恩智浦半导体
应用笔记

文档编号：AN12323
第0版，2018年12月

S32K1xx 固件更新

作者：恩智浦半导体

1 介绍

随着目前技术的进步，车辆正变得更加电子化，而不是机械化。汽车领域的电子创新也在不断增长。因此，车辆中的软件也在增加，从而存在潜在错误的风险。

每次发现一个软件错误时，都需要一个召回过程来更新该软件。这些召回代表了汽车制造商的成本，对最终用户来说，也是一个耗时的过程。

如今，空中更新开始使用a/b交换技术在新车上实现。使用这种技术进行的更新仅限于具有高密度内存的微控制器，如网关或信息娱乐单元。用于边缘节点应用程序的较小的微控制器不会得到更新。

除了负责边缘节点微控制器上的固件更新外，另一个问题是保持更新过程的安全。这是为了保护更新免受可能的窃听、复制攻击。

本应用程序说明重点用于展示S32K1xx微控制器如何使用安全通信过程处理a/b交换更新。它解释了引导加载程序的基本概念、a/b交换更新和安全概念。本文档涵盖了整个系统中固件更新的实现。了解闪存控制器、安全模块、CAN控制器等特定模块的功能细节。请参考S32K1xx设备参考手册。

目录

本应用程序说明中包含了补充软件，以演示a/b交换更新，同时利用S32K144和S32K146微控制器的功能。

2 引导加载程序的概念

在应用程序的固件开发阶段，对微控制器的更新要用闪存编程工具，需要一定数量的微控制器的 I/O 和一个特定的标头来完成交互。一旦固件完成，它将被下载到微控制器，并焊接在最终的 pcb 中。此 pcb 可能包括或不包括编程工具所需的标头，以避免访问在闪存中存储的数据，或因为应用程序需要其应用程序的输入/输出。在某些情况下，托管微控制器的最终模块处于难以访问的位置，例如在汽车应用程序中，电子控制单元被放置在为最终用户隐藏的区域中。那么，如果在设备在现场后，由于故障或升级，需要更新固件怎么办？为了解决这个问题，我们开发了引导加载程序。

引导加载程序是驻留在微控制器的非易失性存储器（NVM）中的一块固件，用于处理应用程序或数据的更新。引导加载程序和应用程序可能位于同一非易失性内存块中，或在单独的块中。应用程序固件和引导加载程序不应该在内存中重叠；因此，引导加载程序通常放置在内存空间的开头。引导加载程序预计不会被更新，因此它所在的区域受到保护，以避免有意或无意的修改。

引导加载程序通过通信协议从主机接收新的固件数据，包括但不限于CAN、LIN、UART、以太网和 USB。此通信协议取决于开发人员的策略、应用程序和微控制器特性。主机使用已知格式发送新固件，例如srec或bin文件。引导加载程序捕获带有新固件数据的数据包，然后更新放置旧固件的内存位置。一旦收到并验证了完整的固件，引导加载程序将跳转到新的应用程序。

在深入了解引导加载程序更新算法的细节之前，有必要了解到NVM中的写入过程。要写入NVM位置，必须首先删除数据，非易失性内存中的数据按扇区删除。这个扇区大小是可以擦除的数据的最小部分，它取决于每个非易失性内存技术，它被称为内存的粒度。一旦该扇区被删除，数据就可以被写入该扇区。写过程还取决于NVM技术，每个NVM都有许多每个写入命令可以写入的字节。最后，要完成写入过程，需要一个程序检查命令来验证数据是否按预期以flash写入，此程序检查将通过写入命令完成。

引导加载程序固件的大小取决于非易失性内存的粒度。粒度越大，扇区大小越小，引导加载器的大小就越灵活。由于引导加载程序将从接收将编程到NVM的主机的新固件数据，因此在写入数据之前必须考虑扇区以前被擦除。

在更新过程中，从NVM读取和执行的引导加载程序固件，启动擦除和写命令，将新的应用程序数据更新到NVM本身，因此必须考虑在内存内同时进行操作。允许的内存同时操作是特定于每个设备的，有些设备允许在写入内存时读取，只要这发生在不同的读取分区或内存块中。如果在读取功能时写入不可用，引导加载程序需要将内存命令放到 ram 位置，以便可以从那里启动，以避免任何冲突。

可以按照软件或硬件的方法来进入更新过程。在一定时间、重置后输入的低或高状态，或通过接收应用程序运行时指示需要更新的特定通信包，通过按下按钮触发更新。通常，在微控制器重置后，引导加载程序固件会在一定时间内检查是否存在启动更新过程的条件，如果不满足此条件，则会跳转到应用程序固件。一些用例还可以在应用程序运行时检查此触发器，允许在运行时进行更新，并避免重置后的停机，因此在下一个重置中，设备立即启动到最新映像。关于如何触发更新过程的决定取决于最终的应用程序、内存和设备特性。

当检测到更新过程触发器时，更新算法开始将新的固件应用程序放到 NVM 中。更新算法包括以下步骤：从通信协议接收数据，验证接收到的数据是否正确，擦除/程序闪存，检查是否已下载完整的数据。

更新过程结束后，引导加载程序应将使用的外设和时钟配置返回到默认状态。如果重置后启动加载程序启动，这是必要的。除了外设和时钟配置外，中断向量表。应用程序固件可能在不同的位置使用中断表，如果是这种情况，指向中断表的指针必须从其默认位置重新定位。一旦处理了这些考虑事项，引导加载程序就可以跳转到应用程序固件。更新过程的流程图如下图所示。

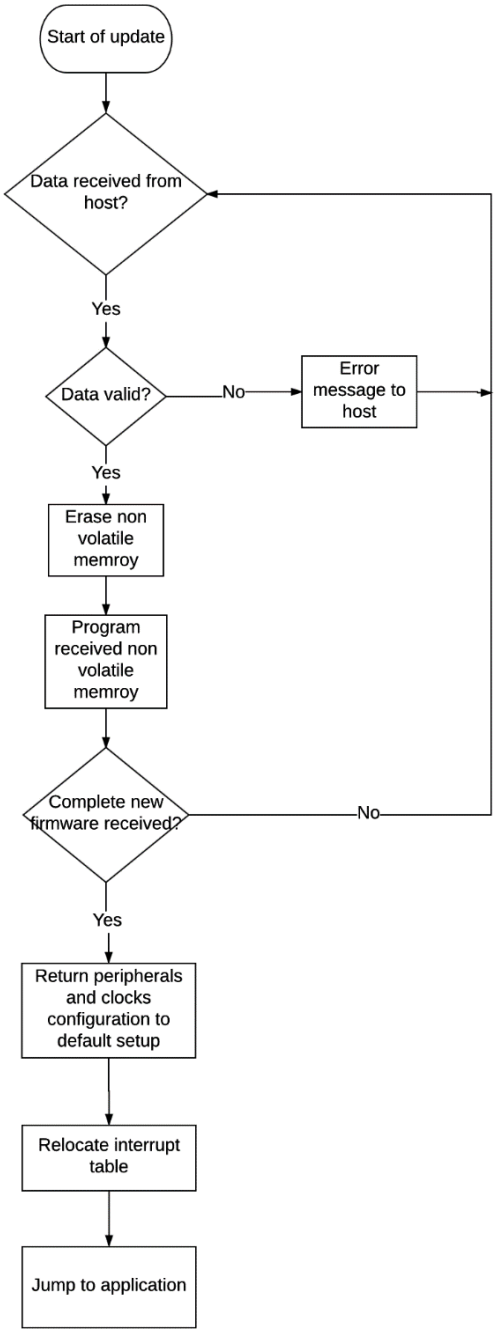

图1. 更新过程

主机和引导加载程序之间共享的应用程序固件很容易被捕获和读取。这是因为潜在的攻击者可以嗅探发生更新过程的网络。为了避免窃听攻击，保护固件的机密性，数据可以进行加密。

在数据在网络中传输之前，可以在主机中进行加密。然后，引导加载程序需要在写入NVM内存之前使用已知密钥解密此数据。除了加密之外，还可以向引导加载程序中添加验证主机真实性和数据新鲜度的机制。

最后，在开发引导加载器固件时，应该考虑在内存中的位置。引导加载程序固件和应用程序可能驻留在同一 NVM 内存块中，因此有必要修改引导加载程序和应用程序固件链接器文件，以避免重叠。有一些设备在内存中有一个专用的位置来放置引导加载程序。链接器文件的修改取决于所使用的微控制器和编译器。

3 固件更新方法

要更新微控制器中的固件，请遵循两种方法中的任何一种。这两种方法是：a/b方法和就地方法。传统上，人们采用了就地的方法。在这种方法中，微控制器只存储一个固件映像，它可能占用所有可用的闪存。在这种情况下，在微控制器运行时无法进行更新，如果更新过程没有正确完成，则很难恢复旧映像。下图显示了这一过程。

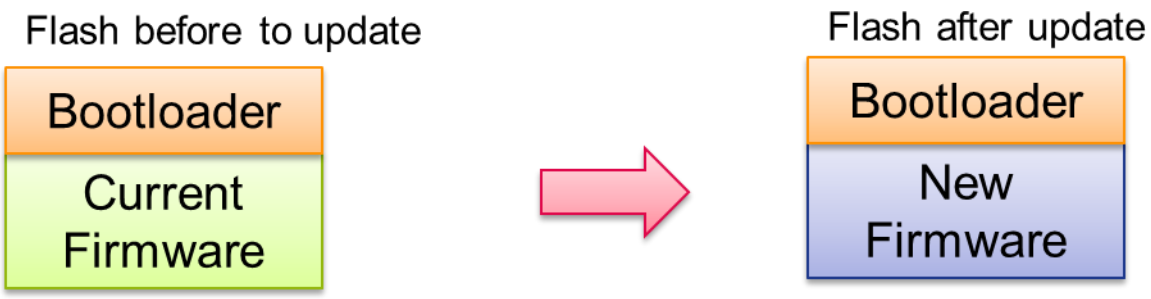

图2. 就地更新

a/b交换方法是指微控制器在不同的闪存区域存储两个图像。在这种情况下，可能可以在微控制器运行时对一个区域进行固件更新。由于微控制器存储了两个不同的固件，旧的固件和更新的固件，如果新的固件不能正常工作，可以立即恢复。主要的缺点是，两个应用程序映像应该占用微控制器的NVM，从而减少了最大固件应用程序的大小。这种方法的另一个挑战是固件重新映射，因为在NVM的不同物理地址中有两个应用程序。A/B交换过程如下图所示。

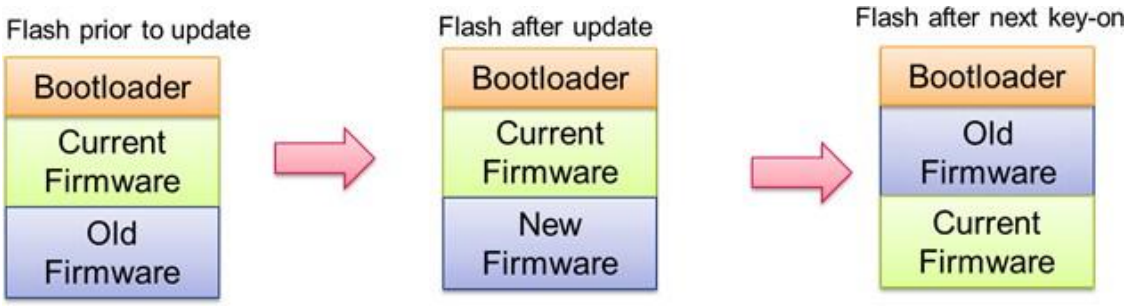

图3. A/B交换更新

因此，这两种方法都有优缺点，但要由汽车制造商决定哪种方法适合该应用。本应用程序说明重点介绍了 S32K1xx 微控制器中的 a/b 交换实现。

4 安全方面的概念

高级加密标准（AES）是一种对称的加密/解密密码。该标准使用 Rijndael 算法使用密码密钥处理数据。每个密钥的大小可以为128、192或256位。下图显示了算法的步骤。

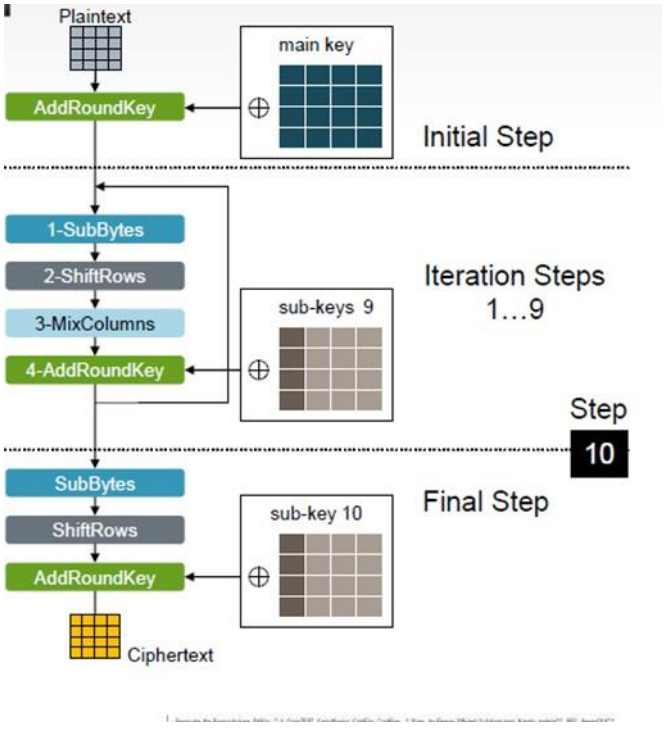

图4. AES算法

有两种可以加密和解密数据的方法。这两种方法是：电子码簿（ECB）和链码簿（CBC）。

ECB方法将纯文本划分为输入值。每个输入的密码粒度大小。加密的输出仅取决于纯文本输入和密钥。每个密码输出都被独立加密。这种密码模式的缺点是，相同的输入值将被解码为相同的输出值。这使得攻击者有机会使用统计分析（例如，在正常文本中，一些字母组合比其他组合频繁得多）。

CBC方法将其划分为输入中的纯文本。每个输入的密码粒度大小。最后一个编码步骤的输出包含当前编码步骤的输入块。因此，需要为第一个编码步骤提供一个额外的值，它称为初始化向量（IV）。使用这种方法，每个密码块取决于迄今为止处理的明文输入。

下图展示了一个例子，显示了两种方法的结果，使用企鹅的图像作为纯文本。

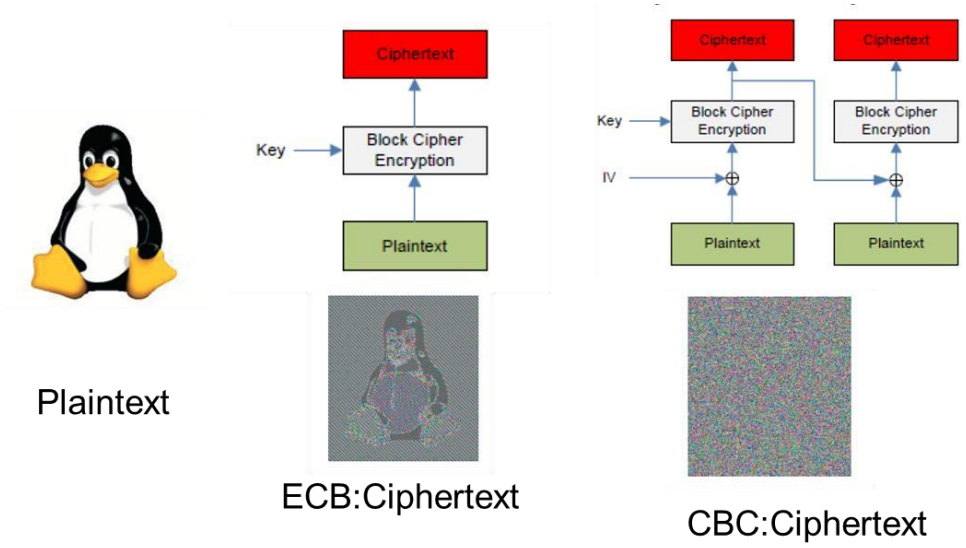

图5. ECB和CBC的示例

基于密码的消息身份验证码（CMAC）提供了一种身份验证消息和数据的方法。CMAC使用AES算法。CMAC算法接受密钥和要验证的任意长度消息作为输入，并输出CMAC。CMAC值通过允许验证者（他们也拥有密钥）检测对消息内容的任何更改，来保护消息的数据完整性和真实性。图中显示了CMAC的组成部分。

图6. CMAC的生成

5 S32K1xx 概述

NXPS32K1xx产品系列的32位汽车微控制器提供了一个高度集成、低功耗、安全、安全的单片机解决方案。快速CPU与灵活的低功耗模式和低泄漏技术过程的结合，不会迫使相对于低功耗对性能造成任何妥协。

S32K1xx系列提供了广泛的内存范围，从128kB到2MB。它共享公共的外设和针脚计数，允许开发人员在MCU系列中或MCU家族之间轻松迁移，以利用更多的内存或特性集成。这种可伸缩性允许开发人员使用S32K1xx产品系列作为其最终产品平台的标准，最大限度地提高硬件和软件的重用，并减少上市时间。

S32K1xx支持CAN灵活数据速率（CAN-FD）以及新的FlexIO可配置外设，允许客户实现未来尚未发明的通信协议，并将通道扩展到现有的片上硬件协议控制器。

安全模块，精简版加密服务引擎（CSEc），包含在 S32K1xx产品中。CSEc如SHE函数规范中所述，实现了一组全面的加密函数，其中包括：通用密钥、AES-128、CBC、ECB、CMAC、伪随机数生成（PRNG）和真随机数生成（TRNG）。这允许保护ecu免受各种攻击场景，并确保系统的完整性。

下图为S32K11x和 S32K14x产品的方框图。

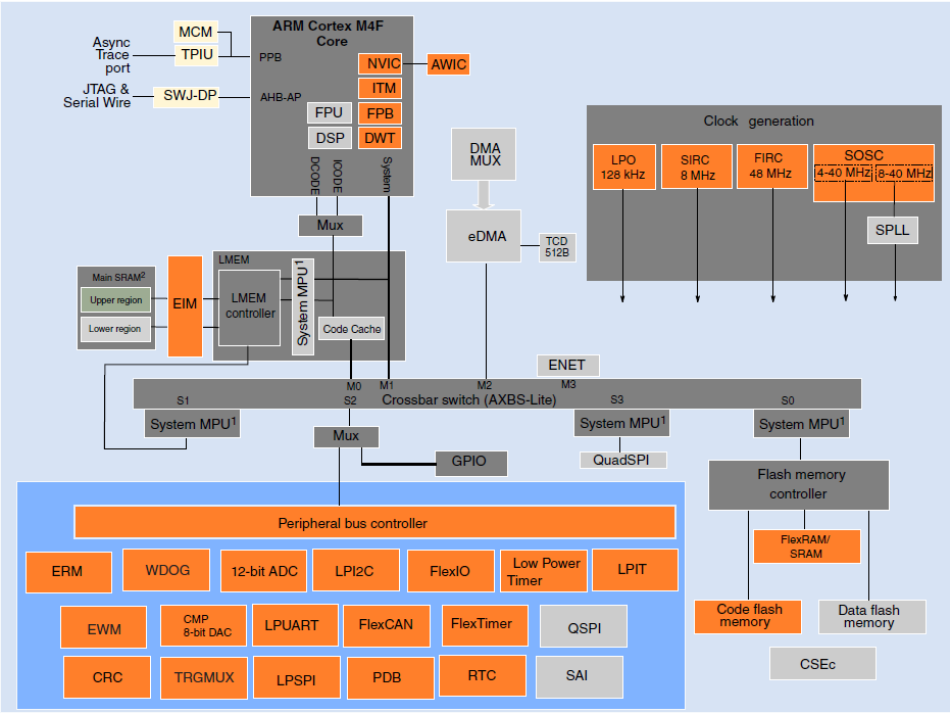

图7. S32K14x方框图

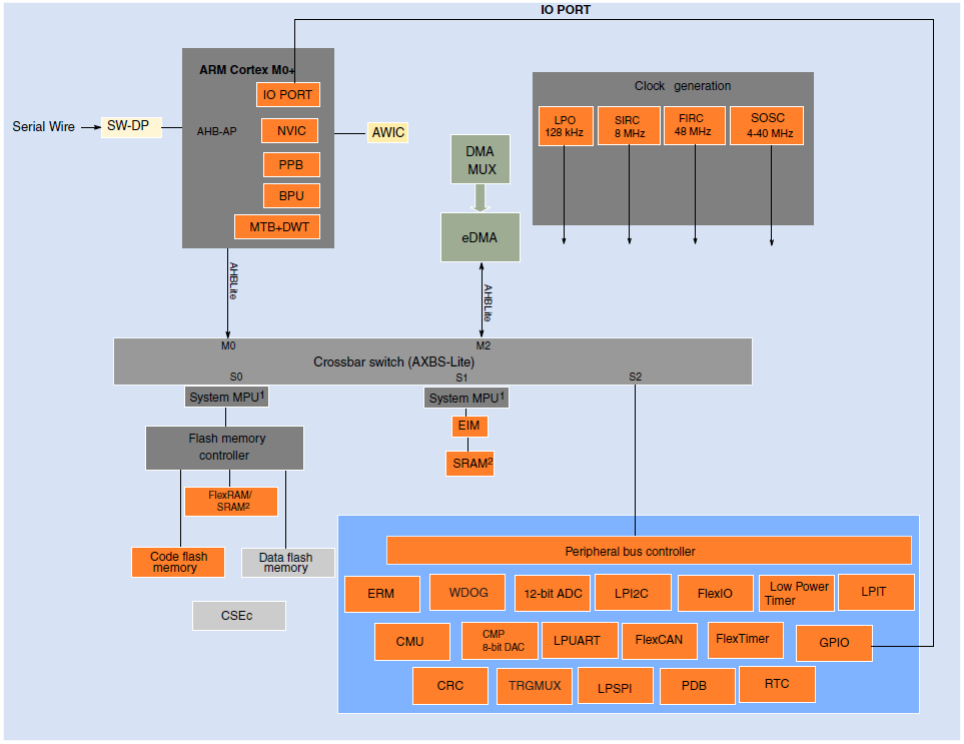

图8. S32K11x方框图

5.1 NVM 体系结构

S32K1xx微控制器具有 C90TFS（薄膜存储）技术闪存。这些设备中有两个闪存，程序闪存（pflash）和闪存。

程序闪存是一种非易失性存储器，其主要目的是执行代码。程序闪存的大小因设备而异，其范围在128kB到1.5MB之间。在程序闪存高于256kB的设备中，该闪存的扇区大小为4kB，而对于程序闪存等于或小于256kB的设备，该扇区大小为2kB。在所有设备中，pflash编程命令一次写入8B。此闪存的块大小为512kB或更小，这取决于设备有一个或多个闪存块。每个块代表一个读取分区，这意味着可以在被删除或写入另一个块时读取一个块。

灵活内存闪存的目的是执行代码，存储数据或功能作为EEPROM与灵活内存内存。每个设备的柔性存储器的大小不同，包括32kB（S32K11x）、64kB（S32K142/4/6）或512kB（S32K148）。对于S32K116/8和S32K142/4/6设备，该存储器中的扇区大小为2kB，而对于S32K148设备的扇区大小为4kB。在所有设备中，flex内存编程命令一次写入8B. flex内存是一个单独的读取分区，因此可以在擦除或写程序闪存时从这个flash中读取，反之亦然。如果 flex内存配置为EEPROM，可以对其进行分区，以便flex存储器存储代码或数据，而另一部分用作EEPROM备份。弯曲内存的大小可划分为：16kB、32kB和64kB。

下表总结了每个 S32K1xx设备的大小和闪存粒度。这些内存在每个设备的内存地图中的位置显示在中图 9。

表1. S32K1xx内存大小

设备	程序闪存	程序闪存扇区大小	Flex内存	Flex内存扇区大小
S32K116	128kB	2kB	32kB	2kB
S32K118	256kB	2kB	32kB	2kB
S32K142	256kB	2kB	64kB	2kB
S32K144	512kB	4kB	64kB	2kB
S32K146	1 MB	4kB	64kB	2kB
S32K148	1.5 MB	4kB	512kB	4kB

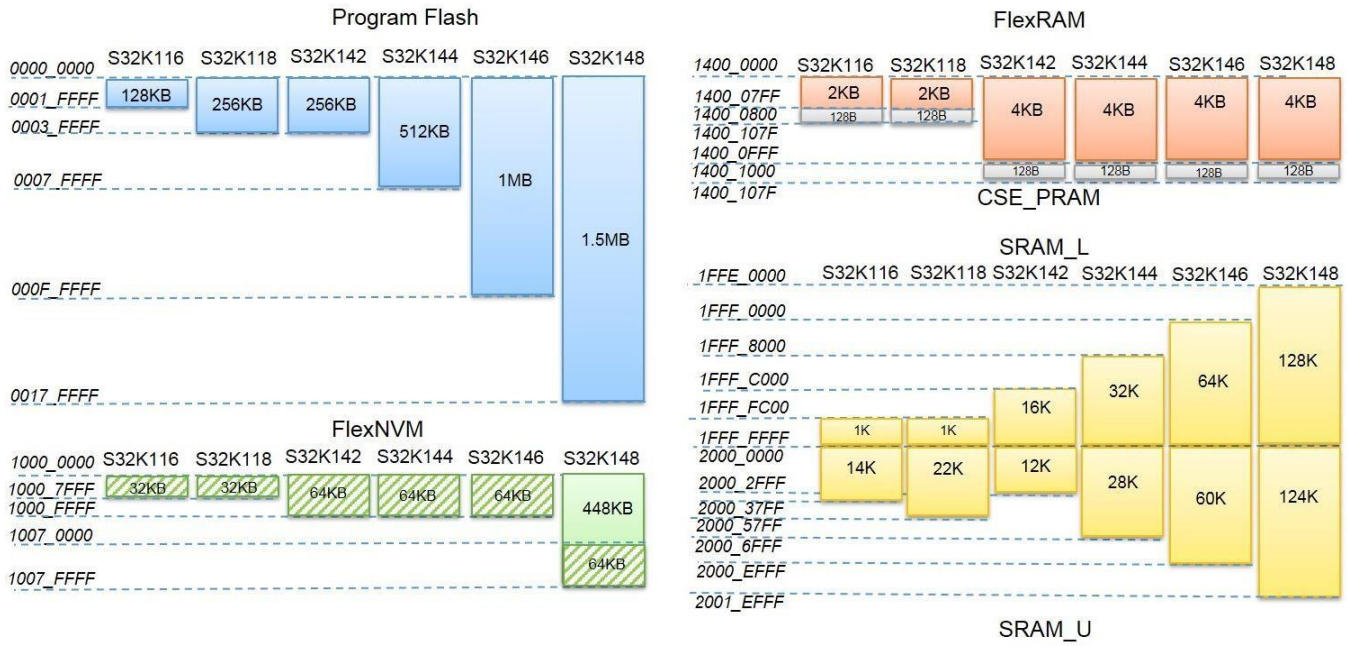

图9. S32K1xx内存大小

这些存储器的微控制器中包含一个闪存控制器。此flash控制器执行要擦除、编程和检查这些内存的命令。除了访问这些内存中的内容的命令外，还包括安全命令。可以从 flash 控制器执行加密、解密、cmac 计算、随机数生成等命令。

6 S321xx 固件更新实现

6.1 S32K144 用例

6.1.1 S32K144 交换内存地图，单个闪存块

在这种情况下，S32K144 设备用于在设备中实现a/b交换更新。请记住，S32K144 有一个 512kB 的程序闪存，和一个 64kB 的灵活内存，总共有两个可用的读取分区。引导加载程序从弹性内存加载到一个 16kb 的空间中。程序 flash定位 a/b 交换所需的两个应用程序映像，包括每个映像的一个头

每个应用程序功能上的标头确定了固件信息。信息，如，但不限于，固件版本、最旧版本、软件开发人员细节、大小、作者、应用程序密钥，可以包含在此标题中。每次更新后，报头信息也会被更新。报头的大小为4kB，一个程序闪存扇区大小。标头不仅可以存储运行的固件信息，还验证最新固件以及是否可以执行固件。

在 ARM® CortexM 核心中，在重置发生后，core 从程序闪存位置读取前两个单词，其中需要中断向量表。第一个单词包含该堆栈指针，第二个字包含重置处理程序的地址。然后，core 将主堆栈指针地址（SP）和程序计数器（PC）设置为重置处理程序的开始位置。由于核心的初始启动过程，有必要保留而不是擦除程序闪存 的第一个扇区。在 S32K144 中，向量表的大小为 1kB 长，因此给定程序闪存 flash的扇区大小，不能删除初始的 4kB 的 flash。

对于这种情况，引导加载程序被分配到 flex 内存区域，因此 pflash 中的第二个字包含指向 flex 内存地址的地址。引导加载程序被放置在 flex 内存上，因为它是一个不同的读取分区，它允许执行修改程序 flash 内容的命令，而不在 RAM 中分配例程。在 flex内存中分配引导加载器的另一个好处是，在 S32K146 中，有两个程序闪存的读取分区。可以在旧固件仍在运行时对新固件的更新。

这两个应用程序的后面是它的头。在每个应用程序之间，一个扇区的间隙留为空白。这是因为在考虑了默认中断表的初始扇区和固件头的两个扇区后，剩余的扇区数并不均匀。然后，每个应用程序固件映像的大小将为 248kB 或 62个扇区。下图显示了结果的闪存地图的摘要。

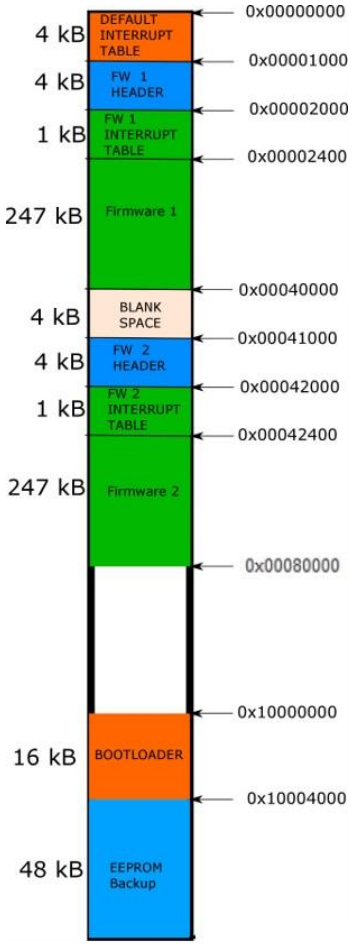

图10. S32K144内存图：A/B交换

6.1.2 S32K144 启动和更新过程

要确定程序闪存中最旧图像的位置，请使用标头上的信息。在这种情况下，通过验证固件在最后一个头地址是否有有效的应用程序键符号 0x55AA55AA 来完成头。标头的此签名是在跳转到应用程序之前的更新过程中完成的最后一个步骤。此键指示固件更新已完成，并且它已有效。

当引导加载程序启动时，它会搜索应用程序头中搜索应用程序密钥。它首先读取头1的最后一个地址的内容，如果在该位置没有找到任何内容，那么分配即将到来的固件的新地址就是该位置。如果在图像一个标头中找到一个应用程序密钥，则读取固件标头2的内容，如果没有找到一个应用程序密钥，则分配即将到来的固件的新地址是图像两个位置。当在两个标头中都找到一个应用程序键时，然后将读取固件的版本，以确定最古老的版本。那么，最老的固件的位置就是分配即将到来的固件的位置。

如果在重置后的定义超时时间内，主机没有请求任何更新（例如 4秒），引导加载程序跳转到使用最新的固件的闪存位置。对于两个头都没有应用程序键的最终情况，这意味着没有有效的固件，然后忽略超时，引导加载程序继续等待来自主机的更新请求。图11显示引导过程的步骤。

第1步。重置后：获取0x000000004的PC值。

第2步。启动加载程序已启动。

第3步。引导加载程序搜索最古老和最新的图像。

- 检查无线软件报头信息。
- 在标头末端找到应用程序键（0x55AA55AA）。
- 如果在任何一个标头中都没有找到应用程序键，请留在引导加载程序中（不要跳转到任何应用程序）。

第4步。超时后：跳转到最新的应用程序。

- 重新定位VTOR表。
- PC从固件中断表中获取值。

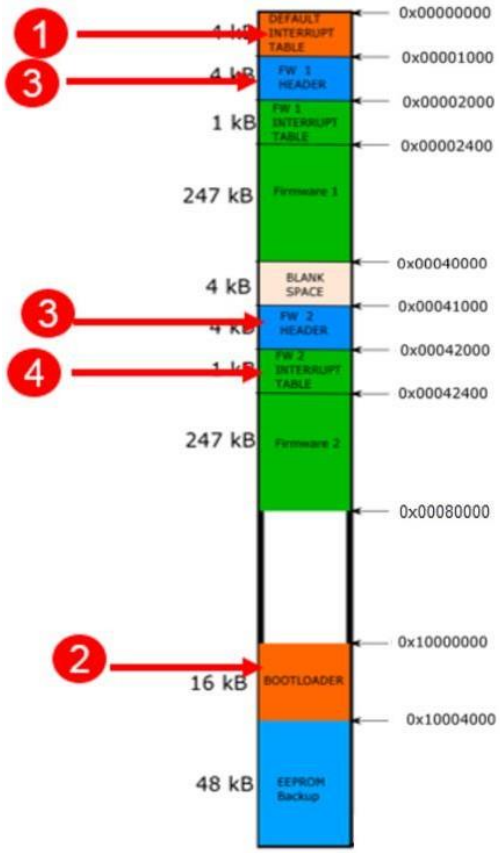

图11. S32K144引导过程

当在重置后的超时时间内触发了新的更新请求时，主机和引导加载程序（边缘节点）之间的通信将开始下载新的固件。主机和引导加载程序之间的通信由最终的应用程序决定，在这种情况下使用 CANFD。在新固件的传输完成后，新固件的标头将被签名并写入 NVM 内存，而旧的固件标头则将被删除。最后，引导加载程序会跳转到新的固件位置。如果更新过程没有完成（例如。）引导加载程序会跳转到旧的固件位置。图12显示此更新过程的步骤。

第1步。重置后：获取 0x000000004 的 PC 值。

第2步。启动加载程序已启动。

第3步。引导加载程序搜索最古老和最新的图像。

- 检查 FW 头信息。
- 在标头末端找到应用程序码（0x55AA55AA）。
- 指定要更新的 FW 位置（最旧的）。

第4步。已收到更新触发器。

- 首先接收标头。
- 验证它是一个新版本。
- 开始在最旧的位置更新新的固件。

第5步。更新已完成。

- 更新新的固件标头。
- 删除/更新旧的固件标头。

第6步。跳转到新的应用。

- 重新定位 VTOR 表。
- PC 从新的固件中断表中获取值。

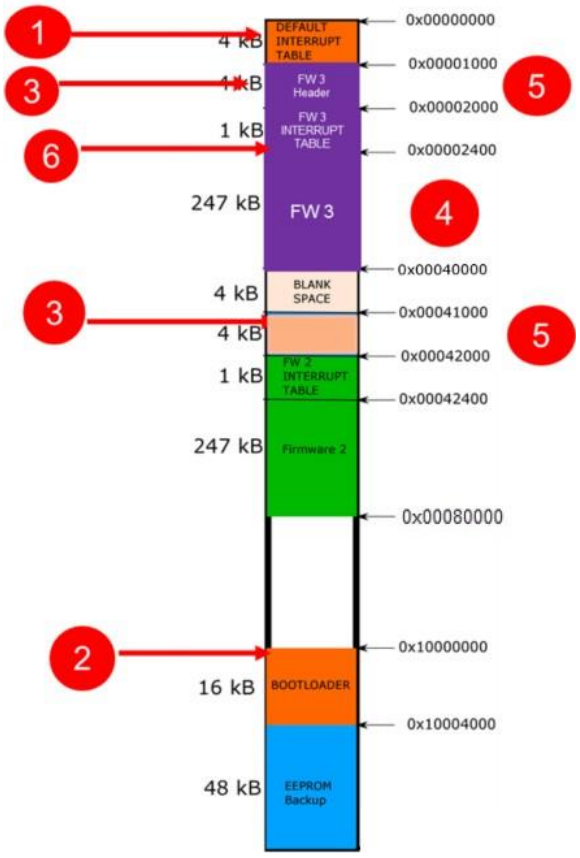

图12. S32K144 更新过程

6.2 S32K146 用例

6.2.1 S32K146 内存地图的A/B交换，与双程序闪存块

对于此用例，将使用 S32K146来演示 a/b交换更新机制。S32K146 有 1MB 的程序闪存，每个两个读取分区为 512KB，一个 64KB 的灵活内存空间。双 pflash 读取分区允许在读取其中一个分区或执行代码时擦除或写入其中一个分区。这意味着，在加载新固件时，可以继续执行当前固件。引导加载程序被放置在弯曲内存的 16kb 空间中。

与 S32K144 用例一样，在两个 pflash 读取分区中相应地加载了两个应用程序。每个固件应用程序都包含一个带有固件信息的头（固件版本、最旧版本、软件开发人员详细信息等）。

重置后，ARMCortexM4core 从向量表中取前两个字来初始化堆栈指针（SP）并加载初始程序计数器（PC），在这种情况下是位于 flex 内存中的引导加载器的入口点。由于这个默认的 ARMCortexM4 引导过程，必须保留向量表所在的第一个闪存扇区。

在第二 pflash 读取分区的开始处，一个扇区的间隙留下空白，以补偿包括向量表的第一读取分区中保留的 flash 扇区，这样对每个固件映像具有相同的可用空间。有了这种结构，除了 4KB 的映像头外，pflash 中的每个固件映像都可以占用高达 504KB（126个扇区）的空间。

下图显示了 S32K146 用例的内存图。

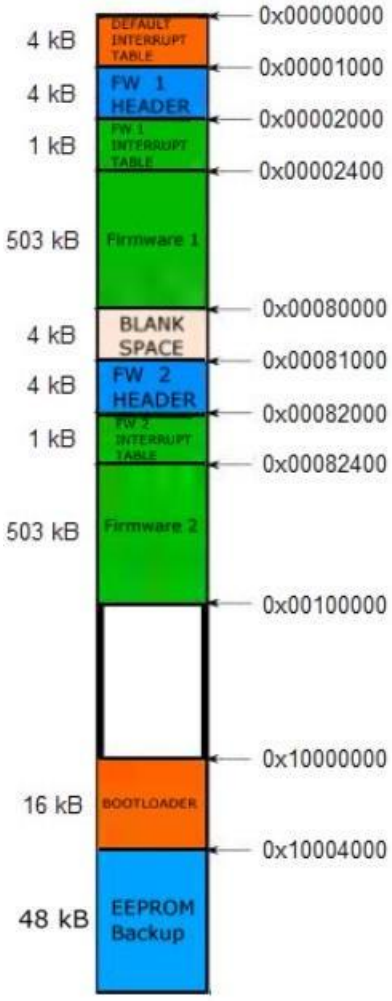

图13. S32K146内存图：A/B交换

6.2.2 S32K146 启动和更新过程

正如在S32K144用例中一样，每个固件标头的信息被用于确定程序闪存中最老的图像的位置。通过验证固件头是否在最后一个头地址上有有效的应用程序键符号（0x55AA55AA），这表明相应的固件有效。

当引导加载程序启动时，它会读取固件头。分配即将到来的新固件的内存块是一个没有有效应用密钥的内存块。如果两个标头都有有效的应用程序密钥，则将比较固件版本，以确定即将到来的固件的位置。具有最旧版本的固件将被覆盖。

引导加载程序在任何重置后都有一个定义的超时时间。S32K146 用例为2秒等待来自主机的更新请求命令。如果没有收到任何请求，则引导加载程序将使用最新的固件跳转到flash位置。如果引导加载程序没有在两个固件头中找到有效的应用程序密钥，则禁用超时，引导加载程序继续等待更新请求。图14显示了 S32K146 用例的引导过程。

第1步。重置后：获取 0x000000004 的 PC 值。

第2步。启动加载程序已启动。

第3步。引导加载程序搜索最古老和最新的图像。

- 检查无线软件报头信息。
- 在标头末端找到应用程序码（0x55AA55AA）。
- 如果在任何一个标头中都没有找到应用程序码，请留在引导加载程序中（不要跳转到任何应用程序）。

第4步。超时后：跳转到最新的应用程序。

- 重新定位 VTOR 表。
- PC 从固件中断表中获取值。

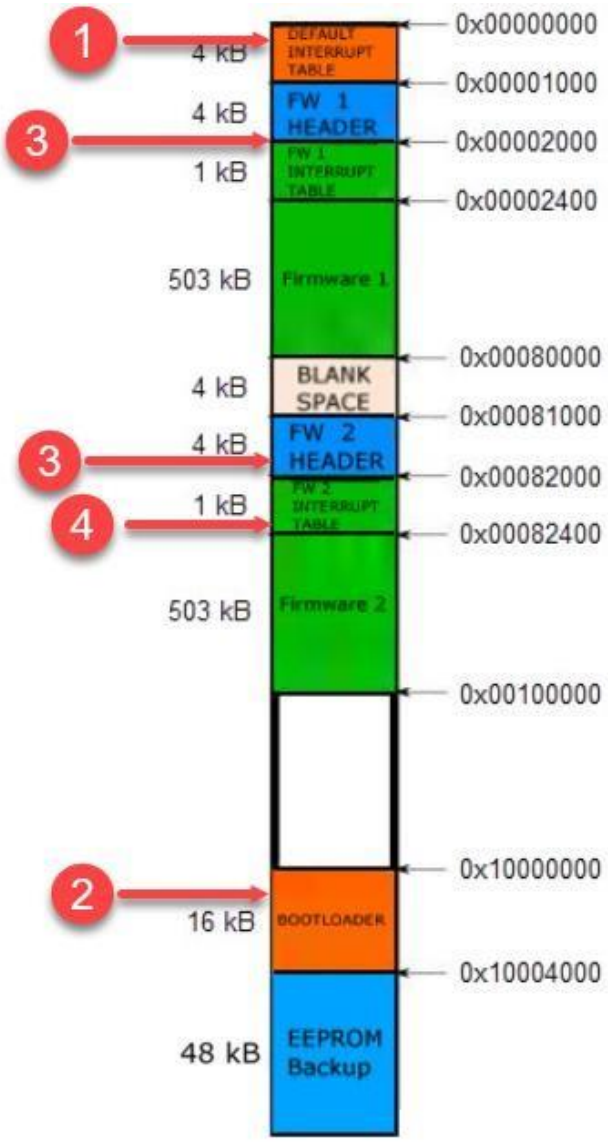

图14. S32K146引导过程

对于S32K146用例中的更新过程，可能有两种情况：

1. 主机在重置后但在超时过期之前请求的更新，因此在引导加载程序跳转到应用程序之前。
2. 在当前应用程序运行时，主机请求的更新。

场景（1）——在跳转到应用程序之前，重置后的更新请求

对于第一个场景，主机会在重置后的超时时间内向引导加载程序（边缘节点）触发更新请求。在这种情况下，处理更新的方式与处理 S32K144 的用例相同，通过 CANFD 接收新固件，将其加载到最旧固件的 pflash 块，当更新完成时，新固件头被签名和存储，而旧固件头被删除。最后，引导加载程序将跳转到新的应用程序中。图15显示了此场景的顺序。

第1步。重置后：获取 0x000000004 的 PC 值。

第2步。启动加载程序已启动。

第3步。引导加载程序搜索最老和最新的图像。

- 检查FW标题信息。
- 在标头末端找到应用程序键（0x55AA55AA）。
- 指定要更新的FW位置（最旧的）。

第4步。已收到更新触发器。

- 首先接收标头。
- 验证它是一个新版本。
- 开始在最旧的位置更新新的固件。

第5步。更新已完成。

- 更新新的固件标头。
- 删除/更新旧的固件标头。

第6步。跳转到新的应用。

- 重新定位 VTOR 表。
- PC 从新的固件中断表中获取值。

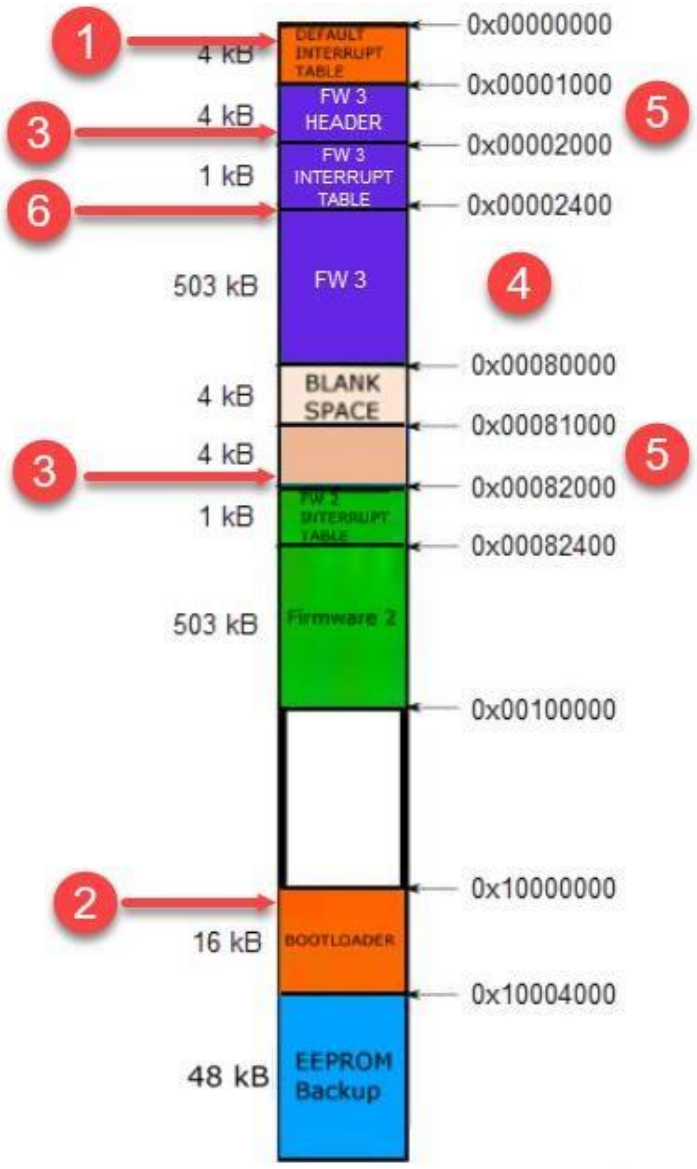

图15. S32K146更新过程（场景1）

场景（2）——在当前应用程序运行时的更新请求

在第二个更新场景中，当收到来自主机的更新请求时，最新的应用程序已经在运行。为了在不完全停止当前固件的执行的情况下处理更新过程，需要一种在引导加载程序和应用程序之间共享 CPU 处理时间的机制。这种机制的实现是灵活的，最终的应用程序将确定从应用程序中窃取的处理时间和完成新固件更新所需的总时间之间的适当平衡。一个重要的考虑是，应用软件需要具有引导加载程序的某种程度的识别，例如通过初始化与主机的通信接口，以及包括给引导加载程序控制的方式，以便处理来自主机的更新命令和数据。

对于此参考实现，每当从主机接收到包含固件数据的新 CANFD 消息时，执行都将从应用程序跳转到引导加载程序。数据被处理并存储在相应的 pflash 块中，执行跳转到停止的应用程序。加载程序专用条目地址与应用程序共享，以便在接收主机命令时知道跳转到哪里。应用程序引导加载程序的跳转循环将继续进行，直到更新过程完成。此时，引导加载程序使用 SRAM 内存中的状态变量向应用程序通知完成状态，然后应用程序触发软件重置以启动新的固件版本。图16 显示了此更新场景的总体过程。

第1步。重置后：获取 0x000000004 的 PC 值。

第2步。启动加载程序已启动。

第3步。引导加载程序搜索最老和最新的图像。

- 检查无线软件报头信息。
- 在标头末端找到应用程序键（0x55AA55AA）。
- 指定要更新的 FW 位置（最旧的）。

第4步。超时后：跳转到最新的应用程序。

- 重新定位 VTOR 表。
- PC 从固件中断表中获取值。

第5步。运行当前的应用程序。

- 继续操作，直到收到来自主机的消息。
- 当一个新的应用程序准备好时，请触发一个重置。

第6步。已收到来自主机的消息。

- 停止应用程序并跳转到引导加载程序。
- 首先接收标头，并验证它是否是一个新版本。
- 对于已收到的后续数据消息，
    - 继续更新最老的位置。
- 跳转到应用程序（**步骤5**）。

第7步。更新已完成。

- 更新新的固件标头。
- 删除/更新旧的固件标头。
- 通过共享变量通知当前应用程序，更新已完成。

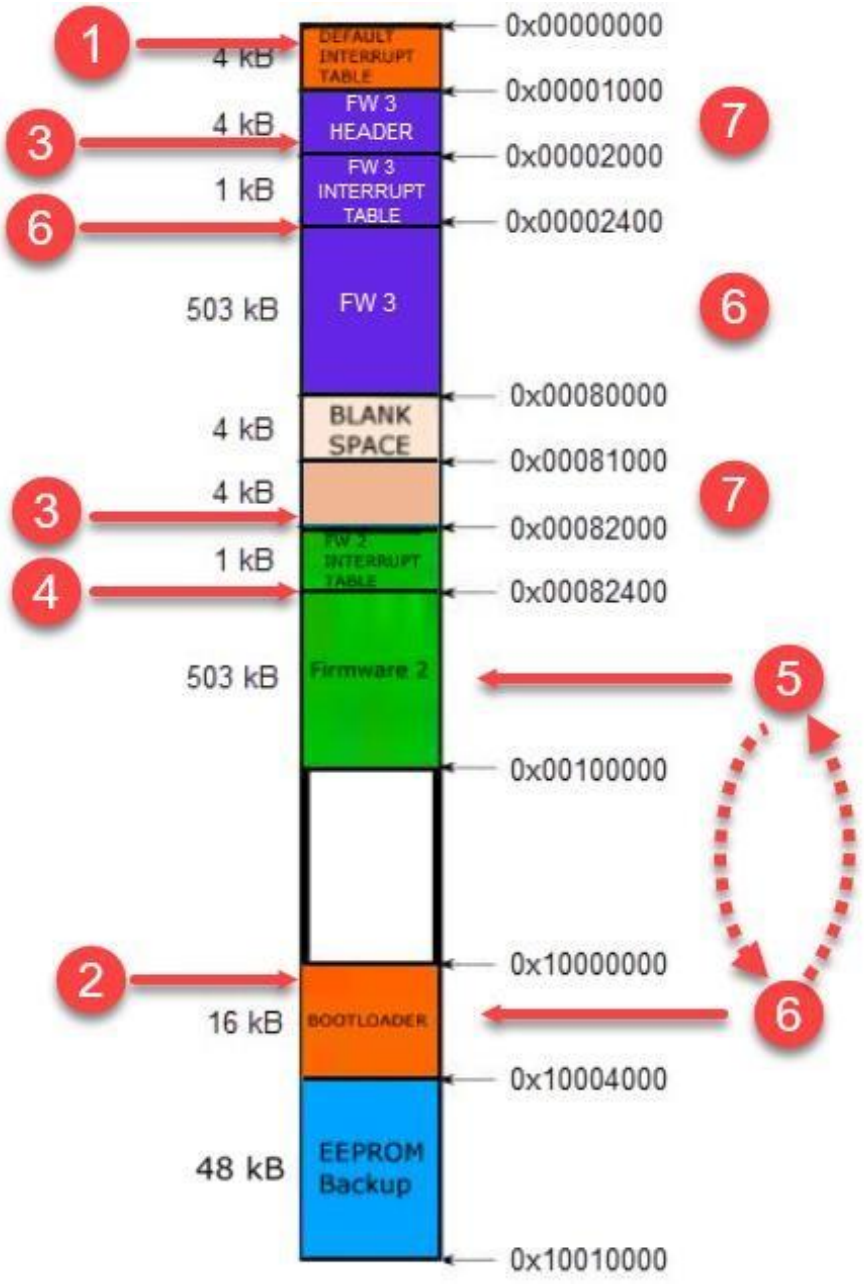

图16. S32K146更新过程（场景2）

6.3 使用位置独立代码（PIC）实现A/B交换

更新过程结束后，程序闪存中将有两个固件映像。图像位于闪存的不同物理地址中。为了能够在特定的地址执行代码，应该修改每个固件链接器，以指示其将被放置的位置。可能很难跟踪将为现场中的每个设备放置新固件的位置。因此，无论执行代码在程序闪存上的地址如何，能够执行该代码都是很方便的。

位置独立的代码意味着固件不顾其在内存中的绝对地址而执行，这意味着在代码中不会出现绝对的跳转。这允许开发固件而不需要修改链接器文件，并且固件可以从它所在位置的任何 pflash 闪存地址运行。

有编译器和工具使代码独立于位置，但有一些限制。例如，IAR 编译器提供了只读位置独立特性（ropi），如下图所示。在这种情况下，在固件的开发过程中，需要考虑以下各个方面。

- 不能使用 C++ 结构。
- 无法使用对象 attribute_ramfunc。
- 指针常数不能用其他常数、字符串文字或函数的地址进行初始化。但是，可写的变量可以在运行时初始化为常量地址。

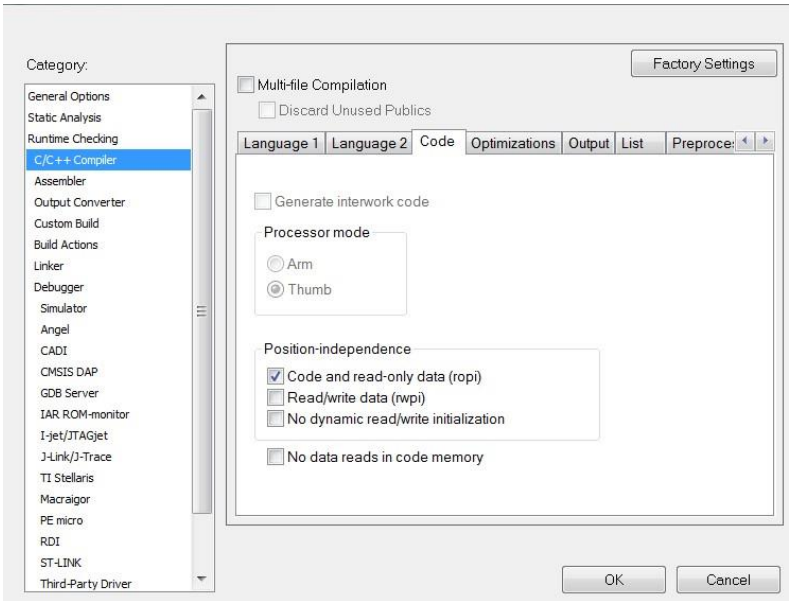

图17. IARropi功能的复选框

还需要考虑生成的位置独立的固件的中断表。即使固件确实包括相对跳转，但也不考虑跳转到中断表地址。中断表中的地址不受 ropi 的影响，并且与声明的地址保持相同在链接器文件中。

为了解决这个问题，中断位于链接器文件的默认地址 0x000000000 开始，然后在更新过程中必须向中断表的每个条目添加一个偏移地址，除了第一个条目是堆栈指针。地址偏移值取决于新固件在程序闪存中的位置。因此，当中断发生时，核心会跳转到程序闪存中应用程序固件放置的确切位置。

6.4 通信过程

通信更新过程包括六个主要步骤：

1. 更新触发器检测。
2. 标题信息的传输。
3. 应用程序数据的传输。
4. 更新触发器检测：在第一步中，接收到来自主机的消息以触发更新过程。
5. 标头信息传输：虽然在用例中，标头是用 flash 编写的最后一部分，但首先收到它，因为它包含固件版本。将此值与当前固件版本进行比较，以验证收到的固件是否比 flash 下载的固件更新。如果接收到的固件版本相同或更旧版本，则停止更新过程，并向主机发送错误消息。如果它是一个较新的版本，则更新过程将继续进行。
6. 应用程序数据传输：在最后一步中，将固件地址和数据共享到引导加载程序。每次引导加载程序收到一条有效的消息时，它都会回复一条确认消息。

下图显示了这个过程。

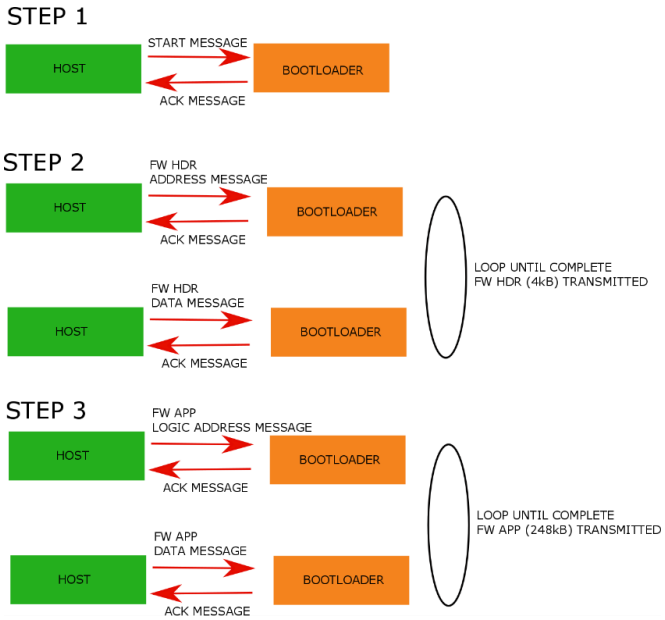

图18. 通信步骤

6.5 安全通信

在更新过程中，主机和引导加载程序之间共享的固件很容易受到攻击。一个可能的攻击可以嗅探网络，以窃取固件的知识产权。另一个可能的攻击是将未经授权的主机和未经授权的固件发送到未授权的引导加载程序，以操作最终的应用程序。为了避免窃听、复制攻击，并确保固件来自真实的源，必须在主机和引导加载程序之间建立安全的通信。对于 S32K1xx 设备，这可能是因为包含了CSEc模块。

侦听攻击，是指攻击者嗅探网络以获得共享的固件的情况。然后，攻击者可以使用这些数据，使用固件重新编程其他非授权模块或窃取知识产权。为了避免这种攻击，在更新过程中共享的数据将被加密。主机使用 AES-128CBC 算法对固件的数据进行加密，在发送到引导加载程序之前使用一个密钥。引导加载程序接收此数据，并使用相同的已知密钥对其进行解密。密钥在更新过程中不共享，而且它以前以脱机方式存储在每个设备上。

使用对数据的加密，可以保护固件的机密性。下一步要考虑的是被共享的消息的完整性和身份验证。未经授权的主机可以将数据发送到引导加载程序，以将数据下载到引导加载程序的闪存中，从而导致设备出现故障。要身份验证消息的来源，并且该消息尚未被修改，请使用CMAC值。

使用已知密钥和要共享的数据从主机中生成每个消息的CMAC值。当引导加载程序接收到具有固件数据的新消息时，它使用与主机相同的键计算接收到的数据的CMAC值。在引导加载器计算出CMAC后，它会将其与接收到的CMAC进行比较。如果CMAC验证正确，则收到的消息被认为是有效的。如果CMAC值不同，则表示它要么不是来自授权的源，要么是数据已损坏。因此，该消息被认为是无效，数据不会下载到flash。

除了所收到的消息的保密性和真实性外，还应考虑消息的新鲜度，以避免回复攻击。潜在的攻击者可以将通过嗅探网络获得的相同消息多次发送给引导加载程序，从而导致设备不需要的功能。为了防止这种攻击，可以在引导加载程序和主机之间的每个消息中共享一个随机数。CSEc模块包括两个随机数生成器，可用于此目的。

CSEc随机数生成器是真随机数生成器（TRNG）和伪随机生成器（PRNG）。TRNG用于生成种子，然后用于PRNG使用CSEc命令生成随机数。随机数生成器应在每次重置后进行初始化，因此将使用新的随机种子。

对于更新过程，引导加载程序生成一个随机数，并与主机共享，以避免应答攻击。引导加载程序向主机发送一个普通的随机数字，然后主机接收这个数字，并使用CBC和一个已知的密钥对其进行加密。下次主机将数据发送到引导加载程序时，它将加密的随机号与固件数据和CMAC一起添加。然后，引导加载程序接收消息并解密随机数。如果解密的随机数与以前共享的随机数相同，则该消息新鲜有效，否则将认为该消息无效。

验证消息的新鲜度后，然后对消息进行解密和CMAC验证。

总之，主机和引导加载程序之间的安全通信包括三个主要方面：

- 加密数据以保护知识产权。
- CMAC生成，以验证源的真实性和数据的完整性。
- 随机数生成，以防止应答攻击和检查源的真实性。

下图显示了主机和引导加载程序之间的安全通信过程。

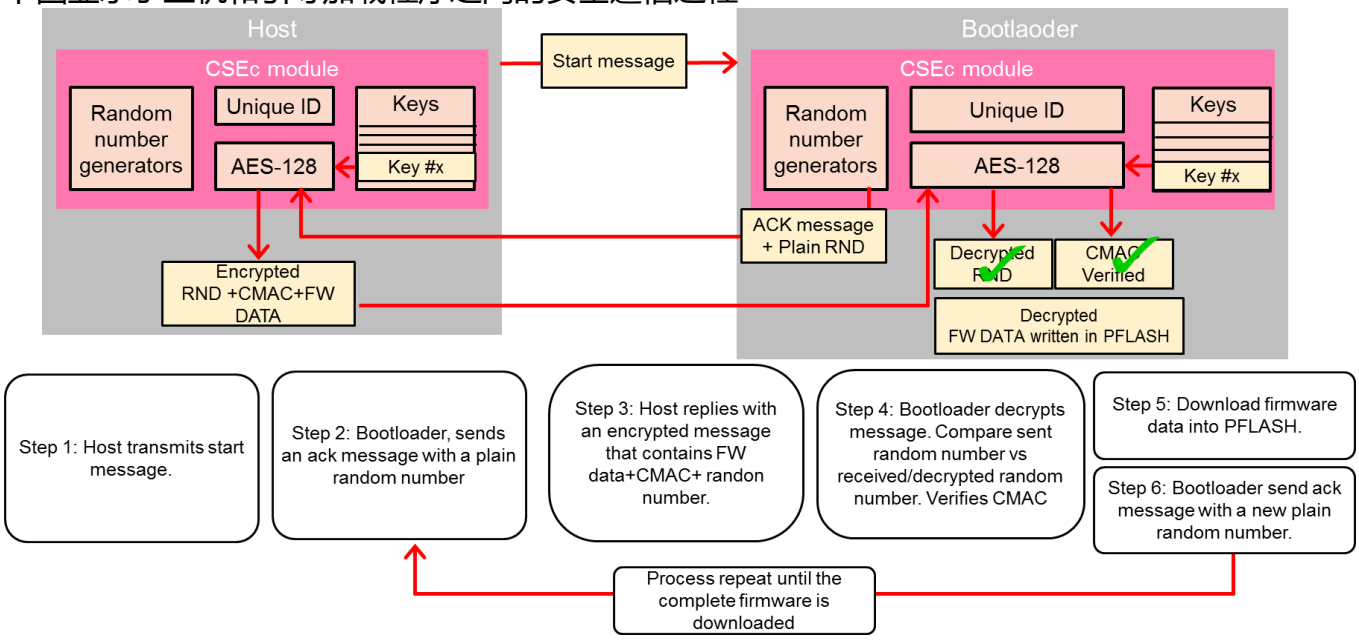

图19. 安全通信步骤

6.6 用例实现

6.6.1 引导加载程序的固件

使用评估板（EVB）和 S32Studin S32K144 和（S32DS）IDE 实现了引导加载程序。实现的引导加载器每次在微控制器中重置时执行。重置后，引导加载程序初始化其时钟和通信配置，并检查最老固件所在的程序闪存物理区域。

在 S32K144 的情况下，引导加载器等待4秒以接收一个触发器以启动更新过程，在这种情况下，触发器是一个特定的消息。如果触发器发生，更新过程将通过 CANFD 通信开始。如果4秒后没有触发，引导加载程序将把时钟和通信外设配置恢复到默认值，并跳转到最新的位置位于固件的位置。要检测到触发消息，需要重置S32K144，因此4秒的周期重新开始。

对于S32K146，引导加载器只等待2秒钟来接收一个触发器，然后恢复时钟和通信外设，最后跳到最新的应用程序。在当前固件执行时，可以从网关发送触发消息以启动更新过程。为此，应用程序反过来应该初始化通信外围设备，以便它准备好从主机接收消息。

引导加载程序被放置在弹性存储器区域的前16kB中。S32K144或S32K146以前被分区为EEPROM备份，其中保留16kB用于数据闪存（引导加载程序执行），而其他48kB用于EEPROM数据备份和CSEc的密钥存储。

下图显示了S32K144引导加载程序的链接器文件中指定的内存区域。

S32K1xx_flexnvm.ld

```
/* Specify the memory areas */
MEMORY
{
  /* Flash */
  m_interrupts          (RX) : ORIGIN = 0x00000000, LENGTH = 0x00000400
  m_flash_config        (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
  m_text                (RX) : ORIGIN = 0x10000000, LENGTH = 0x00004000  /* 16KB for Bootloader*/

  /* SRAM_L */
  m_data                (RW) : ORIGIN = 0x1FFF8000, LENGTH = 0x00008000

  /* SRAM_U */
  m_data_2              (RW) : ORIGIN = 0x20000000, LENGTH = 0x00007000
}
```

图20. S32K144引导加载程序链接器文件

在S32K146引导加载程序用例中，除了为引导加载器保留数据闪存空间外，还保留4个字节的空间来存储加载器入口点，以便运行的应用程序在接收更新命令时知道跳到哪里。此外，在内存中保留4字节的空间，以便在加载程序和应用程序之间共享状态变量。下图显示了S32K146引导加载程序的链接器文件中的内存组织。

S32K1xx_flexnvm.ld

```
/* Specify the memory areas */
MEMORY
{
  /* Flash */
  m_interrupts          (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000400
  m_flash_config        (RX)  : ORIGIN = 0x00000400, LENGTH = 0x00000010
  m_text                (RX)  : ORIGIN = 0x10000000, LENGTH = 0x00004000 - 4 /* 16KB DFlash for loader */
  m_shared_flash        (R)   : ORIGIN = 0x10003FFC, LENGTH = 0x00000004     /* 4 bytes for loader entry function pointer */

  /* SRAM_L */
  m_data                (RW)  : ORIGIN = 0x1FFF0000, LENGTH = 0x00010000

  /* SRAM_U */
  m_data_2              (RW)  : ORIGIN = 0x2000B000, LENGTH = 0x00004000 -4 /* 16KB SRAM for loader */
  m_shared_ram          (RW)  : ORIGIN = 0x2000EFFC, LENGTH = 0x00000004    /* 4 bytes for status variable shared with application */
}
```

图21. S32K146引导加载程序链接器文件

引导加载程序固件组织在以下几层中。

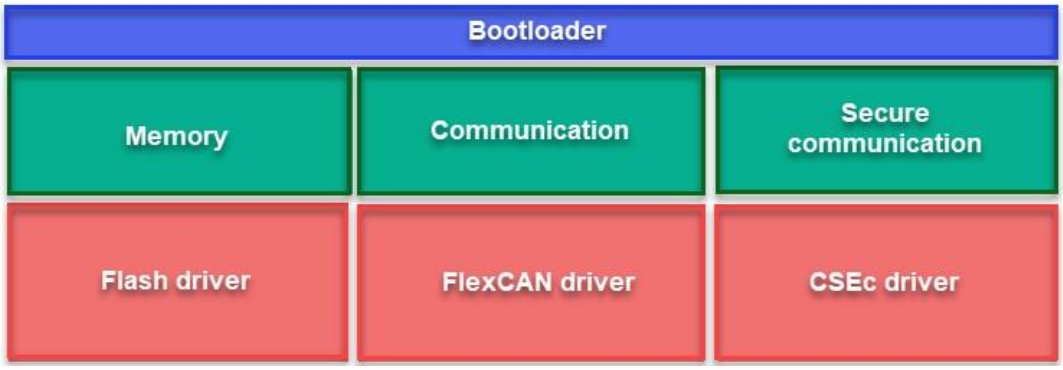

图22. 引导加载程序固件层

内存：引导加载程序和低级闪存驱动程序之间的内存抽象接口，以启动写和擦除等操作。

闪存驱动程序：S32K1xx单片机中的闪存控制器的低级别驱动程序，用于擦除/写/验证操作。

通信：引导加载程序和CANFD通信外围设备之间的抽象接口，以处理来自主机的数据的接收和确认消息的传输。

FlexCAN驱动程序：S32K1xx设备中FlexCAN通信外围设备的低级驱动程序，通过CANFD总线发送和传输数据。

安全通信：引导加载程序使用的安全接口层来请求数据的加密/解密，并验证接收到的消息的真实性和完整性，利用安全外围设备。

CSEc驱动程序：使用S32K1xx中的CSEC外围设备来处理加密/解密操作和CMAC计算。

6.6.2 CANFD 通信

引导加载程序支持CANFD通信协议，以执行更新过程。CANFD标准比特率为500kbps，快速比特率为2Mbps。

传输四种类型的消息，每种消息都具有关联的ID。

- 启动更新过程。
- 固件/标头数据的地址。
- 固件或标头的数据。
- 确认/错误的信息。

启动消息，触发更新过程，它的ID为0x200，其有效负载为4个字节，值为0x15151515。地址消息，传输固件的逻辑地址或头地址，其ID为0x100，有效负载大小为4B。数据消息的末尾表示固件或头的所有数据都已发送，ID与地址消息相同，但有效负载包含0x53535353。固件或报头数据消息，包含将下载到flash的数据，其ID为0x300，有效负载大小为32B。

这些消息从主机传输到引导加载程序，但引导加载程序也会发送确认或错误的消息。确认消息，表示上次消息已正确接收，其ID为0x400，有效负载大小为4B，值为0x04040404。消息错误，表示收到消息时出现错误，或收到的固件版本不是最新版本，ID也为0x400，有效负载大小为4B，值为0x555555555。下表总结了消息ID和有效负载。

表2. 通信信息

消息	ID	有效载荷的大小	有效载荷	方向	产品描述
开始时间	0x200	4	0x15151515	主机->引导加载程序	触发器的更新过程
通讯地址	0x100	4	固件标头的地址或固件逻辑地址 数据结束：0x53535353	主机->引导加载程序	包含fwhdr信息的地址或fw应用程序的逻辑地址。包含数据的结束部分载荷。
数据	0x300	32	固件标头或固件应用程序数据。	主机->引导加载程序	包含固件标头或固件应用程序数据，即下载到pflash。
Ack	0x400	4	确认载荷：0x04040404 错误载荷：0x55555555	引导加载程序->主机	包含已确认的载荷。 包含错误的载荷。

6.6.3 安全的CANFD通信

对于主机和引导加载程序之间的安全通信实现，使用了CANFD，这是由于每条消息的有效负载增加到64B。对于64B的有效载荷，CMAC可以发送16B，随机数可以发送16B，而还有32B可用于发送数据。对于上表中提到的消息，将添加有效负载数据，以包括CMAC和随机数值。

当引导加载程序接收到来自主机的开始消息时，如果这正确，将向主机回复一条确认消息。除了确认值的4B外，还增加了随机数的16B，以避免应答攻击。确认信息总共有20B个长。下图显示了确认消息的组织方式。

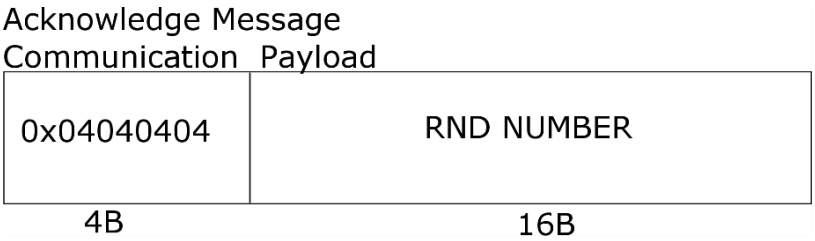

图23. 确认信息

在该地址消息中，除了用于传输该地址的4B之外，引导加载程序还期望接收以前发送的随机数的16B。因此，消息的大小会增加到20B。由于这20个数据被加密，并且在16B的数据包中进行加密，因此需要传输32B。剩下的12个B用0s填充。当引导加载程序收到此消息时，它会解密数据以比较随机数并验证源的真实性。无论在地址消息中发送数据有效负载结束或发送地址，这都适用。下图显示了此消息的组织结构。

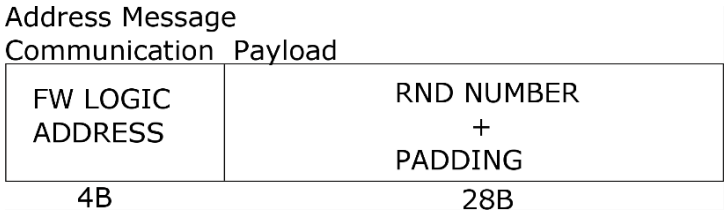

图24. 地址信息

引导加载程序希望在数据消息中接收到64B的有效负载。该64B的数据由固件数据的32B、为固件数据的32B计算的CMAC值的16B和随机数的16B组成。完整的有效负载已被加密。当收到时，数据消息，它解密它比较随机数，并验证真实性。然后比较CMAC值。如果随机数和CMAC都有效，则将数据下载到flash中。如果不匹配，则错误消息将从引导加载程序传输到主机。下图显示了此消息的组织结构。

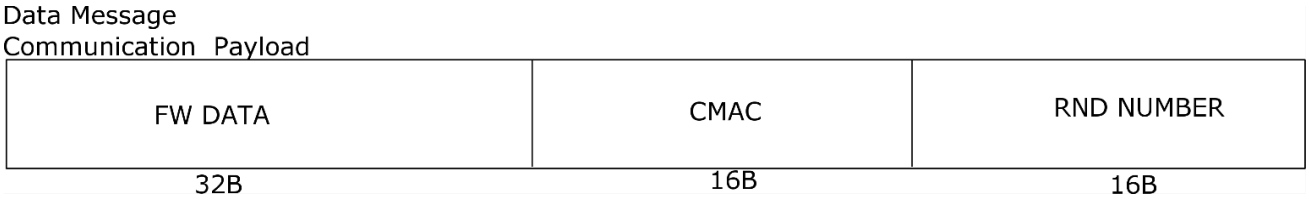

图25. 数据信息

在本例中，可以使用单个密钥来计算CMAC并加密所有消息，但也可以使用多个密钥。一个密钥可用于计算CMAC，而一个其他密钥或密钥可用来加密消息的数据。在这种情况下使用的密钥被加载到RAM中，但它可以存储在灵活的内存安全区域。此键的值为0xc13dd10dbfb5e142316c6d5c2e903ec9。

7 演示程序

要验证a/b交换用例，请使用具有CANFD通信的主机/网关设备。这两种用例的软件组织如下所示：

- S32K144用例：1xS32K144主机/网关项目和1xS32K144引导加载程序项目。
- S32K146用例：1xS32K146主机/网关项目和1xS32K146引导加载程序项目。

主机微控制器在其程序闪存中存储了两个应用程序，固件v1和固件v2，它们用于使用引导加载程序对设备进行编程。用于传输固件的主机的程序代码正在从弹性内存中执行。此程序发送一个开始消息，并等待引导加载程序确认。当接收到时，发送固件头，然后它传输在自己的程序闪存中存储的固件。下图显示了S32K144和S32K146网关项目在闪存中分配程序的方式。

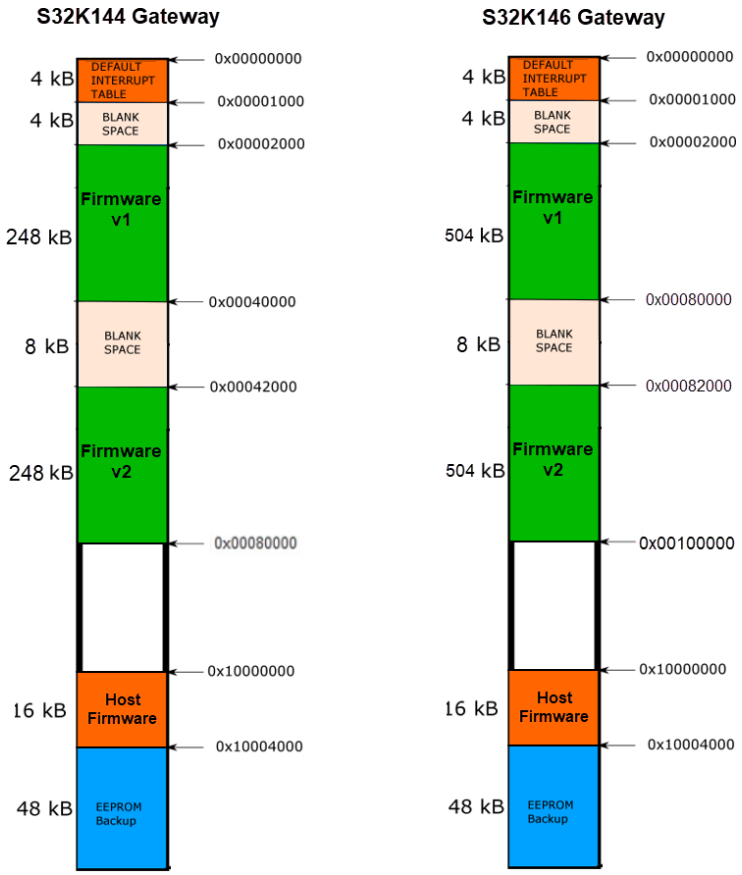

图26. 数据信息

要从主机启动传输，将按SW2或SW3来发送存储在主机闪存中的两个固件版本中的一个。当按下任何一个按钮时，传输就开始启动。当固件传输完成后，主机将准备通过按SW2或SW3发送另一个固件。为了将这两个应用程序存储在程序闪存中，使用S32DS中的主机/网关项目，为每个应用程序固件生成一个bin文件。S32DS项目的链接器被修改，为这些二进制文件分配闪存。

应用程序固件使用IAR IDE和位置独立特性开发的。固件版本被标识为V1和V2。这两个固件都包括从定时器中断内部切换LED（红色或蓝色LED，取决于固件）。该固件还通过LPUART接口将消息打印到PC控制台，使用EVB板上的OpenSDA接口，该接口枚举为虚拟COM端口。打印的消息基本上指示LED应用程序正在运行、应用程序版本和应用程序运行的程序闪存区域。区域A对应下程序闪存区域，而区域B对应上程序闪存区域。对于S32K144，这两个区域（A/B）只是来自同一pflash读取分区的两个不同的存储空间，而对于S32K146，这两个区域被映射到两个可用的pflash读取分区。S32K146用例的应用程序固件在网关触发新的更新过程时额外打印“progress...中的Update”消息，最后在更新完成时打印“新固件版本就绪，重置device...”消息。

下图显示了每个演示用例的设置。

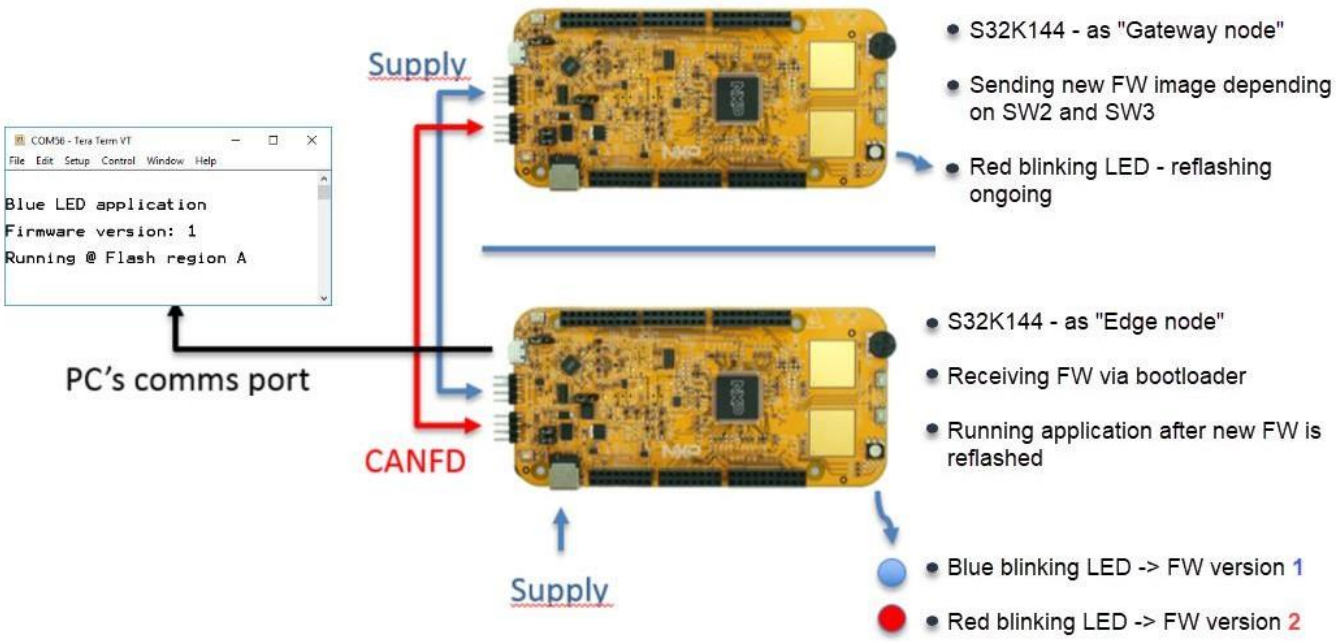

图27. S32K144用例演示概述

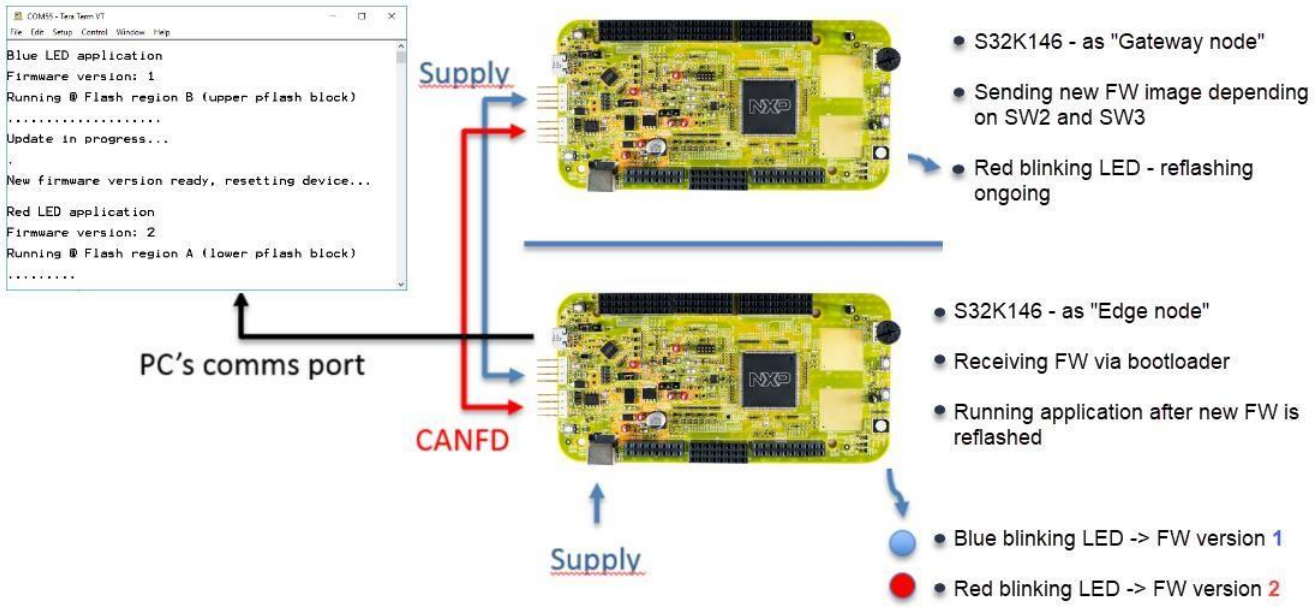

图28. S32K146用例演示程序概述

7.1 设置演示程序

本节介绍每个引导加载程序用例演示的设置。相应的S32设计工作室项目和IAR应用程序项目作为一个软件包与此应用程序说明一起包含在内。

7.1.1 设置S32K144引导加载程序演示

工作区OTA_S32K144_Use_Case_WS包括以下项目：

- S32K144_Memory_Partition：此项目设置了S32K144的灵活内存，分别为48KB的EEPROM备份，代码为16KB。
- S32K144_FOTA_Bootloader：这个项目是S32K144边缘节点微控制器的引导加载程序
- S32K144_FOTA_Gateway：此项目用于充当主机/网关的S32K144设备。该项目包括预编译的应用程序固件的两个二进制文件（v1/v2）。

需要使用以下硬件：

- 2xS32K144EVB电路板。
- 4xfemale-to-femal跨接电缆。
- 1x微型USB电缆。
- 1x12V电源。

按照下一步步骤安装和运行演示：

1) 打开S32DS，选择文件夹OTA_S32K144_Use_Case_WS作为工作区。

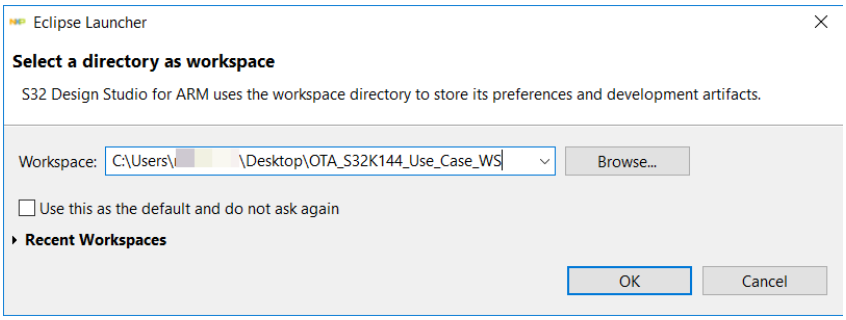

图29. 打开S32K144工作区

2) 导入、构建并将S32K144_Memory_Partition项目下载到每个S32K144EVB，将每个设备的灵活内存分区为48KB的EEPROM备份，16KB的代码。由网关和引导加载程序执行的代码将在flex内存的16kB区域上运行。

图30. 构建和下载S32K144内存分区项目

3) 选择一个S32K144EVB作为网关设备。导入、构建和下载S32K144_FOTA_Gateway项目到此EVB。

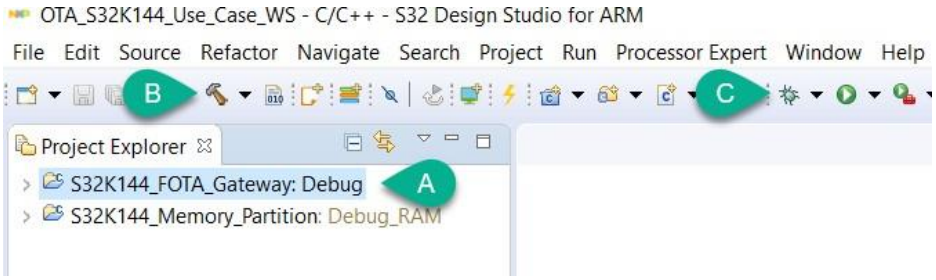

图31. 构建和下载S32K144网关项目

4) 导入、构建和下载S32K144_FOTA_Bootloader到另一个S32K144EVB。

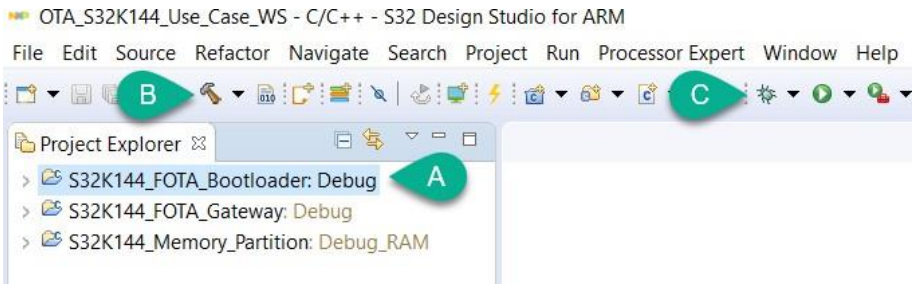

图32. 构建和下载S32K144引导加载程序项目

5) 将所需的电路板连接设置为下一个：

- 使用CAN信号的跨接线和共享的VBAT电源连接它们之间的电路板。
- 在两个电路板中，将跳线J107的位置更改为1-2。这是为了使用外部电源为电路板供电。
- 将12V电源连接到其中一个板（任何板）。两个电池板上的LEDD3都应该亮着。
- 将micro-USB电缆连接到边缘节点EVB（引导加载程序）中的microUSB连接器（J7），并将另一端连接到PC。LED D2应该亮着。

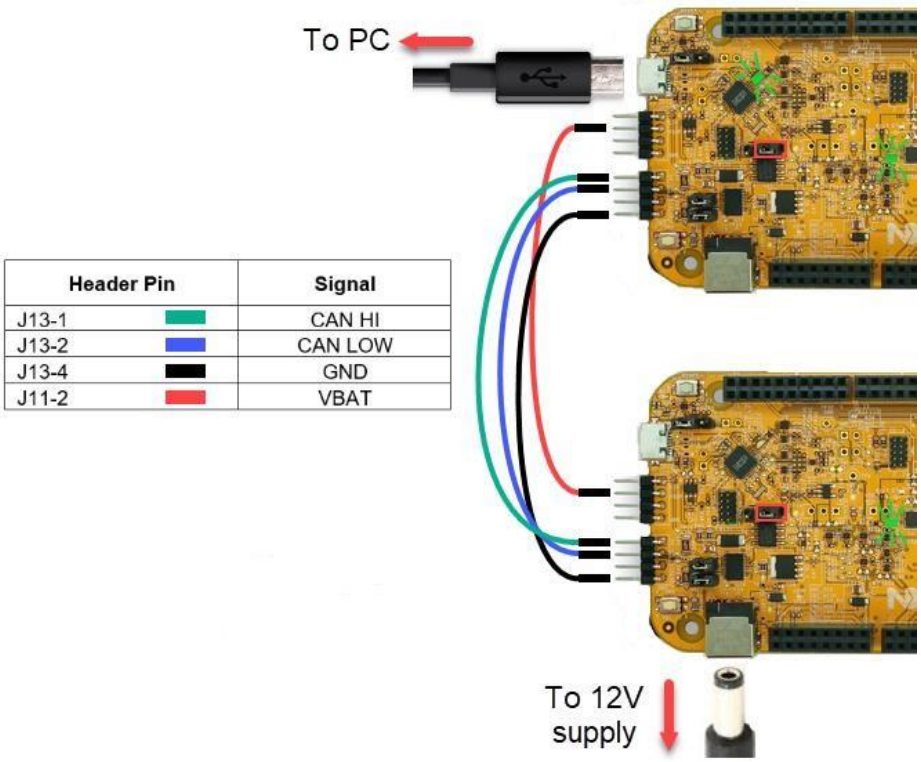

图33. S32K144引导加载程序演示设置

6) 打开串行终端，选择OpenSDA虚拟COM，并将终端设置为9600。
7) 通过按重置按钮SW5来重置边缘节点EVB（引导加载程序板）。
8) 在4秒超时结束之前，按网关板中的SW2（蓝色LED、v1固件）或SW3（红色LED、v2固件）。
9) 更新完成后，边缘节点中固件版本对应的LED将开始闪烁，终端将显示新的固件信息。

7.1.2 设置S32K146引导加载程序演示

工作区OTA_S32K146_Use_Case_WS包括以下项目：

- S32K146_Memory_Partition：此项目设置了S32K146的灵活内存，分别为48KB的EEPROM备份，并为16KB的代码。
- S32K146_FOTA_Bootloader：这个项目是S32K146边缘节点微控制器的引导加载程序。
- S32K146_FOTA_Gateway：此项目是针对作为主机/网关的S32K146设备进行的。该项目包括预编译的应用程序固件的两个二进制文件（v1/v2）。

需要使用以下硬件：

- 2xS32K146EVB电路板。
- 4xfemale-to-female跨接电缆。
- 1x微型USB电缆。
- 1x12V电源。

按照下一个步骤来安装和运行演示程序：

1) 打开S32DS，选择文件夹OTA_S32K146_Use_Case_WS作为工作区。

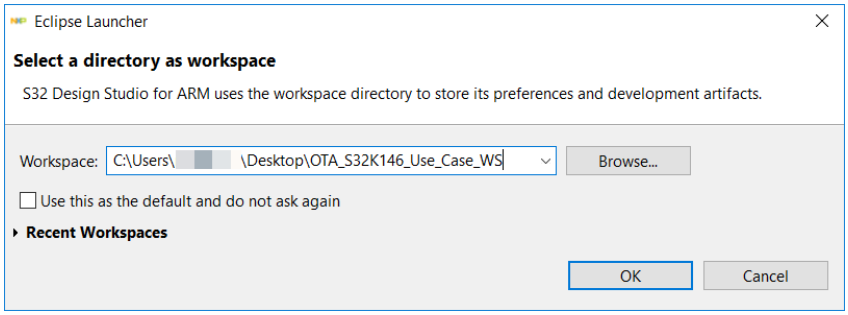

图34. 打开的S32K146工作区

2) 导入、构建并将S32K146_Memory_Partition项目下载到每个S32K146EVB，将每个设备的flex内存分区48KB的EEPROM备份，16KB的代码。由网关和引导加载程序执行的代码将在flex内存的16kB区域上运行。

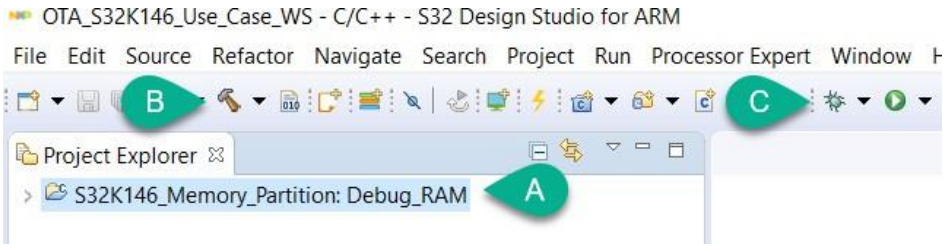

图35. 构建和下载S32K146内存分区项目

3) 选择一个S32K146EVB作为网关设备。导入、构建和下载S32K144_FOTA_Gateway项目到此EVB。

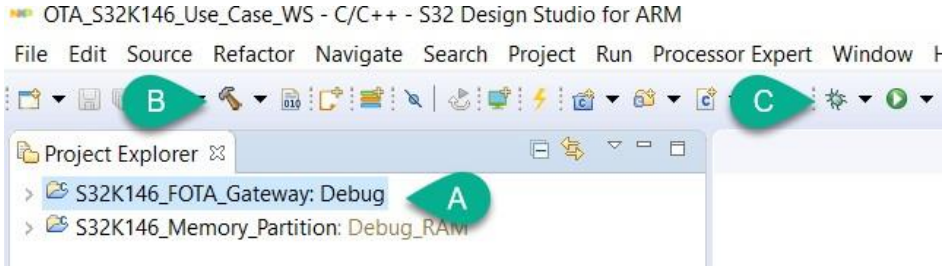

图36. 构建和下载S32K146网关项目

4) 导入、构建和下载S32K146_FOTA_Bootloader到另一个S32K146EVB。

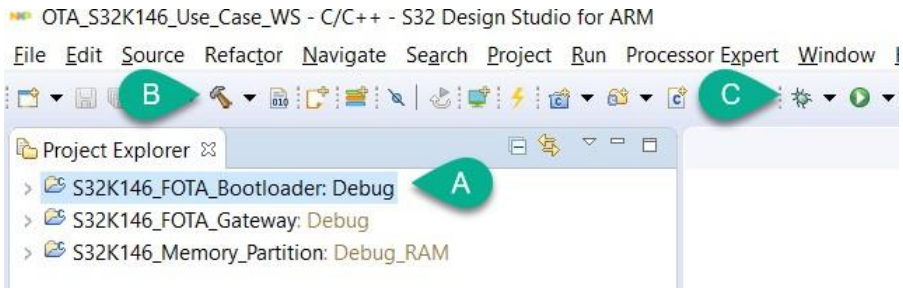

图37. 构建和下载S32K146引导加载程序项目

5) 将所需的电路板连接设置为下一个：

- 使用CAN信号的跨接线和共享的VBAT电源连接它们之间的电路板。
- 在两个电路板中，将跳线J107的位置更改为1-2。这是为了使用外部电源为电路板供电。
- 将12V电源连接到其中一个板（任何板）。两个电池板上的LEDD3都应该亮着。

- 将微型USB电缆连接到边缘节点EVB（引导加载程序）中的微型USB连接器（J7），并将另一端连接到PC。LEDD2应该亮着。

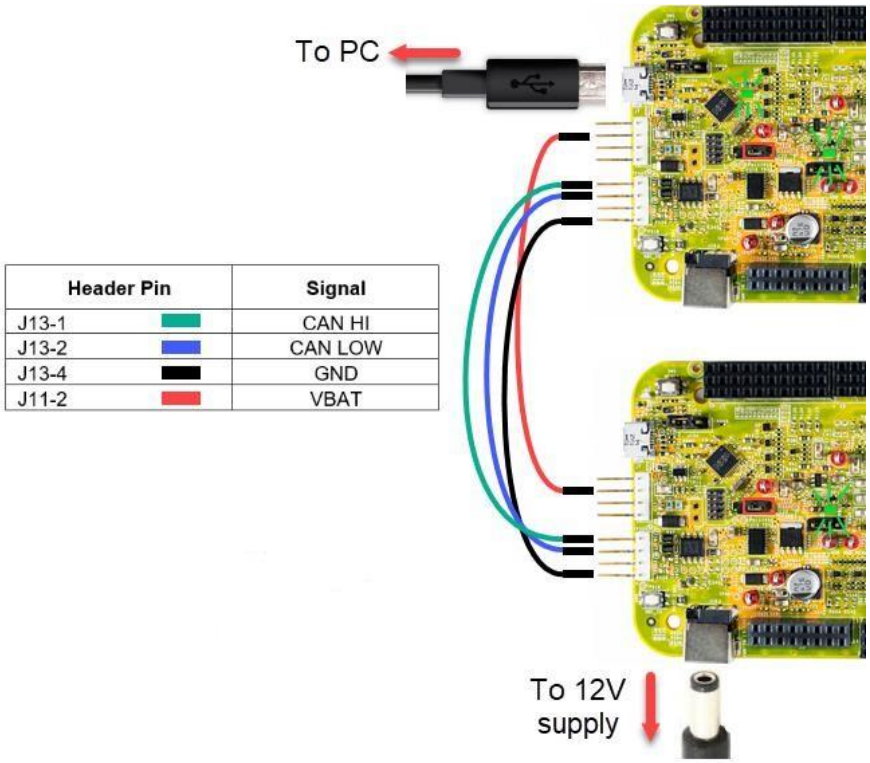

图38. S32K146引导加载程序演示设置

6) 打开串行终端，选择OpenSDA虚拟COM，并将终端设置为9600。

7) 按网关板中的SW2（蓝色LED、v1固件）或SW3（红色LED、v2固件）。

8) 更新完成后，边缘节点中固件版本对应的LED将开始闪烁，终端将显示新的固件信息。

8 参考资料

- S32K系列参考资料手动操作手册
- 定位独立的代码与 IAR
- 使用链接器将一个二进制文件合并到一个S32设计工作室项目中的命令

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

Document Number: AN12323
Rev. 0
12/2018