256 和 MLDSA87）验证应用程序映像的权限和完整性。验证通过后，CSC 会更新回滚计数器、SECRET 密钥和 KEYSTORE。然后，CSC 在 SECRET 闪存区域和可锁定闪存区域中配置防火墙，并确定存储体交换策略。CSC 还会配置 GSC 中的 VTOR，以设置应用程序映像的矢量表偏移。CSC 发出 INITDONE 以触发 SYSRST，随后器件跳转至应用程序映像以启动应用程序。

下面是一些与 CSC 示例执行流程相关的重要说明：

- 需要启用 NONMAIN 闪存中的 CSCEXISTS 和 FLASHBANKSWAPPOLICY 字段，才能启用整个 CSC 序列。请参阅 *MSPM33 NONMAIN 闪存存储器配置指南* 了解配置。
- PB0 表示物理 Bank0。由于存储体交换策略在 INITDONE 置位之前不会生效，因此在 CSC 中使用的闪存地址始终指的是物理地址。
- 如果 PB0 和 PB1 中的两个映像的版本相同，则会以更高的优先级验证并执行 PB0 映像。
- 如果最高版本的映像未通过 SHA256+ECDSA 或 MLDSA 验证，则会立即验证另一个存储体（如果存在）中的映像。

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

- 在非对称认证的情况下，首先计算应用程序代码的安全散列 (SHA2-256) 摘要，然后 ECDSA 和 MLDSA 根据固件中的公钥验证映像签名。
- 对于 M33 器件，SYSRST 不会清除 GSC 中的 VTOR。当 CSC 发出 INITDONE 时，将一起触发 SYSRST。配置 VTOR 后，器件将从应用程序启动。

SECRET 闪存区域是用户指定的一个区域，用于存储机密信息，并且由防火墙进行读取-执行保护。可锁定闪存区域是用户指定的区域，用于存储未修改的信息，并由防火墙进行写保护。如需了解更多详情，请参阅 FLASH 存储器映射。

Copyright © 2026 Texas Instruments Incorporated

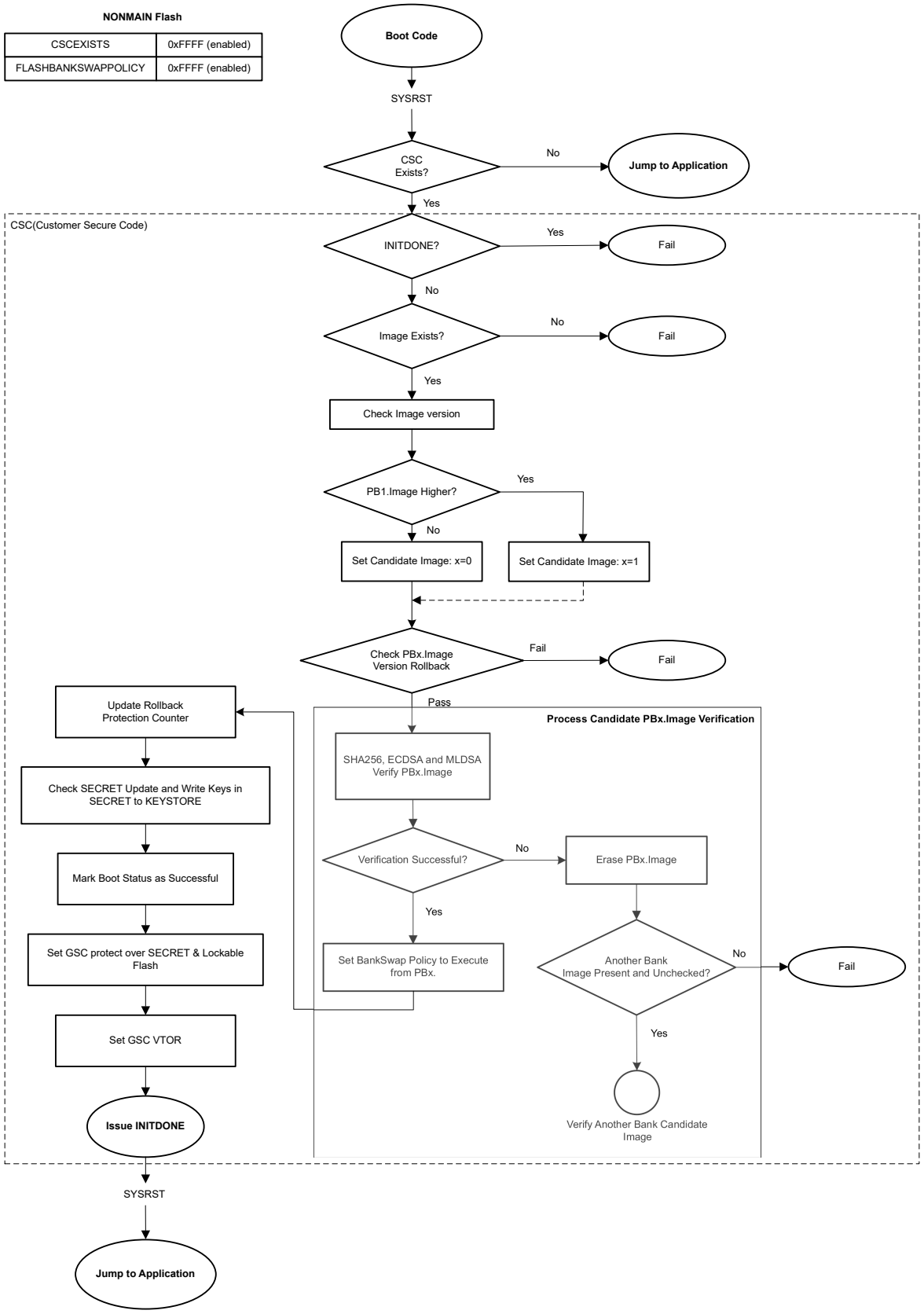


图 2-2. MSPM33 SDK CSC 执行流程

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

2.3 FLASH 存储器映射

图 2-3 展示了 CSC 安全引导中的详细闪存存储器映射。以下是对 CSC 中各个部分的说明：

- SECRET：该机密对 CSC 可见，但受读取保护防火墙保护，从而使应用程序看不到该机密。SECRET 区域可存储应在运行时加载到 KEYSTORE 中的非易失性密钥。因此，应用程序可以在无读取访问权限的情况下使用这些密钥。自定义机密以包含标签及密钥等其他信息。
- Lockable Flash：可锁定的闪存针对需要由 CSC 写入并由应用程序读取但未修改的关键信息提供动态写保护。安全计数器（回滚保护）、密钥库哈希表和映像哈希通常存储在可锁定的闪存中。请注意，可锁定的内容仅在 PB0 中编程到 CSC 区域，以确保两个存储体应用程序都可以访问该区域。
- CSC Interrupt Vectors：这些是客户安全代码的中断向量。在发生 BOOTRST 或带有默认 VTOR 的 SYSRST 时，该中断向量表是从闪存中首先运行的内容。在 CSC 中，VTOR 用应用程序中断矢量的地址进行更新，然后采用配置 VROT 的 SYSRST 会引导器件从应用程序启动。

  备注

  SYSRST 不会复位 VTOR，这意味着在执行 CSC 后，器件在下一个 BOOTRST 之前不会再次进入 CSC。
- CSC Code：主代码和安全基元是客户安全代码的大量内容。通常，CSC 代码和中断向量只需要编程到物理存储体 0，因为 CSC 在存储体交换生效之前执行。在存储体交换期间，由 INITDONE 触发 SYSRST 之后，程序始终从逻辑存储体 0（逻辑地址）中的应用程序开始运行。FLASHCTL 会根据存储体交换策略配置，将该地址映射到 PB0 地址或 PB1 地址。

下面是应用程序映像的各个部分：

- 映像头、映像 TLV 和映像尾部：由签名工具 imgtool 生成，该工具由 MCUBOOT 提供（位于 <mspm33_sdk_path>\source\third_party\mcuboot\scripts 目录下的 Python 脚本中）。这些部分将在 CCS 构建后处理步骤中生成并合并至编译后的应用程序映像中。MSPM33 SDK 中有一个 customer_secure_sample_image 示例，展示了如何在 CCS 中构建已签名的映像。以下是这些图像部分的说明：
  - Image Header：应用程序映像的标头信息，包括标头魔数 (0x96F3B83D)、映像尺寸和映像版本。映像头位于应用程序中断向量表之前 0x200 字节（默认值）的地址处。
  - Image TLV：MCUBOOT 定义了包含映像元数据（放置在映像末尾之后）的类型长度值记录 (TLV)。MSPM33 CSC 中定义的 TLV 包括：TLV 魔数 (0x6907)、映像哈希、ECDSA 公钥哈希、ECDSA 签名、MLDSA 公钥哈希和 MLDSA 签名。如需了解更多详情，请参阅 main · mcu-tools/mcuboot · GitHub 中的 mcuboot/docs/design.md。

    备注

    SHA256 仅针对图像标头、应用程序中断向量和应用程序映像等内容执行。
  - Image Trailer：一个位于图像闪存区域末尾的 16 字节魔数内容。
- Application Interrupt Vectors：这是应用程序使用的单独中断向量。在 CSC 跳转到应用程序期间，向量表偏移寄存器 (VTOR) 指向存储器中的该位置，因此未来所有中断都会发生，并且没有指向这组中断向量的复位（BOOTRST 或更高级别的复位）链接。起始地址是 32 字节对齐的地址。

  备注

  对于 ARM Cortex-M33，要求 0x200 的 VTOR 地址对齐。如需了解更多信息，请参阅 E2E 帖子。
- 应用程序代码：原始应用程序代码。

Copyright © 2026 Texas Instruments Incorporated

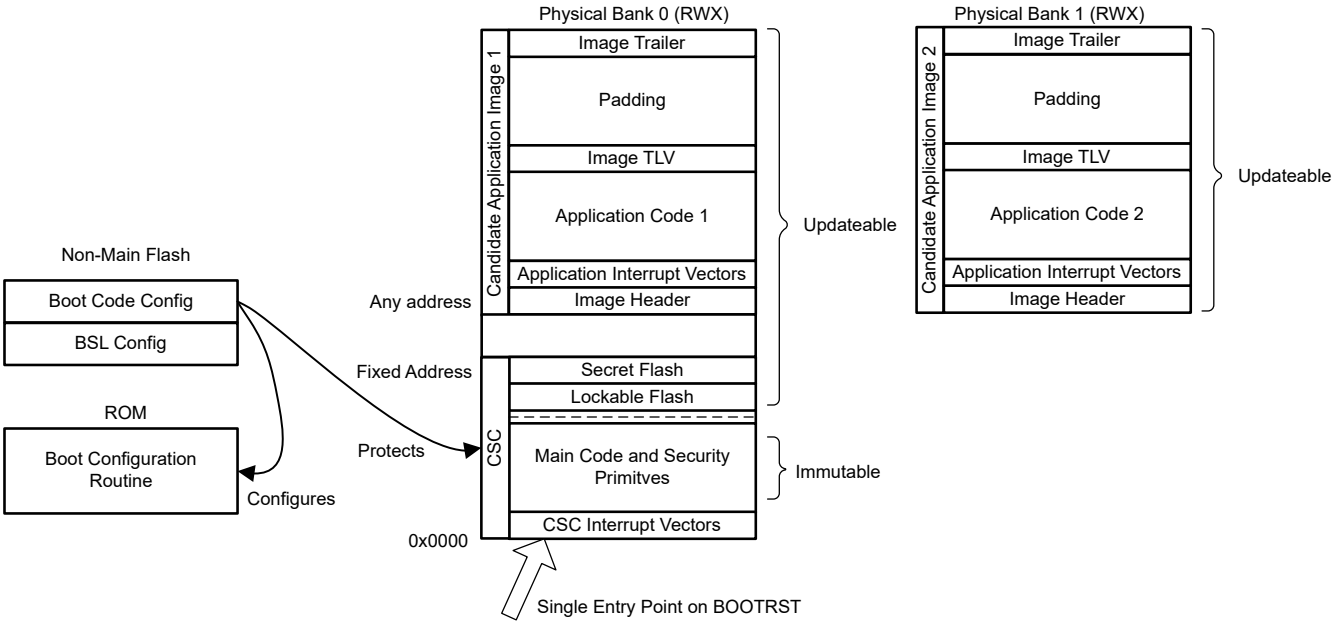


图 2-3. CSC 闪存映射

2.4 存储体交换

PB = 物理存储体。有一个物理存储体 0 (PB0) 和物理存储体 1 (PB1)。这些存储体可以根据存储体交换配置切换地址。

LB = 逻辑存储体。这是映射到物理存储体 0 或物理存储体 1 的两个逻辑存储体。在 INITDONE 之前，逻辑存储体 0 始终映射到地址 PB0，而逻辑存储体 1 映射到 PB1。但是，根据存储体交换的执行情况，在 INITDONE 之后，LB0 可以映射到 PB0 或 PB1。在 INITDONE 之后，逻辑存储体 0 是执行存储体，而逻辑存储体 1 是数据存储体。

备注

由于代码仅从逻辑存储体 0 执行，因此在编译所有代码时，请确保代码是从逻辑存储体 0 运行。因此，如果 CSC 映射在 0x0000 至 0x47FF，而一个应用程序映像从 0x4800 开始映射，则请以 0x4900 为基准来编译每个应用程序映像（即 0x100 的尾部，然后应用程序中断向量放置在 0x4800 + 0x100 = 0x4900 处），即使该应用程序映像被写入了实际物理地址上的逻辑存储体 1。

在存储体交换方法中，实际固件不会在物理位置之间移动，而是配置硬件从适当的存储体执行。

CSC 中的存储体交换机制是通过 DL_SYSCTL_executeFromUpperFlashBank() 和 DL_SYSCTL_executeFromLowerFlashBank() 函数实现，系统会根据需要调用这些函数。

备注

当用户需要配置安全属性单元 (SAU) 以进行安全属性划分时，所配置的地址是基于逻辑地址。在可交换存储体的配置中，请根据逻辑地址仔细配置 SAU。

备注

当用户需要配置全局安全控制器 (GSC) 以进行安全权限归属、FLASH 隐藏保护、静态写保护或其他操作时，GSC 中的相关寄存器基于物理存储体。在可交换存储体的配置中，请根据物理地址仔细配置 GSC。

2.5 CSC 特性

2.5.1 非对称验证

通过加密算法来验证新映像的完整性及真实性。只有已验证的映像才被视为安全映像并且可以执行。

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

2.5.1.1 SHA-256

安全哈希算法 256 (SHA-256) 是一种广泛使用的加密哈希函数，用于从任何输入消息生成固定长度的 256 位摘要。该算法对输入变化很敏感，这意味着 SHA-256 的设计目的是确保即使输入消息的微小变化也会产生不同的输出哈希值。

其中一个核心功能是防冲突，这意味着算法极不可能为两个不同的消息生成相同的哈希值。此属性使 SHA-256 在验证数据完整性方面非常可靠，因为可以轻松检测到任何修改。

实际上，SHA-256 通常用于数字签名和数据完整性检查。通过将整个映像冷凝为独特的摘要，SHA-256 为下一个 ECDSA 算法奠定了坚实基础。

2.5.1.2 ECDSA

椭圆曲线数字签名算法 (ECDSA) 是一种基于椭圆曲线数学运算的数字签名加密算法。与传统算法（如 RSA）相比，ECDSA 在提供高安全性的同时，密钥长度要短得多，这使得该算法非常高效，并专为资源受限的环境而设计。

ECDSA 是 MCUboot 支持的一种认证方法。ECDSA 是非对称算法，意味着有独立的公钥和私钥。公钥将存储在器件闪存中，私钥由开发人员安全保存。CSC 不提供安全私钥管理。

ECDSA 使用映像的哈希值和公钥来验证数字签名 (r, s)，从而检查数据的真实性。与对称加密选项相比，该选项不会在器件上造成关键漏洞。

备注

为了保持最终部署的引导流程，请确保私钥安全且受到管理，以便不能轻松地对映像进行签名。将密钥保留在本地共享驱动器上时，并非属于安全的位置！TI 当前没有提供安全私钥管理。

2.5.1.3 MLDSA

基于模格的数字签名算法 (ML-DSA) 是一种后量子密码算法，旨在抵御量子计算带来的威胁。ML-DSA 基于 PQC 算法家族中的代数格密码套件 (CRYSTALS)，对选择消息攻击尤为有效。

MLDSA 不是 MCUboot 支持的认证方法。实施了一些代码修改以集成 MLDSA。

- 定义 TLV 类型 -
  - #define IMAGE_TLV_MLDSA87 0xCE /* 用于 MLDSA87 签名的自定义 TLV */
- 密钥管理 — 在混合方法中，我们需要多种类型的算法。bootutil_keys 数组需要包含另一个密钥
- 修改映像验证代码 — 更新 bootutil_img_validate 函数以使用 MLDSA 验证函数
- 创建 MLDSA 验证函数 — 创建一个实现 MLDSA 验证功能的新文件（例如 image_mldsa87.c）

MLDSA 是非对称算法，意味着有独立的公钥和私钥。公钥存储在器件闪存中，私钥由开发人员安全保存。CSC 不提供安全私钥管理。

2.5.2 密钥库

KEYSTORE 是一个受保护的 SRAM 存储器，可以安全地存储 AES 密钥，在 INITDONE 之前在 CSC 中配置密钥，并且应用程序可以触发从 KEYSTORE 到 AES 引擎的密钥传输，但不会在 INITDONE 之后直接访问（读取或写入）这些密钥。

只有 CSC 可在 INITDONE 之前将密钥配置为 KEYSTORE。随后，主应用程序可以配置加密引擎来使用其中一个存储的密钥，但其本身始终无法访问（读或写）任何存储的密钥。密钥从 KEYSTORE 到加密引擎的传输被安全地执行，对应用程序代码不可见。

KEYSTORE 内容将在每次 BOOTRST 时被擦除，并且存储将解锁以进行写入。KEYSTORE 内容不受 SYSRST 和低等级复位的影响。

Copyright © 2026 Texas Instruments Incorporated

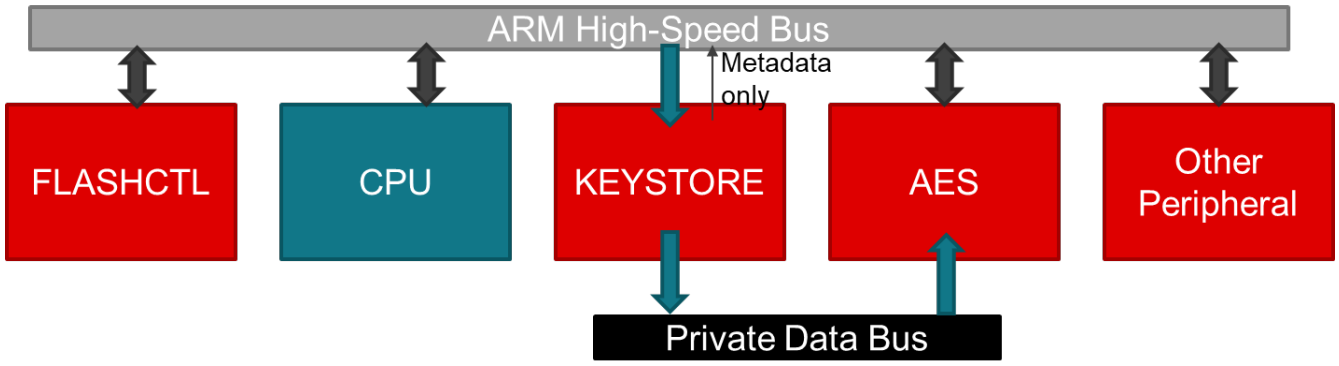


图 2-4. KEYSTORE 工作过程

2.6 CSC 上的保护

MSPM33 在 FLASH 和 SRAM 上提供多种类型的存储器保护机制，用于对 CSC 施加保护。

2.6.1 写擦除保护

MSPM33 MCU 实现了一个静态写保护方案，以将闪存区域中用户定义的扇区锁定，从而防止在运行时针对相应扇区执行任何编程或擦除操作。所需的静态写保护方案在全局安全控制器 (GSC) 中配置，用户可以对 MAIN 闪存、DATA 闪存或 NONMAIN 闪存应用写保护。请参阅 *MSPM33 C3 系列 160MHz 微控制器* 技术参考手册中的 *GSC* 一章。

静态写保护功能支持在闪存存储器中放置一个用户定义的固定区域，一旦该区域经过编程和锁定，该区域就不可由应用程序代码或 ROM 引导加载程序进行修改。

- 如果置于 MAIN 闪存存储器的开头，该区域中的应用程序是 ROM 引导配置例程将执行转移到用户应用程序时执行的第一个代码（这是实施安全启动映像管理器的一项要求）。
- 如果放置在 NONMAIN 闪存中，NONMAIN 闪存中的配置内存随后将被锁定，这可以防止引导加载程序或应用程序代码对安全策略、引导加载程序策略以及静态写保护策略进行任何未经授权的修改。

当引导配置例程将转换到执行 MAIN 闪存中的引导加载程序或用户应用程序代码时，在 GSC 中配置为写入锁定的任何区域都在功能上不可更改。如果应用程序代码或引导加载程序尝试对受静态保护的区域进行任何编程或擦除，都会导致硬件闪存操作错误，并且扇区不会被修改。

备注

用于写保护的 GSC 寄存器的访问类型为“R/W1S”。“W1S”是指该寄存器位只能从 0 置位为 1，且该寄存器只能通过 BOOTRST 或更高级别的复位类型才能被复位，这意味着已施加的写保护无法被直接移除。

MSPM33 还支持在 NONMAIN 闪存中配置写保护策略。在 ROM 引导期间，引导模式根据 NONMAIN 闪存配置 GSC。该配置在进入应用程序代码之前生效，这意味着除非再次修改 NONMAIN 闪存，否则应用程序代码无法移除写保护。请参阅 *MSPM33 NONMAIN 闪存存储器配置指南* 了解配置。

静态写保护可防止应用程序代码或引导加载程序进行任何修改，但通过 SWD 接口发送的批量擦除或恢复出厂设置命令将被接受。如果不需要这种行为，可以使用唯一的密码来保护批量擦除或恢复出厂设置 SWD 命令，也可以禁用这两个命令。要完全消除任何修改受静态写保护的 MAIN 闪存扇区的方法，请按照以下说明操作。

- 在 NOMAIN 中将写保护设置为 MAIN 闪存。
- 禁用 NONMAIN 中的批量擦除和恢复出厂设置命令（或 SW-DP）。
- 在 NONMAIN 中为 NONMAIN 闪存设置写保护，以防止应用程序代码通过修改 NONMAIN 区域的内容来更改底层的写保护方案。

如需了解上述说明，请参阅 *MSPM33 NONMAIN 闪存配置指南*。

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

备注

默认情况下，NONMAIN 配置存储器（包含用户指定的启动安全策略和引导加载程序策略）不受写保护。用户可以在配置期间擦除 NONMAIN，并使用可用于大规模生产的用户指定策略重新编程。

2.6.2 FLASH 隐藏保护

MSPM33 支持隐藏保护 (HDP) 功能，允许应用程序执行一段代码一次，并防止随后对该代码进行读取或执行访问。例如，当运行第二阶段认证代码时，这是一项非常有用的功能，它允许在执行前验证主应用程序，并防止访问任何已存储的对称密钥或私钥。所需的 HDP 方案在全局安全控制器 (GSC) 中配置，用户可以对 MAIN 闪存、DATA 闪存或 NONMAIN 闪存应用 HDP。请参阅 *MSPM33 C3 系列 160MHz 微控制器* 技术参考手册中的 *GSC* 一章。

正确配置并启用 HDP 后，当 CPU 从配置的 HDP 区域执行时，HDP 处于准备状态。任何超出配置的 HDP 区域之外的函数调用或跳转都会自动启用该机制，从而阻止应用程序代码重新进入 HDP 区域。访问、执行或读取会生成不可屏蔽中断 (NMI)。

- 如果在主 Flash 存储体上启用了 HDP，则读取、写入、执行和擦除操作都将被阻止。
- 如果在非主 Flash 存储体上启用了 HDP，则会阻止读取和执行操作。为了启用写入和擦除保护，需要额外配置写擦除保护。

备注

HDP 的 GSC 寄存器的访问类型为“R/W1S”或“WOnce”。“W1S”是指寄存器位只能设置为 0 至 1。“WOnce”是指寄存器位在复位前只能设置一次。这些寄存器只能通过 BOOTRST 或更高级别的复位类型进行复位，这意味着不能直接删除应用的 HDP。

备注

当在 HDP 区域执行代码随后转移到 SRAM 时，SRAM 中的代码可以访问 HDP 区域。一旦 SRAM 中的所有代码执行完毕，程序必须转移返回并退出 HDP 区域，继续在闪存中执行，才能启用 HDP 机制。如果应用程序不希望 SRAM 访问 HDP 区域，则代码必须实现一个跳板函数，以便在转移至 SRAM 代码之前启用 HDP。

2.6.3 数据完整性验证

BCR 支持在将执行从 BCR（位于 ROM 中）转移到用户应用程序（位于 MAIN 闪存中）之前，检查 MAIN 闪存中用户指定地址范围的数据完整性。请参阅 *MSPM33 NONMAIN 闪存存储器配置指南* 了解配置。

完整性检查可用作额外的步骤，以确保在引导 ROM（通常是安全启动映像管理器）之后首先运行的代码具有与预期值匹配的 CRC/SHA256 摘要。此完整性检查降低了闪存中关键代码（可能负责验证其余用户应用软件映像）的任何意外损坏可能会造成安全漏洞的可能性。

可以将起始地址、长度和 ISO-3309 CRC-32 或 SHA2-256 摘要配置到 NONMAIN 配置内存中。在引导过程中，BCR 将计算 MAIN 闪存存储器中指定范围的 CRC-32 摘要，并根据配置的（预期）摘要来验证计算的摘要。如果这些值匹配，则会启动用户应用程序。如果这些值不匹配，则不会启动用户应用程序，结果会导致灾难性的引导错误。

2.6.4 安全和特权保护

部分 MSPM33 器件支持基于 TrustZone 架构的安全防护，以及基于 PMU 的特权保护。请参阅数据表以确认支持状态。

通常，CSC 被设置为安全和特权状态。从非安全或非特权区域进行的任何访问都会导致安全/特权违例。若要在 MSPM33 上配置安全或特权保护，请参阅*在 MSPM33 MCU 中为安全应用启用 TrustZone*。

2.7 CSC 性能

MSPM33 CSC 代码大小与编译器及优化级别相关。默认情况下，CSC 代码大小信息如下所列：

- CSC 区域大小：30KB
- CSC 主代码大小：28KB

Copyright © 2026 Texas Instruments Incorporated

- 机密大小：1KB
- 锁定存储大小：1KB

可以自定义 CSC 示例以修改主代码和 SECRET 或者 Lock Storage Size。如果需要更改 CSC 区域大小，还需要更改应用程序固件的起始地址。有关详细信息，请参阅节 4。

以下是供参考的 CSC 时序性能。测试应用程序映像约为 25.9KB。

表 2-1. CSC 时序性能

ECDSA 验证	MLDSA 验证	SHA256
160MHz 时为 130.3ms	160MHz 时为 104.9ms	160MHz 时的 361.5us

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

3 评估基于 CSC 的安全引导

3.1 环境设置

本文档遵循以下依赖项。

1. 软件和工具：
   a. MSPM33 SDK: mspm33_sdk_1_04_00_01
   b. CCS 20.5
   c. 用于 MCUBoot Image Tool (imgtool.py) 的 Python 3.11 或更高版本
2. 硬件：
   a. MSPM33C321A 样片

在 *MSPM33 SDK* 中，提供了以下两个工程。

1. CSC 示例 - customer_secure_code
   a. CSC 示例提供了使用 ML-DSA-87 和/或 ECDSA-256 验证应用程序映像的功能。
   b. 此示例位于文件夹 *./examples/nortos/LP_MSPM33C321A/boot_manager/ customer_secure_code* 中。
2. 应用程序示例 - customer_secure_code_sample_image
   a. 此示例作为 CSC 示例的配套提供，以演示 CSC 的使用
   b. 此示例位于文件夹 *./examples/nortos/LP_MSPM33C321A/boot_manager/ customer_secure_code_sample_image* 中

TI 建议将这两个工程导入 CCS。对于 CSC 示例用法，用户还可以参阅 SDK 中的 *MSPM33 客户安全代码和引导加载程序 (CSC) 用户指南*。

要运行初始设置，请确保使用最新的 pip 包安装 Python 3.11 或更新版本，并运行下面的命令下载必要的要求。

1. 打开命令行窗口，并运行以下注释以检查您的环境中是否安装了 Python：

```
python --version
```

2. 进入 MSPM33 SDK 安装路径（例如 C:\ti\mspm33_sdk_1_03_00_01\），并在命令行中运行以下命令以满足必要要求：

```
python -m pip install --user -r source/third_party/mcuboot/scripts/requirements.txt
```

3. 位于 /source/third_party/mcuboot/scripts 下的 Python 脚本在对应用程序映像签名时应用 Python 库。

3.2 对 CSC 进行编程

3.2.1 通用

在构建 CSC 并将其刷写到器件上时，还必须配置非主闪存（配置 BootCFG2 以启用 CSC）。CSC 示例包含修改后的非主 FLASH 默认值。

以下是非主闪存的 cmd 文件。

```
MEMORY
{
    FLASH          (RX)  : origin = 0x10000000, length = FLASH_SIZE, fill = 0xFFFFFFFF,
    SECRET         (RX)  : origin = end(FLASH), length = CSC_SECRET_SIZE, fill = 0xFFFFFFFF,
    LOCK_STORAGE   (RX)  : origin = end(SECRET), length = CSC_LOCK_STORAGE_SIZE, fill = 0xFFFFFFFF,
    BCR_CONFIG     (R)   : origin = 0x80101800, length = 0x00000400,
    BSL_CONFIG     (R)   : origin = 0x80101C00, length = 0x00000400,
    SRAM           (RWX) : origin = 0x30000000, length = 0x00040000,

}
```


Copyright © 2026 Texas Instruments Incorporated

```
SECTIONS
{
    .intvecs:  > 0x10000000
    .text    : palign(16) {} > FLASH
    .const   : palign(16) {} > FLASH
    .cinit   : palign(16) {} > FLASH
    .pinit   : palign(16) {} > FLASH
    .rodata  : palign(16) {} > FLASH

    .ARM.exidx    :  palign(16)  {} > FLASH
    .init_array   :  palign(16)  {} > FLASH
    .binit        : palign(16) {} > FLASH
    .TI.ramfunc   : load = FLASH, palign(16), run=SRAM, table(BINIT)

    .secret  : palign(16) {} > SECRET
    .lockStg : (NOINIT) palign(16) {} > LOCK_STORAGE

    .vtable :   > SRAM
    .data   :   > SRAM
    .bss    :   > SRAM
    .sysmem :   > SRAM
    .stack  :   > SRAM (HIGH)

    .BCRConfig  : {} > BCR_CONFIG
    .BSLConfig  : {} > BSL_CONFIG

}
```

图 3-1. 非主 FLASH 的内存分配

SysConfig 在非主 FLASH 上具有默认配置。构建后，SysConfig 将为非主闪存生成 boot_config.c/h：

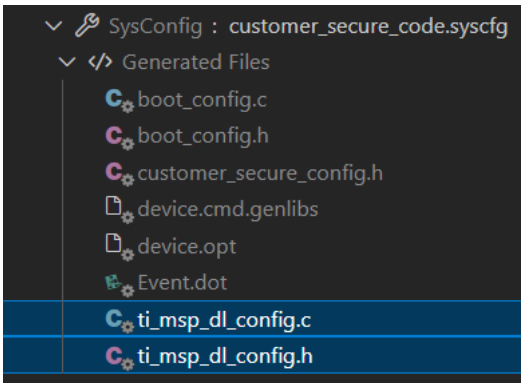


图 3-2. 由 SysConfig 生成的 boot_config.c/h

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

CSC 旨在使用 *customer_secure_code_LP_MSPM33C321A_nortos_ticlang\boot_keys.c* 中存在的示例密钥来验证应用程序。这些密钥是用于验证的公钥（相应的私钥已用于对应用程序映像签名）。

3.2.2 第 1 步 - 构建

若要生成相应的 CSC 二进制文件，请导入 CSC 示例：<MSPM33_SDK>\examples\nortos\LP_MSPM33C321A\boot_manager\customer_secure_code。

现在，构建工程，并在调试文件夹中找到生成的二进制文件。应将 *customer_secure_code_LP_MSPM33C321A_nortos_ticlang.out* 刷写到器件上。此二进制文件具有 ECDSA-256 公钥、ML-DSA87 公钥、CSC 应用程序和非主配置。

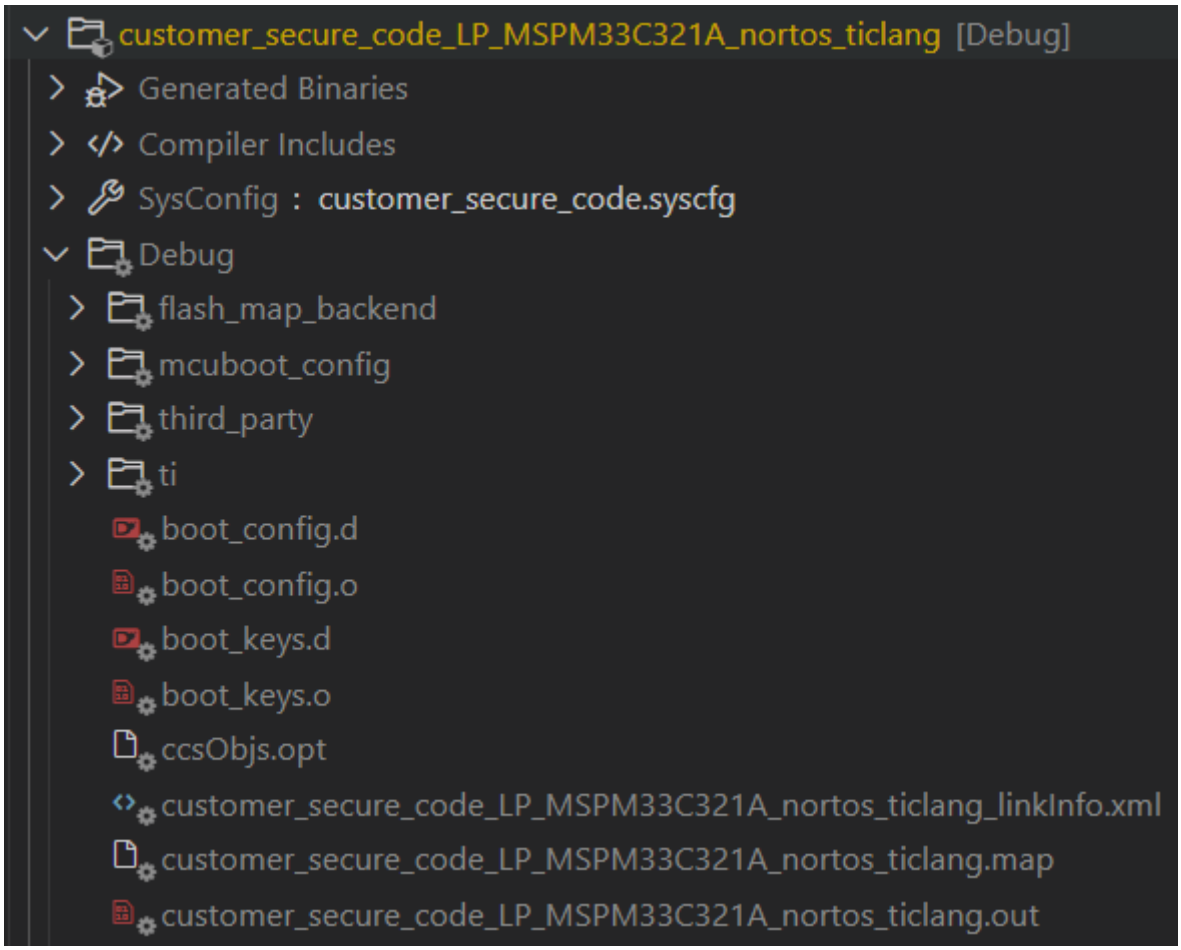


图 3-3. 构建工程以生成二进制文件

3.2.3 第 2 步 - 编程

要在 CCS 中加载映像，请执行以下步骤：

1. 使用 CSC 工程调试文件夹中的 CCXML 文件启动无工程调试，以连接到器件。右键单击 CCXML 文件，然后单击“无工程调试”。
2. 在调试程序设置中，将调试程序配置为**“擦除 MAIN 和 NONMAIN 内存”**。依次进入“运行”->“调试程序属性”->“MSPM33 闪存设置”。
3. 将擦除配置设置为“擦除 MAIN 和 NONMAIN 内存”，如下图所示

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

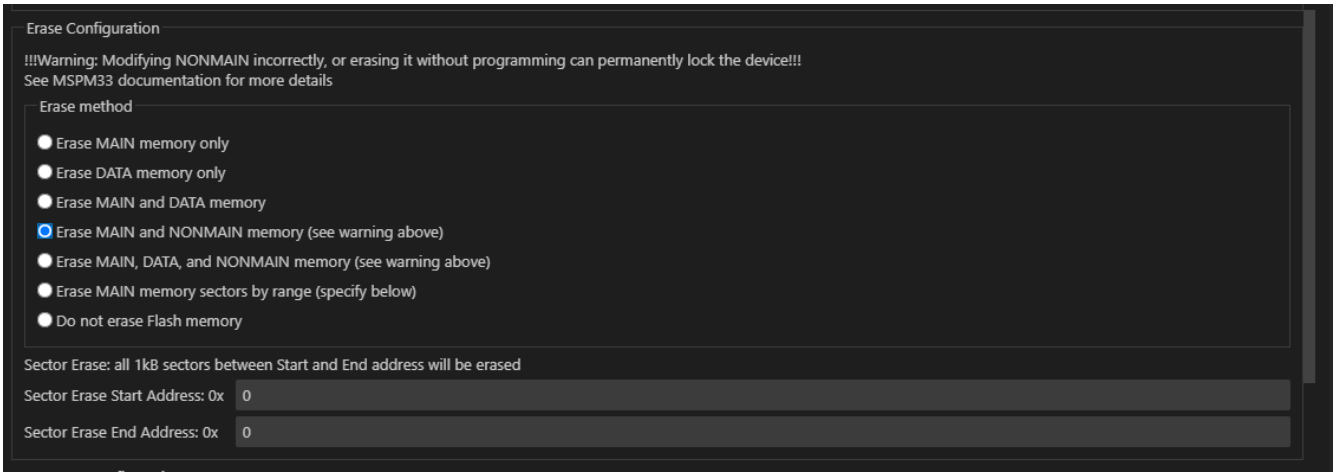


图 3-4. 配置“擦除 MAIN 和 NONMAIN 内存”

4. 加载 CSC 二进制文件。要执行此操作，请选择“运行”->“加载”->“加载程序”。然后选择生成的输出文件。

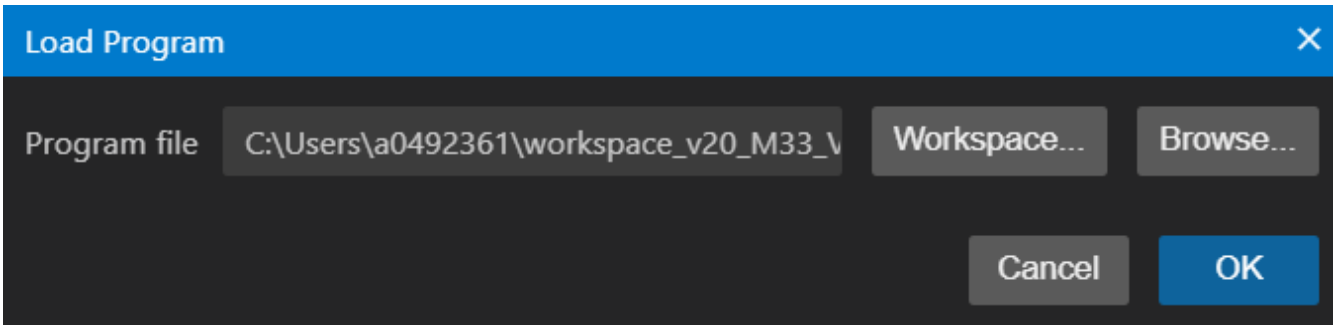


图 3-5. 使用生成的 CSC 二进制文件加载程序

5. 要确认非主配置已按要求执行，请使用内存查看器（位于“视图”->“内存”中），查看地址 0x400B2048(SYSCTL_SECSTATUS) 处的值。如果最高有效位为 4，则表示 CSC 已正确加载。按下 NRST 按钮并重新连接到内核。请参考下图中的示例

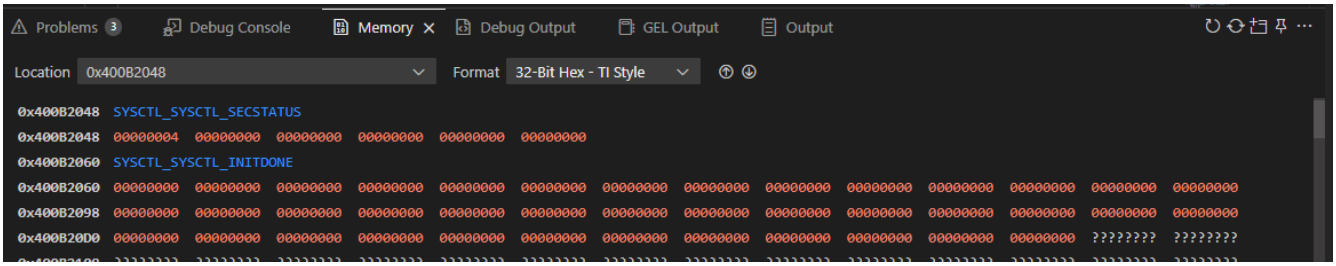


图 3-6. 使用生成的 CSC 二进制文件加载程序

备注

完成上述步骤后，请将擦除配置改回仅“擦除 MAIN”。错误地写入非主闪存可能会使器件不可恢复。

3.3 编程应用程序

3.3.1 通用

若要创建可由 CSC 验证的应用程序映像，程序员需要对应用程序映像签名，并在应用程序代码中放入相应的哈希值。这是使用 MCUBOOT 提供的 Image Tool 计算得出的，该工具位于 SDK 的 *<MSPM33_SDK>\source\third_party\mcuboot\scripts* 目录下。

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

本文档利用 MCUBoot 中的 Image Tool (*imgtool.py)* 对二进制文件签名。Image Tool 位于 MSPM33 SDK 安装中的 *./source/third_party/mcuboot/* 文件夹中。以下是使用 Key0.pem（私钥）对二进制文件进行签名的参考命令。

```
python scripts/imgtool.py sign --pad-header -k key0.pem -v 1.0.0 -H 0x70 -S 0x100000 --align 16 -F
0x10000080 <output_image.bin> <input_image.bin>
```

备注

1. 本文档中的说明基于在 Windows 系统上使用 Python。如果使用的是 Linux 系统，请将调用的“python”替换为“python3”。
2. Image Tool 不支持 .out 文件签名。将 .out 文件转换为 .bin 文件。后续章节将介绍这一点。
3. 修改 CCS 工程设置，以将 Image tool 和 .bin 文件生成集成到构建后处理步骤中，从而自动生成已签名的二进制文件。

有关 Image Tool 的命令列表，请参阅 *Imgtool 自述文件*。

3.3.2 步骤 1 - 应用程序映像生成 – 映像 1（版本 1.0.0，位置为存储体 0 – 0x10000）

要生成应用程序映像，您可以使用任何工程，但在本文档中，我们重点介绍 CSC 示例应用程序。我们创建了两个单独版本的二进制文件，并将两者都刷写到器件上。每个二进制文件都有相应的版本号，并且 CSC 会引导最新版本映像。这用于突出器件的更新功能和版本区分功能。

1. 首先，导入以下位置的 CSC_sample：*<MSPM33_SDK>\examples\nortos\LP_MSPM33C321A\boot_manager\customer_secure_sample_image*。对于第一个应用程序，通过将闪存的起始位置设置为 0x10200 来构建工程。在 .cmd 文件中找到位置：

```
51  MEMORY
52  {
53    FLASH        (RX)  : origin = 0x00010200, length = FLASH_SIZE, fill = 0xFF,
54    SRAM           (RWX) : origin = 0x30000000, length = 0x0000FFFF
55
56  }
57
58  SECTIONS
59  {
60      .intvecs:  > 0x00010200
61      .text    : palign(16) {} > FLASH
62      .const   : palign(16) {} > FLASH
63      .cinit   : palign(16) {} > FLASH
64      .pinit   : palign(16) {} > FLASH
65      .rodata  : palign(16) {} > FLASH
```

图 3-7. 应用程序映像 1 的闪存的起始位置

2. 将这些构建后步骤添加到 Properties-Build-Steps：

```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/
input1.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/
requirements.txt
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --
key ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --
pad-header -v 1.0.0 --slot-size 0x3000 --load-addr 0x10000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input1.bin ${PROJECT_BUILD_DIR}/dual_signed_output1.bin
```


Copyright © 2026 Texas Instruments Incorporated

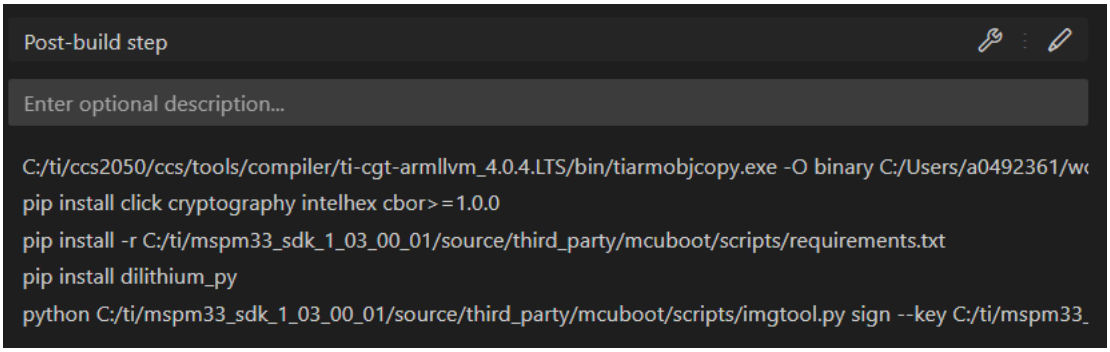


图 3-8. 属性中的构建后步骤

3. 构建工程，您将在工程调试文件夹中看到“dual_signed_out1.bin”文件。

3.3.3 步骤 2 - 应用程序映像生成 – 映像 2（版本 2.0.0，位置为 0x20000）

1. 第二个应用程序具有存储体交换功能，使用与第一个应用程序相同的链接器文件，使得应用程序的起始位置为 0x10200。
   a. 此处无需修改，因为第 1 个应用程序和第 2 个应用程序的逻辑起始地址相同。

```
51  MEMORY
52  {
53    FLASH         (RX)  : origin = 0x00010200, length = FLASH_SIZE, fill = 0xFF,
54    SRAM            (RWX) : origin = 0x30000000, length = 0x0000FFFF
55
56  }
57
58  SECTIONS
59  {
60      .intvecs:  > 0x00010200
61      .text    : palign(16) {} > FLASH
62      .const   : palign(16) {} > FLASH
63      .cinit   : palign(16) {} > FLASH
64      .pinit   : palign(16) {} > FLASH
65      .rodata  : palign(16) {} > FLASH
```

图 3-9. 应用程序映像 2 的闪存的起始位置

2. 更改构建后步骤以包括以下命令：

```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/
input2.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/
requirements.txt
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --
key ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --
pad-header -v 2.0.0 --slot-size 0x3000 --load-addr 0x10000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input2.bin ${PROJECT_BUILD_DIR}/dual_signed_output2.bin
```


English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

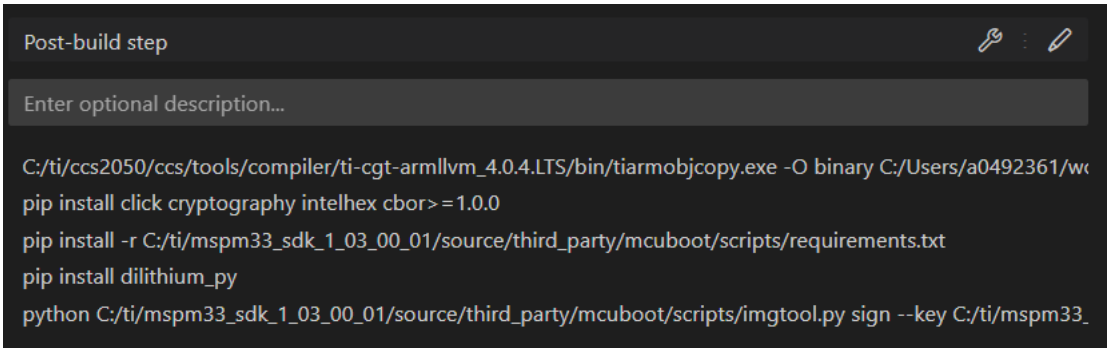


图 3-10. 属性中的构建后步骤

3. 现在，再次构建工程。这将生成“dual_signed_output2.bin”。图 3-11 展示了构建两个二进制文件后调试文件夹的外观。

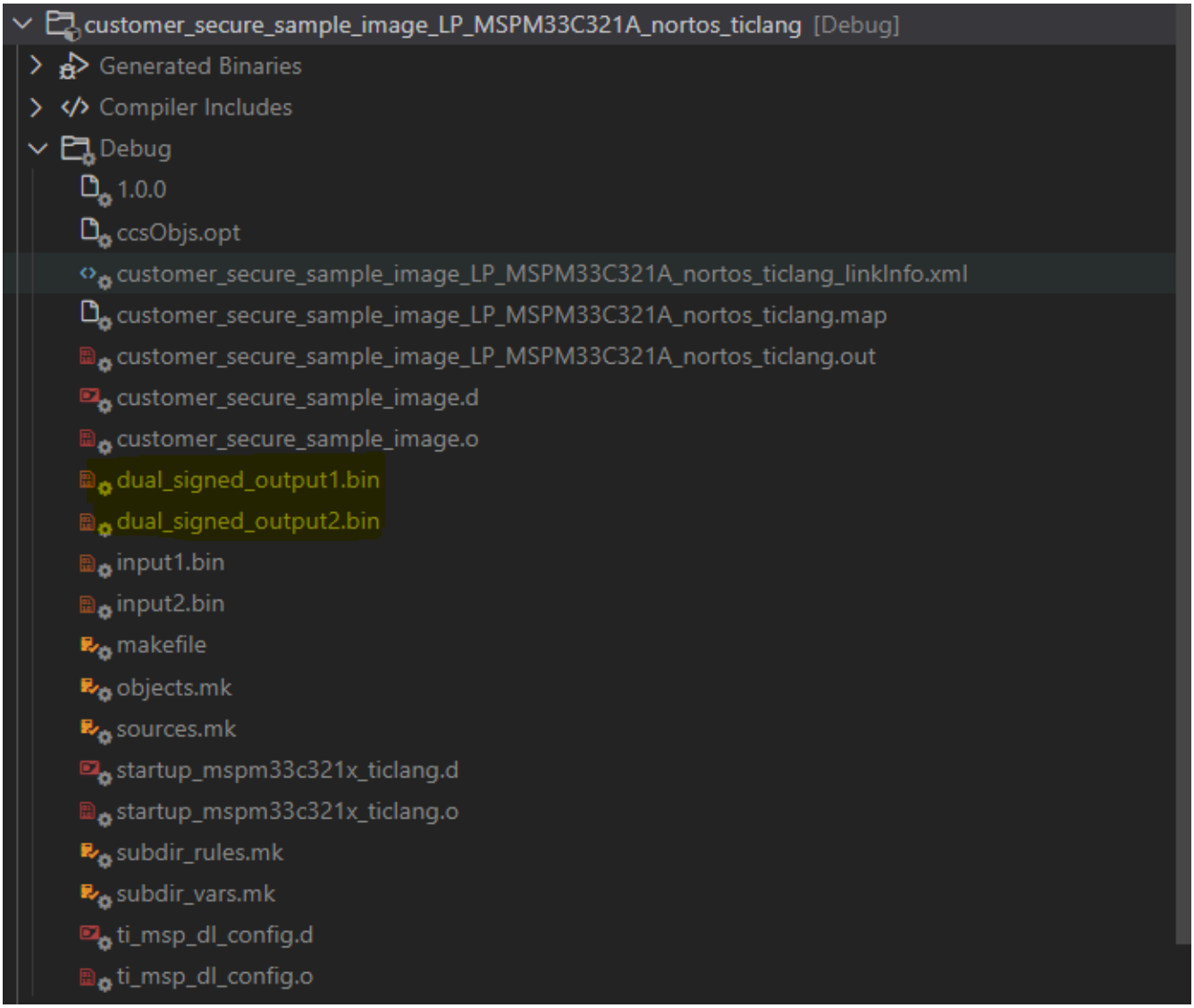


图 3-11. 工程中的调试文件夹

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

3.3.4 第 3 步 - 编程

具有不同版本的两个应用程序映像都采用相同的地址分配（位置 0x10000）构建。在 0x10000 (bank0) 处刷写一个应用程序映像。在 0x90000 (bank1) 处刷写第二个应用程序映像。

1. 在刷写 CSC 映像之前，请将擦除配置设置为*不擦除闪存存储器*。

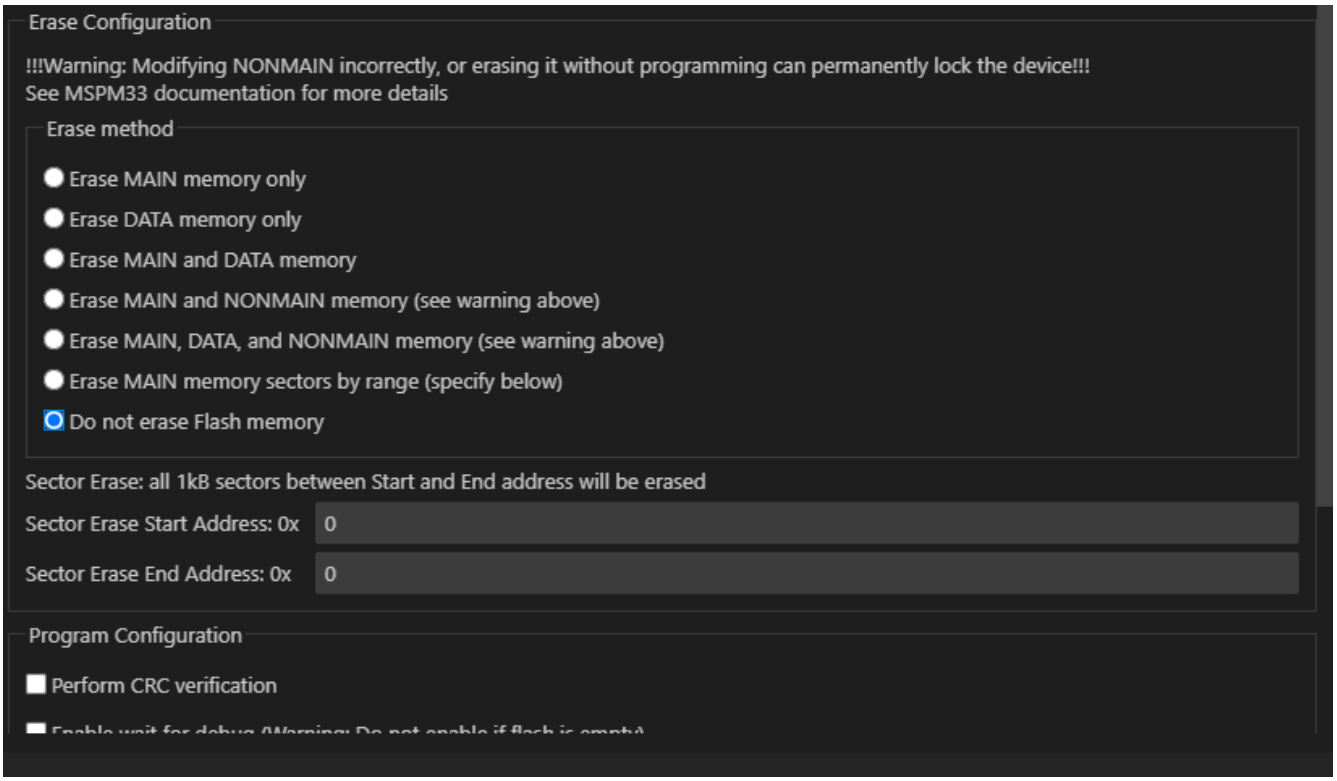
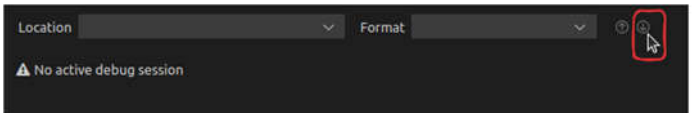


图 3-12. 配置“*不擦除闪存存储器*”

2. 若要将“dual_signed_output1.bin”刷写到器件中，请打开内存视图，然后导入文件。“dual_signed_output1.bin”的位置为 0x10000。如需参考，请使用以下图像获取“dual_signed_output1.bin”



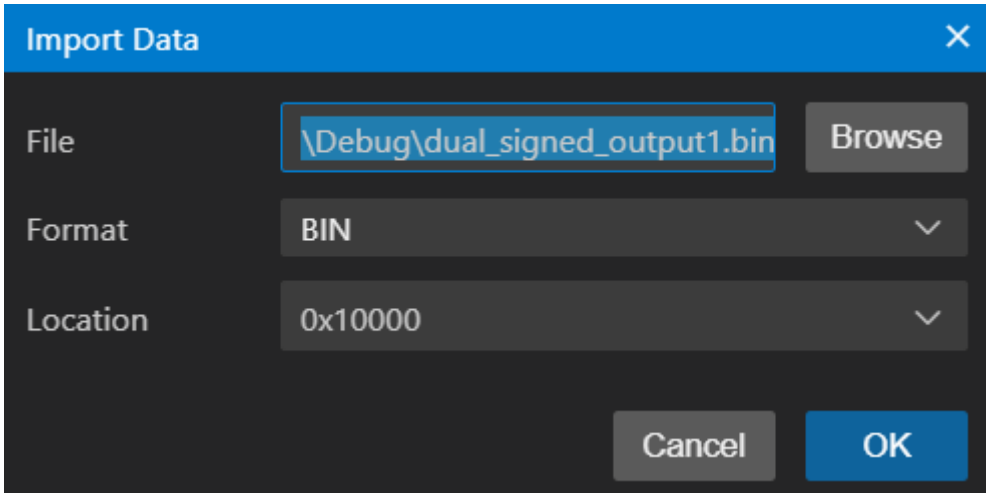


图 3-13. 在内存视图中导入图像

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

3. 若要将“dual_signed_output2.bin”刷写到工程中，请打开内存视图。“dual_signed_output2.bin”的位置为 0x90000。
4. 编程后，在内存视图中找到 0x0000 (图 3-14)、0x10000 (图 3-15) 和 0x90000 (图 3-16) 中的数据。

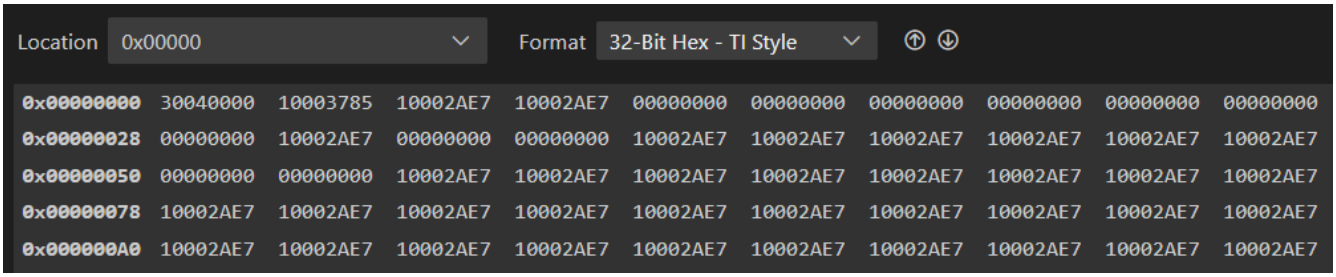


图 3-14. 0x00000 处的内存视图



图 3-15. 0x10000 处的内存视图

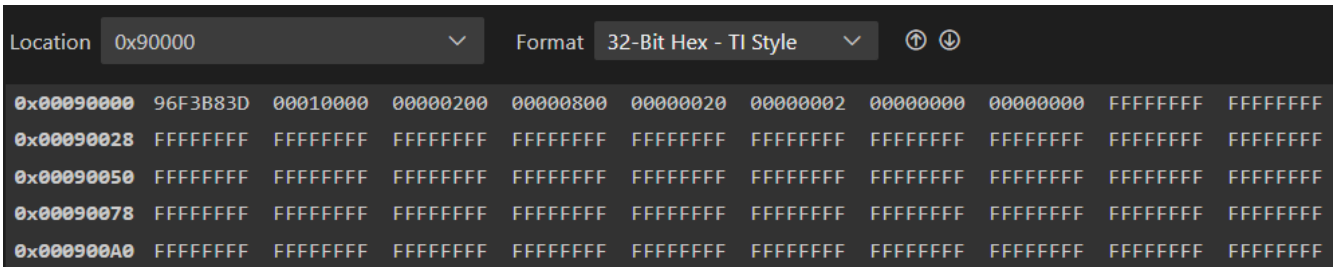


图 3-16. 0x90000 处的内存视图

3.4 运行 CSC 应用程序

在加载 CSC、映像 1 和映像 2，并执行一次引导复位后，CSC 示例能够使用 boot_keys.c 中存在的密钥，正确地对已签名的映像 1 和映像 2 进行认证。验证成功后，CSC 会跳转至该应用程序，在此程序中会点亮一个蓝色 LED。成功闪烁并运行示例后，用户可以看到蓝色 LED 亮起。

重新引导后，加载正在引导的应用程序的符号。**要将符号加载到应用程序中，请依次导航至运行和加载，然后选择符号**。在示例应用程序中，PA2 会被置位，在刷写整个 CSC 和示例映像并重新引导开发板后，可以观察到这一现象。

我们生成的两个映像具有不同的版本，用于展示此器件的更新功能。在构建后步骤的最后一步中对版本进行了澄清。“dual_signed_output1.bin”的版本被设为 1.0.0，“dual_signed_output2.bin”的版本被设为 2.0.0，这意味着实际运行的是 bank1 中的“dual_signed_output2.bin”。

1. 要检查 MCU 是否已进入应用程序，请查看“调用栈”菜单，确认器件是否正运行在 0x10000 地址处。请参考下图中的示例。

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated



图 3-17. 调用栈

2. 若要检查存储体交换，运行示例后，您可以看到两个存储体（一个在地址 0x10000 处，另一个在 0x90000 处）交换的版本信息不同。
   a. 交换在从 CSC 应用程序发出 INIT_DONE 后发生。
   b. 在开始执行 CSC 代码之前，可以看到版本号：

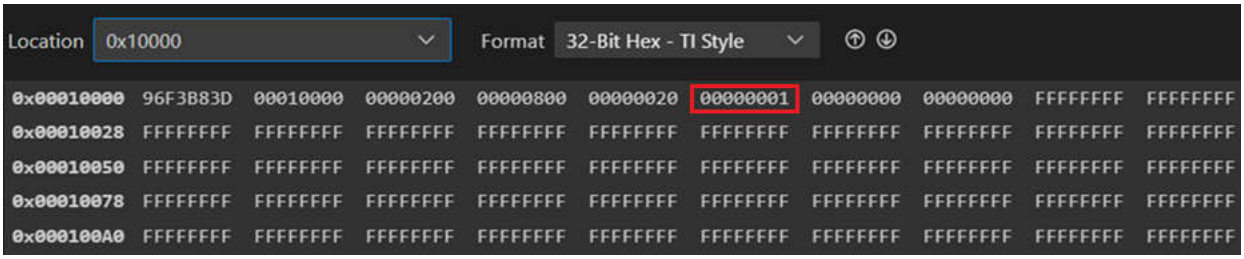


图 3-18. 0x10000 处的内存视图

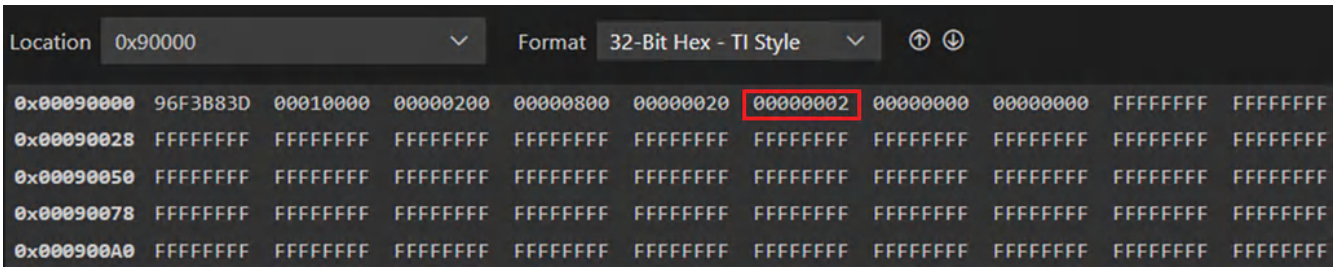


图 3-19. 0x90000 处的内存视图

   c. 在执行 CSC 代码之后，将执行存储体交换：

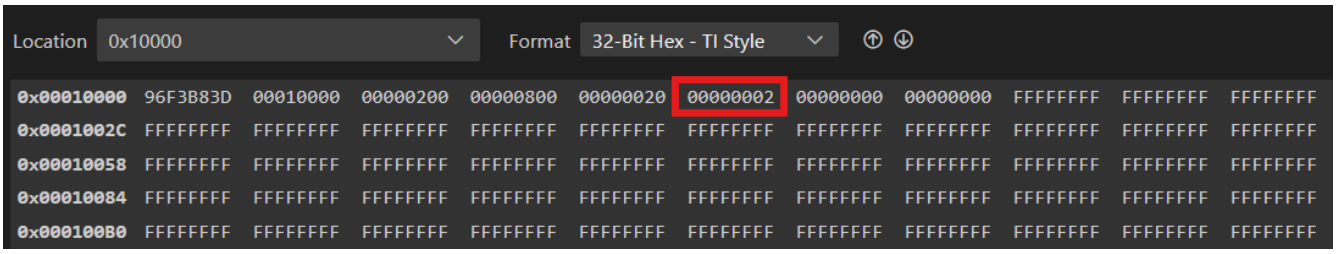


图 3-20. 0x10000 处的内存视图

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated



图 3-21. 0x90000 处的内存视图

Copyright © 2026 Texas Instruments Incorporated

4 问答

4.1 如何修改应用程序映像版本

用户可以修改应用程序更新的应用程序版本。请按照以下步骤操作。

4.1.1 应用程序映像

1. 修改命令以进行签名

```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/
input2.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/
requirements.txt
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --
key ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --
pad-header -v 1.0.0 --slot-size 0x3000 --load-addr 0x10000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input2.bin ${PROJECT_BUILD_DIR}/dual_signed_output2.bin
```

2. 例如，如果更新版本是 v 1.1.0，则将 v 1.0.0 修改为 v 1.1.0

```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/
input2.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/
requirements.txt
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --
key ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --
pad-header -v 1.1.0 --slot-size 0x3000 --load-addr 0x10000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input2.bin ${PROJECT_BUILD_DIR}/dual_signed_output2.bin
```

4.2 如何修改应用程序映像地址

用户可以修改自定义应用程序的应用程序映像地址。请按照以下步骤操作

4.2.1 应用程序映像

1. 修改应用程序映像的 .cmd 文件。

```
51  MEMORY
52  {
53    FLASH         (RX)  : origin = 0x00010200, length = FLASH_SIZE, fill = 0xFF,
54    SRAM            (RWX) : origin = 0x30000000, length = 0x0000FFFF
55
56  }
57
58  SECTIONS
59  {
60      .intvecs:  > 0x00010200
61      .text    : palign(16) {} > FLASH
62      .const   : palign(16) {} > FLASH
63      .cinit   : palign(16) {} > FLASH
64      .pinit   : palign(16) {} > FLASH
65      .rodata  : palign(16) {} > FLASH
```

2. 例如，如果更新地址为 0x30000。将 0x10200 修改为 0x30200

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

```
MEMORY
{
  FLASH         (RX)  : origin = 0x00030200, length = FLASH_SIZE, fill = 0xFF,
  SRAM           (RWX) : origin = 0x30000000, length = 0x0000FFFF

}

SECTIONS
{
    .intvecs:  > 0x00030200
    .text    : palign(16) {} > FLASH
    .const   : palign(16) {} > FLASH
    .cinit   : palign(16) {} > FLASH
    .pinit   : palign(16) {} > FLASH
    .rodata  : palign(16) {} > FLASH
```

4.2.2 应用程序映像签名命令

1. 修改命令以进行签名。

```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/
input2.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/
requirements.txt
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --
key ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --
pad-header -v 1.0.0 --slot-size 0x3000 --load-addr 0x10000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input2.bin ${PROJECT_BUILD_DIR}/dual_signed_output2.bin
```

2. 例如，如果更新地址为 0x30000。将 0x10000 修改为 0x30000

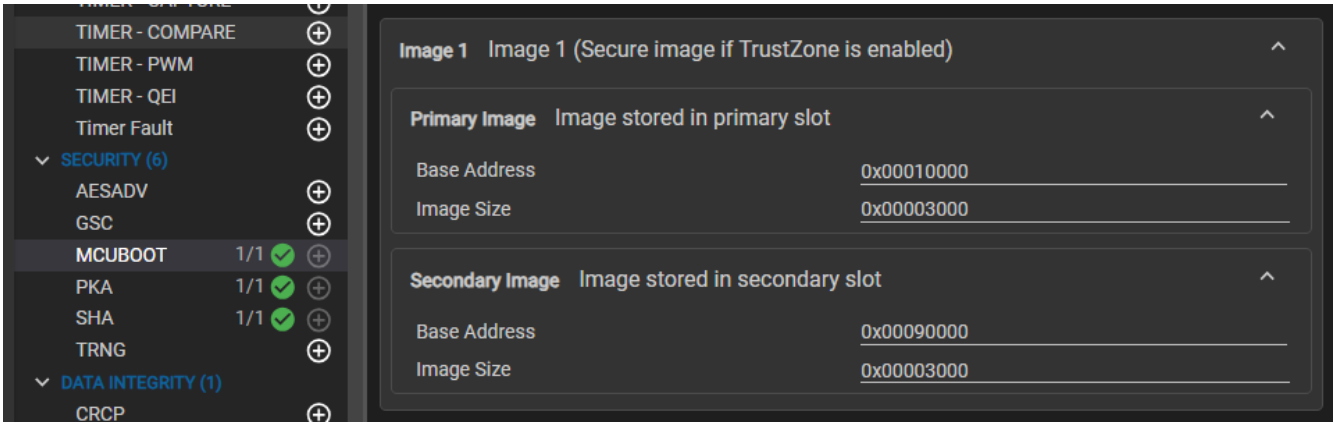
```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/
input2.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/
requirements.txt
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --
key ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --
pad-header -v 1.0.0 --slot-size 0x3000 --load-addr 0x30000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input2.bin ${PROJECT_BUILD_DIR}/dual_signed_output2.bin
```

4.2.3 CSC 示例代码

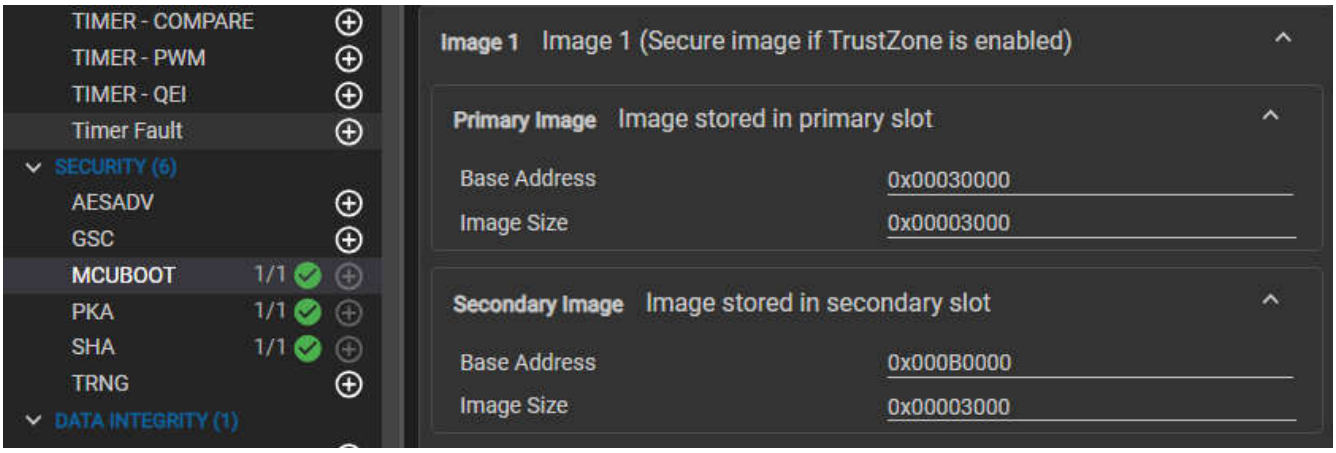
1. 修改 customer_secure_config.h 中的地址。现在，用户可以使用 SysConfig 配置地址。

```
#define CSC_APPLICATION_IMAGE1_BASE_ADDR                                (0x00010000)
#define CSC_APPLICATION_IMAGE1_SIZE                                     (0x00003000)
#define CSC_APPLICATION_IMAGE2_BASE_ADDR                                (0x00090000)
#define CSC_APPLICATION_IMAGE2_SIZE                                     (0x00003000)
```


Copyright © 2026 Texas Instruments Incorporated



2. 例如，如果更新地址为 0x30000。将主映像的地址从 0x10000 修改为 0x30000。将辅助映像的地址从 0x90000 修改为 0xB0000。
   a. 两个不同版本的应用程序映像均构建用于地址 0x30000，但其中一个需要刷写至 bank0，而另一个应用程序需要刷写至 bank1
   b. 存储体 0 – 0x00000 + 0x30000 = 0x30000
   c. 存储体 1 – 0x80000 + 0x30000 = 0xB0000



4.3 如何修改插槽大小

用户可能需要为应用程序映像提供更大的插槽大小。

4.3.1 应用程序映像

1. 修改命令以进行签名

```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/
input2.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/
requirements.txt
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --
key ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --
pad-header -v 1.0.0 --slot-size 0x3000 --load-addr 0x10000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input2.bin ${PROJECT_BUILD_DIR}/dual_signed_output2.bin
```

2. 例如，如果更新插槽大小为 0x8000。将 0x3000 修改为 0x8000

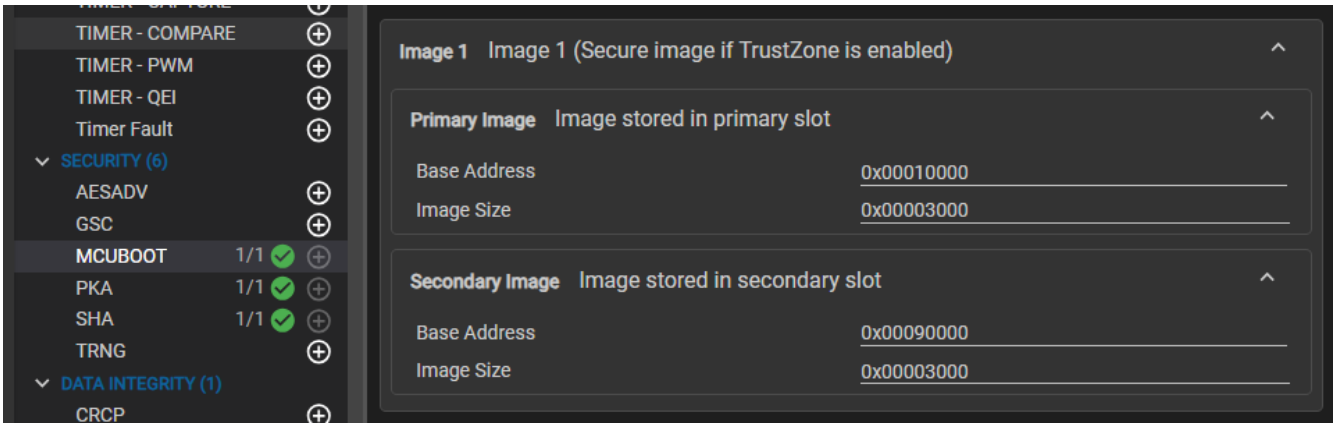
```
${CG_TOOL_OBJCOPY} -O binary ${PROJECT_BUILD_DIR}/${ProjName}.out ${PROJECT_BUILD_DIR}/input2.bin
pip install click cryptography intelhex cbor>=1.0.0
pip install -r ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/requirements.txt
```


English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

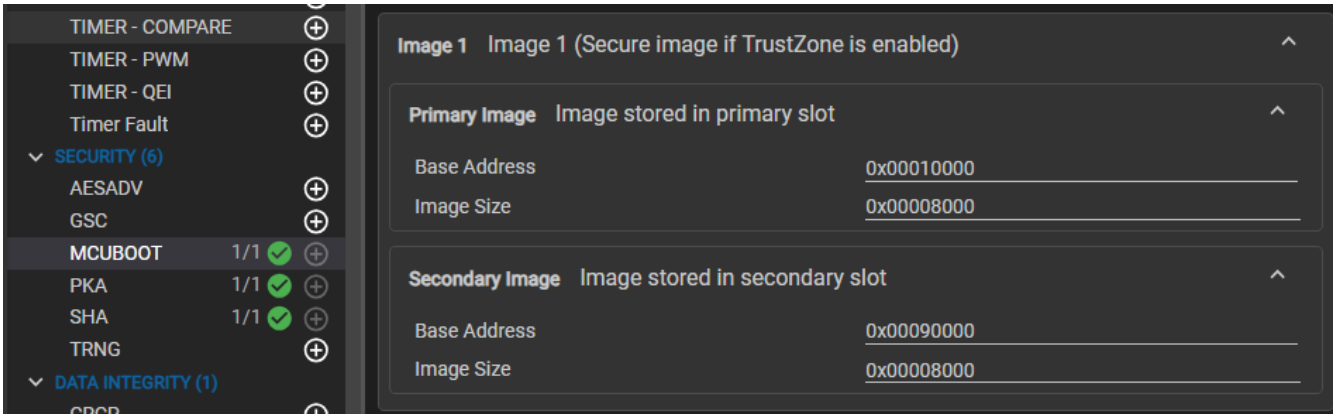
```
pip install dilithium_py
python ${COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/scripts/imgtool.py sign --key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/ecdsa-p256.pem --second-key $
{COM_TI_MSPM33_SDK_INSTALL_DIR}/source/third_party/mcuboot/mldsa_key.pem --header-size 0x200 --pad-
header -v 1.0.0 --slot-size 0x8000 --load-addr 0x10000 --align 8 --pad ${PROJECT_BUILD_DIR}/
input2.bin ${PROJECT_BUILD_DIR}/dual_signed_output2.bin
```

4.3.2 CSC 示例代码

1. 修改 customer_secure_config.h 中的大小。用户可以使用 SysConfig 配置大小。



2. 例如，如果更新大小为 0x8000，请将 0x3000 修改为 0x8000。



4.4 如何修改应用程序映像大小

用户可以配置应用程序映像的大小。

4.4.1 应用程序映像

1. 修改 cmd 文件中的 FLASH_SIZE，这将影响 input_bin 的大小

```
48  /* Region of Flash that is read-protected after INITDONE */
49  //#define FLASH_SIZE    0x0800
50  #define FLASH_SIZE    0x60000
```

4.4.2 修改插槽大小

要修改插槽大小，请参阅如何修改插槽大小。验证插槽大小是否大于 input_bin 的大小。

4.5 对应用程序映像或 CSC 执行更多自定义操作

有关更多说明，请参阅 *<MSPM33_SDK>\docs\english\middleware\boot_manager\Secure_Booting_Users_Guide* 或《安全引导用户指南》

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

5 参考资料

- 德州仪器 (TI)，*MSPM33 C3 系列 160MHz 微控制器* 技术参考手册
- 德州仪器 (TI)，*MSPM33C 安全性 用户指南*
- 德州仪器 (TI)，*MSPM33 NONMAIN 闪存存储器配置指南*
- 德州仪器 (TI)，*在 MSPM33 MCU 中启用信任区以实现安全应用*
- *德州仪器 (TI)，MSPM33 客户安全代码及引导加载程序 (CSC) 用户指南*
- *映像工具自述文件 Imgtool.md*

6 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

日期	修订版本	注释
August 2026	*	初始发行版

English Document: SDAA472
Copyright © 2026 Texas Instruments Incorporated

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

版权所有 © 2026，德州仪器 (TI) 公司

最后更新日期：2025 年 10 月