Application Report

C2000™ MCU 在有器件复位时的实时固件更新

Sira Rao and Baskaran Chidambaram

摘要

本文档详细介绍了包含双组闪存的器件在有器件复位时的实时固件更新 (LFU)，及其相关挑战和解决建议。为了方便演示，使用了基于 LED 的示例（作为 C2000ware 的一部分提供）。

内容

插图清单

商标

1 引言

对服务器电源、计量等应用而言，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，也会导致相关实体的停运。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 LFU 所需资源

如果器件拥有各种类型的充足资源（CPU 带宽、内存和可用外设），则 LFU 是可行的：

- CPU 带宽 – 新固件必须使用通信外设传输，并写入闪存存储器，同时应用仍在运行。这意味着 CPU 需要有足够的可用带宽来支持 LFU。
- 存储器 – 此处使用的非易失性存储器是闪存存储器。闪存读取和写入操作无法在同一闪存组中同时执行。但读取和写入操作可以在不同闪存组中同时执行。因此，对器件而言理想方案是包含两组闪存。如果器件只有一组闪存，实现 LFU 将特别具有挑战性，但仍是可行的，前提条件是：
  - 新固件更新到闪存时，完整的应用代码（或控制用户所关注的输出的那部分代码）和闪存 API 需要从 RAM 存储器运行。这意味着应有大小足够的 RAM 存储器可供利用。
  - 一些器件支持 ROM 中的闪存 API；在这些器件中，应用代码可从 RAM 运行，闪存 API 可从 ROM 存储器运行，从而降低了 RAM 要求。
- 可用外设 – 使用新映像的备用通信外设可从主机传输到器件。

在包含多组闪存的器件中更易于实现 LFU。在此文档和参考示例中，假设器件都包含两组闪存。考虑使用的器件为具有双组闪存的 TMS320F28004x。它们各自的地址空间都是 64K x 16，地址范围为 0x80000-0x8FFFF 和 0x90000-0x9FFFF。

3 存储器布局

闪存存储器的分区如图 3-1 所示。

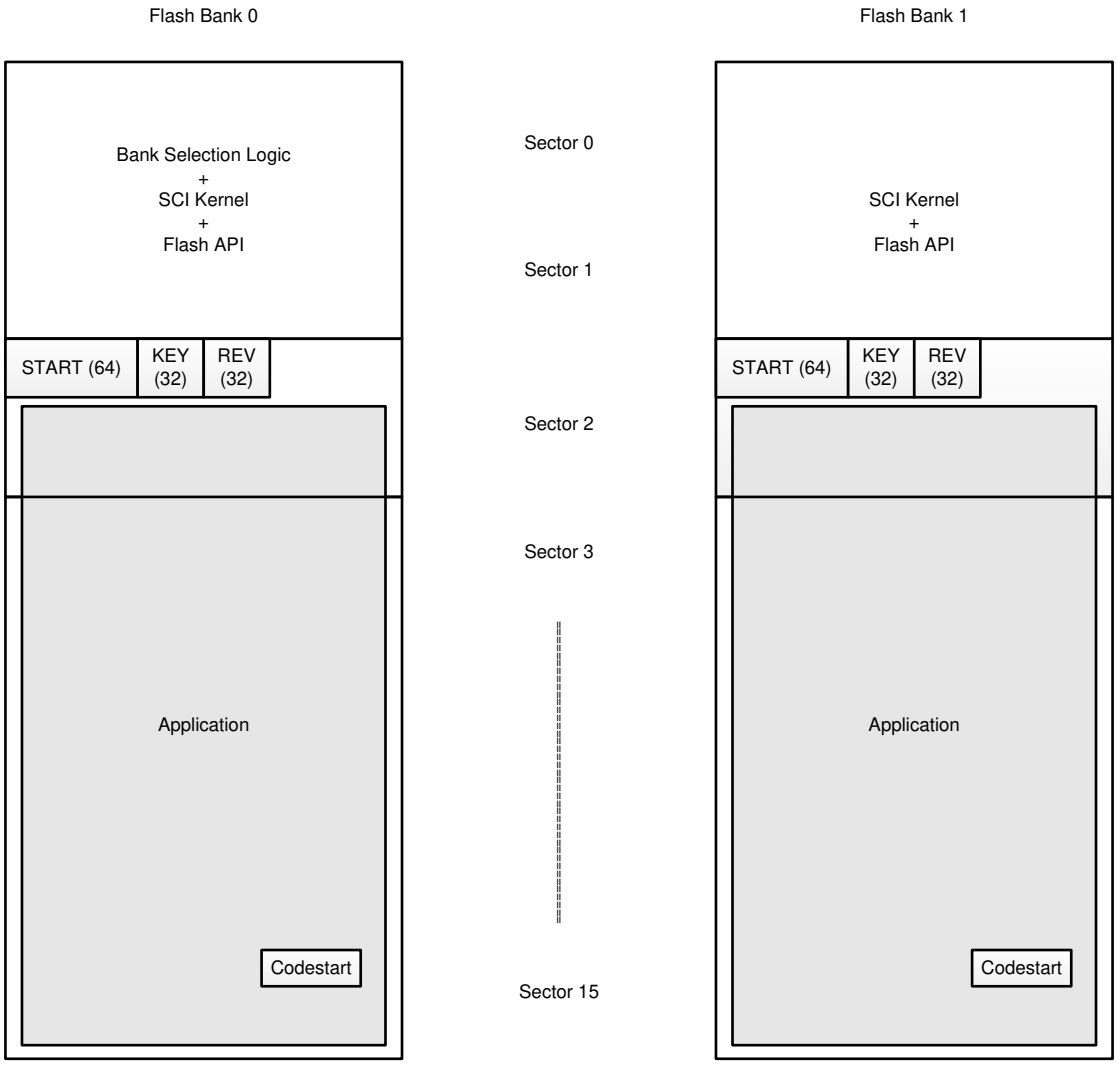

图 3-1. 闪存存储器组 0 和组 1 的内容

假设每组闪存有 16 个扇区，两个扇区已分配给静态代码（代码在不同应用中均保持不变）。节 4 进行了相关介绍。

保留了扇区 2 中的几个位置，用于存储以下条目：

- START – 当这个 64 位的字段由串行通信接口 (SCI) 闪存内核在特定闪存组中设置（设为 0x5A5A5A5A5A5A5A5A），则说明相应的闪存组已被擦除（应用扇区），而且编程/验证即将开始。在此示例中，START 字段在 BANK0 中位于地址 0x82000，在 BANK1 中位于地址 0x92000。

- REV – 这代表 32 位固件版本号，由 SCI 闪存内核设置，由组选择逻辑用于确定在闪存组 0 和 1 之间哪一个是最新映像。REV 始于 0xFFFFFFFF，每个后续闪存编程周期都会递减。因此具有较低版本号的闪存组被视为最新版本。为了简单起见，固件版本字段由闪存内核处理。实际上，最近下载的映像可能并不是最新固件版本，应用映像包含固件版本，但此模型假设内核会更新 REV 字段。在此示例中，REV 字段在 BANK0 中位于地址 0x82006，在 BANK1 中位于地址 0x92006。假设 REV 在 BANK0 中为 FFFF FFFA，在 BANK1 中为 FFFF FFF9，则 BANK1 将被视为最新固件并执行。
- KEY – 如果此位置包含有效 KEY (0x5B5B5B5B)，闪存组中的映像将被视为有效。这个 KEY 将由 SCI 闪存内核写入，并由组选择逻辑读取并测试。在此示例中，KEY 字段在 BANK0 中位于地址 0x82004，在 BANK1 中位于地址 0x92004。

扇区 2 的其余部分和扇区 3-15 可用于存储应用映像。这样扇区 0 和 1 中的静态代码可编程一次，在 LFU 期间保持不变。

4 LFU 中的静态代码

用于支持 LFU 的静态代码包含以下内容：

- 组选择逻辑 – 如有两组（或更多）包含应用固件的闪存，则必须有确定从哪个闪存组引导的逻辑。此逻辑的常见实现方式取决于固件版本号。如前所述，在此示例中，较低版本号为最新映像。组选择逻辑位于默认闪存引导地址（在 F28004x 中为 0x80000），因此当引导 ROM 代码执行完成后，执行将传输到组选择逻辑。组选择逻辑仅包含在 Bank0 中，不在 Bank1 中。
- 闪存内核 – 闪存内核的任务是使用外设从主机接收固件映像，并调用闪存编程 API 将其写入闪存存储器。在本文档中使用的是 SCI 闪存内核，因为会使用 SCI 外设传输新的固件映像。详细的分步流程记录在标头为 flashapi_ex2_ldfu.c 的文件中。
  - 在空白器件（两组闪存中均无应用固件）中，组选择逻辑将确认两组闪存中均无有效的 KEY，并将等待通过 SCI 传输的 LFU 命令。将使用闪存内核来更新 bank1 上的固件，因为内核代码首先在 bank0 上编程，因此将从 bank0 执行。
  - 如果一组或两组闪存包含有效应用，组选择逻辑会将控制传输到相应组的代码入口点 (codestart)。在此示例中，Bank0 的 codestart 地址为 0x8EFF0，Bank1 的为 0x9EFF0。
  - 在 LFU 期间，应用将调用闪存内核，以接收并更新固件。
- 闪存 API – 闪存 API 提供擦除和编程闪存存储器的接口。这些 API 需要从未在更新的闪存组运行。

静态代码配置为单一示例 - flashapi_ex2_sci_kernel 工程（包含在 C2000Ware 中，地址为 <C2000Ware>\driverlib\f28004x\examples\flash）。此示例支持多种构建配置，下方列出了其中与 LFU 相关的配置：

- BANK0_LDFU - 将组选择逻辑和闪存内核链接到组 0（地址为 0x80000 - 0x81FFF）。在闪存中使用闪存 API 符号。
- BANK0_LDFU_ROM - 将组选择逻辑和闪存内核链接到组 0（地址为 0x80000 - 0x81FFF）。在 ROM 中使用闪存 API 符号；F28004x 的 Rev A 无法与这个构建配置一同使用，因为它不支持 ROM 中的闪存 API。
- BANK1_LDFU - 将闪存内核链接到组 1 (0x90000 - 0x91FFF)。在闪存中使用闪存 API 符号。
- BANK1_LDFU_ROM - 将闪存内核链接到组 1 (0x90000 - 0x91FFF)。在 ROM 中使用闪存 API 符号；F28004x 的 Rev A 无法与这个构建配置一同使用，因为它不支持 ROM 中的闪存 API。

更多详细信息，请参阅 flashapi_ex2_sci_kernel 工程（包含在 C2000Ware 中，路径为 <C2000Ware>\driverlib\f28004x\examples\flash）中的 flashapi_ex2_sciKernel.c。

5 LED 示例应用和 LFU 流程

此示例（flashapi_ex3_live_firmware_update 工程）旨在演示 LFU，其中，LED 会定期闪烁。这是使用实时器件固件更新（实时 DFU 或 LDFU）命令实现的，该命令由 SCI 内核发出，可与串行闪存编程器（PC 工具）配套使用。

在此示例中，将执行 SCI 自动波特锁，自动波特锁使用的字节将回传。初始化并启用两个中断的命令是：SCI Rx FIFO 中断和 CPU Timer 0 中断。CPU Timer 0 中断每秒出现一次；CPU Timer 0 的中断服务例程 (ISR) 会根据正在运行的构建配置切换 LED。

- 根据 BANK0_FLASH 构建配置切换 LED1。“BANK0”是此构建配置中的预定义符号，定义此符号后，工程会将 GPIO 设置为与 LED1 关联。
- 根据 BANK1_FLASH 构建配置切换 LED2。“BANK1”是此构建配置中的预定义符号，定义此符号后，工程会将 GPIO 设置为与 LED2 关联。

上述构建配置生成的应用映像，将用于在此文档中演示 LFU。请注意，除了上述两种构建配置的不同外，两种应用映像没有其他差别。因此，这是相对简单的 LFU 演示示例。

SCI Rx FIFO 中断的 FIFO 中断级别被设为 10 字节。串行闪存编程器数据包中的字节数（使用 LDFU 命令时）为 10。从串行闪存编程器向器件发送命令时，SCI Rx FIFO ISR 会从 FIFO 中的 10 字节数据包中接收命令。如果该命令与实时器件固件更新（实时 DFU）命令匹配，代码会跳转至位于相应闪存组的 SCI 闪存内核 (flashapi_ex2_ldfu.c) 内部的实时 DFU 函数 (liveDFU())。如果 Bank0 上的应用正在执行，控制将传递到 Bank0 上位于 0x81000 处的 liveDFU()。如果 Bank1 上的应用正在执行，控制将传递到 Bank1 上位于 0x91000 处的 liveDFU()。在此函数中，执行传递到 ldfuLoad() 函数，以擦除相应闪存组，将十六进制格式的程序（为相应 SCI 引导格式）加载到闪存中，并验证程序。然后将看门狗配置为进行复位。最后，启用看门狗，以便复位。器件复位后，会进行引导并加载新固件。

图 5-1 所示为代码进入应用的 main() 后的概要代码流程。更多详细信息，请参阅位于 flashapi_live_firmware_update 工程（包含在 C2000Ware 中，路径为 <C2000Ware>\driverlib\f28004x\examples\flash）中的 flashapi_ex3_live_firmware_update.c。

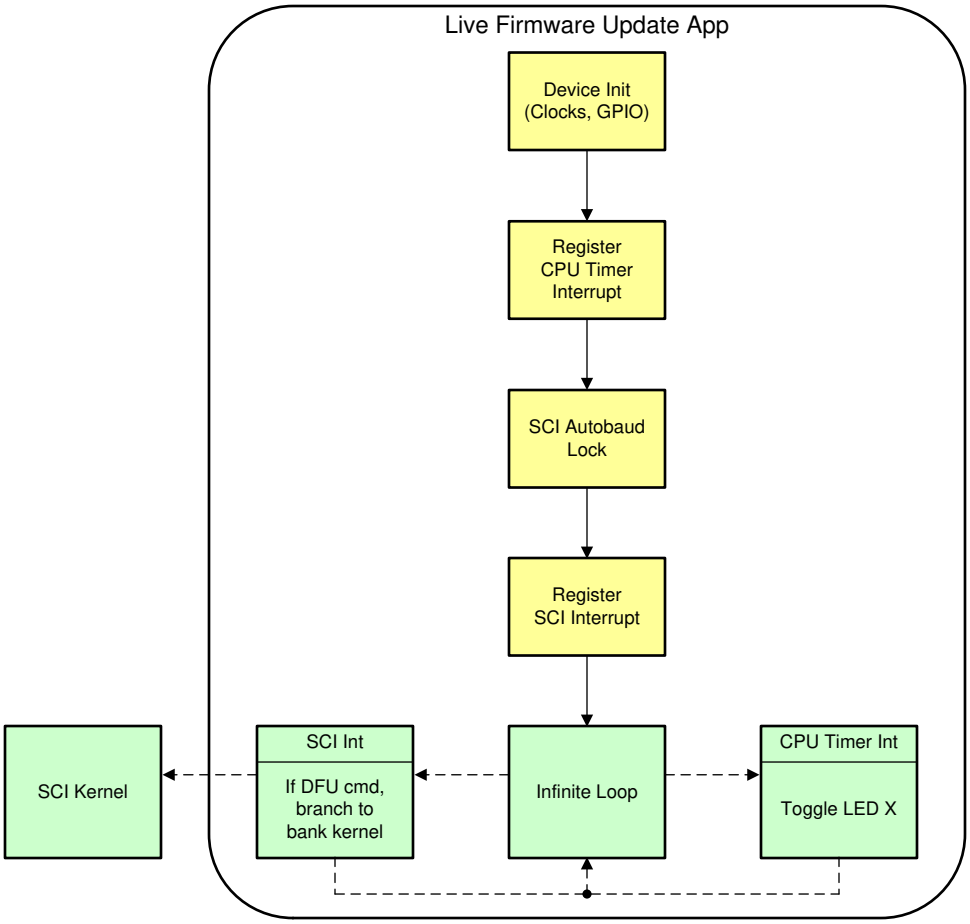

图 5-1. 进入应用的 main() 后的代码流程

6 运行 LED 示例

6.1 串行闪存编程器更新

C2000ware 随附的串行闪存编程器 (serial_flash_programmer.exe) 将内核和应用映像作为参数。通常，内核首先传输到 SCI 引导加载程序，并从器件上的 RAM 或闪存执行。然后内核程序通过 SCI（从在 PC 上运行的串行编程器中）获得应用映像，并在闪存存储器中对应用映像进行编程。

对于 LFU 而言，静态内容（包括闪存内核）首先编程到闪存组 0 和 1 的闪存扇区 0 和 1。节 6.2 进行了相关介绍。之后，需要修改串行闪存编程器，目的是仅传输应用映像。在 serial_flash_programmer.cpp 中为“#define kernel”这一行添加注释则可实现此目标。在 Visual C 中编译工程（调用 serial_flash_programmer_appln.exe）可以重新生成串行闪存编程器。预先构建的可执行文件位于 <C2000Ware>\utilities\flash_programmers\serial_flash_programmer\。因此这时用户无需执行操作。

6.2 静态代码编程 – 通过 Code Composer Studio™ (CCS) 加载

图示步骤中使用的硬件是 ControlCARD 集线站 Rev4.1 上的 F28004x ControlCARD。如果 JTAG 连接可用，则 CCS 可用于为闪存组 0 和 1 加载静态代码。注意：

NOTE

在加载映像之前，请确保在 CCS（或您的目标配置文件 – 右键单击选择属性）中如图 6-1 所示进行设置。在 BANK0_LDFU 和 BANK1_LDFU 配置中构建 flashapi_ex2_sci_kernel 工程，并分别加载至目标。CCS 闪存插件会将内容加载至闪存。

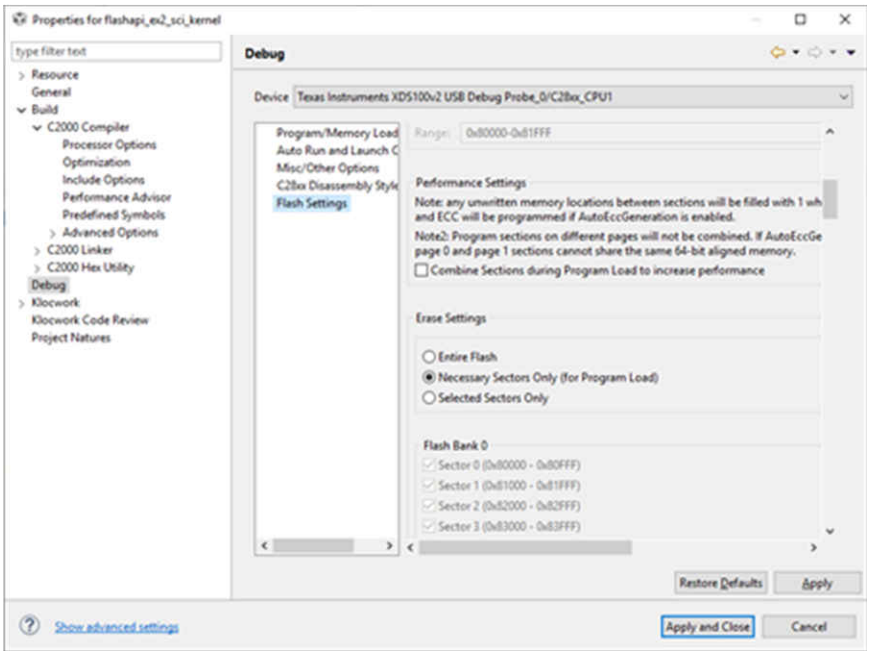

图 6-1. 将闪存设置为仅擦除必要扇区

1. 加载 BANK0 静态映像（flashapi_ex2_sci_kernel 工程的 BANK0_LDFU）。

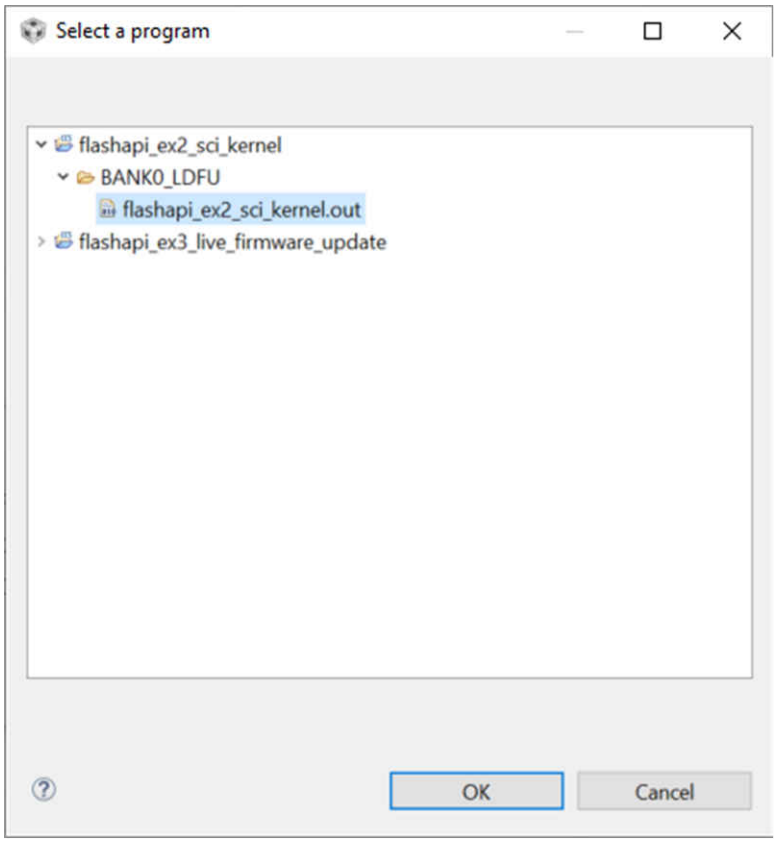

图 6-2. 选择要加载到闪存组 0 的内核

将静态内容加载到 BANK0 后，首先会执行组选择逻辑，确定在两个组中均未对应用固件进行编程。

控制将传递到闪存内核，它将做好在 BANK1 中对应用进行编程的准备。图 6-3 所示为 CCS 视图（程序不会在 main() 停止，而是会继续运行并等待 SCI 命令）：

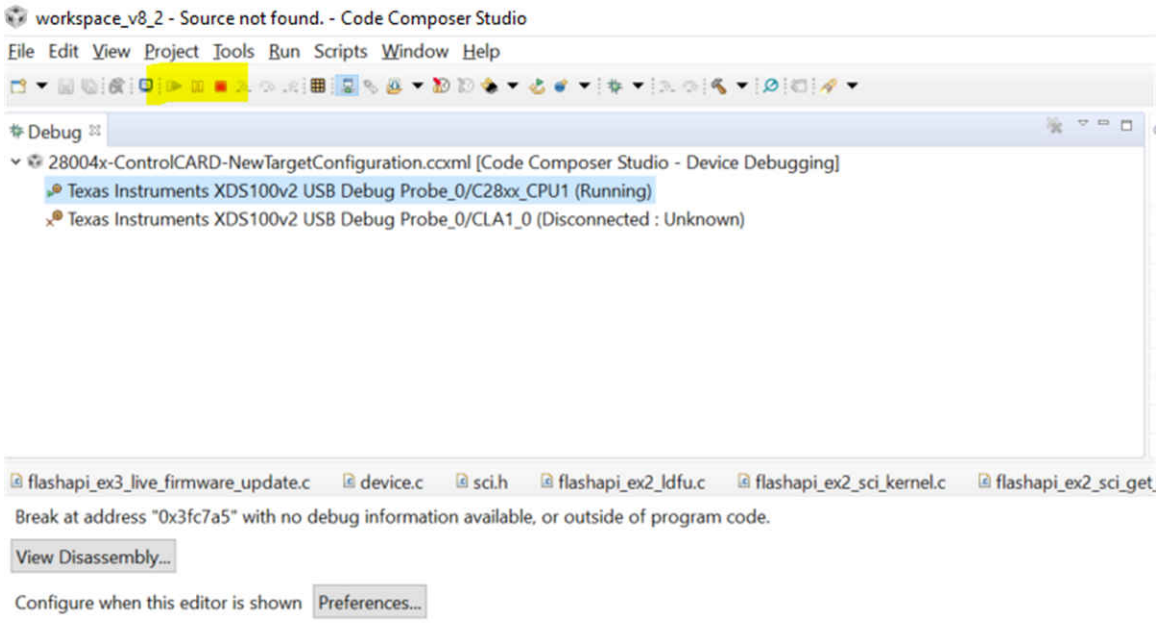

图 6-3. 编程闪存组 0 内核后的 CCS 窗口视图

2. 在 CCS 中打开“Memory Browser”窗口，并输入地址 0x80000，验证 BANK0 的内核内容。

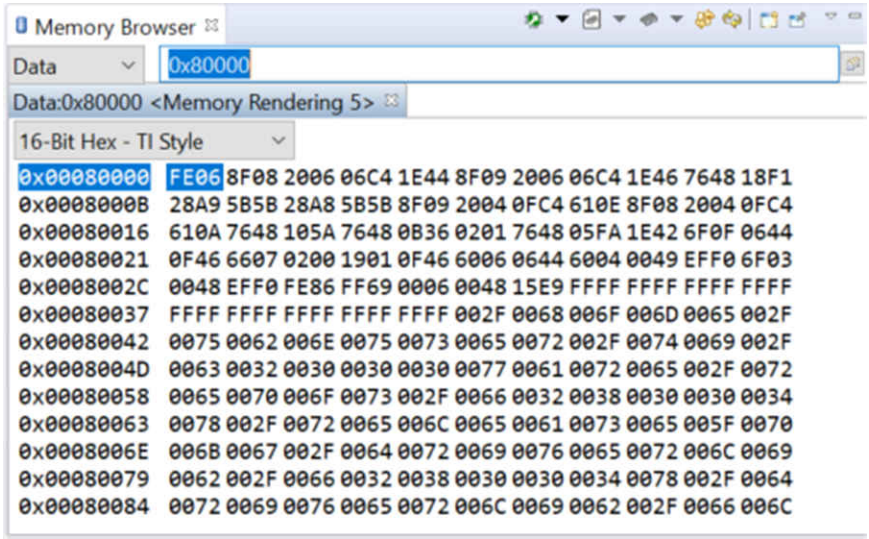

图 6-4. 验证闪存组 0 内核编程成功的 CCS 存储器浏览器视图

3. 现在切换到 Windows 命令提示，并执行以下命令：serial_flash_programmer_appln.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel.txt -a flashapi_ex3_live_firmware_updateBANK1FLASH.txt -b 9600 -p COMx 其中 x = PC 和目标板之间相应 JTAG 连接的 COM 端口。可以在设备管理器中寻找端口，从而找到该 COM 端口号。在目标和计算机之间如果连接了提供 JTAG 和 SCI 功能的 USB 电缆，则会出现该端口号。此示例使用的波特率为 9600。以构建配置 BANK1_FLASH 构建 CCS 工程 flashapi_ex3_live_firmware_update 将生成 flashapi_ex3_live_firmware_updateBANK1FLASH.txt。

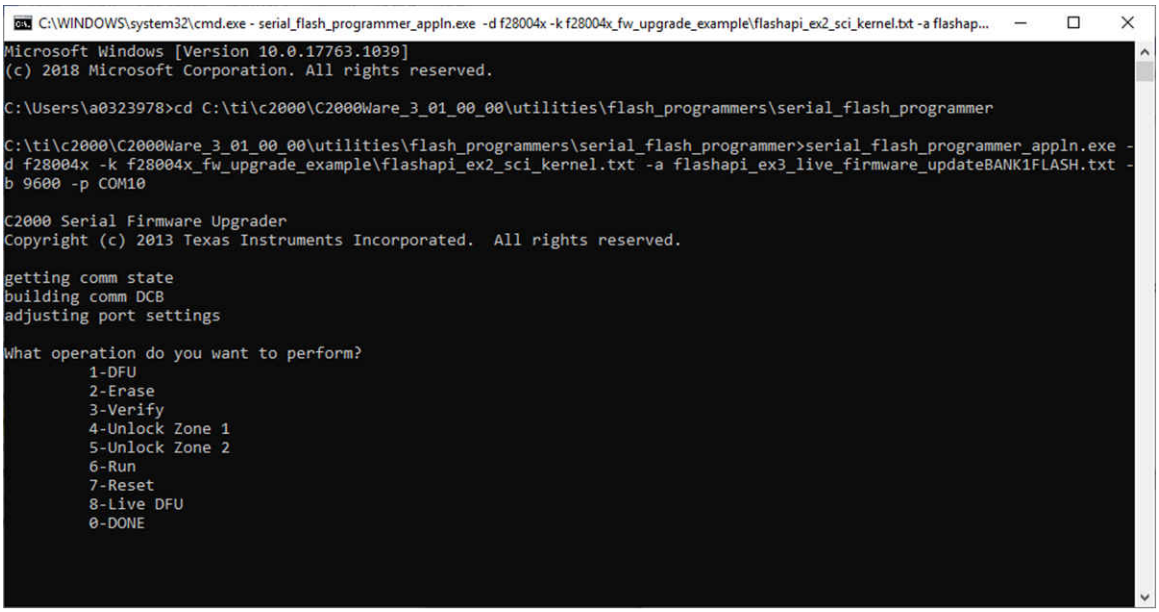

图 6-5. 根据 Windows 命令提示调用的 LFU 串行命令

4. 选择 LDFU (8) 命令后，内核将在 BANK1 中接收并对应用进行编程。应用大小约为 36KB。下载时间约为 30 秒。

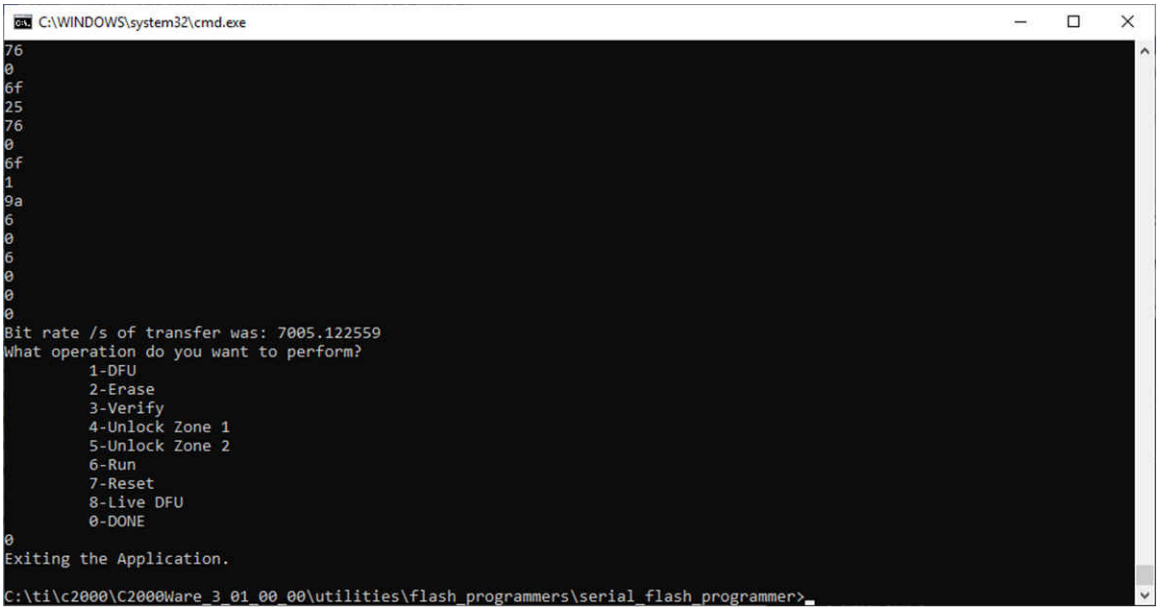

图 6-6. 成功完成对闪存组 1 进行编程的 LFU 命令

5. 传输完成后，输入 0 指示命令操作结束。在 CCS 中打开“Memory Browser”窗口，并输入地址 0x92000，验证 BANK1 的应用内容。

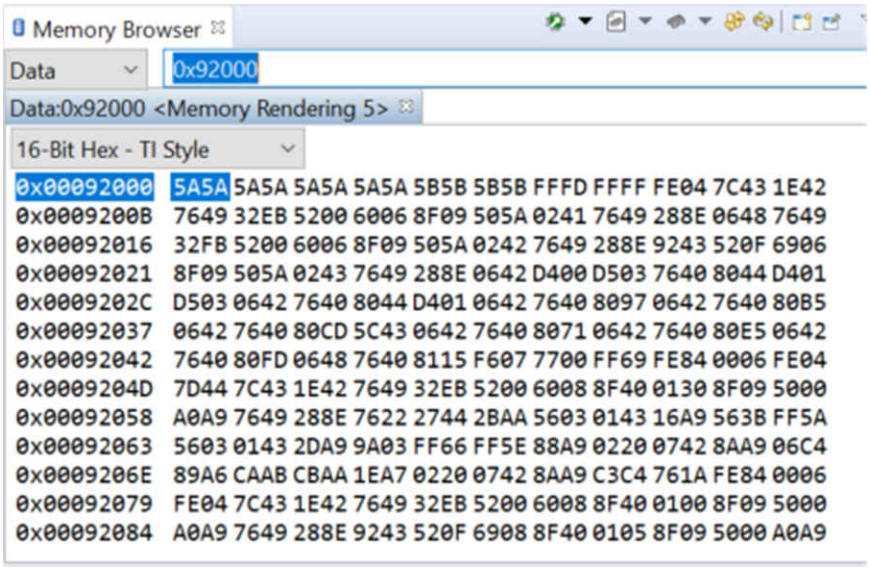

图 6-7. 验证闪存组 1 中的应用编程成功的 CCS 存储器浏览器视图

内核还将更新 BANK1 扇区 2 中的 KEY 和版本号。现在静态映像在编程后的 BANK0 中，应用映像在 BANK1 中编程。

6. 此时，用户将电路板复位就可以看到 LED2 开始闪烁。
7. 接下来加载 BANK1 静态映像（flashapi_ex2_sci_kernel 工程的 BANK1_LDFU）。

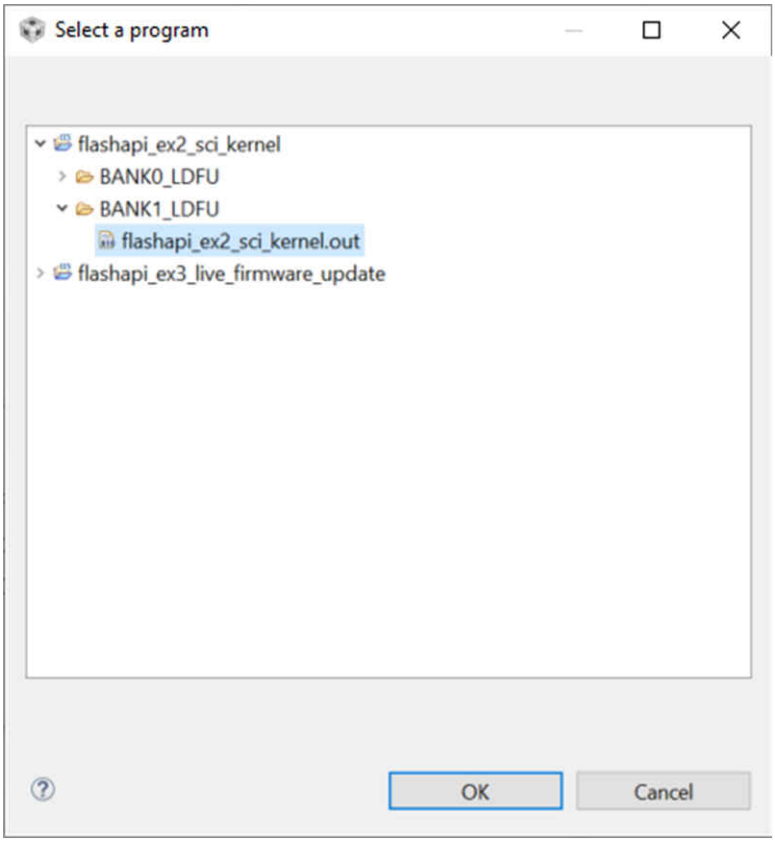

图 6-8. 选择要加载到闪存组 1 的内核

8. 在 CCS 中打开“Memory Browser”窗口，并输入地址 0x90000，验证 BANK1 的内核内容。

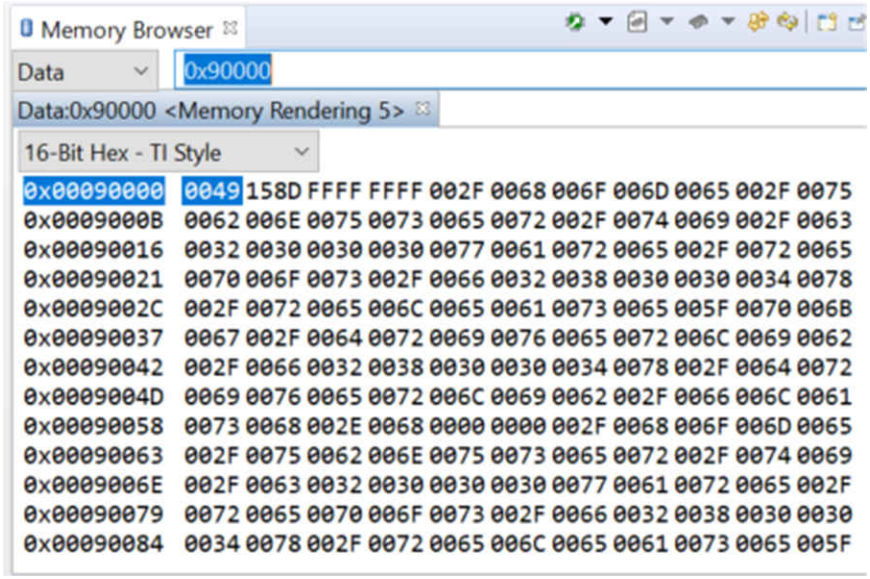

图 6-9. 验证闪存组 1 内核编程成功的 CCS 存储器浏览器视图

静态内容被加载到 BANK1 之后，在 main() 停止执行。Bank1 的内核将出现这种情况，因为组选择逻辑仅位于 Bank0 上，不在 Bank1 上。因此，对于 Bank1 来说，执行流程遵循传统流程，codestart 延伸至 main()。

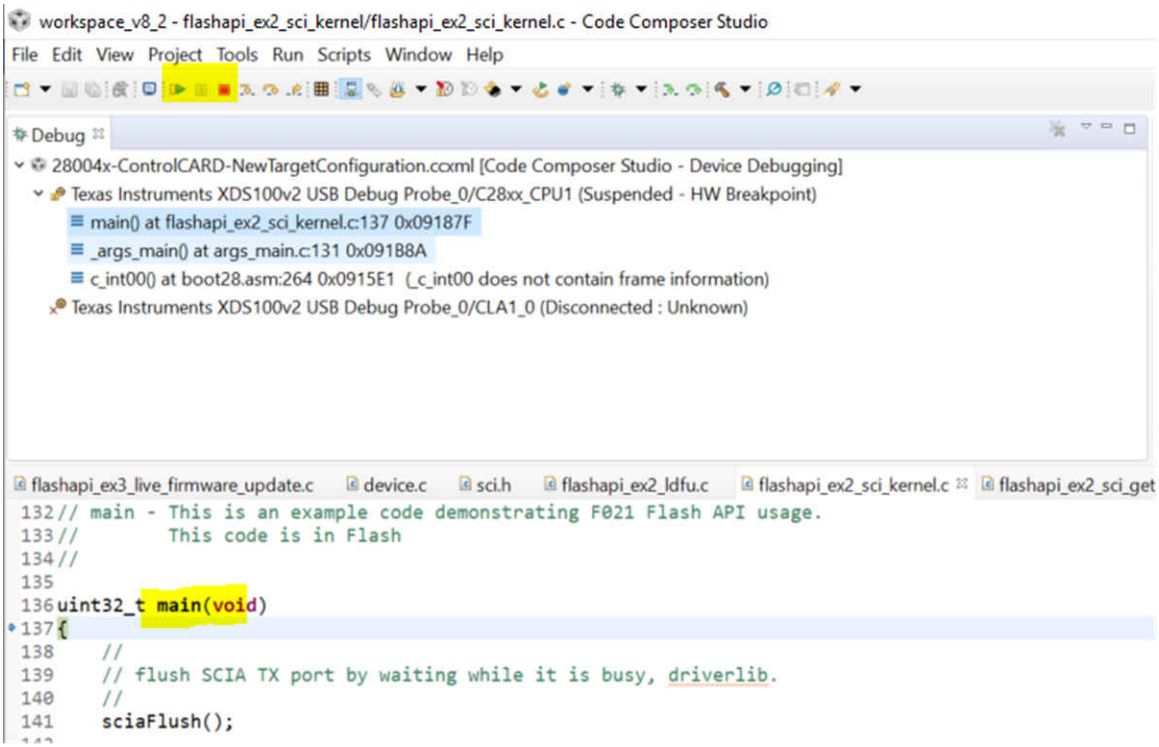

图 6-10. 编程闪存组 1 内核后的 CCS 窗口视图

9. 在 CCS 中按“Run”（运行），组选择逻辑将运行，并从 BANK1 执行应用。然后，在 PC 命令行中执行以下命令。

   serial_flash_programmer_appln.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel.txt -a flashapi_ex3_live_firmware_updateBANK0FLASH.txt -b 9600 -p COMx 其中 x = PC 和目标板之间相应 JTAG 连接的 COM 端口。以构建配置 BANK0_FLASH 构建 CCS 工程 flashapi_ex3_live_firmware_update 将生成 flashapi_ex3_live_firmware_updateBANK0FLASH.txt。

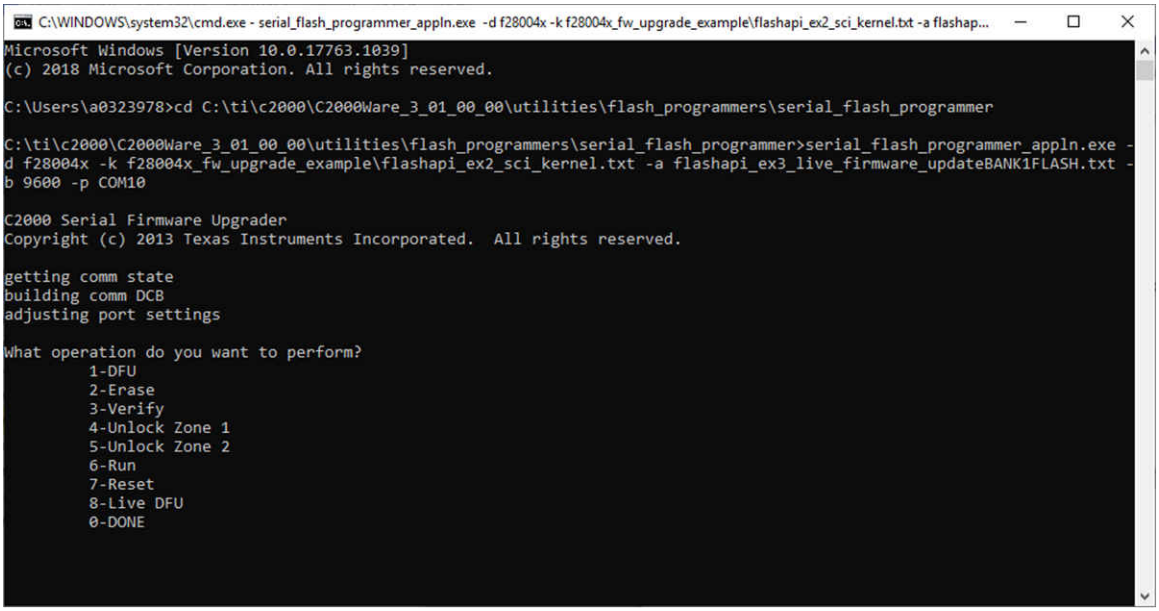

图 6-11. 根据 Windows 命令提示调用的 LFU 串行命令

10. 控制将从应用传递到闪存内核，选择 LDFU (8) 命令后，内核将在 BANK0 中接收并对应用进行编程。
11. 传输完成后，输入 0 指示命令操作结束。

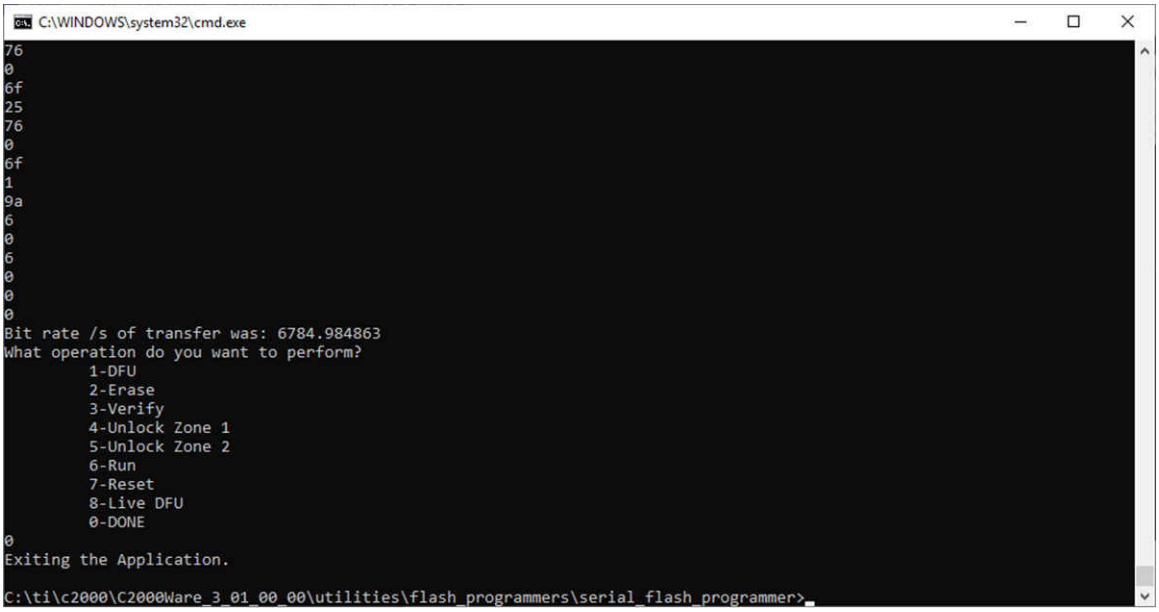

图 6-12. 成功完成对闪存组 0 进行编程的 LFU 命令

内核还将更新 BANK0 扇区 2 中的 KEY 和版本号。现在静态映像在编程后的 BANK1 中，应用映像在 BANK0 中编程。

12. 此时，与之前一样，用户将电路板复位就可以看到 LED1 开始闪烁。

6.3 应用的实时固件更新

对静态内容进行编程后，断开调试器的连接，并将引导模式开关设为闪存引导模式。器件进行引导时，将跳转至闪存。默认的闪存入口点是 0x80000，这是 Bank0 中编程静态代码（组选择逻辑 + SCI 闪存内核）的位置。组选择逻辑将执行，并确定闪存组 0 和 1 中均存在有效映像（基于 KEY 和版本号）。将选择运行 Bank0，因为在节 6.1 中，它是较晚编程的，因此会根据 REV 字段判定为最新版本。因此将执行 Bank0 中的应用固件。

此应用每秒钟闪烁一次 LED1。同时，应用还会监控串行端口，检查是否存在用于实时固件更新的映像。

若要执行实时固件更新，应为 BANK1 配置构建更新的应用，并从主机 PC 的命令行中执行图 6-13 中显示的命令。列出菜单后输入“8”来选择 LDFU。 serial_flash_programmer_appln.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel.txt -a flashapi_ex3_live_firmware_updateBANK1FLASH.txt -b 9600 -p COMx.

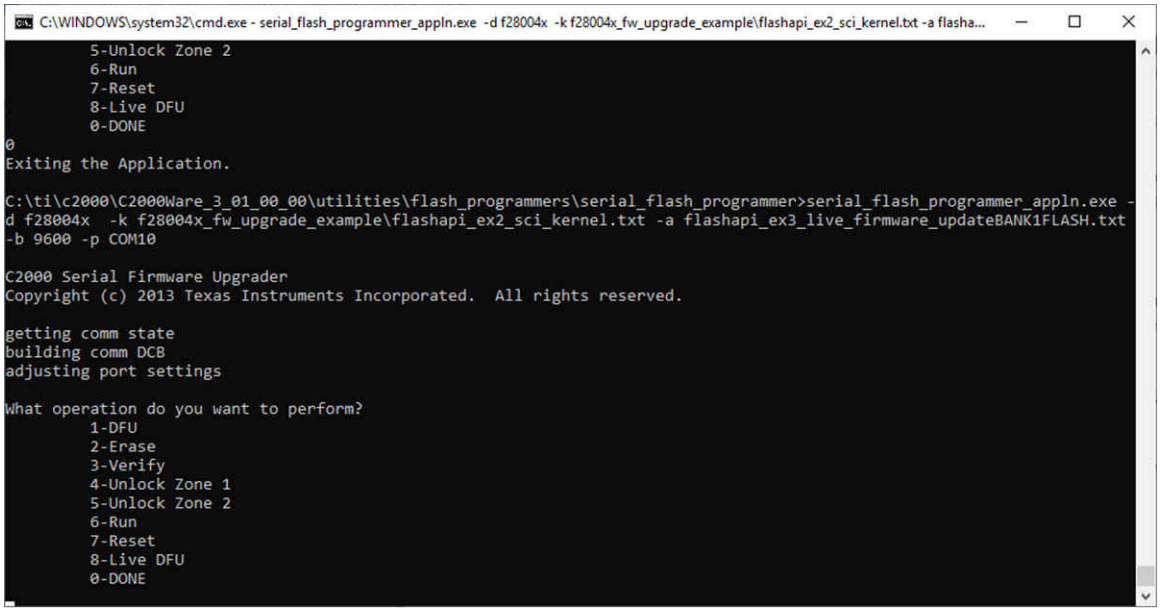

图 6-13. 根据 Windows 命令提示调用的 LFU 串行命令

如果它接收一个映像，控制将跳转至 Bank0 中的闪存内核，并更新 Bank1 上的固件映像。传输完成后，输入 0 以指示命令操作结束。在将新映像下载到 Bank1 的过程中，应用会继续在 Bank0 上运行（LED1 继续照常每秒闪烁）。这是因为 LFU 处理在后台循环中进行，不是在 SCI 接收中断中进行。这样就可以处理其他中断（例如开关 LED 的 CPU 计时器中断）。

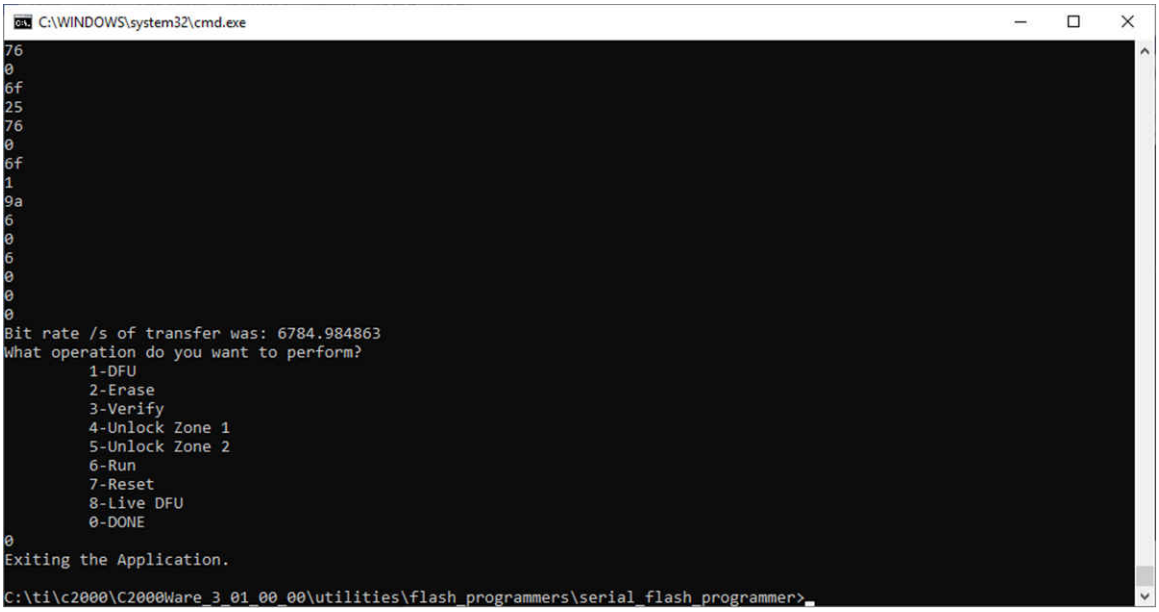

图 6-14. 成功完成对闪存组进行编程的 LFU 命令

看门狗计时器复位后，器件也会复位，控制会传递至 Bank1 中最新的应用映像，LED2 将开始闪烁。

NOTE

不再需要手动将器件复位。

重复此过程，以更新交替组中的映像。图 6-15 所示为上述流程。

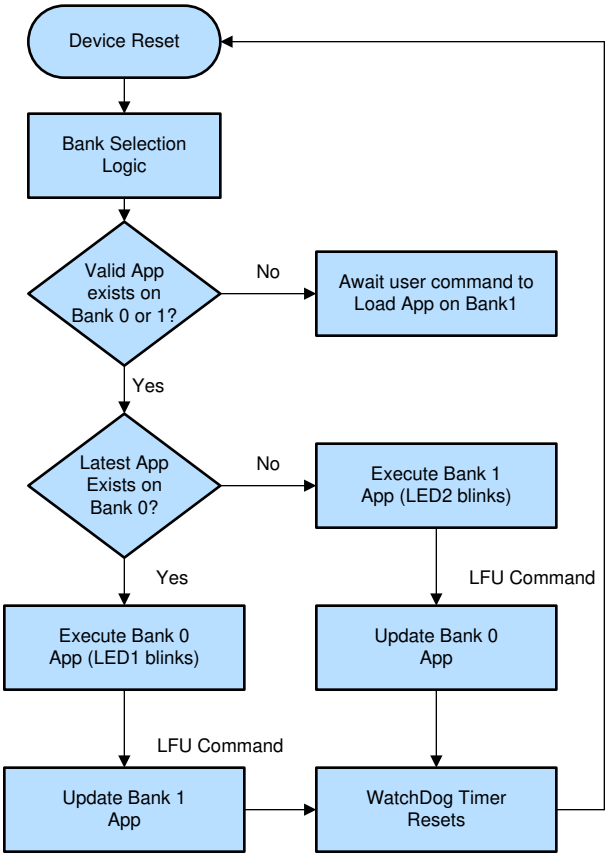

图 6-15. LFU 代码流程图

6.4 限制和疑难解答

- 用户需要注意，由于组选择逻辑位于 Bank0 中，如果在 Bank0 上更新应用时闪存损坏，则位于 Bank0 上其他扇区的静态内容也有可能会损坏。其中可能包括组选择逻辑 + SCI 闪存内核 + 闪存 API（如果从闪存运行）。这时用户必须重复执行静态代码编程涉及的步骤，使系统再次运行。
- 根据节 6.2 对闪存内核进行编程时，闪存的擦除设置应设为“Necessary Sectors Only”，否则，在 BANK1 上对内核进行编程时，BANK1 上的应用将被擦除。

7 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 * (March 2020) to Revision A (August 2021)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式。	2

重要声明和免责声明

TI 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安全、安保或其他要求。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 (https:www.ti.com/legal/termsofsale.html) 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。

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

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