嵌入式微控制器应用中的无线(OTA)更新：设计权衡与经验教训

作者：Benjamin Bucklin Brown

摘要

许多嵌入式系统部署在操作人员难以或无法接近的地方。物联网(IoT)应用尤其如此，这些应用通常大量部署并且电池寿命有限。实例包括监控人员或机器健康状况的嵌入式系统。这些挑战加上快速迭代的软件生命周期，导致许多系统需要支持无线(OTA)更新。OTA更新用新软件替换嵌入式系统的微控制器或微处理器上的软件。虽然很多人非常熟悉移动设备上的OTA更新，但在资源受限的系统上设计和实施会带来许多不同的挑战。本文将介绍针对OTA更新的若干不同软件设计，并讨论其优缺点。我们将了解OTA更新软件如何利用两款超低功耗微控制器的硬件特性。

构建模块

服务器和客户端

OTA更新用新软件替换器件上的当前软件，新软件以无线方式下载。在嵌入式系统中，运行此软件的器件通常是微控制器。微控制器是一种小型计算器件，其存储器、速度和功耗均很有限。微控制器通常包含微处理器（核心）和用于执行特定操作的数字硬件模块（外设）。工作模式下典型功耗为30 µA/MHz至40 µA/MHz的超低功耗微控制器是此类应用的理想选择。使用这些微控制器上的特定硬件外设并将其置于低功耗模式，是OTA更新软件设计的重要组成部分。图1显示了一个可能需要OTA更新的嵌入式系统实例。可以看到，一个微控制器与射频收发器和传感器相连，这可用在物联网应用中，利用传感器收集有关环境的数据，并利用无线收发器定期报告数据。系统的这一部分称为边缘节点或客户端，是OTA更新的目标。系统的另一部分称为云或服务器，是新软件的提供者。服务器和客户端利用无线收发器通过无线连接进行通信。

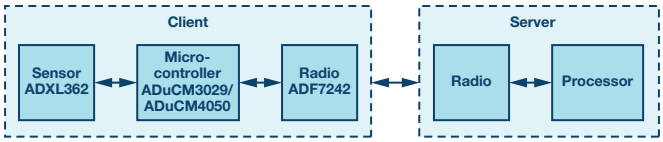


图1. 示例嵌入式系统中的服务器/客户端架构

何为软件应用程序?

OTA更新过程的大部分操作是将新软件从服务器传输到客户端。软件从源格式转换为二进制格式之后，作为一个字节序列进行传输。这个转换过程包括编译源代码文件（例如*c*、*cpp*），将其链接成一个可执行文件（例如*exe*、*elf*），然后将可执行文件转换为可移植的二进制文件格式（例如*bin*、*hex*）。概言之，这些文件格式包含一个字节序列，此字节序列需要存放在微控制器中存储器的特定地址。通常，我们将通过无线链路发送的信息概念化为数据，例如更改系统状态的命令或系统收集的传感器数据。就OTA更新而言，数据就是二进制格式的新软件。在很多情况下，二进制文件非常大，无法通过单次传输从服务器发送到客户端，这意味着需要将二进制文件放入多个不同的数据包中，此过程称为“分包”。为了更好地说明此过程，图2演示了软件的不同版本如何生成不同的二进制文件，从而在OTA更新期间发送不同的数据包。在这个简单例子中，每个数据包包含8字节数据，前4个字节表示客户端存储器中用来存储后4个字节的地址。

主要挑战

基于对OTA更新过程的这种高层次描述，OTA更新解决方案必须应对三大挑战。第一个挑战与存储器有关。软件解决方案必须将新软件应用程序组织到客户端器件的易失性或非易失性存储器中，以便在更新过程完成时可以执行它。解决方案必须确保将前一版

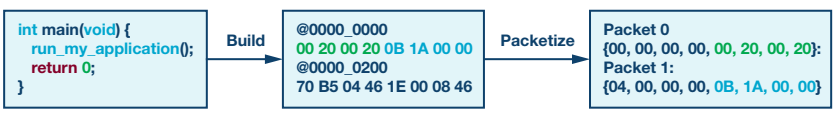


图2. 软件应用程序的二进制转换和分包过程

本的软件保留为后备应用程序，以防新软件出现问题。此外，当复位和断电重启时，我们必须让客户端器件的状态——例如当前运行的软件版本以及它在存储器中的位置——保持不变。第二大挑战是通信。新软件必须以离散数据包的形式从服务器发送到客户端，每个数据包都要放在客户端存储器中的特定地址。分包方案、数据包结构和数据传输协议必须在软件设计中考虑周全。最后一个主要挑战是安全性。当新软件以无线方式从服务器发送到客户端时，我们必须确保服务器是可信任方。这种安全挑战称为身份验证。我们还必须对新软件进行模糊处理以防被窃，因为其中可能包含敏感信息。这种安全挑战称为保密。安全性的最后一个要素是完整性，即确保新软件在通过无线方式发送时不会损坏。

第二阶段引导加载程序(SSBL)

了解引导序列

主引导加载程序是一种软件应用程序，永久驻留在微控制器的只读存储器中。主引导加载程序所在的存储区域称为信息空间，有时用户无法访问。每次复位都会执行该应用程序，一般完成一些必要的硬件初始化，并且可能将用户软件加载到存储器中。但是，如果微控制器包含片内非易失性存储器（如闪存），则引导加载程序不需要进行任何加载，只需将控制权转移到闪存中的程序即可。如果主引导加载程序不支持OTA更新，则必须有第二阶段引导加载程序。与主引导加载程序一样，SSBL会在每次复位时运行，但将实施OTA更新过程的一部分。此引导序列如图3所示。本节将说明为什么需要第二阶段引导加载程序，并解释如何设计这个应用程序是一个重要设计权衡。

经验教训：务必有一个SSBL

从概念上讲，省略SSBL并将所有OTA更新功能放入用户应用程序似乎更简单，因为这样的话，OTA过程可以无缝利用现有的软件框架、操作系统和设备驱动程序。图4显示了一个选择此方法的系统的存储器映射和引导序列。

应用程序A是部署在现场微控制器上的原始应用程序。此应用程序包含OTA更新相关软件，当服务器请求时，利用该软件可下载应用程序B。下载完成且应用程序B经过验证之后，应用程序A将对应用程序B的复位处理程序执行分支指令，以将控制权转移给应用程序B。复位处理程序是一小段代码，用作软件应用程序的入口点，并在复位时运行。在这种情况下，复位是通过执行一个分支来模拟，这相当于函数调用。这种方法有两大问题：

- 许多嵌入式软件应用程序采用实时操作系统(RTOS)，其允许将软件拆分为多个并发任务，每个任务在系统中具有不同的职责。例如，图1所示的应用程序可能有用于读取传感器的RTOS任务，对传感器数据运行某种算法的RTOS任务，以及与无线电接口的RTOS任务。RTOS本身始终处于活动状态，负责根据异步事件或特定的基于时间的延迟切换这些任务。因此，从RTOS任务分支到新程序是不安全的，因为其他任务会在后台继续运行。对于实时操作系统，终止某个程序的唯一安全方法是通过复位。

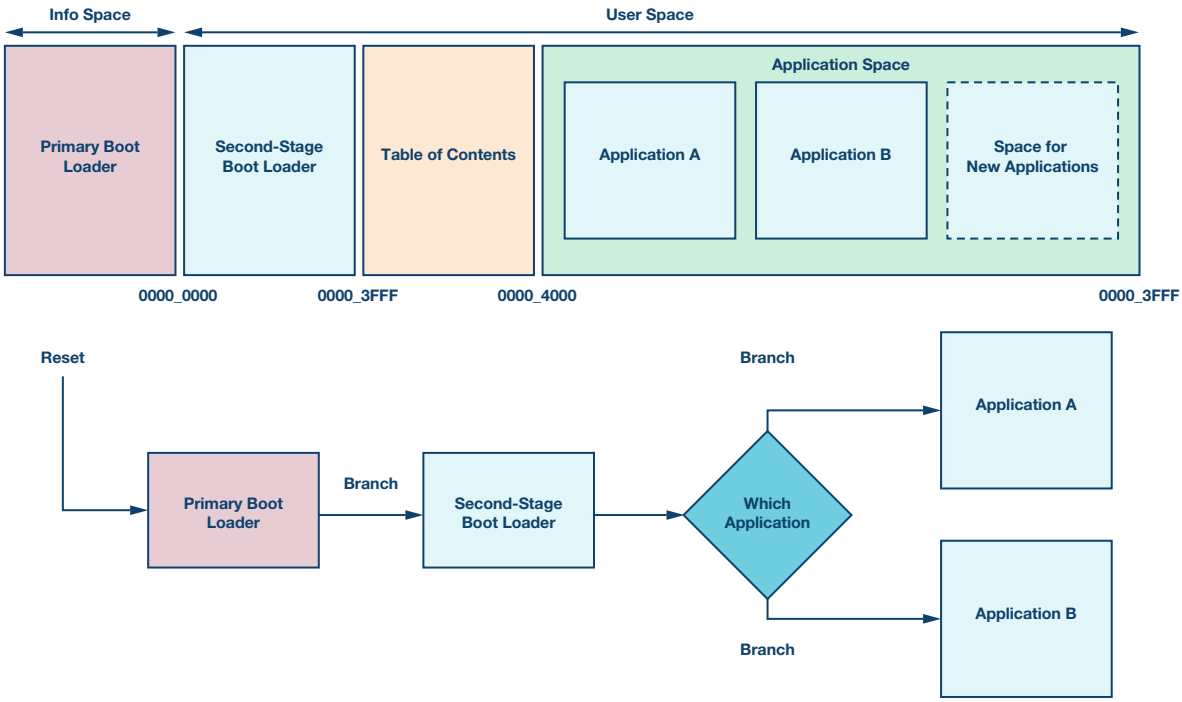


图3. 使用SSBL的存储器映射和引导流程示例

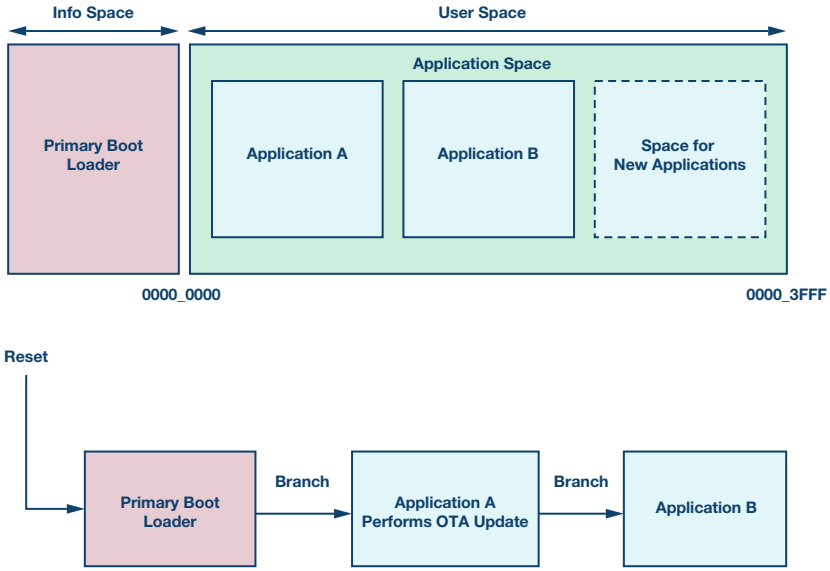


图4. 没有SSBL的存储器映射和引导流程示例

- 基于图4，上述问题的解决办法是让主引导加载程序分支到应用程序B而不是应用程序A。但在某些微控制器上，主引导加载程序总是运行具有中断向量表(IVT)的程序；IVT是应用程序的一个关键部分，描述中断处理函数，位于地址0。这意味着必须以某种形式重定位IVT，使其复位映射到应用程序B。如果在IVT重定位期间发生断电重启，则系统可能会处于永久破损状态。

将SSBL固定在地址0可以解决这些问题，如图3所示。SSBL不是RTOS程序，因此可以安全地分支到新应用程序。地址0处的SSBL的IVT永远不会重新定位，所以不必担心断电重启会将系统置于灾难性状态。

设计权衡：SSBL的作用

我们花了很多时间讨论SSBL及其与应用软件的关系，但SSBL程序有何作用？至少，该程序必须确定当前应用程序是什么（其开始位置），然后分支到该地址。微控制器存储器中各种应用的位置一般保存在目录(ToC)中，如图3所示。这是常驻内存中的一个共享区域，SSBL和应用软件均利用它来相互通信。当OTA更新过程完成时，新的应用程序信息会更新ToC。OTA更新功能的某些部分也可以被推送到SSBL。开发OTA更新软件时，确定推送哪些部分是重要的设计决策。上述最小SSBL将非常简单，易于验证，并且在应用程序的生命周期中很可能不需要修改。但是，这意味着每个应用程序都要负责下载和验证下一个应用程序。这可能导致通信协议栈、设备固件和OTA更新软件的代码重复。另一方面，我们可以选择将整个OTA更新过程推送到SSBL。在这种情况下，应用程序只需在ToC中设置一个标志以请求更新，然后执行复位。SSBL随后执行下载序列和验证过程。这将最大限度地减少代码重复并简化应用专用软件。然而，这会引入一个新的挑战，那就是可能需要更新SSBL本身（即更新更新代码）。最终，决定SSBL中放置哪些功能将取决于客户端器件的存储器限制、下载的应用程序之间的相似性以及OTA更新软件的可移植性。

设计权衡：缓存和压缩

OTA更新软件中的另一个关键设计决策是在OTA更新过程中如何组织存储器中传入的应用程序。微控制器上通常有两类存储器：非易失性存储器（例如闪存）和易失性存储器（例如SRAM）。闪存用于存储应用程序的程序代码和只读数据，以及其他系统级数据，例如ToC和事件日志。SRAM用于存储软件应用程序的可修改部分，例如非常数全局变量和堆栈。图2所示的软件应用程序二进制文件仅包含非易失性存储器中存在的程序的某些部分。在启动例程期间，应用程序将初始化属于易失性存储器的部分。

在OTA更新过程中，每次客户端器件从服务器收到一个包含该二进制文件一部分的数据包时，便会将其存储到SRAM中。该数据包可以是压缩的，也可以是未压缩的。压缩应用程序二进制文件的好处是文件会变小，从而要发送的数据包会减少，下载过程中存储数据包所需的SRAM空间相应地减小。这种方法的缺点是压缩和解压缩会增加更新过程的处理时间，并且必须在OTA更新软件中捆绑压缩相关代码。

新应用软件应该存放在闪存，但在更新过程中到达SRAM，因此OTA更新软件需要在更新过程中的某个时刻执行对闪存的写操作。暂时将新应用程序存储在SRAM中的操作称为缓存。概言之，OTA更新软件可以采取三种不同的缓存方法。

- 不缓存：每次包含新应用程序一部分的数据包到达时，便将其写入闪存中的目标位置。这种方案非常简单，可以最大限度地减少OTA更新软件中的逻辑数量，但要求完全擦除新应用程序对应的闪存区域。此方法会消磨闪存并增加开销。
- 部分缓存：保留一个SRAM区域用于缓存，当新数据包到达时，将其存储在该区域中。当该区域填满时，将数据写入闪存以清空该区域。如果数据包无序到达或新应用程序二进制文件中存在间隙，这种方案可能会变得很复杂，因为需要一种方法来将SRAM地址映射到闪存地址。一种策略是让缓存

充当闪存一部分的镜像。闪存被划分为若干称为页面的小区域，这是可供擦除的最小区域。得益于这种自然划分，一个好办法是在SRAM中缓存闪存的一页，当其填满或下一数据包属于其他页面时，便将该页写入闪存以清空缓存。

- 完全缓存：在OTA更新过程中将整个新应用程序存储在SRAM中，只有从服务器完全下载好新应用程序之后才将其写入闪存。这种方法克服了前述方法的缺点，写入闪存的次数最少，OTA更新软件无需复杂的缓存逻辑。但是，这会限制所下载新应用程序的大小，因为系统的可用SRAM量通常远小于可用闪存量。

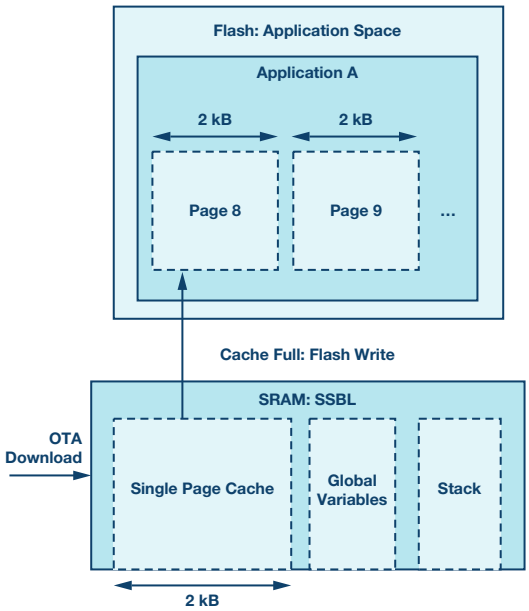


图5. 使用SRAM缓存闪存的一页

图5显示了OTA更新过程中的第二种方案——部分缓存，来自图3和图4的应用程序A所对应的闪存部分被放大，并且显示了用于SSBL的SRAM的功能存储器映射。示例闪存页面大小为2 kB。最终，此设计决策将取决于新应用程序的大小和OTA更新软件容许的复杂度。

安全和通信

设计权衡：软件与协议

OTA更新解决方案还必须解决安全和通信问题。如图1所示，许多系统会在硬件和软件中实现通信协议，以支持系统的正常（非OTA更新相关）操作，例如交换传感器数据。这意味着服务器和客户端之间已经建立了（可能是安全的）无线通信的方法。类似图1所示的嵌入式系统可以使用的通信协议有低功耗蓝牙®(BLE)或6LoWPAN等。有时候，这些协议支持安全性和数据交换，OTA更新软件在OTA更新过程中可以利用。

OTA更新软件中必须构建的通信功能量最终将取决于现有通信协议提供的抽象程度。现有通信协议具有用于在服务器和客户端之间发送和接收文件的工具，OTA更新软件可以简单地将该工具用于下载过程。但是，如果通信协议较为原始，只有发送原始数据的工具，那么OTA更新软件可能需要执行分包处理，并提供元数据和新应用程序二进制文件。这也适用于安全挑战。如果通信协议不支持，OTA更新软件可能要负责对无线保密发送的字节进行解密。

总之，在OTA更新软件中实施哪些功能，例如自定义数据包结构、服务器/客户端同步、加密和密钥交换等，将取决于系统的通信协议提供了什么内容以及对安全性和稳健性的要求。下一节将提出一个完整的安全解决方案，其解决了之前介绍的所有挑战，我们将展示如何在此解决方案中利用微控制器的加密硬件外设。

解决安全挑战

我们的安全解决方案需要让新应用程序以无线方式保密发送，检测新应用程序中的任何损坏，并验证新应用程序是从受信任的服务器而不是恶意方发送的。这些挑战可通过加密操作来解决。具体而言，该安全解决方案可以使用两种加密操作：加密和哈希处理。加密使用客户端和服务器共享的密钥（密码）来对无线发送的数据进行模糊处理。微控制器的加密硬件加速器可能支持的特定加密类型是AES-128或AES-256，具体取决于密钥大小。除了加密数据，服务器还可以发送一个摘要以确保没有损坏。摘要通过对数据包进行哈希处理来生成，这是一种用于生成唯一代码的不可逆数学函数。在服务器产生消息或摘要之后，如果其任何部分遭到修改，比如在无线通信期间有一位发生翻转，则客户端在对数据包执行相同的哈希函数处理并比较摘要时，会注意到此修改。微控制器的加密硬件加速器可能支持的特定哈希处理类型是SHA-256。图6显示了微控制器中的加密硬件外设的框图，OTA更新软件驻留在Cortex-M4应用层中。此图还显示了其支持将受保护密钥存储在外设中，OTA更新软件解决方案可以利用这一点来安全存储客户端密钥。

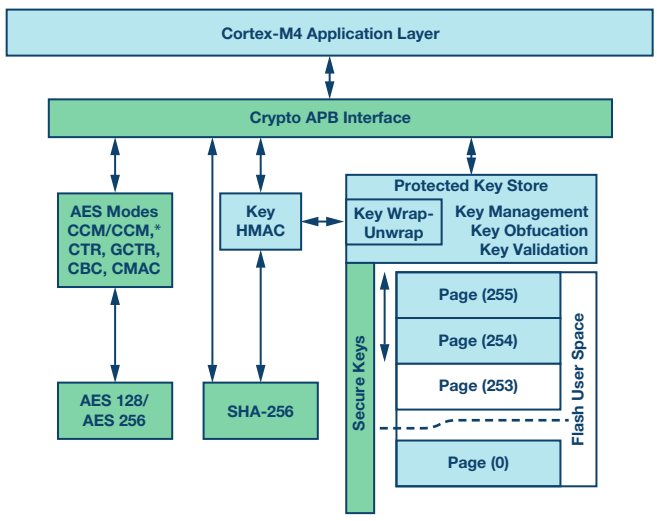


图6. ADuCM4050上的加密加速器的硬件框图

解决身份验证这一最终挑战的常见技术是使用非对称加密。对于此操作，服务器会生成一个公钥-私钥对。私钥只有服务器知道，客户端知道公钥。服务器使用私钥可以生成给定数据块的签名，例如要无线发送的数据包的摘要。签名被发送给客户端，后者可以使用公钥验证签名。这样，客户端就能确认消息是从服务器而不是恶意第三方发送的。此序列如图7所示，实线箭头表示函数输入/输出，虚线箭头表示无线发送的信息。

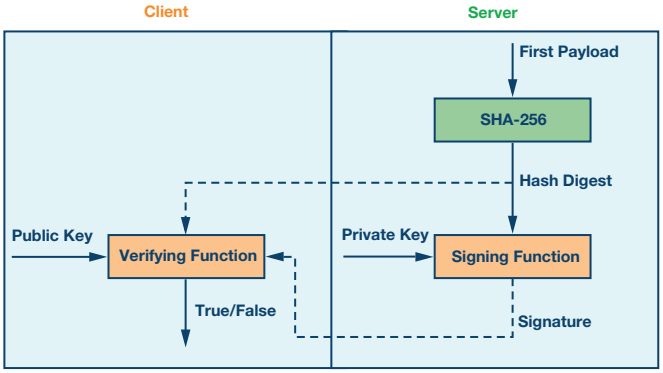
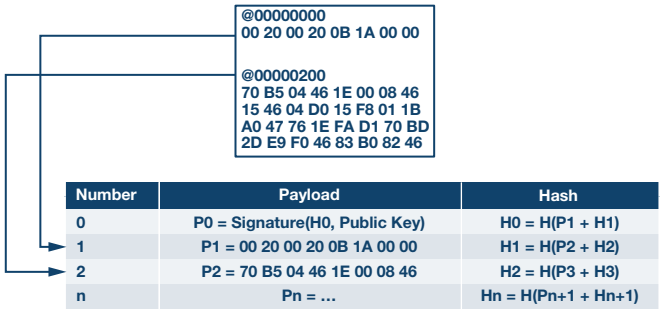


图7. 使用非对称加密验证消息

多数微控制器没有用于执行这些非对称加密操作的硬件加速器，但可以使用Micro-ECC等专门针对资源受限器件的软件库来实现。该库需要一个用户定义的随机数生成功能，这可以利用微控制器上的真随机数发生器硬件外设来实现。虽然这些非对称加密操作解决了OTA更新期间的信任挑战，但是会消耗大量处理时间，并且需要将签名与数据一同发送，这会增加数据包大小。我们可以在下载结束时使用最后数据包的摘要或整个新软件应用程序的摘要执行一次此检查，但如此的话，第三方将能把不受信任的软件下载到客户端，这不太理想。理想情况下，我们希望验证所收到的每个数据包都来自我们信任的服务器，而且没有每次都需要签名的开销。这可以利用哈希链来实现。

哈希链将本节讨论的加密概念整合到一系列数据包中，以便在数学上将它们联系在一起。如图8所示，第一个数据包（编号0）包含下一个数据包的摘要。第一个数据包的有效载荷不是实际的软件应用程序数据，而是签名。第二个数据包（编号1）的有效载荷包含二进制文件的一部分和第三个数据包（编号2）的摘要。客户端验证第一个数据包中的签名并缓存摘要H0以供以后使用。当第二个数据包到达时，客户端对有效载荷进行哈希处理并将其与H0进行比较。如果它们匹配，客户端便可确定该后续数据包来自可信服务器，而无需费力进行签名检查。生成此链的高开销任务留给服务器完成，客户端只需在每个数据包到达时进行缓存和哈希处理，确保到达的数据包完整无损并验明正身。



Number	Payload	Hash
0	P0 = Signature(H0, Public Key)	H0 = H(P1 + H1)
1	P1 = 00 20 00 20 0B 1A 00 00	H1 = H(P2 + H2)
2	P2 = 70 B5 04 46 1E 00 08 46	H2 = H(P3 + H3)
n	Pn = ...	Hn = H(Pn+1 + Hn+1)

图8. 将哈希链应用于数据包序列

实验设置

解决本文所述存储器、通信和安全设计挑战的超低功耗微控制器是ADuCM3029和ADuCM4050这些微控制器包含本文讨论的用于OTA更新的硬件外设，例如闪存、SRAM、加密加速器和真随机数发生器。这些微控制器的器件系列包(DFP)为在这些器件上构建OTA更新解决方案提供了软件支持。DFP包含外设驱动，以便为使用硬件提供简单灵活的接口。

硬件配置

为了验证本文讨论的概念，我们利用ADuCM4050创建了OTA更新软件参考设计。对于客户端，一个ADuCM4050 EZ-KIT®使用收发器子板马蹄形连接器连接到ADF7242。客户端器件如图9左侧所示。对于服务器，我们开发了一个在Windows PC上运行的Python应用程序。Python应用程序通过串行端口与另一个ADuCM4050 EZ-KIT通信，后者也以与客户端相同的配置连接一个ADF7242。但是，图9中右边的EZ-KIT不执行OTA更新逻辑，只是将从ADF7242接收到的数据包中继给Python应用程序。

图9. 实验硬件设置

软件组件

软件参考设计对客户端器件的闪存进行分区，如图3所示。主要客户端应用程序具有非常好的移植性和可配置性，以便其他方案或

其他硬件平台也可以使用。图10显示了客户端器件的软件架构。请注意，虽然我们有时将整个应用程序称为SSBL，但在图10中，并且从现在开始，我们在逻辑上将真正的SSBL部分（蓝色）与OTA更新部分（红色）分开，因为后者不一定需要完全在上述应用程序中实现。图10所示的硬件抽象层使OTA客户端软件可移植并独立于任何底层库（以橙色显示）。

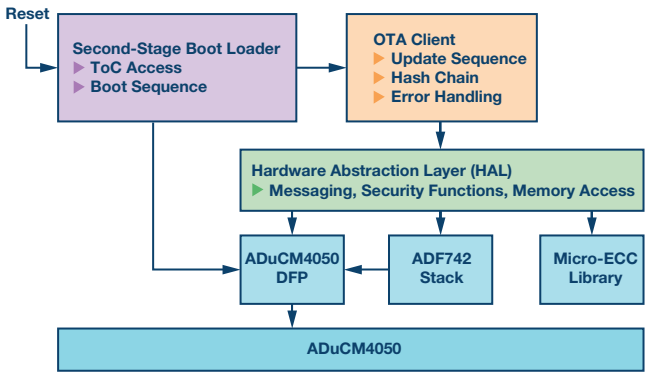


图10. 客户端软件架构

软件应用程序实现图3中的引导序列（一个用于从服务器下载新应用程序的简单通信协议）和哈希链。通信协议中的每个数据包都有12字节的元数据头、64字节的有效载荷和32字节的摘要。此外，它还有如下特性：

- 缓存：根据用户配置，支持不缓存或缓存闪存的一页。
- 目录：ToC设计为仅容纳两个应用程序，并且新应用程序总是下载到最旧的位置，以保留一个备用应用程序。这称为A/B更新方案。
- 消息传递：支持ADF7242或UART进行消息传递，具体取决于用户配置。使用UART进行消息传递可免除图9左侧的EZ-KIT，仅保留右侧套件用于客户端。这种有线更新方案对初始系统启动和调试很有用。

结果

除了满足功能要求并通过各种测试之外，软件的性能对于判断项目成功与否也很重要。通常使用两个指标来衡量嵌入式软件的性能：占用空间和周期数。占用空间是指软件应用程序在易失性(SRAM)和非易失性（闪存）存储器中占用的空间大小。周期数是指软件执行特定任务所使用的微处理器时钟周期数。它与软件运行时间相似，但在执行OTA更新时，软件可能进入低功耗模式，此时微处理器处于非活动状态，不消耗任何周期。虽然软件参考设计没有针对任何一个指标进行优化，但它们对于程序基准测试和比较设计权衡非常有用。

图11和图12显示了在ADuCM4050上实现的OTA更新软件参考设计的占用空间（不缓存）。这些图根据图10所示的组件进行划分。如图11所示，整个应用程序使用大约15 kB的闪存。鉴于ADuCM4050包含512 kB闪存，此占用空间非常小。真正的应用软件（为OTA更新过程开发的软件）仅需1.5 kB左右，其余用于库，例如DFP、Micro-ECC和ADF7242堆栈。这些结果有助于说明SSBL应在系统中扮演什么角色的设计权衡。15 kB占用空间的大部分是用于更新过程。SSBL本身仅占用大约500字节的空间，另外还有1 kB到2 kB的DFP代码，用于访问闪存驱动之类的器件。

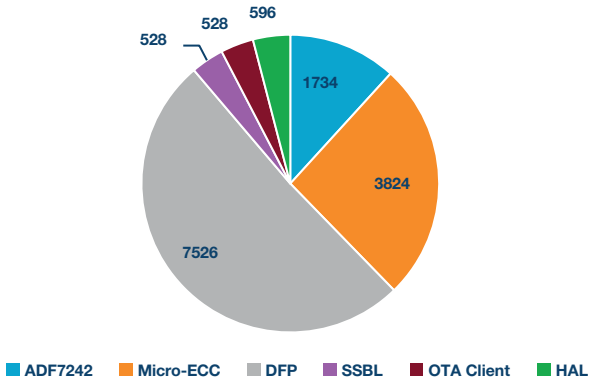


图11. 闪存占用空间（字节）

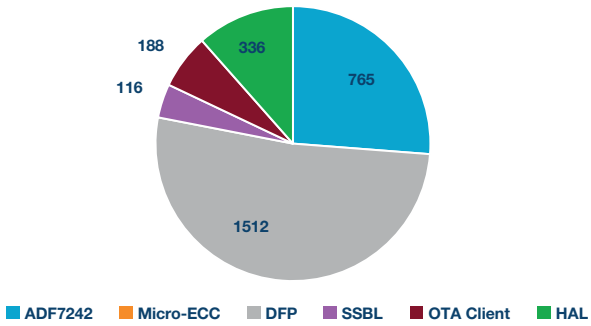


图12. SRAM占用空间（字节）

为了评估软件的开销，我们在每次接收数据包时计数周期，然后计算每个数据包平均消耗的周期数。每个数据包都需要AES-128解密、SHA-256哈希处理、闪存写入和某种数据包元数据验证。数据包有效载荷为64字节且不缓存时，处理单个数据包的开销为7409个周期。使用26 MHz内核时钟时，大约需要285微秒的处理时间。该值是利用ADuCM4050 DFP中的周期计数驱动程序计算的（未调整周期数），并且是100 kB二进制文件下载期间（约1500个数据包）的平均值。为使每个数据包的开销最小，DFP中的驱动程序应利用ADuCM4050上的直接存储访问(DMA)硬件外设来执行总线事务，并且驱动程序在每次事务处理期间将处理器置于低功耗休眠状态。每个事务中不存在一个万能的状态，如果我们禁用DFP中

的低功耗休眠并将总线事务更改为不使用DMA，则每个数据包的开销将增加到17,297个周期。这说明了高效使用器件驱动程序对嵌入式软件应用程序是有影响的。虽然减少每个数据包的数据字节数也可以降低开销，但每个数据包的数据字节数翻一倍达到128时，周期数仅有少量增加，相同实验得到的周期数为8362。

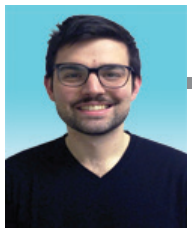
周期数和占用空间也解释了先前讨论的权衡——缓存数据包数据而不是每次都写入闪存。使能缓存一页闪存后，每个数据包的开销从7409减少到5904个周期。此20%减幅来自于更新过程跳过了大多数数据包的闪存写入，仅在缓存已满时才执行闪存写入。其代价是SRAM占用面积增加。不使用缓存时，HAL只需要336个字节的SRAM，如图12所示。但是，当使用缓存时，必须保留一个相当于闪存一整页的空间，故SRAM占用增加到2388字节。HAL使用的闪存也会少量增加，原因是需要额外代码来判断缓存何时必须清空。

这些结果证明，设计决策对软件性能会有切实的影响。不存在一个万能的解决方案，每个系统都有不同的要求和约束，OTA更新软件需要视具体情况具体对待。希望本文阐明了在设计、实现和验证OTA更新软件解决方案时遇到的常见问题和权衡。

参考文献

Nilsson、Dennis Kengo和Ulf E. Larson。“智能车辆的无线安全固件更新”。ICC研讨会——2008年IEEE国际通信会议，2008年5月。

Benjamin Bucklin Brown [benjamin-b.brown@analog.com]于2016年从麦吉尔大学毕业并获得电气工程学士学位后加入ADI公司。目前他在消费电子检测与处理技术(CSPT)部门工作，担任嵌入式软件工程师，为专用集成电路开发固件。此前，他曾在物联网平台技术部门工作，为ADuCM3029和ADuCM4050微控制器开发器件驱动程序和软件参考应用程序。



Benjamin Bucklin Brown