Design Guide: TIDM-02011

具有 C2000™ 实时 MCU 的实时固件更新参考设计

说明

该参考设计说明了两个 C2000™ 实时 MCU（其中一个具有支持 LFU 的硬件特性）上无需器件复位的实时固件更新 (LFU)。LFU 在 C28x CPU 和控制律加速器 (CLA) 上都有说明。此设计使用的软件可帮助用户缩短产品上市时间。对于与服务器电源单元 (PSU) 类似、需要停机时间尽可能短的高可用性系统，无器件复位的 LFU 是一项重要考虑因素。该参考设计还提供了一个示例来说明固件无线升级 (FOTA) 功能。

资源

TIDM-02011	设计文件夹
TIDM-DC-DC-BUCK	产品文件夹
TMS320F28003x、TMS320F28004x	产品文件夹
BOOSTXL-BUCKCONV	产品文件夹
LAUNCHXL-F280039C、 LAUNCHXL-F280049C	产品文件夹
C2000WARE-DIGITALPOWER-SDK	软件文件夹

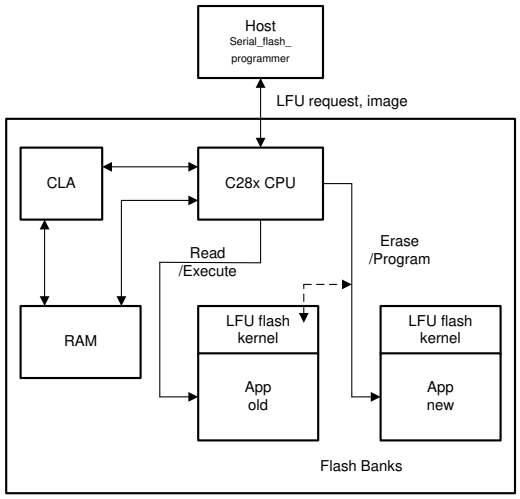
咨询我们的 TI E2E™ 支持专家

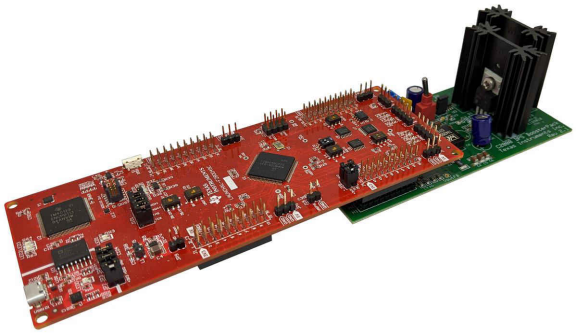
特性

- 基于 C2000™ 数字电源 BoosterPack™ 插件模块的 LFU 参考设计
- 展示无需器件复位的 LFU 切换的参考软件
- 说明在 MCU 上使用 LFU 硬件特性的 LFU 示例
- 说明无实时中断损失的无缝切换的 LFU 示例
- C28x CPU 和 CLA 的 LFU 示例
- 编译器 LFU 支持集成嵌入式应用程序二进制接口 (EABI) 输出格式
- FOTA 示例

应用

- 商用网络和服务器 PSU





English Document: TIDUEY4

1 系统说明

对于服务器电源、计量等类似应用而言，系统需要持续运行以减少停机时间。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。还有一种备选方法是实时固件更新 (LFU)，在系统运行期间仍可更新固件。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。

本参考指南详细介绍了在 TMS320F28003x 或 TMS320F28004x 器件上使用两个闪存组执行的无器件复位 LFU，详述了所涉及的具体挑战以及如何解决这些问题。*C2000™ 数字电源降压转换器 BoosterPack* 参考设计实现了 LFU。本文档说明了在 C28x CPU 或 CLA 上运行主控制循环的 LFU 功能。

1.1 关键系统规格

表 1-1. 关键系统规格

参数	规格
LFU 切换时间	LFU 切换必须在可用的空闲时间内完成。LFU 切换时间表示禁用中断的时间。空闲时间表示中断服务例程 (ISR) 之间的最长间隔。它将取决于系统参数，如中断率和 ISR CPU 负载

2 系统概览

2.1 方框图

图 2-1 显示了基于 LFU 的系统的方框图。

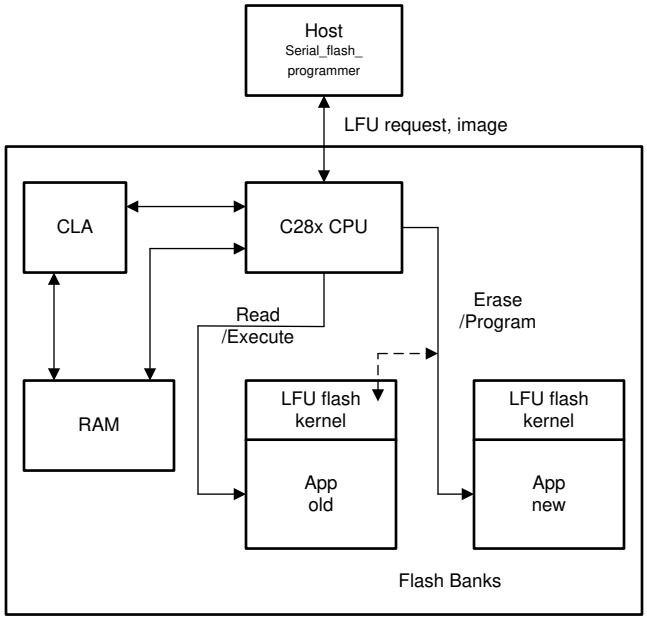


图 2-1. TIDM-02011 方框图

2.2 设计注意事项

2.2.1 构建块

LFU 设计由许多构建块组成：发出 LFU 命令的桌面主机应用程序、与主机通信并启用 LFU 的目标器件闪存上的自定义引导加载程序、将主机连接到目标的通信外设（例如，SCI/UART、CAN、I2C 等）、要下载和执行的与 LFU 兼容的应用程序、具有 LFU 支持的编译器、具有 LFU 相关硬件支持的 MCU，以及具有多个物理上分隔的闪存组的闪存。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。

2.2.2 闪存分区

图 2-2 显示了如何对双组闪存进行分区。每组中的两个扇区分配给自定义引导加载程序，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）闪存组已编程，以及哪个组包含较新的应用程序固件版本。这暗示了该函数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。
- 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。
- KEY – 如果此位置包含特定模式，则认为组中的固件有效。

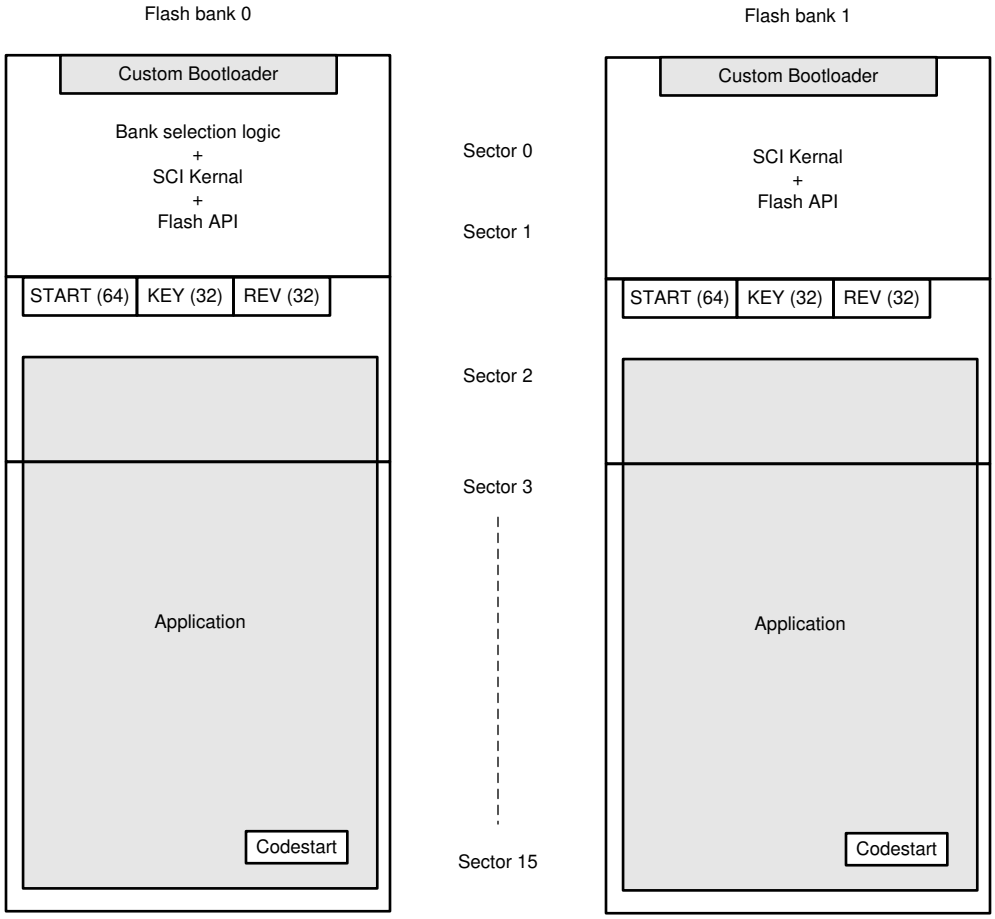


图 2-2. 双闪存组分区

2.2.3 LFU 切换概念

为 LFU 准备固件时的关键注意事项是运行连续性和 LFU 切换时间，运行连续性是通过状态的持续性来实现的，这意味着在固件升级期间将 RAM 中公用的静态变量和全局变量保存在相同的地址，并避免在新固件生效时重新初始化这些变量。对 LFU 的编译器支持用于实现状态的持续性。

激活新固件涉及从旧固件分支到新固件的 LFU 入口点、执行编译器的 LFU 初始化例程、到达新映像的 main() 内部以及执行任何其他初始化。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 [2]** 时，LFU 会被简化，其中任一闪存组都可以映射到固定地址空间，被视为*运行的*闪存组。*未运行的*闪存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，用户需要跟踪应用程序固件将驻留的闪存组，并在链接器命令文件中进行必要的分配和调整。

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），因此重新初始化所有中断向量不太现实。通常，只使用其中一小部分，其余的分配给默认向量。F28003x 器件包含 LFU 特定的硬件特性（中断向量交换、RAM 块交换），可缩短 LFU 切换时间。

如果阵列大小发生变化或向结构中添加了变量，则用户需进行适当管理，即在开发周期的早期使用 pragma 指令将阵列和结构放置在固定位置，但要有足够的余量来满足它们在将来固件中的潜在增长。如果采用这种方法，则只需要初始化新增字段。

2.2.4 应用程序 LFU 流程

图 2-3 显示了 LFU 软件流程图。器件复位后，总是从组选择逻辑开始执行，该逻辑将根据固件版本字段确定要从哪个闪存组执行，并将控制权传递给相应的应用程序。在执行必要的系统初始化和启用中断之后，实时控制循环在与特定中断向量相对应的 ISR 内执行。在 ISR 之间的空闲时间内，将执行由较低优先级函数组成的后台循环。如果主机发出 LFU 命令，它会在 MCU 中触发 SCI 接收中断，相应的 ISR（优先级低于控制循环 ISR）会执行和识别主机命令请求。

在后台循环中，解析此命令，识别 LFU 请求，将控制权传递给自定义引导加载程序（即 SCI 闪存内核），从而可以从主机下载新的应用程序映像并对相应的闪存组进行编程。如果闪存组 1 上的应用程序正在运行，则控制权将传递给组 1 上的 SCI 闪存内核，以便对组 0 进行编程。一旦新应用程序映像位于闪存中，就可以开始切换到新固件的过程。

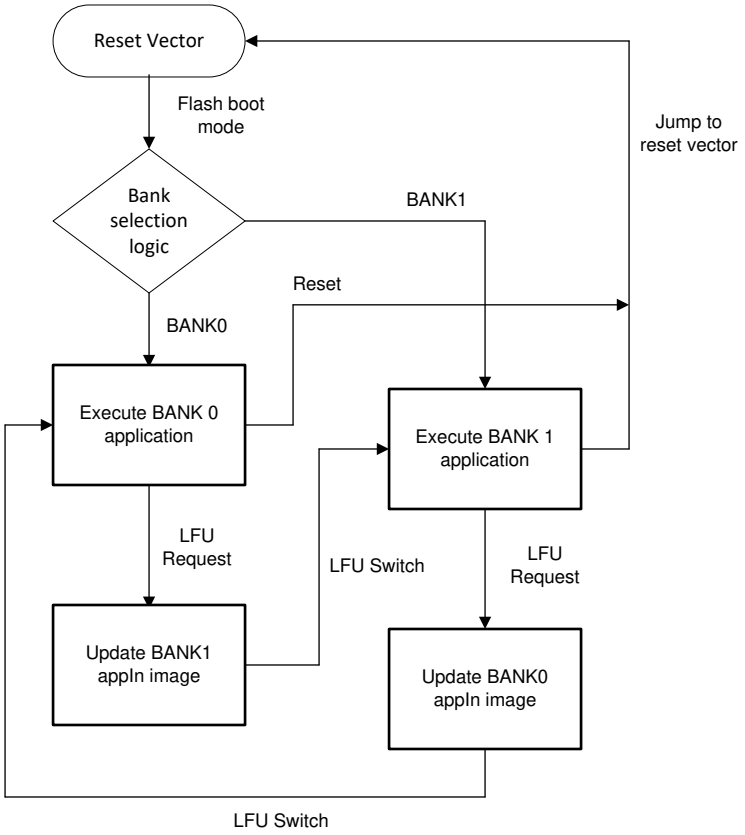


图 2-3. LFU 软件流程图

3 硬件、软件、测试要求和测试结果

3.1 硬件要求

用户需要以下组件：

1. F28003x Launchpad (LAUNCHXL-F280039C)

   或 F28004x Launchpad (LAUNCHXL-F280049C)
2. BoosterPack (BOOSTXL-BUCKCONV)
3. 用于将 Launchpad 连接到计算机的 Micro-USB 转 USB 电缆
4. 9V、2A 直流工作台电源
5. 两条香蕉头转裸线电缆
6. 示波器或类似仪器（例如，Saleae 逻辑分析仪）
7. 万用表

3.2 软件要求

用户需要以下软件：

1. Code Composer Studio™ (CCS) 软件 v10.1.0 或更高版本，运行 TI 编译器 C2000 v21.6.0.LTS 或更高版本。
2. DigitalPower SDK v4.01.00.00 或更高版本，包括用于此设计的软件。该软件在适用于 C2000 MCU 的 DigitalPower 软件开发套件 (SDK) 中提供。

3.2.1 软件包内容

表 3-1 列出了运行 LFU 示例所需的关键目标可执行文件，还提到了工程位置、构建输出可执行文件需使用的**特定工程构建配置**，以及需将输出可执行文件放置在何处。

表 3-1. F28004x 示例的软件包内容

文件/文件夹名称	运行控制循环的位置？	工程构建配置	说明
flashapi_ex2_sci_kernel.out	不适用	BANK0_LDFU_ROM	构建 **flashapi_ex2_sci_kernel** 工程后的输出文件，这是自定义引导加载程序（此工程将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28004x\examples\flash\CCS）
flashapi_ex2_sci_kernel.out	不适用	BANK1_LDFU_ROM	同上
buck_F28004x_lfuBANK0FLASH.txt	CPU	BANK0_FLASH	**buck_F28004x_lfu** 工程的输出文件经过转换后变为 .txt，该工程是应用程序（将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28004x\ccs） 工程是用编译器预定义符号 BUCK_CONTROL_RUNNING_ON_CPU 构建的 将生成的 .txt 文件复制到 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
buck_F28004x_lfuBANK1FLASH.txt	CPU	BANK1_FLASH	同上
buck_F28004x_lfu_controlloopBANK0FLASH.txt	CPU	BANK0_FLASH	**buck_F28004x_lfu_controlloop** 工程的输出文件经过转换后变为 .txt，该工程是应用程序（将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28004x\ccs） 工程是用编译器预定义符号 BUCK_CONTROL_RUNNING_ON_CPU 构建的 将生成的 .txt 文件复制到 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
buck_F28004x_lfu_controlloopBANK1FLASH.txt	CPU	BANK1_FLASH	同上
buck_F28004x_lfuBANK0FLASH_cla.txt	CLA	BANK0_FLASH	**buck_F28004x_lfu** 工程的输出文件经过转换后变为 .txt，该工程是应用程序（将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28004x\ccs） 工程是用编译器预定义的符号 BUCK_CONTROL_RUNNING_ON_CLA 构建的。对生成的 .txt 重命名以包含 “_cla” 将生成的 .txt 文件复制到 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
buck_F28004x_lfuBANK1FLASH_cla.txt	CLA	BANK1_FLASH	同上
buck_F28004x_lfu_controlloopBANK0FLASH_cla.txt	CLA	BANK0_FLASH	**buck_F28004x_lfu_controlloop** 工程的输出文件经过转换后变为 .txt，该工程是应用程序（将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28004x\ccs） 工程是用编译器预定义的符号 BUCK_CONTROL_RUNNING_ON_CLA 构建的。对生成的 .txt 重命名以包含 “_cla” 将生成的 .txt 文件复制到 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
buck_F28004x_lfu_controlloopBANK1FLASH_cla.txt	CLA	BANK1_FLASH	同上

表 3-1. F28004x 示例的软件包内容 (continued)

文件/文件夹名称	运行控制循环的位置？	工程构建配置	说明
serial_flash_programmer_appln.exe	.exe	-	这是主机端串行闪存编程器可执行文件，用于将应用程序加载至目标器件上的闪存 它位于 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer

表 3-2. F28003x 示例的软件包内容

文件/文件夹名称	运行控制循环的位置？	工程构建配置	说明
flash_kernel_ex3_sci_flash_kernel.out	不适用	BANK0_LDFU	构建 **flash_kernel_ex3_sci_flash_kernel** 工程后的输出文件，这是自定义引导加载程序（此工程将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28003x\examples\flash\CCS）
flash_kernel_ex3_sci_flash_kernel.out	不适用	BANK1_LDFU	同上
buck_F28003x_lfuBANK0FLASH.txt	CPU	BANK0_FLASH	**buck_F28003x_lfu** 工程的输出文件经过转换后变为 .txt，该工程是应用程序（将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28003x\ccs） 工程是用编译器预定义符号 BUCK_CONTROL_RUNNING_ON_CPU 构建的 将生成的 .txt 文件复制到 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
buck_F28003x_lfuBANK1FLASH.txt	CPU	BANK1_FLASH	同上
buck_F28003x_lfuBANK0FLASH_cla.txt	CLA	BANK0_FLASH	**buck_F28003x_lfu** 工程的输出文件经过转换后变为 .txt，这是应用程序（此工程将位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28003x\ccs） 工程是用编译器预定义符号 BUCK_CONTROL_RUNNING_ON_CLA 构建的。对生成的 .txt 重命名以包含“_cla” 将生成的 .txt 文件复制到 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
buck_F28003x_lfuBANK1FLASH_cla.txt	CLA	BANK1_FLASH	同上
serial_flash_programmer_appln.exe	.exe	-	这是主机端串行闪存编程器可执行文件，用于将应用程序加载至目标器件上的闪存 它位于 <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer

备注

buck_F28003x_lfu 工程还包含 BANK0_FLASH_BANK10COPY 构建配置，这是一个备用配置，其中应用程序始终构建为加载到组 1 中并从组 0 运行。在该配置中，BANK1_TO_0COPY 是一个预定义符号，它允许实现修改后的功能。

同样，flash_kernel_ex3_sci_flash_kernel 工程也包含 BANK0_LDFU_BANK1TO0COPY 构建配置，这是支持上述用例的备用配置，即应用程序始终构建为加载到组 1 中并从组 0 运行。在该配置中，BANK1_TO_0COPY 是一个预定义符号，它允许实现修改后的功能。

这允许开发人员为 LFU 构建映像而无需知道映像将驻留在哪个组中。这样做的缺点是组 1 到组 0 的复制，这需要在激活新映像之前发生。该复制是由闪存内核完成的。这会占用一定的时间，在此期间应用程序无法运行。

3.2.2 软件结构

图 3-1 显示了 F28003x 的 LFU 解决方案的软件目录结构。F28004x 存在相同的结构。解决方案文件夹包含一个 tidm_02011 文件夹，其中容纳了 TIDM-DC-DC-BUCK 解决方案的 LFU 实现。存在以下文件夹：

- /lfu 包含特定于 LFU 的源文件和头文件
- /drivers 包含 HAL（硬件抽象层）源文件和头文件
- /ccs 包含 CCS 工程规格
- /cmd 包含链接器命令文件
- /buck 包含 buck_main.c、buck_clatasks.cla、main.syscfg 以及其他头文件
- /examples 包含自定义引导加载程序（SCI 闪存内核）

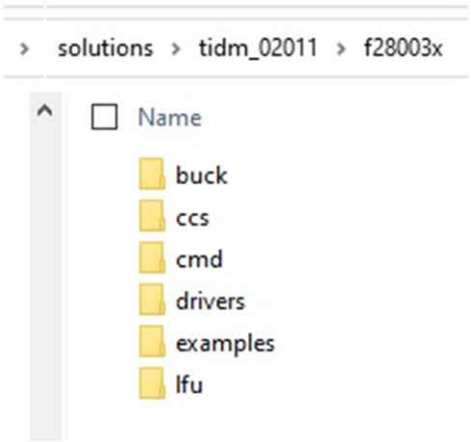


图 3-1. F28003x 的 LFU 解决方案软件目录结构

3.3 TIDM-DC-DC-BUCK 简介

TIDM-DC-DC-BUCK 解决方案说明了如何在 C2000 MCU 上实现数字电源控制。它在 TIDM-02011 中用于说明 LFU。此解决方案的显著特点如下所述：

- 在时钟频率为 100MHz 的 TMS320F28004x 双组闪存 MCU 上运行

  该参考设计 TIDM-02011 还添加了对上述 TIDM-DC-DC-BUCK 示例的支持，以在时钟频率为 120MHz 的 TMS320F28003x MCU（最多包含 3 个闪存组）上运行。
- 控制循环 ISR 以 200kHz 的频率运行。请参阅 buck_settings.h 中的 BUCK_DRV_EPWM_SWITCHING_FREQUENCY
- 当控制循环 ISR 未执行时，会运行一系列后台任务：
  - 它在 A、B 和 C 型任务之间轮换
  - A 型任务以 1kHz 的频率运行（在 A1、A2、A3 函数之间轮换）
  - B 型任务以 100Hz 的频率运行（在 B1、B2、B3 函数之间轮换）

- C 型任务以 10Hz 的频率运行（在 C1、C2、C3 函数之间轮换）
- ISR 以及选择后台任务函数从 RAM 运行。

3.4 测试设置

本文档的其余部分演示了假定使用 LaunchPad 的测试结果。如果用户希望改用 ControlCard，则 CONTROLCARD 需要在构建应用程序工程之前成为应用程序工程中的预定义符号。

3.4.1 使用 CCS 将自定义引导加载程序和应用程序加载到闪存

1. 通过在引导选择开关上将 GPIO24 移到 OFF (1) 并将 GPIO32 移到 OFF (1)，将 LaunchPad 设置为闪存引导模式。有关详细信息，请参阅 C2000™Piccolo™F28004x 系列 LaunchPad™ 开发套件用户指南或 *C2000™ Piccolo™ F28003x 系列 LaunchPad™ 开发套件*。
2. 将 Micro USB 电缆连接到计算机和 Launchpad，以使电路板通电。
3. 使用 CCS 将自定义引导加载程序下载到闪存组 0。
   a. 对于 F28004x，使用 CCS 对自定义引导加载程序（位于 <C2000Ware_DigitalPower_SDK_path>\c2000ware\driverlib\f28004x\examples\flash\CCS）的 Bank0_LDFU_ROM 构建配置 .out 进行编程。
   b. 对于 F28003x，使用 CCS 对自定义引导加载程序（位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28003x\examples\flash\CCS）的 Bank0_LDFU 构建配置 .out 进行编程。

   *对于此步骤，使用可擦除整个闪存的目标配置文件。*有关详细信息，请参阅图 3-2。

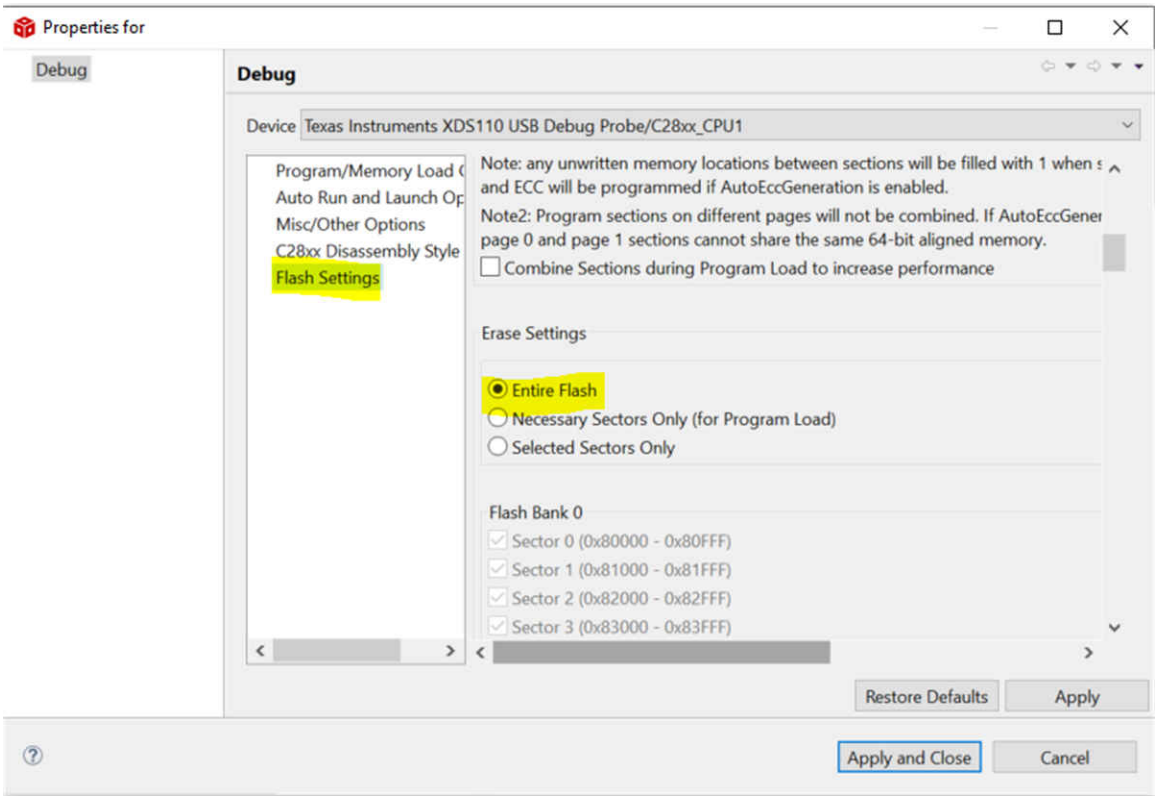


图 3-2. 带有用于擦除整个闪存的擦除设置的目标配置文件

4. 在闪存中编程自定义引导加载程序后，在 CCS 中点击“Run”，然后从 Windows 命令提示符执行以下命令：
   - cd <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
   - 在 F28004x 上：serial_flash_programmer_**appln**.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28004x_lfuBANK1FLASH.txt -b 9600 -p COM11

- 在 F28003x 上：serial_flash_programmer_**appln**.exe -d f28003x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28003x_lfuBANK1FLASH.txt -b 9600 -p COM11
- 在上述命令中，需要 f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt，但不使用它。这是因为使用了 serial_flash_programmer_appln.exe，它只对应用程序固件进行编程，而不对内核进行编程。
- 另请注意，在上述命令中，“COM11”必须替换为与相关连接关联的特定 COM 端口。这可以通过“Device Manager”–“Ports”–“XDS110 Class Application/User UART”来识别
- 输入“8 – Live DFU” – 这会将应用程序固件的 Bank1_Flash 构建配置编程到闪存组 1
- 完成后输入“0 – Done”

5. 此时，自定义引导加载程序在闪存组 0 上进行编程，应用程序在闪存组 1 上进行编程。
6. 接下来，自定义引导加载程序在闪存组 1 上进行编程，应用程序在闪存组 0 上进行编程。使用 CCS 将自定义引导加载程序下载到闪存组 1。对于 F28004x，使用 CCS 对自定义引导加载程序（位于 <C2000Ware_DigitalPower_SDK_path>\c2000ware\driverlib\f28004x\examples\flash\CCS）的 Bank1_LDFU_ROM 构建配置 .out 进行编程。对于 F28003x，使用 CCS 对自定义引导加载程序（位于 <C2000Ware_DigitalPower_SDK_path>\solutions\tidm_02011\f28003x\examples\flash\CCS）的 Bank1_LDFU 构建配置 .out 进行编程。

   *对于此步骤，使用仅擦除必要扇区而不是整个闪存的目标配置文件。*有关详细信息，请参阅图 3-3。

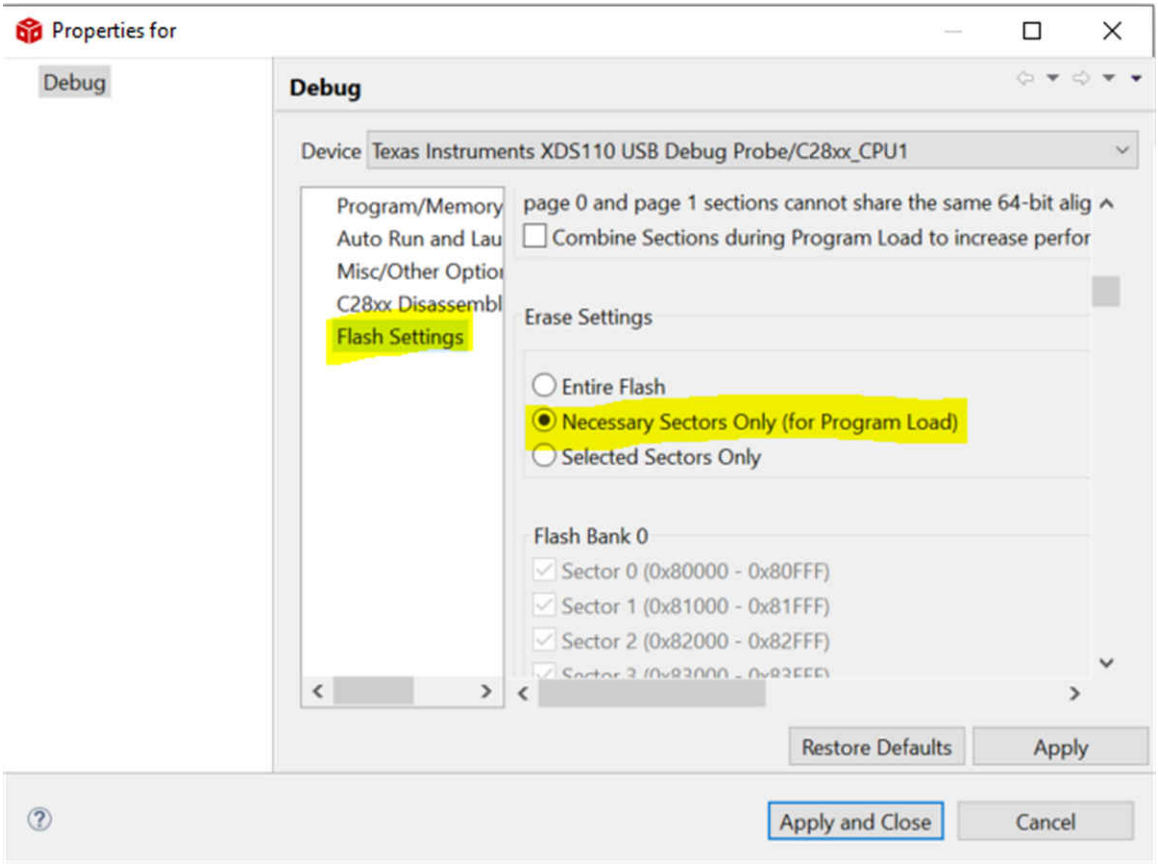


图 3-3. 带有仅擦除必要扇区的擦除设置的目标配置文件

7. 在闪存中编程自定义引导加载程序后，在 CCS 中点击“Run”，然后从 Windows 命令提示符执行以下命令：
   - cd <C2000Ware_DigitalPower_SDK_path>\c2000ware\utilities\flash_programmers\serial_flash_programmer
   - 在 F28004x 上：serial_flash_programmer_**appln**.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28004x_lfuBANK0FLASH.txt -b 9600 -p COM11

- 在 F28003x 上：serial_flash_programmer_**appln**.exe -d f28003x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28003x_lfuBANK0FLASH.txt -b 9600 -p COM11
- 在上述命令中，需要 f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt，但不使用它。这是因为使用了 serial_flash_programmer_appln.exe，它只对应用程序固件进行编程，而不对内核进行编程。
- 另请注意，在上述命令中，“COM11”必须替换为与相关连接关联的特定 COM 端口。这可以通过“Device Manager”–“Ports”–“XDS110 Class Application/User UART”来识别
- 输入“8 – Live DFU” – 这会将应用程序固件的 Bank0_Flash 构建配置编程到闪存组 0
- 完成后输入“0 – Done”

8. 重置电路板。现在，两个闪存组都有自定义引导加载程序和应用程序映像。

3.5 测试结果

3.5.1 在 CPU 上运行控制循环时运行 LFU 演示

使用自定义引导加载程序和应用程序映像对器件的两个闪存组进行编程后，LFU 演示现在可以在独立模式下运行。

1. 切换到闪存引导模式（此时应已处于此模式）。
2. 将 BoosterPack 连接至 LaunchPad，如图 3-4 所示。LaunchPad 位于 BoosterPack 上方。LaunchPad 接头 J5-J7 连接到 BoosterPack 接头 H1-H2。LaunchPad 接头 J6-J8 连接到 BoosterPack 接头 H3-H4。这表示工程的 main.syscfg 中的“Launchpad Site2”（“Powerstage Parameters”–“Hardware”–“Launchpad Site”）。

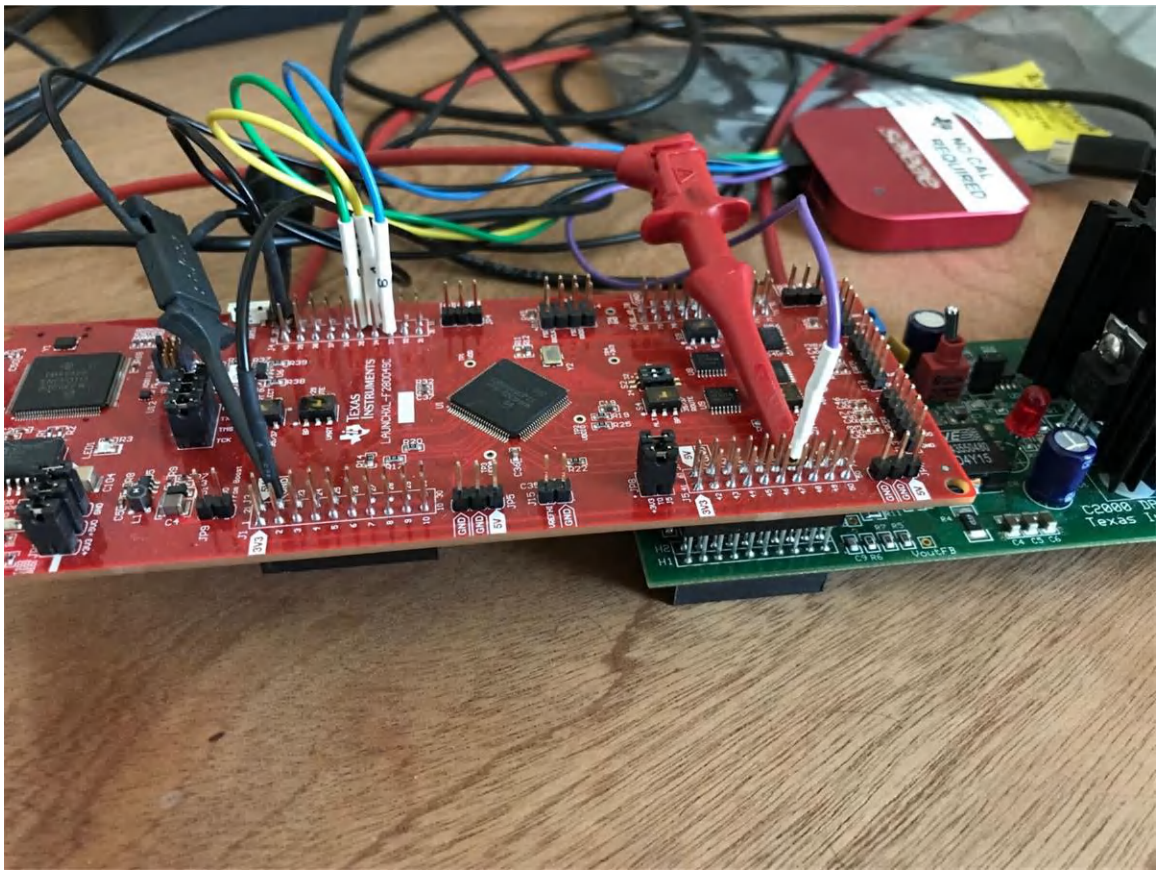


图 3-4. 将 BoosterPack 连接至 F28004x LaunchPad

3. 以正确的极性（[JP1 +]Vin 和 [JP1 GND]GND）将香蕉头转裸线电缆从直流工作台电源连接至位于 JP1 处的 BoosterPack。
4. 将直流工作台电源设置为输出 9V。启用电源。
5. 将 SW1 转动至 ON 位置。

6. 连接示波器（或类似设备）以感测输出电压以及 2 个附加信号 – ISR CPU 负载以及 LFU 切换时间。可根据以下说明进行连接。同时使用万用表监测稳压输出电压。
   - 输出电压 – 在 LaunchPad 的接头 J7 上，信号 67。这表示稳压输出电压。
   - ISR CPU 负载 - 在 LaunchPad 的接头 J2 上，信号 15。这表示控制循环 ISR 的 CPU 负载。
   - LFU 切换时间 - 在 LaunchPad 的接头 J2 上，信号 14。这表示从旧应用程序映像到新应用程序映像执行 LFU 所花的时间。
7. 将 Micro USB 电缆连接到计算机和 Launchpad，以使电路板通电。请注意，务必在直流工作台电源已经为 BoosterPack 供电且 SW1 接通**后**执行此步骤。
8. 这将导致控制循环 ISR 开始执行。默认情况下，TIDM-DC-DC-BUCK 的 Build2 将运行。这是使用 VMC（电压模式控制）的闭环电压调节。但是，软件初始化步骤尚未完成，因此后台任务尚未执行。这是 SCI 自动波特率锁定步骤。若要启用 SCI 自动波特率锁定，请从 Windows 命令提示符执行命令，如下所示：
   - 在 F28004x 上：serial_flash_programmer_**appIn**.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28004x_IfuBANK1FLASH.txt -b 9600 -p COM11
   - 在 F28003x 上：serial_flash_programmer_**appIn**.exe -d f28003x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28003x_IfuBANK1FLASH.txt -b 9600 -p COM11
   - 发出此命令后，暂时不要选择任何选项
   - 这对于自动波特率锁定来说就足够了，后台任务将开始执行。发生这种情况时，感测操作将开始，稳压输出电压在万用表上将显示为 1V（反映 2V 的稳压输出电压）。
   - 如果 BoosterPack 上有一个红色 LED，它将亮起。
   - LaunchPad 上的红色 LED4 (GPIO23) 由 buck_main.c 中 B1() 内的 BUCK_HAL_toggleRunLed() 控制。由于这是一个后台任务函数，它将开始切换。应用程序从组 0 运行时的切换频率值设置为比从组 1 运行时要小。
9. 上述编程步骤首先在闪存组 1 上对 TIDM-DC-DC-BUCK 应用程序进行编程，然后在闪存组 0 上进行编程。但两者的固件版本都是 0xFFFE。当它们相等时，自定义引导加载程序中的组选择逻辑将认为编号较小的组（即闪存组 0）是最新的应用程序版本，并将执行此版本。
   - 当代码从组 1 运行时，LaunchPad 上的绿色 LED5 (GPIO34) 亮起；当代码从组 0 运行时，此 LED 将熄灭。由于代码现在正在从组 0 运行，此 LED 将熄灭。
10. 在步骤 8 中，发出一条命令以启用 SCI 自动波特率锁定，并且命令提示符正在等待用户输入
    - 输入“8 – Live DFU” – 这会将 TIDM-DC-DC-BUCK 应用程序的 Bank1_Flash 构建配置编程到闪存组 1
    - 完成后输入“0 – Done”
    - 当新映像下载到闪存时，LaunchPad 上的 LED4 将不会切换，因为在映像下载过程中后台任务会停止。
    - 将 Bank1_Flash 映像编程到闪存组 1 后，此映像将自动开始执行。用户现在将注意到以下情况：
      - LaunchPad 上的绿色 LED5 亮起。这是因为代码现在正在从组 1 运行。
      - BoosterPack 上的红色 LED 仍亮起。但 LaunchPad 上的红色 LED4 未切换。这是因为后台任务尚未启用。与上述类似，可以发出命令以启用 SCI 自动波特率锁定。可以使用以下命令
        - 在 F28004x 上：serial_flash_programmer_**appIn**.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28004x_IfuBANK0FLASH.txt -b 9600 -p COM11
        - 在 F28003x 上：serial_flash_programmer_**appIn**.exe -d f28003x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28003x_IfuBANK0FLASH.txt -b 9600 -p COM11
        - 发出此命令后，暂时不要选择任何选项
      - 在整个过程中，输出电压继续保持在 1V。**这是因为在 LFU 之后没有发出器件复位，并且在中断之间的空闲时间内发生了从旧应用程序固件到新应用程序固件的切换。**
11. 对于下一次 LFU 切换，用户可以将示波器设置为根据“LFU 切换时间”信号触发。这将允许用户直观地检查切换发生的时间、需要多长时间，等等。
    - 在步骤 8 中，发出一条命令以启用 SCI 自动波特率锁定，并且命令提示符正在等待用户输入
    - 输入“8 – Live DFU” – 这会将 TIDM-DC-DC-BUCK 应用程序的 Bank0_Flash 构建配置编程到闪存组 0
    - 完成后输入“0 – Done”

- 当新映像下载到闪存时，LaunchPad 上的 LED4 将不会切换，因为在映像下载过程中后台任务会停止。
- 将 Bank0_Flash 映像编程到闪存组 0 后，此映像将自动开始执行。用户现在将注意到以下情况：
  - LaunchPad 上的绿色 LED5 熄灭。这是因为代码现在正在从组 0 运行
  - BoosterPack 上的红色 LED 仍亮起。但 LaunchPad 上的红色 LED4 未切换。这是因为后台任务尚未启用。与上述类似，可以发出命令以启用 SCI 自动波特率锁定。可以使用以下命令
    - 在 F28004x 上：serial_flash_programmer_**appln**.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28004x_lfubank1FLASH.txt -b 9600 -p COM11
    - 在 F28003x 上：serial_flash_programmer_**appln**.exe -d f28003x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28003x_lfubank1FLASH.txt -b 9600 -p COM11
    - 发出此命令后，暂时不要选择任何选项
  - 在整个过程中，输出电压继续保持在 1V。**这是因为在 LFU 之后没有发出器件复位，并且在中断之间的空闲时间内发生了从旧应用程序固件到新应用程序固件的切换**。
  - 请参考图 3-5 以了解详细信息，并直观确认上述声明。所示的信号为 LFU 切换、CPU ISR 负载和稳压输出电压。

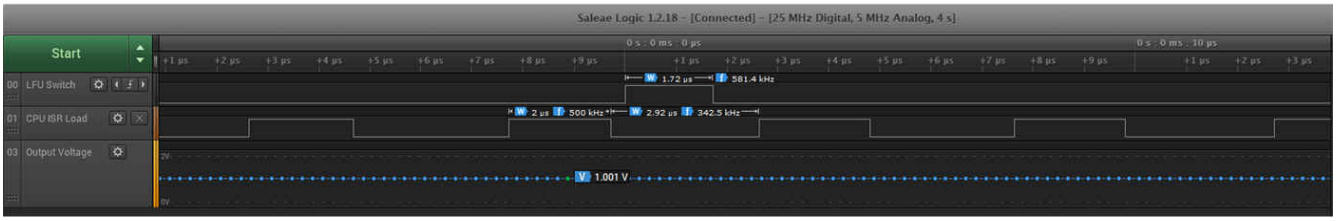


图 3-5. LFU 切换时间（控制循环在 CPU 上运行）

12. 根据需要重复上述步骤。当组 0 处于运行状态时，发出 LFU 命令以进行编程并切换到组 1。当组 1 处于运行状态时，发出 LFU 命令以进行编程并切换到组 0。

3.5.2 在 CLA 上运行控制循环时运行 LFU 演示

如果用户希望为在 CLA 上运行控制循环的情况下构建的工程生成 .txt 文件，唯一需要做的是将预定义的编译器符号 BUCK_CONTROL_RUNNING_ON_CPU 更改为 BUCK_CONTROL_RUNNING_ON_CLA。有关详细信息，请参阅图 3-6。生成的 .txt 文件以 _cla 重命名，以区别于在 CPU 上运行的相应 .txt 文件。

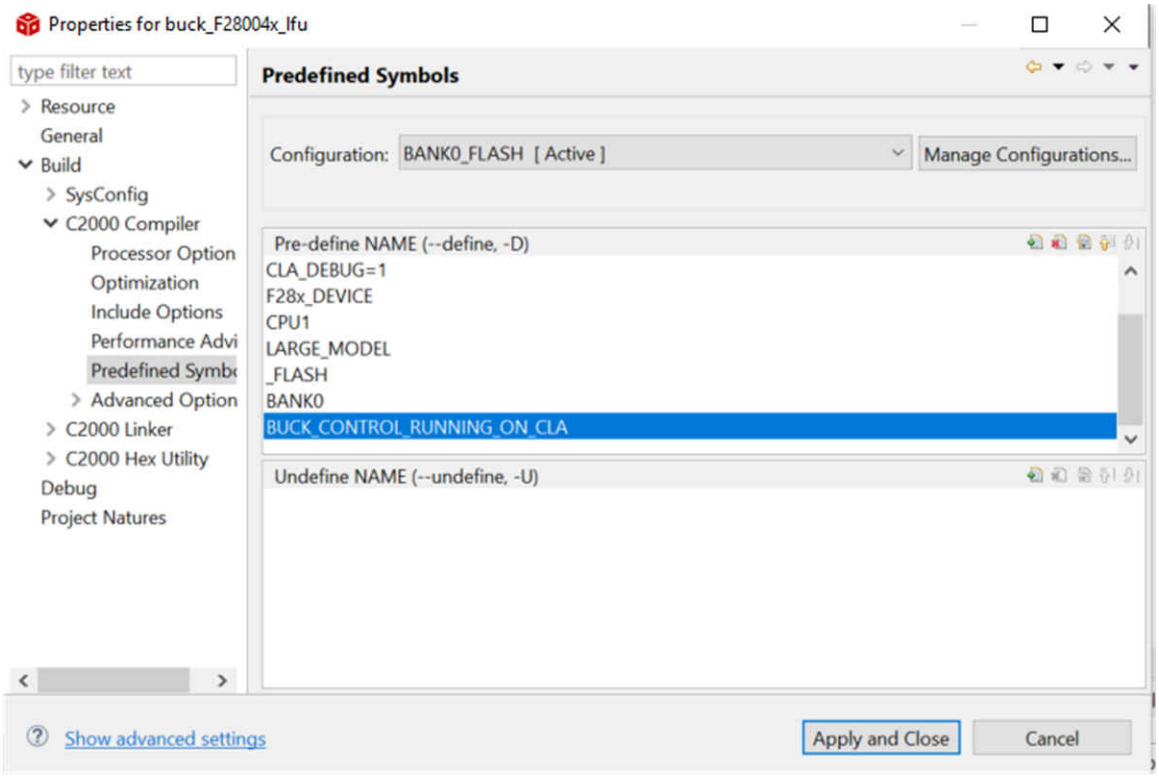


图 3-6. 用于工程 CLA 构建的预定义符号

控制循环在 CPU 上运行时用户运行过 LFU 演示后，则控制循环在 CLA 上运行时运行 LFU 演示就会很简单，只需注意以下几点：

1. 如果器件已经包含与 CPU 端控制循环相对应的应用程序文件，则可以使用与上一节相同的 LFU 命令执行此更新，但与 CLA 构建相对应的已更新的 .txt 名称除外（见表 3-1）。例如，当从 BANK0_FLASH 更新到 BANK1_FLASH 时，请执行以下命令：
   - 在 F28004x 上：serial_flash_programmer_**appIn**.exe -d f28004x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28004x_lfuBANK1FLASH_**cla**.txt -b 9600 -p COM11
   - 在 F28003x 上：serial_flash_programmer_**appIn**.exe -d f28003x -k f28004x_fw_upgrade_example\flashapi_ex2_sci_kernel-CPU1-RAM.txt -a buck_F28003x_lfuBANK1FLASH_**cla**.txt -b 9600 -p COM11
2. CLA 设置函数在器件复位时（而不是在 LFU 切换之后）出现在 main() 中，因此，在运行 LFU 后复位器件很重要，可为执行此初始化提供便利。例如，在 CPU 上运行控制循环时，假设最后更新了 BANK0_FLASH。这意味着组 0 上的固件正在执行。因此，用户需要执行 LFU 命令来更新 BANK1_FLASH（使用 CLA 可执行文件）。LFU 更新完成后，需要复位器件。***器件复位只需执行一次***。

   然后，用户可以使用 CLA 固件可执行文件，在不复位器件的情况下执行其他 LFU 更新。
3. 图 3-7 和图 3-8 演示控制循环在 CLA 上运行时的 LFU 切换。具有一个每 1ms 切换一次 GPIO 的后台任务，该 GPIO 在 Launchpad 的接头 J4（信号 33）上提供。请注意，在这种情况下，LFU 切换通常可能与 ISR 执行重叠，因为 ISR 在 CLA 上执行，而 LFU 在 CPU 上执行。这通常不会造成问题，但在某些情况下，这是不可接受的。图 3-7 和图 3-8 所示的信号为 LFU 切换、CLA ISR 负载、CLA 后台任务执行和稳压输出电压。

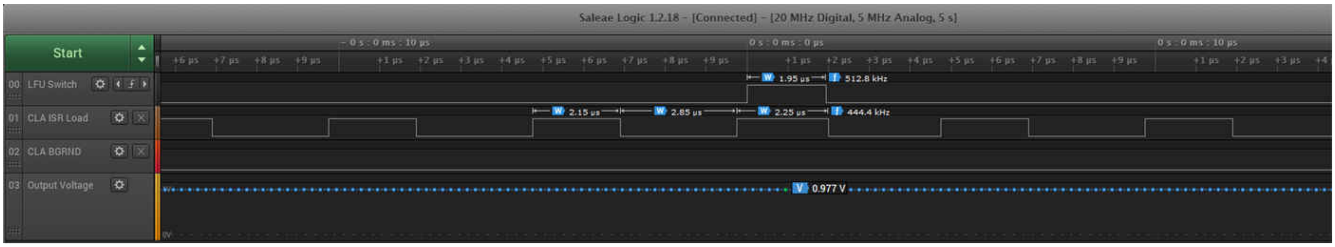


图 3-7. LFU 切换时间（控制循环在 CLA 上运行）

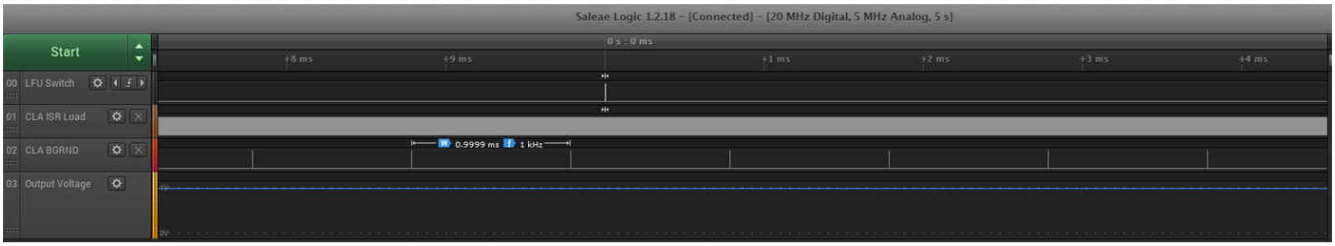


图 3-8. CLA 后台任务

3.5.3 CPU 上的 LFU 流程

在上面的 LFU 演示中，使用了两个应用程序映像。一个在闪存组 0 上运行，另一个在闪存组 1 上运行。BANK0_FLASH 构建配置被视为“旧”或“参考”固件，而 BANK1_FLASH 构建配置被视为“新”固件。这两个应用程序在其他方面都是相同的。它们之间没有源代码差异；但是，新固件有 25 个已定义和已初始化的新浮点变量。这两个应用程序通过 TIDM-DC-DC-BUCK 解决方案工程的两个构建配置来实现 – 即 BANK0_FLASH 和 BANK1_FLASH。顾名思义，BANK0_FLASH 从闪存组 0 执行，BANK1_FLASH 从闪存组 1 执行。这两个构建配置共享相同的源文件，但包含不同的链接器命令文件。此外，在代码中的不同位置，宏“#ifdef BANK0”和“ifdef BANK1”控制执行，它们运行相同的控制循环 ISR 和后台任务。

以下是控制循环在 CPU 上运行时的简要 LFU 流程。

1. 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。此函数检查任一闪存组或这两个闪存组中是否存在有效的应用程序，如果存在一个，则选择较新的版本并转移到该地址（0x8EFF0 或 0x9EFF0）。这些是各个应用程序的 code_start 位置，执行从这里进入 C 运行时初始化例程 (_c_int00)，并进入相应应用程序的 main()。如果这两个闪存组都不包含有效的应用程序，则执行将等待主机完成自动波特率锁定，并等待主机通过 SCI 发送映像。
2. 用户通过 Windows 命令提示符调用 LFU 命令“8 Live DFU”。
3. 目标器件在 SCI 接收中断 ISR 中接收命令 ID“0x700”。
4. 在 main() 中，在后台循环中调用函数 BUCK_LFU_runLFU()。当命令 ID 与“0x700”匹配时，SCI 中断被禁用，执行分支到自定义引导加载程序中 Live DFU (liveDFU()) 函数的地址。如果组 0 中的应用程序正在执行，则分支到组 0 中的自定义引导加载程序（地址 0x81000）。如果组 1 中的应用程序正在执行，则分支到组 1 中的自定义引导加载程序（地址 0x91000）。
5. 自定义引导加载程序中的 liveDFU() 从主机接收应用程序映像并将其编程到闪存中。完成后，执行取决于是否定义了宏 LFU_WITH_RESET。如果是，则看门狗配置为生成复位信号，然后启用，因此发生器件复位。如果未定义宏，则执行分支到**新应用程序映像的 LFU 入口点**。这是组 0 的 0x8EFF8 和组 1 的 0x9EFF8。这与常规的闪存启动入口点不同。
6. 函数 c_int_lfu() 位于组 0 上的 0x8eff8 和组 1 上的 0x9eff8。该函数可在不复位器件的情况下执行 LFU 切换。在该函数中：
   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。因此，它将初始化在 BANK1_Flash 构建配置中定义的 25 个新浮点变量。
   b. 设置一个标志以指示 LFU 正在进行中。在 F28004x 上，这是使用软件变量 lfuSwitch 实现的。在 F28003x 上，这是通过使用 LFU_setLFUCPU() 设置 LFUConfig SysCtl 寄存器的 LFU.CPU 位实现的。
   c. 调用 main()

7. 在 main() 中，初始化进程取决于 LFU 是否正在进行中。这是通过访问 F28003x 上的 LFUConfig SysCtl 寄存器的 LFU.CPU 位或 F28004x 上的 lfuSwitch 实现的。如果该值为 0，则初始化会像发生器件复位一样进行。
8. 如果该值不为 0，则执行有限初始化。首先，执行 init_lfu()。该函数将代码从闪存复制到 RAM，对应于用户已指示需要从 RAM 运行的程序代码。接下来，在 F28003x 上，它使用 Shadow_Interrupt_Register() 更新停用的 PIE 中断向量表。在 F28004x 上，中断向量表交换硬件特性不可用，因此不执行该特性。在 F28003x 上，还会更新一组停用的函数指针。在 F28004x 上，RAM 块交换硬件特性不可用，因此不执行该特性。
9. 接下来，将变量 lfuSwitch_start 设置为 Lfu_switch_wait_for_isr。执行在此等待，直到下一个控制循环 ISR 开始执行，其中 lfuSwitch_start 从 Lfu_switch_wait_for_isr 移到 Lfu_switch_ready_to_switch。**这有助于将 LFU 切换同步到控制循环 ISR 的末端，从而最大限度地利用控制循环中断之间的空闲时间**。
10. 当执行继续时，将发生 LFU 切换步骤。首先，禁用全局中断。在 F28003x 上，执行 PIE 中断向量表交换和 RAM 块交换。在 F28004x 上，每个使用的 PIE 中断向量都需要在此处单独更新。同样，每个函数指针都需要在此处单独更新。然后执行 C28x CPU 端堆栈指针初始化，并重新启用全局中断。**这表示 LFU 切换结束**。
11. 在短时间内禁用全局中断。如果在此期间发生外设中断，它将继续保持锁定状态，并在重新启用全局中断时中断 CPU。这样做是为了避免在以下情况（不太可能发生）下出现不可预测的行为：发生中断，以及在更新向量表之时访问此向量表。
12. 图 3-9 显示了另一个 LFU 用例，其中将在固件升级期间更新控制循环参数。在实践中，这可以使用补偿设计器实时完成，这里提供了这方面的用例进行说明。这对应于 **buck_F28004x_lfu_controlloop** 工程。在此工程中，BANK0_FLASH 构建配置包含对应于较小增益 (Kdc = 4000) 的系数。BANK1_FLASH 构建配置包含对应于较大增益 Kdc=38904 的系数（请参阅 buck.h 中的函数 BUCK_initControlLoopGlobals()）。当启用有源负载时，这会导致 BANK0_FLASH 编译配置的瞬态性能较差，而 BANK1_FLASH 编译配置的瞬态性能更好。如果器件已经包含与 **buck_f28004x_lfu** 工程（或甚至 CLA）对应的应用程序文件，则可以使用与上一节相同的 LFU 命令运行此更新，但与此工程对应的已更新的 .txt 名称除外，如表 3-1 所示。在 **buck_f28004x_lfu_controlloop** 工程中，在器件复位时（而不是在 LFU 切换之后）在 main() 中启用有源负载，因此在运行 LFU 后复位器件非常重要，可为执行此初始化提供便利。***器件复位只需执行一次***。

    然后，用户可以使用 controlloop 工程可执行文件，在不复位器件的情况下执行其他 LFU 更新。

备注

目前，仅在 F28004x 上创建启用了有源负载的 Controlloop 示例。

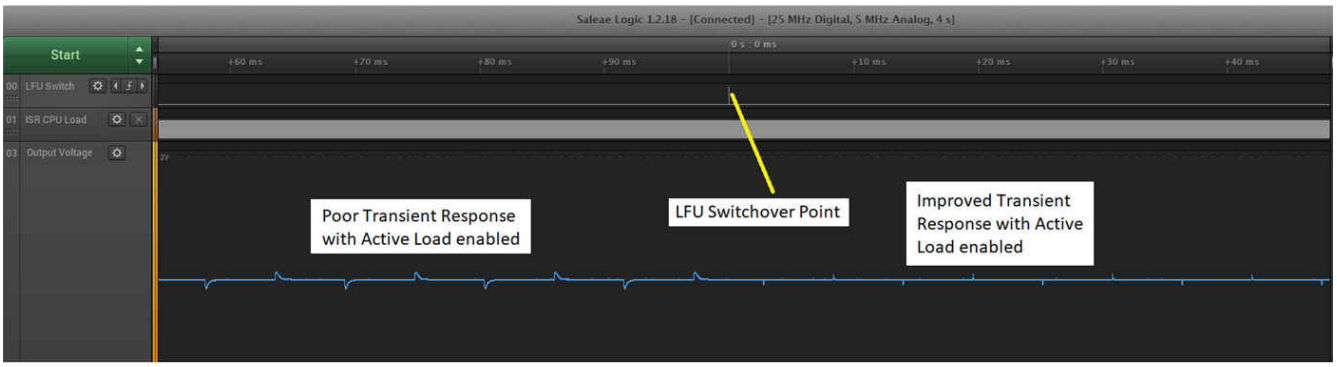


图 3-9. LFU 切换和瞬态性能改善（控制循环在 CPU 上运行）

3.5.4 CLA 上的 LFU 流程

以下是控制循环在 CLA 上运行时的简要 LFU 流程。

1. 闪存分区和高级 LFU 软件流都保持不变。
2. 用户通过 Windows 命令提示符调用 LFU 命令 *8 Live DFU*。
3. 目标器件在 SCI 接收中断 ISR 中接收命令 ID *0x700*。
4. 在 main() 中，在后台循环中调用函数 BUCK_LFU_runLFU()。当命令 ID 与 *0x700* 匹配时，将禁用 SCI 中断，执行分支到自定义引导加载程序中 Live DFU 函数的地址。如果组 0 中的应用程序正在执行，则分支到组 0 中的自定义引导加载程序（地址 0x81000）。如果组 1 中的应用程序正在执行，则分支到组 1 中的自定义引导加载程序（地址 0x91000）。

5. 自定义引导加载程序中的 liveDFU() 从主机接收应用程序映像并将其编程到闪存中。完成后，执行取决于是否定义了宏 LFU_WITH_RESET。如果是，则看门狗配置为生成复位信号，然后启用，因此发生器件复位。如果未定义宏，则执行分支到**新应用程序映像的 LFU 入口点**。这是组 0 的 0x8EFF8 和组 1 的 0x9EFF8。这与常规的闪存启动入口点不同。
6. 函数 c_int_lfu() 位于组 0 上的 0x8eff8 和组 1 上的 0x9eff8。该函数可在不复位器件的情况下执行 LFU 切换。在该函数中：
   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。因此，它将初始化在 BANK1_Flash 构建配置中定义的 25 个新浮点变量。
   b. 设置一个标志以指示 LFU 正在进行中。在 F28004x 上，这是使用软件变量 lfuSwitch 实现的。在 F28003x 上，这是通过使用 LFU_setLFUCPU() 设置 LFUConfig SysCtl 寄存器的 LFU.CPU 位实现的。
   c. 调用 main()
7. 在 main() 中，初始化进程取决于 LFU 是否正在进行中。这是通过访问 F28003x 上的 LFUConfig SysCtl 寄存器的 LFU.CPU 位或 F28004x 上的 lfuSwitch 实现的。如果该值为 0，则初始化会像发生器件复位一样进行。
8. 如果该值不为 0，则执行有限初始化。首先，执行 init_lfu() 并执行上一节中描述的操作。
   a. 在这种情况下，在 CLA 上运行控制时，需要使用 memcpy() 将代码从闪存复制到 RAM。这对应于链接器命令文件中的 Cla1Prog 和 .const_cla 段以及控制循环 ISR。在准备此 memcpy 时，应将相应的 LSRAM 段重新配置，确保 CPU 是这些段的主器件。在 memcpy 之后，这些 LSRAM 段再次配置为在 CPU 和 CLA 之间共享。
   b. 禁用 CLA 后台任务。
9. 当控制循环在 CLA 上运行并且存在 CLA 后台任务时，为了确定执行 LFU 切换的正确时间，逻辑略有不同。首先，BUCK_LFU_getBackgroundTaskControlRegister() 用于读取 MCTLBGRND 寄存器的 BGSTART 位。如果回读值为 0，则表示 CLA BGRND 任务既没有运行，也没有挂起。应用程序会视为已准备好进行 LFU 切换。如果读回值为 1，则 CPU 将变量 lfuSwitch_start 设置为 Lfu_switch_waiting_to_switch_cla。从 CLA BGRND 任务到 CPU 的任务结束中断会引发执行 ISR BUCK_LFU_CLA_BGRND_ISR，其中 lfuSwitch_start 从 Lfu_switch_waiting_to_switch_cla 更改为 Lfu_switch_ready_to_switch_cla。

备注

LFU 切换会等待，直到 CLA 后台任务停止，因为与其他 CLA 任务不同，可以抢占 CLA 后台任务，而不必运行到结束。如果切换导致执行中的后台任务停止，它可能会将该任务保留为“非干净”状态。目标是让切换仅在旧固件中的所有任务都完成执行后发生。

10. 此外，在 BUCK_LFU_CLA_BGRND_ISR 内部，在上述变量更改后会调用 BUCK_LFU_setupCLALFU()，其中会执行以下步骤：
    a. CLA 任务向量和后台任务向量映射到相应的任务。
    b. 注册对应于从 CLA BGRND 任务到 CPU 的任务结束中断的 ISR。
    c. 启用 CLA 后台任务。
11. 更新 CLA 任务向量后，外设中断将继续发生。但是，CLA 任务始终运行到结束。由于这一属性，不会发生上下文冲突。
12. 这里要注意的另一个要点是：.scratchpad 段（对应于 CLA 任务和函数）需要分配给一个单独的存储器段（不同于 .bss 和 .bss_cla 段）。在 LFU 期间，当 C28x CPU 初始化新变量时，CLA ISR 可能正在运行。ISR 可能正在访问位于 .bss 和 .bss_cla 段中的变量，可能也在使用 .scratchpad。同时，变量初始化将更新 .bss_cla 段。为避免任何 .scratchpad 损坏，它们的分离很重要。
13. 另请注意，在这种情况下，LFU 切换期间的 CLA ISR 时间可能会略微增加。这是 CLA 和 C28x CPU 之间的 LSRAM 内存访问冲突造成的。CLA ISR 使用 .scratchpad 和 .bss（两者均位于 RAMLS7 块内）运行，而 C28x CPU 在 .bss_cla（也位于 RAMLS7 块内）中初始化 CLA 的新变量。
14. 图 3-10 显示了另一个 LFU 用例，其中将在固件升级期间更新控制循环参数。在实践中，这可以使用 Compensation Designer 实时完成，这里提供了这方面的用例进行说明。这对应于 **buck_F28004x_lfu_controlloop** 工程，该工程使用编译器预定义符号 BUCK_CONTROL_RUNNING_ON_CLA 进行构建。在此工程中，BANK0_FLASH 构建配置包含对应于较小增益 (Kdc = 4000) 的系数。BANK1_FLASH 构建配置包含对应于较大增益 Kdc=38904 的系数（请参阅 buck.h 中的函数 BUCK_initControlLoopGlobals()）。当启用有源负载时，这会导致 BANK0_FLASH 构建配置的瞬态性能较差，而 BANK1_FLASH 构建配置的瞬态性能更佳。如果器件已经包含与 **buck_f28004x_lfu** 工程（或甚至 CLA）对应的应用程序文件，则可以使用与上一节相同的 LFU 命令运行此更新，但与此工程对

应的已更新的 .txt 名称除外，如表 3-1 所示。在 **buck_f28004x_lfu_controlloop** 工程中，在器件复位时（而不是在 LFU 切换之后）在 main() 中启用有源负载，因此在运行 LFU 后复位器件非常重要，可为执行此初始化提供便利。例如，在 CPU 上运行控制循环时，假设最后更新了 BANK1_FLASH。这意味着组 1 上的固件正在执行。因此，用户需要执行 LFU 命令来更新 BANK0_FLASH（使用 CLA controlloop 工程可执行文件），**而非 BANK1_FLASH**。LFU 更新完成后，需要复位器件。***器件复位只需执行一次***。

然后，用户可以使用 controlloop 工程可执行文件，在不复位器件的情况下执行其他 LFU 更新。

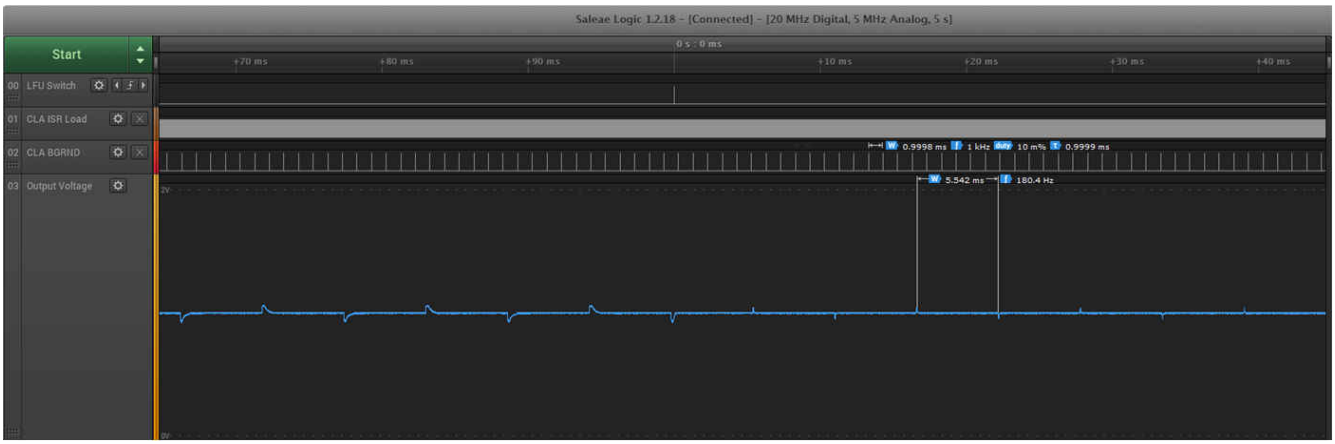


图 3-10. LFU 切换和瞬态性能改善（控制循环在 CLA 上运行）

3.5.5 假设

1. 在设计中，F28003x 上的 LFU 实施仅限于 2 个闪存组。然而，该器件最多可以包含 3 个闪存组，并且 LFU 实施可以扩展到涵盖 3 个组。
2. 组选择逻辑仅存在于闪存组 0 上。
3. 一旦 LFU 命令处理开始，TIDM-DC-DC-BUCK 的后台任务就会停止运行。如果用户想要实施 LFU，并且希望后台循环或部分后台循环在 LFU 命令处理期间继续运行，则可能需考虑将上述循环移到 ISR（例如，CPUTimerISR）中。
4. 使用“#pragma INTERRUPT(ISR_name, HPI)”指定控制循环 ISR。HPI 指高优先级中断，它使用快速上下文保存模式，但无法嵌套。SCI 接收中断 ISR 未指定为 HPI。因此，它默认为 LPI 或低优先级中断，这是可以嵌套的。此外，控制循环 ISR 由 ADCB1 中断触发，该中断属于 F28004x/F28003x 上的中断组 1，优先级高于 SCIA_RX 中断（该中断属于中断组 9）。
5. 分配给未显式分配的 PIE 向量的默认 ISR 保持不变。
6. 在 LFU 期间禁用 SFRA – 这是因为 SFRA 和 LFU 主机共享同一 SCI 外设。在当前硬件配置下，不可能同时支持这两者，因为 Launchpad 仅支持从主机到器件的一个 SCI 通道。
7. 由于不支持闪存组交换，必须将特定的固件版本映射到特定的闪存组。
8. 应用程序固件的对象输出类型为 EABI。
9. .bss（未初始化的数据）成为 NOINIT，因为 EABI 在默认情况下会将未初始化的变量初始化为 0。全局变量的初始化由 main() 中的用户函数完成。这并非必需的，也可以通过 C 运行时初始化例程来完成。

备注

NOINIT 不影响 __TI_auto_init_warm()。

10. 该注释特定于 F28004x - C2000Ware 包含为 COFF 和 EABI 构建的闪存 API 库。基于 EABI 的库是一个从 ROM 运行的闪存 API 库。这包括在应用程序工程中。为了保持一致性，ROM 构建配置也用于定制引导加载程序 (flashapi_ex2_sci_kernel) 工程。
11. 如果用户的应用程序包含多个 ISR（包括嵌套 ISR），则 LFU 切换需要在没有 ISR 运行时的空闲时间发生。如果存在持续时间不同的空闲时间段，则用户可以根据其具体的应用用例选择理想空闲时间段。整个 LFU 过程（包括最终 LFU 切换）在后台（例如在空闲时间）发生。
12. C28x 和 CLA LFU 切换是异步发生的，前提是 C28x 上运行的代码和 CLA 上运行的代码足够独立，从而允许进行该异步切换。如果这些代码相互依赖，需要进行同步切换，则需要修改 LFU 切换的流程。首先，需要确

定 CLA LFU 切换时间。这对应于从 CLA 到 C28x CPU 的“CLA 后台任务停止”中断。现在 CLA 已为切换做好准备，但不应进行切换。现在需要确定 C28x LFU 切换时间。确定该时间之后，C28x 和 CLA 就已为切换做好准备，可以同时进行切换。

3.5.6 为 LFU 准备固件

为了执行 LFU，同时在旧和新的应用程序映像之间进行更加实质性的更改，用户需要注意以下事项：

1. LFU 编译器支持有助于维持公用全局变量的状态（在 RAM 中保留它们的地址，并避免在 LFU 切换期间对它们进行初始化）。

   __TI_auto_init_warm() 与旧应用程序的 ISR 一同执行，因此 __TI_auto_init_warm() 花费多长时间并不重要。**这意味着对需要初始化的变量的数量没有限制**。
2. LFU 切换时间 – 当控制循环 ISR 在 C28x CPU 上运行时，这很重要。在 F28003x 上，就需要在 LFU 上更新的中断向量和函数指针的数量而言，器件上的 LFU 硬件特性（如 PIE 向量交换和 RAM 块交换）可提供极大的灵活性。无论向量或函数指针的数量如何，只需一个单周期交换实现即可。但 F28004x 上不存在这些硬件特性，因此每个 PIE 向量和函数指针都需要单独更新，这会成比例地增加 LFU 切换时间。如果这超过了空闲时间，那么中断执行就会受到影响，这是不可接受的。
3. LFU 切换时间 - 一般来说，当控制循环 ISR 在 CLA 上运行时，这不会成为问题。禁用全局中断仅影响 C28x CPU，不会影响 CLA。LFU 期间不会禁用和重新启用 CLA 任务（后台任务除外）。
4. 另一个需要考虑的重要方面是自定义引导加载程序（SCI 闪存内核）和应用程序之间的 RAM 内存重叠：
   a. 一般来说，应避免 RAM 段在 SCI 闪存内核和应用程序之间发生重叠。如果无法实现，请使用为 SCI 闪存内核和应用程序生成的 .map 文件来验证未发生 RAM 内存重叠，因为这会破坏功能。
   b. 对于使用 CLA 的 LFU，一些 LSRAM 段在应用程序中指定为程序，一些指定为数据。确保这不会与 SCI 闪存内核发生冲突。换而言之，请勿将应用程序放在 SCI 闪存内核用于数据的段中，反之亦然。

3.5.7 LFU 编译器支持

本节介绍了如何利用编译器支持实现 LFU。

1. LFU 支持所需的编译器版本为 21.6.0.LTS 或更高版本。
2. **假设 BANK0_FLASH 构建配置是旧固件，则需要提供指向其输出可执行文件的路径，作为 BANK1_FLASH 构建配置的参考映像**。这将允许编译器识别公用变量及其位置，以及识别新变量。这在 BANK1_FLASH 构建配置工程规格中按如下方式实现（对于 F28004x）：
   --lfu_reference_elf=${CWD}\..\BANK0_FLASH\buck_F28004x_lfu.out

   同样，对于 F28003x，--lfu_reference_elf=${CWD}\..\BANK0_FLASH\buck_F28003x_lfu.out
3. 编译器为变量定义了 2 个新属性，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 LFU 初始化例程 __TI_auto_init_warm() 期间进行初始化。下面列出了有关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;
4. 如果用户如上所述将 BANK0_FLASH 映像作为参考映像来编译 BANK1_FLASH 配置，则生成的 .map 文件将包含与“preserve”变量对应的 .TI.bound 段。此外，如果用户使用“update”属性指定变量（C28x 端或 CLA 端），则 .map 文件将包含单个 .TI.update 段（其中集合了所有“update”变量）。**它们将不会放在 .bss 或 .data 或 .bss_cla 段中。用户需要在链接器命令文件中定义和分配 .TI.update 段。**
5. 为了让应用程序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*（**不要与上述相应的变量属性混淆**），在 BANK1_FLASH 构建配置工程规格中明确指定如下：
   --lfu_default=preserve

   此模式具有以下属性：

   a. 如果提供了参考（*旧*）映像，则不需要将公用变量指定为 *preserve*。这将是公用变量的默认属性，RTS 库将**不**会在 LFU 初始化例程中初始化它们。这有助于保持状态。
   b. 没有指定任何属性的任何新变量将会得到分配地址，但这些变量也**不**会在热启动 LFU 例程中初始化。**如果用户希望 LFU 初始化例程初始化新变量，则需要使用 *update* 属性声明新变量。**
6. 本版本编译器支持的 LFU 模式的完整列表称为“none”和“preserve”。它们具有以下属性：
   a. none：默认情况下，不保留任何全局变量和静态变量地址，也不在热启动期间初始化任何变量。

i. 如果明确指定了“preserve”属性，则保留变量的地址。
ii. 如果明确指定了“update”属性，则在热启动期间初始化变量的值。地址可以在内存中移动。

b. preserve：保留在参考 ELF 中找到的所有全局变量和静态变量地址，除非为变量指定了“update”属性。
i. 无需为公用变量指定“preserve”属性。如果为参考 ELF 中的变量明确指定了“preserve”属性，则其行为与未指定此属性时的行为相同。
ii. 如果明确指定了“update”属性，则在热启动期间初始化变量的值。否则，不会在热启动期间初始化。在这两种情况下，地址都可以在内存中移动。

7. RTS 库提供了一个 LFU 初始化例程 (__TI_auto_init_warm())。它根据上述规则初始化任何新变量。
   a. 该例程执行 C28x CPU 端全局变量和静态变量的初始化，包括零初始化（默认）和非零初始化（如果指定了非零值）。
   b. 该例程只执行 CLA 端全局变量和静态变量的零初始化，不支持 CLA 端全局变量和静态变量的非零初始化。**即使编译器不支持在启动 C 初始化例程中进行 CLA 端全局变量和静态变量的初始化（零或非零），它的确也支持在 __TI_auto_init_warm() 中进行零初始化**。
   c. 如前所述，该例程不受 NOINIT pragma（应用于链接器命令文件中的一个或多个段）的影响。

有关更多信息，请参阅 TMS320C28x 优化 C/C++ 编译器用户指南的 LFU 部分。

3.5.8 稳健性

与 LFU 相关的闪存编程事件的顺序为：

- 在 liveDFU() 中，自定义引导加载程序将 START 字段写入正在编程的闪存组中的特定闪存位置
- 接着，主机将数据逐块（多个字节）传输到器件，数据存储在缓冲区中，校验和返回到主机
- 然后，自定义引导加载程序擦除相应的闪存扇区（如果尚未擦除）
- 在对整个应用程序映像进行编程后，自定义引导加载程序将写入 KEY 字段并更新已编程闪存库中的 VERSION 字段。这表明该闪存库中存在有效的应用程序映像。

如果 LFU 过程因断电或通信问题而中断：

- 如果中断没有导致器件复位（例如通信问题导致故障发生），则自定义引导加载程序将无法完成应用程序映像的下载。但是，来自应用程序的旧 ISR 将继续执行，而不会执行后台任务。在执行器件复位之前，无法启动另一个 LFU 命令。
- 将会对正在写入的闪存组进行部分编程。
- 但是，由于版本未更新，在下一次器件复位时，自定义引导加载程序将分支到闪存中的旧应用程序，并恢复完整的服务，从而能够再次执行 LFU。

3.5.9 LFU 用例

可以通过 LFU 设想许多用例。下面列出了这些用例。A、B、C、D 等指固件版本。

1. A→B →C →D →E
   - 这是我们准备的典型用例
   - A 用作构建 B 的参考映像，B 用作 C 的参考映像，依此类推
2. A→B →A →B →A
   - 我们可能会遇到的另一个用例，会用于测试目的，或者在现场，如果您希望当在新映像中发现问题时恢复到原始映像
   - A 将用作构建 B 的参考映像，因此，当您从 A 切换到 B 时，可以使用编译器的热初始化例程。因为 A 是作为参考提供的，所以编译器知道 A 和 B 之间的变量差异，并将 B 特有的变量放在“TI.update”段。当 LFU 从 A 切换到 B 时，这是将由编译器在其 __TI_auto_init_warm() 例程中进行初始化的唯一段
   - 当从 B 切换到 A 时，情况就不同了。A 是独立编译的，因此它没有“.TI.update”段，编译器不知道哪些变量是 A 所特有的（相对于 B），因此 __TI_auto_init_warm() 不会执行任何操作
   - **这个用例是否可行？是**，用户仍可以从 B 切换回 A。只是用户无法利用编译器的 __TI_auto_init_warm() 来初始化 A 所特有的任何变量。用户需要使用宏来手动初始化 A 的 main() 中的这些特有变量。例如，如果 A 在闪存组 0 中，而 B 在闪存组 1 中，则 A 在 main() 中可以具有如下初始化代码：
     #ifdef BANK0
     [初始化 A 所特有的变量]

#endif

- **事实上，当前 LFU 示例说明了 A-B-A-B 切换**。不同之处在于，在当前示例中，B 与 A 相比具有新变量，但 A 与 B 相比没有任何特有变量。还请记住，并未使用由编译器提供的热启动例程，因此，即使从 A 切换到 B 使用的也是手动初始化

3. 跳过了更新 – 假设现场位置在固件版本变为可用时并不进行更新，而是跳过更新。例如：

   现场位置 1：A→B →C →D →E

   现场位置 2：A →C →D →E

   现场位置 3：A→B → D →E

   这个用例是否可行？否。这里有两个问题。

   - 第一，LFU 本质上是递增的。因此，每个映像都建立在另一个映像的基础上。由编译器生成的 .TI.update 段特定于生成该映像时使用的参考 elf。如果用户希望相对于较旧的映像进行更新，则需要通过手动方式来了解这两个映像之间的变量差异，并且需要对特有变量进行手动初始化。更大的问题是状态。假设 B 引入了一个新变量“var_x”，该变量随后成为所有将来映像的公用变量。用户正在从 A 更新到 D。现在，D 假设 var_x 是一个公用变量，因为它位于 C 中，所以不会将其初始化。然而，相对于 A 而言，var_x 是新变量。因此，不对它进行初始化可能会导致问题。
   - 第二，在我们的实现方案中，用户还需要知道映像所针对的特定组。所以，在这个例子中，A、C、E 将位于组 0 上，而 B、D 将位于组 1 上。因此，不可能如现场位置 2 中所示从 A 更新到 C。
   - 如果用户在不使用旧固件作为参考的情况下构建新固件，则无法保证公用全局变量保留在相同地址，因此无法保留状态。因此，在 LFU 切换之后，除非在切换期间执行了整个 C 初始化例程，否则应用程序的行为可能会出乎意料。这可能会耗时太多，超过可用的 LFU 切换时间。

4 FOTA 示例

4.1 摘要

本节说明 F28003x 上的 LFU 示例，其中固件可执行文件始终加载到闪存组 1 中并从闪存组 0 运行。在将固件可执行文件编程到闪存组 1 中之后，会发生从闪存组 1 到闪存组 0 的复制。这种方法的优点是用户只需为其工程维护一个链接器命令文件，而无需跟踪 LFU 操作后固件将在其中运行的闪存组。

该示例也可以用作 FOTA 的参考。

4.2 引言

在无法重新映射闪存组的情况下，每个闪存组都映射到一个固定的存储器地址。在 LFU 期间，固件可执行文件会编程到当前不活动的闪存组中，而应用程序继续从当前活动的闪存组运行。在使用该方法时，用户需要知道其固件可执行文件的目标闪存组。因此，用户需要为其工程维护 2 个链接器命令文件（如果用户使用 2 个闪存组）。这可能很麻烦，因此，此处提出并实施了一种替代解决方案。

在该方法中，固件可执行文件始终构建为加载到闪存组 1 中并从闪存组 0 运行。这可以只用一个链接器命令文件来完成。与应用程序中的函数加载到闪存中并从 RAM 运行以提高性能类似，此处也需要一个存储器副本。这是在 LFU 引导加载程序（即闪存内核）中实现的。完成闪存组 1 到组 0 的存储器复制大约需要 1 秒的时间。

如前所述，该示例可以用作 **FOTA 的参考**，有几个功能特性没有实现：

- 回滚 – 要支持回滚，需要先从闪存组 0 复制到闪存组 2，然后再从组 1 复制到组 0。这可以完全按照闪存内核中展示的组 1 到组 0 复制的方式完成。
- 复位 – 该示例在组 1 到组 0 复制之后不实现完整的器件复位。这就是 FOTA 的工作方式。在该示例中，存储器复制完成后，闪存内核直接跳转到应用程序的入口点，在此处调用 c_int00，然后调用 main()。

4.3 硬件要求

运行该示例需要以下硬件：

1. F28003x ControlCARD 和 USB-C 电缆
2. ControlCARD 扩展坞 [R4.1]

3. 逻辑分析仪，如 Saleae Logic 8

4.4 软件要求

运行该示例所需的软件为：

1. flash_kernel_ex3_sci_flash_kernel 工程
2. buck_F28003x_lfu 工程

4.5 运行示例

运行示例的步骤为：

1. 启动 CCS，导入 tidm_02011 目录中的以下工程 - buck_f28003x_lfu 和 flash_kernel_ex3_sci_flash_kernel。构建 flash_kernel_ex3_sci_flash_kernel 工程的 BANK0_LDFU_BANK1TO0COPY 构建配置。
2. 构建 buck_F28003x_lfu 工程的 BANK0_FLASH_BANK10COPY 构建配置，将 **BANK0_V1** 声明为预定义符号。将构建的 .txt 文件从 buck_F28003x_lfuBANK0FLASH.txt 重命名为 buck_F28003x_lfuBANK0FLASH_v1.txt，并将其复制到 C2000Ware_DigitalPower_SDK_xx_xx_xx\c2000ware\utilities\flash_programmers\serial_flash_programmer
3. 构建 buck_F28003x_lfu 工程的 BANK0_FLASH_BANK10COPY 构建配置，将 **BANK0_V2** 声明为预定义符号。将构建的 .txt 文件从 buck_F28003x_lfuBANK0FLASH.txt 重命名为 buck_F28003x_lfuBANK0FLASH_v2.txt，并将其复制到 C2000Ware_DigitalPower_SDK_xx_xx_xx\c2000ware\utilities\flash_programmers\serial_flash_programmer
4. 启动 F28003x 的目标配置文件（擦除所有闪存），连接到 ControlCARD 上的 F28003x 目标，并将 flash_kernel_ex3_sci_flash_kernel.out 编程到器件中。这会将 SCI 闪存内核（即 LFU 引导加载程序）放置在闪存组 0 的扇区 0 和 1 中。
5. 编程完成后，执行在 bankSelect() 函数中停止。点击“Run”。
6. 打开 Windows 命令提示符，将目录更改为 DigitalPower SDK 中的 serial_flash_programmer 目录。然后发出常用的 LFU 命令将 buck_F28003x_lfuBANK0FLASH_v1.txt 编程到目标中，这会将应用程序固件可执行文件编程到组 1 中，然后将其从组 1 复制到组 0，然后跳转到组 0 中的应用程序入口点并开始执行。ControlCARD 右上角的 LED D2 将开始闪烁。断开 CCS。这样就完成了“生产编程”步骤。
7. 要在现场测试 LFU/FOTA 更新，请使用常用的 LFU 命令重复上述步骤，将 buck_F28003x_lfuBANK0FLASH_v2.txt 编程到目标中，这会将应用程序固件可执行文件编程到组 1 中，然后将其从组 1 复制到组 0，然后跳转到组 0 中的应用程序入口点并开始执行。
8. 使用 _v1 时，LED D2 的闪烁频率低于使用 _v2 时的频率。由于在 ISR 中发生 LED 闪烁，因此即使在 LFU 过程中也会继续闪烁。当闪存组 1 到组 0 复制和新代码初始化发生时，LED 闪烁短暂停止，持续约 1 秒。

5 设计和文档支持

5.1 软件文件

若要下载此参考设计的软件，请转至适用于 C2000 MCU 的 DigitalPower 软件开发套件 (SDK)。

5.2 文档支持

1. 德州仪器 (TI)，*TMS320F28003x 实时微控制器技术参考手册*
2. 德州仪器 (TI)，*TMS320F28004x 实时微控制器技术参考手册*
3. 德州仪器 (TI)，*C2000™ MCU 在无器件复位时的实时固件更新用户指南*
4. 德州仪器 (TI)，*C2000™ MCU 在有器件复位时的实时固件更新用户指南*
5. 德州仪器 (TI)，*TIDM-DC-DC-BUCK C2000™ 数字电源 BoosterPack™*
6. 德州仪器 (TI)，*C2000™Piccolo™ F28004x 系列 LaunchPad 开发套件*
7. 德州仪器 (TI)，*C2000™Piccolo™ F28003x 系列 LaunchPad 开发套件*
8. 德州仪器 (TI)，*C2000Ware 中的 SCI 闪存内核 F28004x 工程*
9. 德州仪器 (TI)，*C2000Ware 中 LFU LED 闪烁 F28004x 示例*
10. 德州仪器 (TI)，TMS320C28x 优化 C/C++ 编译器用户指南

5.3 支持资源

TI E2E™ 支持论坛是工程师的重要参考资料，可直接从专家获得快速、经过验证的解答和设计帮助。搜索现有解答或提出自己的问题可获得所需的快速设计帮助。

链接的内容由各个贡献者“按原样”提供。这些内容并不构成 TI 技术规范，并且不一定反映 TI 的观点；请参阅 TI 的《使用条款》。

5.4 商标

C2000™, TI E2E™, BoosterPack™, Code Composer Studio™, Piccolo™, F28004x 系列 LaunchPad™, LaunchPad™, are trademarks of Texas Instruments.
所有商标均为其各自所有者的财产。

6 术语

CCS	Code Composer Studio
CLA	控制律加速器
ISR	中断处理例程
LFU	实时固件更新
MCU	微控制器单元
PSU	电源单元
SCI	串行通信接口
UART	通用异步接收器-发送器

7 关于作者

Sira Rao 是德州仪器 (TI) C2000™ 业务部门软件团队的负责人。他于 2007 年毕业于佐治亚理工学院，获得电气工程博士学位。他的兴趣包括研究嵌入式系统、计算机架构和信号处理。

8 修订历史记录

Changes from Revision C (August 2022) to Revision D (December 2022) **Page**

- 更新了 *F28003x 示例的软件包内容* 表.....5
- 更新了*测试设置* 主题.....9
- 更新了 *CLA 上的 LFU 流程*主题.....16
- 在*假设* 主题中添加了其他步骤.....18
- 更新了 *LFU 用例* 主题.....20
- 添加了 *FOTA 示例* 主题.....21

Changes from Revision B (July 2022) to Revision C (August 2022) **Page**

- 更新了 *CPU 上的 LFU 流程* 主题.....15
- 更新了 *CLA 上的 LFU 流程*主题.....16

Changes from Revision A (April 2021) to Revision B (July 2022) **Page**

- 更新的说明.....1
- 在 TMS320F28003x MCU 上添加了对 LFU 的支持.....1
- 添加了 F28003x 和 LAUNCHXL-F280039C.....1
- 更新了特性.....1
- 添加了 F28003x.....2
- 添加了 LFU 切换时间的定义.....2
- 添加了编译器、MCU LFU 硬件支持.....3
- 将主题标题更新为“LFU 切换概念”.....3
- 更新了注释，指示对 LFU 的编译器支持可用.....3
- 更新了 LFU 切换时间的定义.....3
- 添加了有关 F28003x LFU 硬件特性的注释.....3
- 更新了说明，删除了关于 RAM 中 ISR 的注释.....4
- 添加了 LAUNCHXL-F280039C.....5
- 更新了编译器和 DigitalPower SDK 版本要求.....5
- 将表 3-1 的标题更新为 *F28004x 示例的软件包内容* 5
- 更新了 F28004x 自定义引导加载程序的路径（从 C2000Ware 到 DPSDK）.....5
- 从表中删除了 flashapi_ex2_sci_kernel-CPU1-RAM.txt 和 serial_flash_programmer.exe 行，以简化演示.....5
- 为 F28003x 示例的软件包内容添加了新表.....5
- 更新了 F28003x 的内容.....8
- 更新了 LFU 解决方案软件目录结构图像.....8
- 添加了指示 TIDM-02011 在 F28003x 上运行 TIDM-DC-DC-BUCK 示例的注释。原始 TIDM-DC-DC-BUCK 设计仍仅在 F28004x 上运行.....8
- 删除了*在不使用 CCS 的情况下将自定义引导加载程序和应用程序加载到闪存*这一主题。.....9
- 删除了更新的“在不使用 CCS 的情况下将自定义引导加载程序和应用程序加载到闪存”.....9
- 添加了关于 F28003x 执行的详细信息.....11
- 添加了关于 F28003x 执行的详细信息.....13
- 更新了 F28003x 实现.....15
- 添加了关于 F28003x LFU 实施仅限于两个组的注释。可以扩展到三个组.....18
- 删除了关于 memcpy() 的注释.....18
- 添加了关于 NOINIT 的注释.....18
- 将标题更新为*为 LFU 准备固件* 19
- 未更新编译器支持和初始化变量.....19
- 未更新 LFU 切换时间.....19
- 删除了与编译器初始化错误相关的注释.....19
- 更新了编译器版本，支持 LFU.....19

Changes from Revision * (December 2020) to Revision A (April 2021) **Page**

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