Application Notes
ZHCADF7 – Nov. 2023

快速实现 C2000 串口程序升级

Brian Wang *North and West China*

ABSTRACT

在以服务器电源、伺服驱动器为代表的工业应用上，为了能够对产品进行持续的维护和优化，近年来越来越多出现了远程固件升级的需求。传统的升级方式多通过烧录口手动进行程序升级，需要大量人力且效率低下。远程升级允许用户采用 OTA 的方式通过系统中的上位机进行批量化的远程操作可以节约大量人力。

TI C2000 系列产品并没有直接支持远程升级功能，需要使用者自行进行升级程序开发。升级程序涉及 C2000 的上电引导，Flash 操作，程序跳转，升级文件转换等多方面考虑因素，对于开发者而言有一定的开发门槛。本文将以串口为例，介绍利用 C2000 开发包中的资源快速进行远程升级程序（Bootloader）的开发。

Contents

Figures

1 远程升级的基本概念介绍

Figure 1 为实现远程升级的基本程序架构。为了实现远程升级，程序代码必须包含 Bootloader 和 APP Code 两部分。其中 APP Code 为用户的应用程序代码，Bootloader Code 则为独立于 APP Code，专门用于对其进行升级的代码，需要负责启动引导， 上位机通讯，APP Code 接收、升级等一系列功能。

对于 C2000 系列 MCU，芯片上电后首先执行内置 ROM Code 完成芯片初始化流程。为了实现远程升级，我们需要首先跳转进入 Boot Loader Code，判断需要进行程序升级，或是跳转进入 APP Code。而当 APP Code 正在执行时，如果想要进行升级，则可以通过 MCU 复位或程序跳转重新进入 Boot loader 以执行相应的程序升级判断功能。

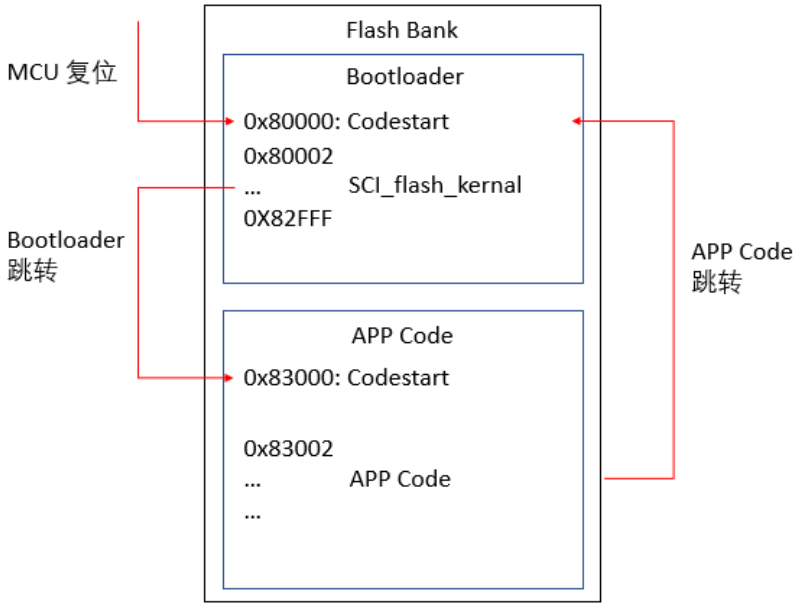

Figure 1.　远程升级程序流程图

为了快速实现程序升级功能，我们主要需要解决以下问题：

1. Bootloader ：

    1）Bootloader 需要负责判断是否需要升级；

    2）在需要升级的情况下通过某种通信协议(CAN\UART\SPI 等) 接收 APP Code 数据并写入相应的地址；

    3）升级结束后跳转导 APP Code 当中执行相应的功能；

2. APP Code ：

    1）实现正常的用户应用程序功能（该部分与正常的程序开发一致）；

    2）在需要升级的情况下跳转到 Bootloader 当中；

3）将 APP Code 转化成 Bootloader 能够‘听懂’的格式，即与 Bootloader 中协议相符的 HEX 或 BIN 格式

3. 上位机软件：

将格式转化完成的 APP Code 按照与 Bootloader 中协议相符的通信协议发送给 C2000 MCU。

为了实现可靠的升级过程，我们需要在 Flash 的物理分区上将两部分功能完全区分开来。这样无论升级过程当中发生任何意外，我们只需重新复位 MCU 即可进入 Bootloader 再次进入升级。从程序空间分布上来说，为了保证 MCU 复位后首先进入 Bootloader，我们将 Boot Loader 的程序开始段始终放在 Flash Sector 0（以 F280039 为例为 0X80000）,而对于 APP Code 只需要保证分配的 Flash 程序空间不与 Bootloader 重叠即可。而由于在程序跳转时将重新执行 c_int00， 因而程序中的变量等 RAM 中数据将被重新初始化，因此对于 RAM 的分配不需要进行特殊考虑。

2 通过以上介绍可以看到从零开始 C2000 程序升级功能的开发包含很大的工作量。为了方便广大工程师朋友快速完成此类需求开发，在 C2000Ware 中提供了基于串口的 Bootloader 例程以及对应的上位机工具，本文将介绍如何基于这些资源快速完成基于串口的程序升级功能开发。SCI Bootloader 介绍

对 C2000 较为熟悉的用户会了解到 C2000 芯片本身也提供了 SCI, SPI，CAN 等多种 BOOT 模式，为何依然要开发定制化的 Boot Loader 程序？

首先， C2000 BootROM 中确实内置多种 Boot 模式代码，但芯片自带 Boot 程序只能将接收到的程序写进芯片的 RAM 中，而不能进行任何 Flash 操作，每次上电需要重新写入；另外芯片进入不同的 BOOT 模式必须通过 GPIO 口状态或者写入 OTP 来控制，而无论是 GPIO 口还是 OTP 在量产产品都无法多次更改；最后芯片自带的 Boot 模式必须遵循固定的通信方式和通信协议，无法适应用户特定的通信要求。因此需要进行自定义 Boot Loader 程序的开发。

完整的 Bootloader 开发需要涉及通讯、Flash API 等多方面的工作。如果完全从零开始会是一个非常复杂的工作，而考虑到这一点 C2000 的 SDK C2000Ware 里已经为我们提供了一个基于 SCI 的 Boot loader 例程，用户可以在\C2000Ware_x_xx_xx_xx\driverlib\f28004x\examples\flash 目录下找到一个名为 flash_ex2_sci_flash_kernel 的工程。

导入 CCS 打开后, 将编译选项选择为 BANK0_NO_LDFU 即为实现最基础程序升级功能的 Example Code：

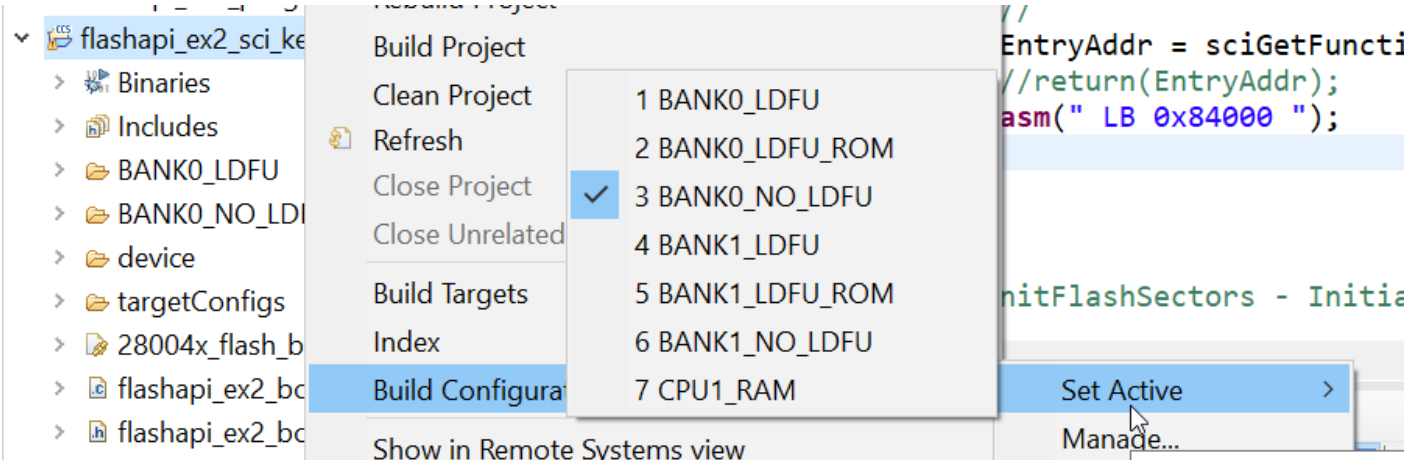

Figure 2. SCI Bootloader 例程编译选项

该程序在默认状态下已经提供了程序升级的几个主要功能:

命令	功能	描述
DFU	程序升级	将通过 SCI 接收到的 Image 写入到对应的 Flash 空间中，包含 Flash 空间的擦除，写入和校验全过程
Erase	Flash 擦除	擦除上位机指定的 Flash Sector
Verify	程序验证	将通过 SCI 接收到的 Image 与 Flash 中的数据进行对比校验
Unlock Zone 1	解锁 DCSM Zone1	使用通过 SCI 接收到的密码解锁 DCSM Zone1
Unlock Zone 2	解锁 DCSM Zone1	使用通过 SCI 接收到的密码解锁 DCSM Zone2
Run	MCU 运行	MCU 跳转到指定的 Flash 地址
Reset	MCU 复位	MCU 通过看门狗溢出复位

命令帧格式如下：

Header	数据长度	命令	数据	校验和	Footer
2*Bytes	2*Bytes	2*Bytes	数据长度决定	2*Bytes	2*Bytes
0x1BE4	数据长度	命令代号	数据	命令及数据校验和	0xE41B

各功能的源码可以在 flashapi_ex2_sci_get_function.c 中的 sciGetFunction() 函数中找到。对于例程不能满足的功能，用户可以自行修改源码。

为配合上位机软件的正常运行，需要将原例程的跳转函数修改如下(L172 及 L173)，其中跳转地址 0x83000 为 APP Code 的起始地址。考虑到 Bootloader 程序一经烧录通常就不再进行修改，可以将 APP Code 的程序入口地址固定下来。请注意在当前 C2000Ware_4_01_00_00 中必须进行此修改，上位机软件才能正常运行。

```
171     EntryAddr = sciGetFunction(SCI_BOOT);
172     //return(EntryAddr);
173     asm(" LB 0x83000 ");
```

Figure 3. 修改 Bootloader 程序跳转命令

最后，需要对 Bootloader 的代码空间进行规划，需要确保:

Begin 位于 Flash 开头的 0x80000 且程序空间不于 Bootloader 重叠， 示例如下：

```
   /* Flash sectors */
   /* BANK 0 */
   BEGIN           : origin = 0x080000, length = 0x00000
   FLASH_BANK0_BOOT: origin = 0x080002, length = 0x002FFE/* 将Bank0_3 合并为Boot程序空间 */

   .text : > FLASH_BANK0_BOOT,   PAGE = 0, ALIGN(4) /*将所有Flash内容分配到FLASH_BANK0_BOOT
当中，此处仅举一个例子 */
```

编译后将程序写入MCU当中，我们开始着手进行应用程序代码的准备工作。

3 用于升级的应用代码准备

用于升级的应用代码（APP Code）在功能开发上与正常代码的开发并没有区别，在此我们以 C2000Ware 中的 LED 闪灯例程为例介绍如何处理应用代码
X\ti\c2000\C2000Ware_4_01_00_00\driverlib\f28004x\examples\led

首先，应用代码所使用的 Flash Sector 与 Bootloader 程序在地址上不能有重叠，以上面一章所示的 Bootloader 为例，可以利用 0x083000 之后的 flash 空间作为 APP Code 的 Flash 空间；

Begin 为 APP Code 的入口地址，也是跳转的 APP Code 时的跳转地址。此处常见的问题是为什么不跳转到 APP Code 的 Main 函数地址而要跳转到 Begin 地址？ 纠其原因，Begin 段所包含的内容是 codestartbranch.asm 文件，其中最重要的功能是跳转到 BootROM 中的_c_int00 函数。c_int00 函数中将进行系统堆栈、全局变量等初始化工作，这对于保证跳转前后的程序运行不相互影响至关重要。许多开发者发现直接跳转到 Main 函数入口之后，程序运行会发生很多难以解释的问题。其原因都是 MCU 堆栈和全局变量均维持了跳转前的状态，影响了跳转之后的程序运行。

应用代码中另外一个需要考虑到的问题就是如何在 MCU 正常运行 APP Code 时让 MCU 进入升级流程当中。通常有几种处理方法：

1. 当需要进行 MCU 升级，手动 Reset 或者软件 Reset MCU。 按照上一章的配置，MCU 复位后即会进入 Bootloader 程序，可以通过上位机进行程序升级；

2. 若无法进行 MCU 复位，可以在 MCU 收到升级指令之后可以通过跳转指令跳转到 Bootloader 的程序入口处，再通过上位机进行程序升级；

3. 更为复杂的程序升级可能会涉及到程序版本号管理、升级密钥等进阶功能。可以在 Flash 中再使用一个单独的 Sector，写入当前程序的版本号等其它标志位信息，在 Bootloader 将新 Image 的信息与当前信息进行对比，判断升级过程是否安全。该类功能多由用户根据自身的规范和需求自行开发，在此暂不深入讨论。

综合以上，对于 APP Code 的 CMD 进行如下修改：

```
   /* Flash sectors */
   /* BANK 0 */
   BEGIN           : origin = 0x083000, length = 0x00000
   FLASH_BANK0_APP:  origin = 0x083002, length = 0x002FFE/* 将Bank0_3 合并为Boot程序空间 */

   .text : > FLASH_BANK0_APP,   PAGE = 0, ALIGN(4) /*将所有Flash内容分配到FLASH_BANK0_BOOT
当中，此处仅举一个例子 */
```

编译得到.out 文件之后，需要使用 hex2000 工具将.out 文件转换成 host 软件支持的 hex 文件格式。首先将生成的.out 文件到 HEX2000 工具所在的目录下：C:\ti\ccs1210\ccs\tools\compiler\ti-cgt-c2000_22.6.0.LTS\bin

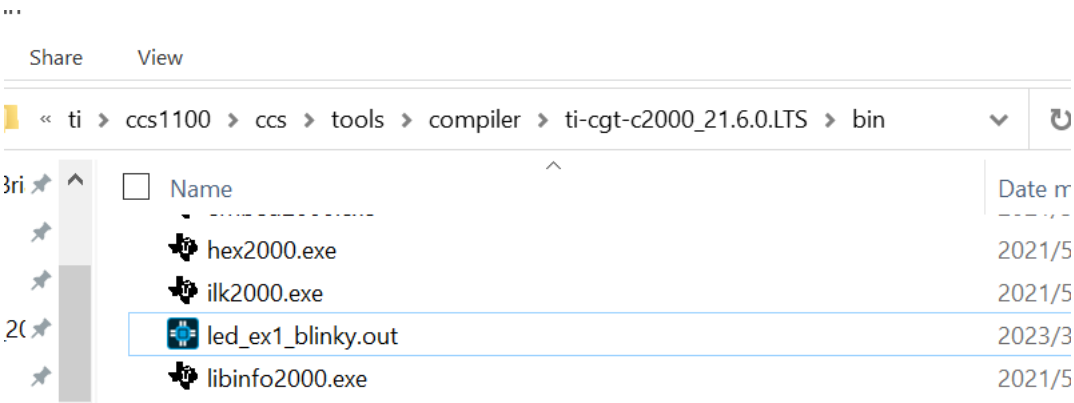

Figure 4.　APP Code .out 文件及 HEX2000 转换工具

编译得到.out 文件之后，调用 WINDOWS 自带的命令行工具，按照以下格式进行文件转换即可得到符合上位机格式要求的 HEX 格式的 TXT 后缀文件：

ccsxx\ccs\tools\compiler\ti-cgt-c2000_22.6.0.LTS\bin\hex2000.exe -boot -a -sci8 <.out filename> -o <.txt filename>

```
C:\Users\a0224161>cd C:\ti\ccs1100\ccs\tools\compiler\ti-cgt-c2000_21.6.0.LTS\bin

C:\ti\ccs1100\ccs\tools\compiler\ti-cgt-c2000_21.6.0.LTS\bin>hex2000.exe -boot -a -sci8 led_ex1_blinky.out -o led_ex1_blinky.txt
Translating to ASCII-Hex format...
   "led_ex1_blinky.out" codestart ==> (BOOT TABLE)
   "led_ex1_blinky.out" .cinit ==> (BOOT TABLE)
   "led_ex1_blinky.out" .econst ==> (BOOT TABLE)
   "led_ex1_blinky.out" .TI.ramfunc ==> (BOOT TABLE)
   "led_ex1_blinky.out" .text ==> (BOOT TABLE)
```

Figure 5.　HEX2000 工具命令行

4 上位机软件的使用

很多 C2000 系列 MCU 的用户主要从事 MCU 嵌入式软件的开发工作，对于上位机软件的开发没有丰富的经验。基于这点考虑，C2000Ware 里集成了一个开发好的上位程序，路径为：C2000Ware_XX\utilities\flash_programmers\serial_flash_programmer.

该路径下包含两个不同的上位机版本， 其中 serial_flash_programmer.exe 与 MCU BOOTROM 配合使用，并不能进行程序的烧录。 而 serial_flash_programmer_appln.exe 与 sci_flash_kernel 程序配合使用，本文的升级即使用这个上位机程序。

以 F280049 的 Control Card 为例，MCU 串口通过板载的 XDS100 仿真器与 PC 上位机连接。按照第一章的步骤处理 bootloader 程序，并通过 Uniflash 或者 CCS 将 bootloader 程序烧录到 MCU 当中。MCU 复位上电后即运行在 BootLoader 程序当中。

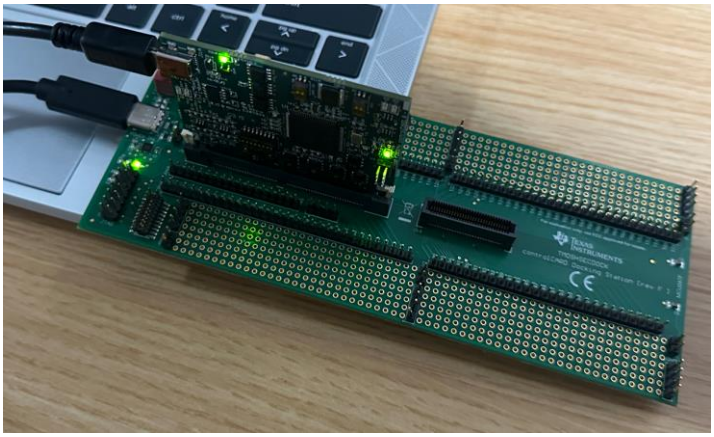

Figure 6. 演示所用硬件环境

通过前文的介绍，我们已经对于 C2000 程序升级的基本方法有了一定的认识。 在此以一个 LED 闪灯程序为例，详细演示如何利用 C2000Ware 中的资源快速实现

将第 3 章中转换得到的.txt 文件放在 serial_flash_programmer_appln.exe 目录下，然后通过命令行执行以下命令进入上位机程序： C2000XX\utilities\flash_programmers\serial_flash_programmer\

serial_flash_programmer_appln.exe -d <device> -a <.txt file name> -b <baud rate> -p <COM>

```
C:\Users\a0224161>cd C:\ti\c2000\C2000Ware_4_01_00_00\utilities\flash_programmers\serial_flash_programmer

C:\ti\c2000\C2000Ware_4_01_00_00\utilities\flash_programmers\serial_flash_programmer>serial_flash_programmer_appln.exe
-d f28004x -a led_ex1_blinky.txt -b 9600 -p COM16

C2000 Serial Firmware Upgrader
Copyright (c) 2021 Texas Instruments Incorporated.  All rights reserved.

This application may be used to download images to a Texas Instruments
```

Figure 7. 上位机软件命令行

选择 1-DFU 开始进行程序升级，等候直到升级完成。

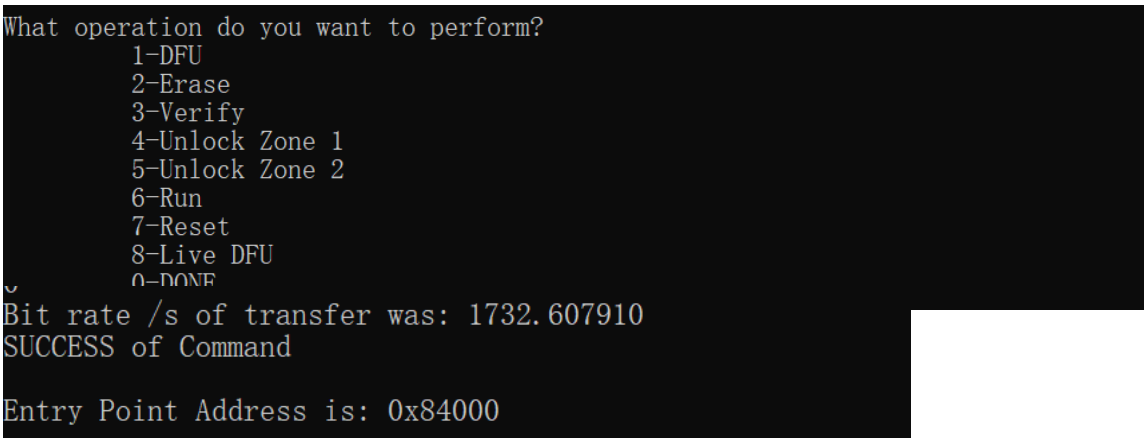

Figure 8. 上位机软件升级过程

随后选择 Run，输入任意地址，完成程序跳转。在第二章中，我们已经将主程序运行结束后的跳转地址固定为 0X83000，因此此处我们可以输入任意地址，并不影响程序运行。

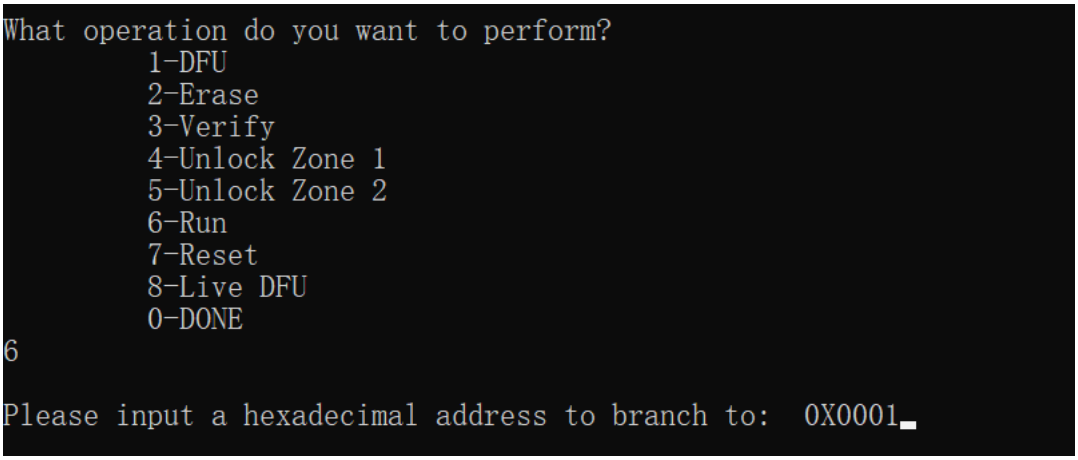

Figure 9. 演示所用硬件环境

程序正常跳转, Control Card 上 LED D2 开始闪烁, 升级流程结束:

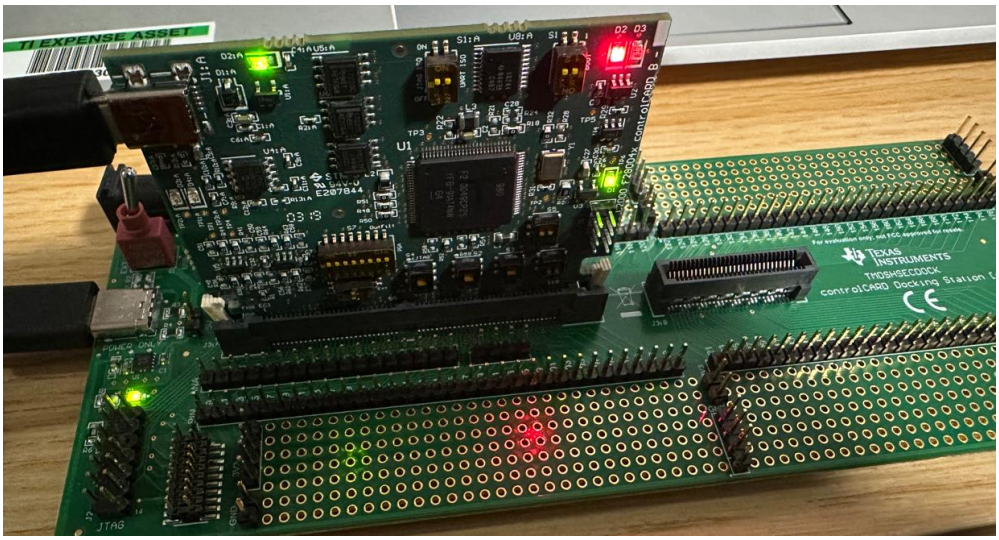

Figure 10. LED 闪灯

5 参考文献

1. *Live Firmware Update Without Device Reset on C2000™ MCUs* (SPRUIU9B)
2. *Serial Flash Programming of C2000™ Microcontrollers* (SPRABV3E)
3. *TMS320F28004x Real-Time Microcontrollers Technical Reference Manual (Rev. F)* (SPRUI33F)

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