Application Note

C2000™ MCU 在无器件复位时的实时固件更新

Baskaran Chidambaram and Sira Rao

摘要

本文档详细介绍了带双组闪存的器件（如 TMS320F28003x）在无器件复位功能时的实时固件更新 (LFU)。

内容

插图清单

商标

1 引言

对服务器电源等应用而言，系统应持续运行，减少停机次数。但通常在因错误修复、新增功能和/或性能改进而进行固件升级期间，系统无法提供服务，导致停机。冗余模块可以解决这个问题，但会使系统总体成本增加。还有一种备选方法被称为实时固件更新 (LFU)，可在系统运行期间更新固件。无论器件复位与否，都可升级到新固件，但不复位时的操作更为复杂。

2 关键创新

建议的解决方案有两项关键创新：

- 编译器 LFU 初始化例程，用于初始化新固件中的变量，并与旧固件的控制 ISR *协同执行*。在此之前，新固件的下载和安装（编程）会与旧固件的控制 ISR 同步进行。这些步骤可能很耗时，但协同执行意味着在执行 LFU 的必要步骤时不会损害应用程序功能。
- 在理想时间（空闲时间开始）切换到新固件，并在很短的时间内（< 100 个 CPU 时钟周期）禁用中断，可以通过 MCU 中的*硬件 LFU 支持*（交换中断向量和函数指针）来实现。此步骤非常短，将其与上一步分离可以非常快速地激活新固件。

3 LFU 的构建块

LFU 设计包含多个构建块：

- 发出 LFU 命令的桌面主机应用程序
- 目标器件闪存上的 LFU 引导加载程序，用于与主机通信并实现 LFU
- 将主机连接到目标的通信外围设备（例如，串行通信接口 (SCI)）
- 要下载并激活的 LFU 就绪应用程序
- 支持 LFU 的编译器
- 具有 LFU 相关硬件支持的目标 MCU，例如具有多个物理上独立的闪存组的闪存存储器等。两个或更多个闪存组允许执行驻留在一个闪存组上的应用程序固件，并同时更新另一个闪存组。

4 建议解决方案的详细信息

4.1 闪存组组织

图 4-1 展示了如何对双组闪存分区。每组中的两个扇区分配给 LFU 引导加载程序，该引导加载程序由闪存组选择逻辑、SCI 内核和闪存 API 组成。这些在固件升级期间不会发生变化。组 1 不包含组选择逻辑。组中的其余闪存扇区被分配给应用程序。组选择逻辑使引导加载程序能够确定哪个（如果有）闪存组已使用应用固件编程，以及哪个组包含较新的应用程序固件版本。因此，组选择逻辑是软件系统的入口点。SCI 内核函数实现从主机传输映像，并通过闪存编程 API（在闪存或 ROM 中）对闪存进行编程。保留了扇区 2 中的几个位置，用于存储以下信息：

- START – 指示闪存擦除已完成，程序/验证即将开始。
- KEY – 如果此位置包含特定模式，则认为组中的固件有效
- 固件版本号 (REV) – 由组选择逻辑用于确定组 0 和 1 之间的较新固件版本

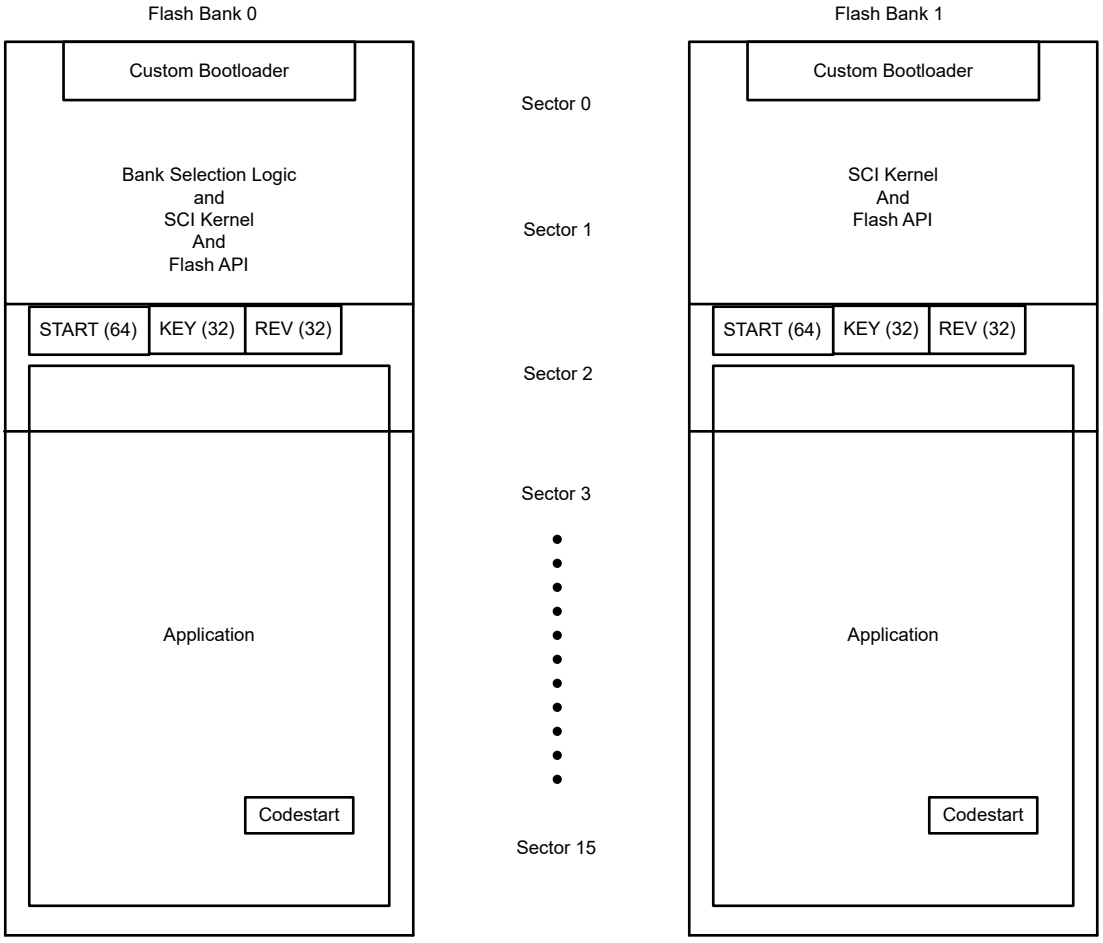

图 4-1. 双组闪存分区

4.2 影响性能的 LFU 概念和因素

创建 LFU 就绪固件时的关键注意事项是 LFU 和 LFU 切换时间期间的操作连续性。两者密切相关。运行连续性是通过状态的持续性来实现的，在固件升级期间将 RAM 中当前的静态变量和全局变量保存在相同的地址，并避免在新固件激活时重新初始化这些变量。LFU 编译器支持将启用此功能。

激活新固件涉及分支到新固件的 LFU 入口点、执行编译器的 LFU 初始化例程、到达新映像的 main() 内部以及执行其他初始化。此时会短暂禁用中断，执行需要禁用中断的初始化（例如中断向量更新、函数指针更新），然后再重新启用中断。这个最后的时间间隔被定义为 LFU 切换时间。

当有硬件支持交换闪存组时，LFU 会简化，其中任一闪存组都可以映射到固定地址空间，被视为*活动* 闪存组。*非活动* 组映射到不同的地址空间，并且是更新的组。C2000™ MCU 当前不支持闪存组交换，因此需要跟踪运行和未运行的组，并且您需要使用链接器命令文件创建面向特定组的应用固件。

函数指针和中断向量需要在 main() 内重新初始化，因为它们在闪存组之间的位置不同。C2000 MCU 支持大量中断向量（通常为 192 个），因此对所有中断向量重新初始化不太现实。通常，只使用其中一小部分，其余的分配给默认向量。LFU 特定的硬件特性（中断向量交换、RAM 块交换），可*显著* 缩短 LFU 切换时间。

4.3 LFU 的硬件支持

4.3.1 独立闪存组

为了实现从旧固件到新固件的无缝控制传输，多组闪存支持是一项关键功能。器件上使用的闪存技术不允许同时读取和写入闪存组，因此该模型允许一个组执行固件，并允许对其他组进行编程。

4.3.2 中断向量表交换

对切换时间的多个分量进行了分析以评估优化机会，并发现中断矢量映射条目的更新是影响切换时间的主要因素之一。要更新的向量的数量从几个到整个表（192 个向量）不等。单个条目的更新可能需要大约 5 个周期，因此向量表本身的更新可能需要多达 960 个周期（200MHz 时为 4.8us）。

为了减少切换时间，实现了影子向量存储器以及将其与活动向量存储器交换的能力。切换代码可以在应用程序执行过程中更新影子矢量存储器。更新向量存储器后，交换将在一个时钟周期内完成。这两个存储器都可以乒乓方式用于连续的软件升级。

中断向量交换的代表性实现如图 4-2 所示。图 4-2(a) 是交换之前的配置，图 4-2(b) 是交换之后的配置。整个向量存储器被分成两个块 – 块 A 从地址 0x0000_0D00 到 0x0000_0EFF，块 B 从地址 0x0100_0900 到 0x0100_0AFF。块 A 保存活动向量表，块 B 保存影子向量表。在 LFU 期间，影子存储器条目在切换之前得到更新，并且在切换期间执行交换。这会将切换时间从最多 960 个周期减少到一个周期。

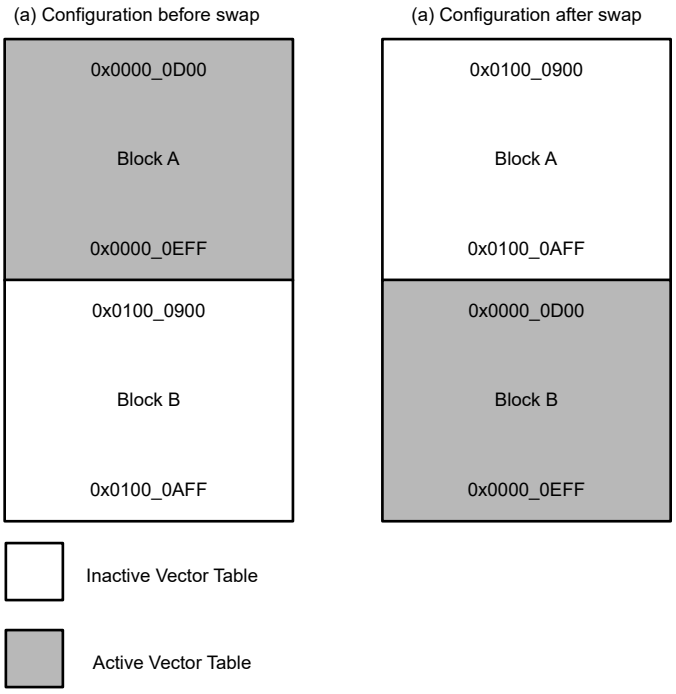

图 4-2. 中断向量交换

4.3.3 RAM 块交换

与向量表交换类似，可以交换物理 RAM 内存块，如图 4-3 所示。

如果物理存储器块 1 包含当前固件的函数指针，则在 LFU 切换之前，块 2 物理存储器中相同的相对位置可以填充新固件的函数指针。在 LFU 切换期间，简单交换操作由仅占用 1 个 CPU 时钟周期的用户应用代码启动。这允许用户应用代码在 LS0 中维护函数指针，但有两个不同的物理块映射到 LS0 地址范围。交换之后，之前映射到 Block1 地址空间的物理 RAM 块现在将映射到 Block0 地址空间，反之亦然，从而在新固件中实现无缝函数指针访问。

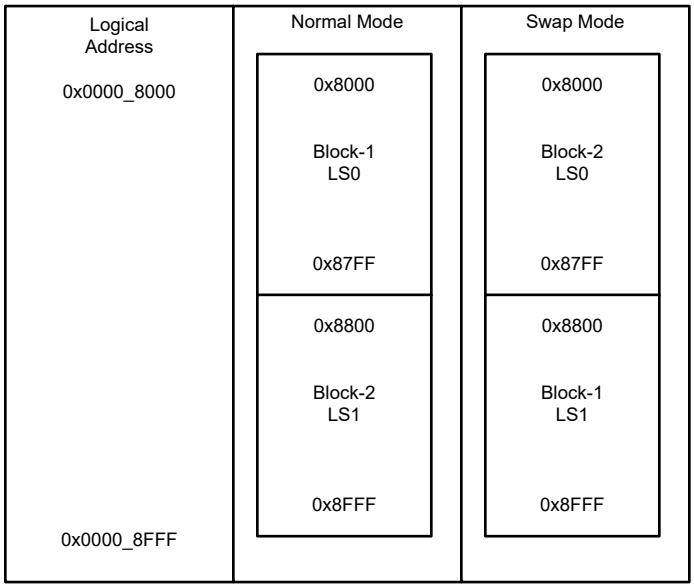

图 4-3. RAM 块交换

4.3.4 硬件寄存器标志

执行可以在器件复位后通过 C 语言初始化例程调用 main()，或在 LFU 入口点调用 LFU 编译器初始化例程，然后调用 main()，到达 main()。在前一种情况下，需要在 main() 中进行特定于器件的初始化，但在 LFU 情况下不需要。为了区分这两者，支持寄存器中的硬件标志，该标志用于指示 LFU 当前是否处于活动状态。

4.4 LFU 编译器支持

编译器为 LFU 提供以下支持：

1. LFU 初始化例程，名为 __TI_auto_init_warm
2. 变量的 LFU 属性
3. LFU 模式

编译器要求在构建新的固件可执行文件时将旧固件可执行文件作为参考映像提供。这允许编译器识别公用变量及其位置，以及识别新变量和删除的变量。

编译器为变量定义了 2 个新 LFU 属性，分别称为 *preserve* 和 *update*。“Preserve”用于在固件升级期间维持公用变量的地址。“Update”用于指示编译器可以无约束地为其分配地址的新变量，还在编译器的 LFU 初始化例程（名为 __TI_auto_init_warm()）期间进行初始化。下面列出了如何使用这些属性的示例：

float32_t __attribute__((preserve)) BUCK_update_test_variable1_cpu;

float32_t __attribute__((update)) BUCK_update_test_variable2_cpu;

使用上述分配，生成的存储器映射文件包含与 preserve 变量对应的“.TI.bound”部分和一个收集所有更新变量的“.TI.update”部分。

为了让应用开发人员处理起来更轻松，可以使用不同的 **LFU 模式**。默认模式称为 *preserve*（不要与上面描述的相应变量属性混淆），它具有以下属性：

- 如果提供了参考 (*旧*) 映像，甚至不需要将公用变量指定为 ***preserve***。这将是常用变量的默认属性，并且它们不会在编译器的 LFU 初始化例程中进行初始化。因此，它会保持该状态。
- 未指定任何属性的所有新变量都将被分配地址，但这些变量不会在编译器的 LFU 初始化例程中初始化。编译器的 LFU 初始化例程仅在使用 ***update*** 属性声明时才初始化变量。

4.5 应用程序 LFU 流程

与 LFU 相关的详细步骤概述如下：

1.*引导至组选择逻辑并执行应用*：器件复位时，执行从默认引导至闪存入口点 0x80000 处开始，这是组选择逻辑函数所在的位置。此函数检查闪存组中的有效应用，选择最新的版本，并跳转到相应固件版本的入口点 (codestart)。入口点是连接到应用的 C 运行时初始化例程和 main() 的网关。

2.*启动 LFU*：用户通过主机从主机端发起的 LFU 命令在目标 MCU 上调用 LFU。

3.*在应用中接收 LFU 命令*：(图 4-4 中的步骤 1) 应用在其 SCI 接收中断 ISR (CommandLogISR) 中接收 LFU 命令。

4.*解析 LFU 命令并跳转到 LFU 引导加载程序*：(图 4-4 中的步骤 2) 特定后台任务函数 (运行 LFU) 解析 LFU 命令，禁用 SCI 中断，并跳转到位于固定地址的同一闪存组中 LFU 引导加载程序内的 LFU 函数。

5.*将新固件和程序下载到闪存*：(图 4-4 中的步骤 3) LFU 引导加载程序中的 LFU 函数从主机接收应用映像，并将其编程到非活动闪存组中。此时，旧固件中的后台任务函数已停止执行，但旧固件中的控制 ISR 仍在继续执行，以保持应用功能不受影响。

6.*跳转到新固件的 LFU 入口点*：(图 4-4 中的步骤 4) 成功下载新固件并对其进行编程后，自定义引导加载程序会跳转到新应用映像的 LFU 入口点 (C_int_LFU)，该入口点位于每个闪存组的固定地址，与常规闪存引导入口点 (codestart) 不同。

6.*执行编译器 LFU 初始化例程并跳转到新固件的 main()*：LFU 入口点处的函数执行以下操作：

a. 调用编译器的 LFU 初始化例程 (__TI_auto_init_warm)。这将初始化已指示为需要初始化的任何变量。(图 4-4 中的步骤 5) 。

b. 在硬件 LFU 寄存器中设置一个标志，以指示 LFU 正在进行中。

c. 调用 main()。(图 4-4 中的步骤 6) 。

7.*在切换之前，在 main() 中执行 LFU 特定初始化*：(图 4-4 中的步骤 7) 在 main() 中，通过检查上述标志，初始化进程取决于 LFU 是否正在进行中。

如果设置了该标志，LFU 初始化函数 (Init_lfu) 会通过任何用户指定的代码从闪存复制到 RAM 存储器。接下来，它使用与新固件对应的中断向量位置更新非活动中断向量表。类似地，一组非活动函数指针也对应于新固件中的函数指针位置进行更新。

8.*等待更优 LFU 切换点*：(图 4-4 中的步骤 8) 使用一个带有软件标志的简单状态机来确定控制 ISR 的结束和空闲时间的开始。这是更优切换时间 (IdentifyIdleTime)，因为它允许更大限度地利用控制循环中断之间的空闲时间。

9.*执行 LFU 切换*：(图 4-4 中的步骤 9) 请注意，即使新固件的 main() 中已经有执行，旧的控制循环 ISR 仍在执行，以使应用功能不受影响。确定更优 LFU 切换点后，会发生 LFU 切换步骤 (ActivateApp)。首先，禁用全局中断。执行硬件中断向量表交换和 RAM 块交换。然后堆栈指针被重新初始化，全局中断被重新启用。现在，新固件的 ISR 和后台任务函数开始执行，代表 LFU 切换完成。由于全局中断在短时间内被禁用，在这段时间内发生的中断将继续保持锁存状态，并在全局中断被重新启用时中断 CPU。

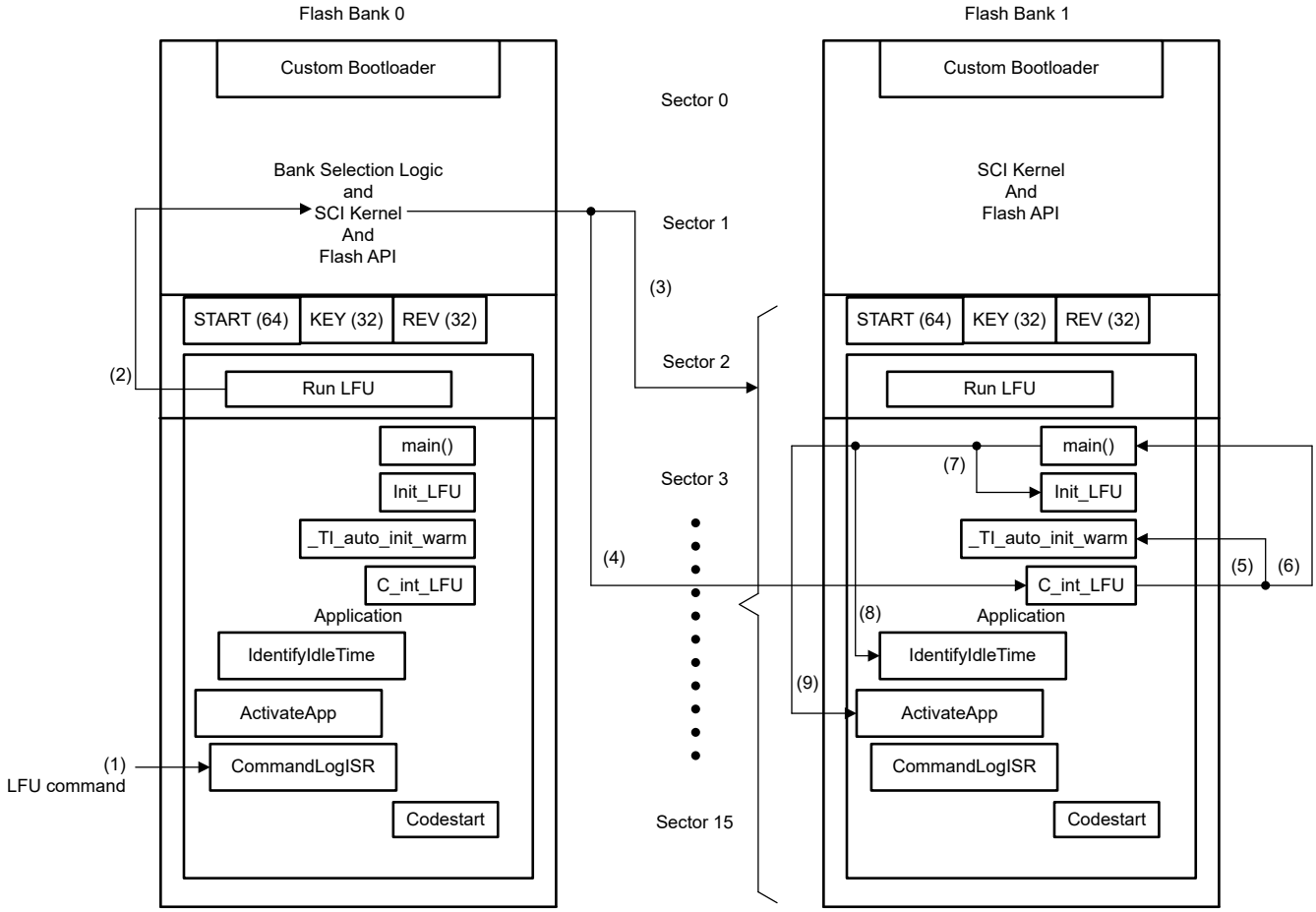

图 4-4. 应用程序 LFU 流程

5 结果和结论

在图 5-1 中，描述了 LFU 切换之前发生的步骤：下载固件和编程闪存组、LFU 编译器初始化例程（函数 __TI_auto_init_warm()），然后在 main() 中执行 LFU 特定初始化（init_lfu() 函数）。

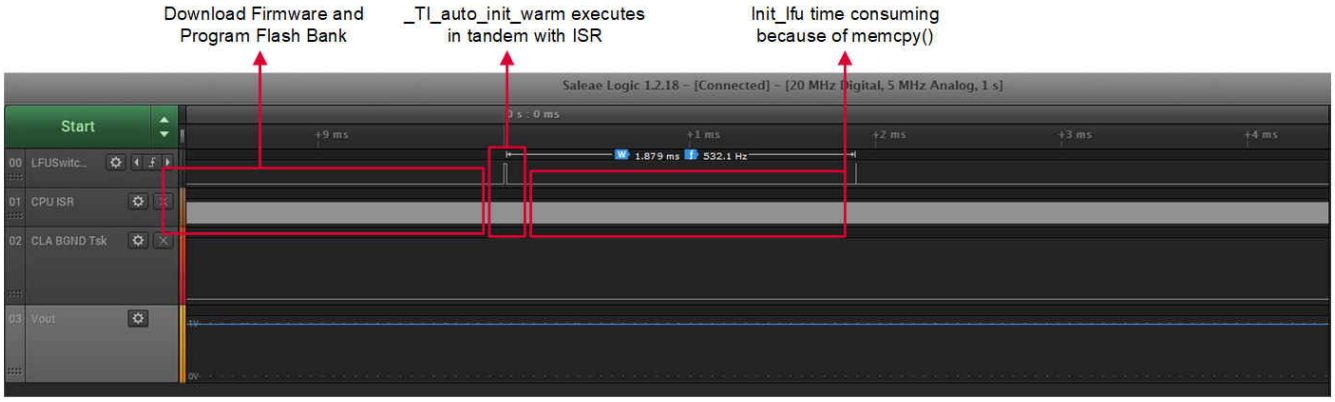

图 5-1. LFU 切换前的步骤

图 5-2 展示了实际的 LFU 切换。最顶部的波形 (00) 表示 LFU 切换，第二个波形 (01) 表示 ISR CPU 负载（旧固件为 80%，新固件为 40%），底部波形 (03) 表示稳定的输出电压。切换在 0.6µs（或 72 个 CPU 时周期）内发生，其中包括函数调用和 GPIO 设置/复位时间。ISR 结束后大约 40 个周期发生切换。退出 ISR 大约需要 20 个周期，然后退出等待理想切换时间的循环大约需要 15 个周期。

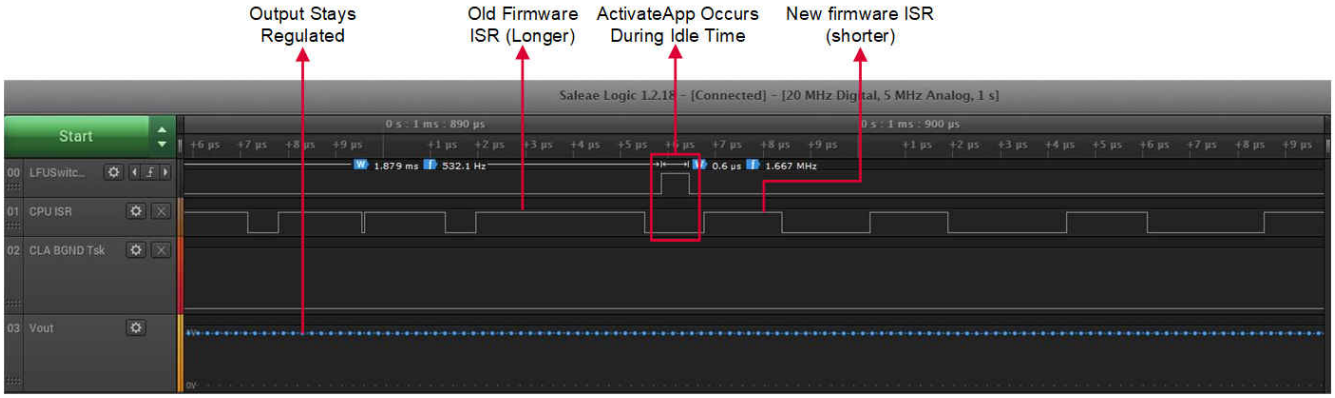

图 5-2. LFU 切换步骤

本应用手册演示了适合于实时控制应用程序的 LFU 的系统实现，尤其是需要在无停机条件下运行的高可用性系统。借助可用的 LFU 构建块，可以在 10 秒的 CPU 时钟周期内完成到新固件的切换，包括全新的应用程序 LFU 软件流程、硬件 LFU 支持和编译器 LFU 支持。

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

Changes from Revision A (August 2021) to Revision B (September 2022)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式。	2
• 已添加 节 2。	2

重要声明和免责声明

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、某特定用途方面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功能安全、信息安全、监管或其他要求。

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 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/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。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265