



Matt Kukucka

摘要

在无法可靠地使用 JTAG 调试探针对目标器件进行编程的情况下，通常需要对嵌入式处理器进行编程。在这些情况下，工程师需要依靠利用 USB（通用串行总线）或控制器局域网灵活数据速率（CAN-FD）等外设的编程方法。通过在引导 ROM 中添加几个引导加载实用程序以将固件加载到片上 RAM 中，C2000™ 器件可在这方面提供帮助。这些实用程序很有用，但实际上，最初理解和调试它们时会带来一些困惑。本文档介绍一些基本引导加载配置，并说明如何利用最常见的引导模式将应用程序代码加载到片上闪存中。

内容

1 简介.....	4
2 配置引导模式.....	6
2.1 独立引导.....	6
2.1.1 引导模式选择引脚 (BMSP).....	6
2.1.2 引导定义表 (BOOTDEF).....	8
2.1.3 引导 ROM OTP 配置寄存器.....	9
2.1.4 CPU2 启动流程.....	10
2.2 仿真引导.....	12
3 对闪存编程.....	13
3.1 闪存 API.....	14
3.2 闪存内核.....	14
4 将代码引导加载到闪存.....	16
4.1 C2000 Hex Utility.....	16
4.2 常见引导模式.....	17
4.2.1 引导至闪存.....	17
4.2.2 SCI 引导.....	20
4.2.3 CAN 引导.....	26
4.2.4 CAN-FD 引导.....	34
4.2.5 USB 引导.....	42
5 常见问题解答.....	47
5.1 通过基于软件的实现选择 BMSP GPIO.....	47
5.2 从闪存而非 RAM 运行闪存内核.....	47
5.3 在调试引导 ROM 时没有定义符号.....	50
5.4 使用片上闪存工具向 OTP 中写入值.....	52
5.5 使用闪存 API 插件向 OTP 中写入值.....	54
6 总结.....	55
7 参考资料.....	56

插图清单

图 1-1. 引导加载程序总体设计流程.....	5
图 3-1. CCS v12 中的片上闪存工具位置.....	13
图 3-2. CCS v20 中的片上闪存工具位置.....	13
图 4-1. 添加编译后处理步骤以调用 Hex Utility.....	16
图 4-2. 在 CCS 中打开“Target Configuration”菜单.....	18
图 4-3. 启动 CCS 中的目标配置.....	18
图 4-4. 连接到 CCS 中的目标核心.....	18

图 4-5. 导航到 CCS 中的 Memory Browser.....	19
图 4-6. 模拟零引脚引导至闪存 (0x0009 0000).....	19
图 4-7. 查找转换后的 SCI 内核输出文件.....	21
图 4-8. F2800157 LaunchPad UART 布线原理图.....	22
图 4-9. 将 SCI TX 或者 RX (GPIO28 或 GPIO29) 信号路由至 BoosterPack (BP) 引脚.....	22
图 4-10. 将 XDS TX、RX 跳转到 SCI TX、RX GPIO.....	23
图 4-11. 在 CCS 中打开 “Target Configuration” 菜单.....	23
图 4-12. 启动 CCS 中的目标配置.....	24
图 4-13. 连接到 CCS 中的目标核心.....	24
图 4-14. 导航到 CCS 中的 Memory Browser.....	24
图 4-15. 模拟 SCIRXDA 为 GPIO28 且 SCITXDA 为 GPIO29 的零引脚 SCI 引导.....	25
图 4-16. 在器件管理器中查找 XDS COM 端口.....	25
图 4-17. 命令串行闪存编程器下载应用程序.....	26
图 4-18. 将应用程序加载到闪存后运行应用程序.....	26
图 4-19. F280039C LaunchPad CAN 收发器和连接器原理图.....	28
图 4-20. 为正确的 LaunchPad CAN GPIO 分配添加预定义符号.....	28
图 4-21. 包括本地 CAN 时序头文件.....	29
图 4-22. 确认 CAN 时序设置.....	29
图 4-23. 生成 DCAN 内核 txt 输出.....	29
图 4-24. 设置 CAN 路由开关.....	30
图 4-25. 在 CCS 中打开 “Target Configuration” 菜单.....	31
图 4-26. 启动 CCS 中的目标配置.....	31
图 4-27. 连接到 CCS 中的目标核心.....	31
图 4-28. 导航到 CCS 中的 Memory Browser.....	32
图 4-29. 模拟 CANRXA=GPIO5 且 CANTXA=GPIO4 的零引脚 CAN 引导.....	32
图 4-30. 已加载的 DCAN 闪存编程器内核.....	34
图 4-31. 已加载的 DCAN 闪存编程器应用程序.....	34
图 4-32. F280039C LaunchPad CAN 收发器和连接器原理图.....	36
图 4-33. 为正确的 LaunchPad CAN GPIO 分配添加预定义符号.....	36
图 4-34. 生成 CAN 内核 txt 输出.....	37
图 4-35. 设置 CAN 路由开关.....	38
图 4-36. 在 CCS 中打开 “Target Configuration” 菜单.....	38
图 4-37. 启动 CCS 中的目标配置.....	39
图 4-38. 连接到 CCS 中的目标核心.....	39
图 4-39. 导航到 CCS 中的 Memory Browser.....	39
图 4-40. 模拟 CANRXA=GPIO5 且 CANTXA =GPIO4 的零引脚 MCAN 引导.....	40
图 4-41. 已加载的 MCAN 闪存编程器内核.....	41
图 4-42. 已加载的 MCAN 闪存编程器应用程序.....	41
图 4-43. 生成正确的 USB 内核 Bin 输出.....	42
图 4-44. 配置 F2837xD controlCard 以使用 CCS 和板载 XDS 仿真器进行调试.....	43
图 4-45. 在 CCS 中打开 “Target Configuration” 菜单.....	43
图 4-46. 启动 CCS 中的目标配置.....	44
图 4-47. 连接到 CCS 中的目标核心.....	44
图 4-48. 导航到 CCS 中的 Memory Browser.....	44
图 4-49. 配置 F2837xD controlCard 以模拟 USB 引导.....	45
图 4-50. 浏览 C2000Ware 以查找 USB Windows 驱动程序.....	45
图 4-51. 设备管理器中显示的 F28x7x USB 引导加载程序器件.....	46
图 5-1. 将工程文件添加到 CCS 中.....	48
图 5-2. 从编译器构建中排除 RAM 链接器命令文件.....	48
图 5-3. 为工程创建新构建配置.....	49
图 5-4. 创建 CPU1_FLASH 构建配置.....	49
图 5-5. 添加预定义符号 _FLASH 以来初始化闪存函数.....	50
图 5-6. 在 CCS 中打开 “Target Configuration” 菜单.....	51
图 5-7. 启动 CCS 中的目标配置.....	51
图 5-8. 连接到 CCS 中的目标核心.....	51
图 5-9. 没有加载引导 ROM 符号时的 CCS 视图.....	51
图 5-10. 在 CCS 中导航至 Load Symbols.....	52
图 5-11. 定位引导 ROM 源文件.....	52

图 5-12. 示例 1：已编程的闪存插件引导配置.....	53
图 5-13. 示例 2：已编程的闪存插件引导配置.....	54

表格清单

表 2-1. F280015x 器件默认引导模式.....	6
表 2-2. 器件默认引导模式选择引脚.....	6
表 2-3. 按器件系列划分的引导配置类型.....	7
表 2-4. BOOTPIN-CONFIG 位字段.....	8
表 2-5. BOOTDEF 位字段.....	9
表 2-6. F280015x 器件引导模式.....	9
表 2-7. F280015x 引导 ROM 寄存器.....	10
表 2-8. CPU1TOCPU2IPCBOOTMODE 寄存器详细信息.....	11
表 2-9. 仿真引导寄存器位置.....	12
表 5-1. 生成的零引脚引导配置.....	53
表 5-2. 双引导模式选择引脚配置.....	54

商标

C2000™, LaunchPad™, Code Composer Studio™, and BoosterPack™ are trademarks of Texas Instruments.

Microsoft Visual Studio® is a registered trademark of Microsoft Corporation.

所有商标均为其各自所有者的财产。

1 简介

备注

本文档仅适用于基于 **C28x** 的微控制器。有关 **C29x** 引导加载的详细信息，请参阅器件特定的技术参考手册。

随着应用越来越复杂，错误合并、特性添加和嵌入式固件修改功能变得日益重要，尤其是对于在现场维持器件安全与安全完整性而言。**C2000** 器件通过在引导 ROM (出厂编程的只读存储器) 中提供简单的加载实用程序来实现固件更新。

ROM 加载程序通常称为引导加载程序，它驻留在目标器件的引导 ROM 中并允许通过软件从外部主机加载应用程序代码。引导加载程序是 JTAG 调试探针的可靠替代方案，后者需要可直接访问目标器件的昂贵专用硬件。

尽管取决于器件，但用户可以选择各种外设引导加载程序作为将固件加载到目标器件中的介质，例如：

1. 串行通信接口 (SCI)
2. 串行外设接口 (SPI)
3. 内部集成电路 (I2C)
4. 控制器局域网 (CAN)
5. 控制器局域网灵活数据速率 (CAN-FD)
6. 通用串行总线 (USB)
7. 并行 GPIO

每个 **C2000** 器件都有一个默认引导模式的子集可供选择。但是，如果用户需要访问默认引导表中未提供的引导模式，或者需要灵活地使用不同的 GPIO 分配，则必须配置一次性可编程 (OTP) 存储器。OTP 寄存器允许选择未在默认引导表中提供的其他引导模式。

如果用户选择使用外设引导加载程序将新代码加载到器件上，则必须事先以特定格式生成应用程序映像，如 [节 4.1](#) 中所述。准备好应用程序后，引导 ROM 和主机器件之间的数据传输可以继续所选外设引导加载程序中进行。然后，引导加载程序将应用程序代码加载到片上 RAM 中并执行。

外设引导加载程序在每个 **C2000** 器件的 ROM 中都存在并且简单易用，但它仅限于将代码加载到 RAM 中。闪存内核通过提供一种将代码从 RAM 写入闪存的中间方法来弥合 ROM 与闪存之间的差距，如 [节 3.2](#) 中所述。

不过，如果引导流出现一个错误，开始使用 **C2000** 引导加载的用户可能会遇到一些难以调试的缺陷 (有关一般设计概述，请参阅 [图 1-1](#))。本报告旨在阐明的引导加载程序设计分为四个阶段：

1. 选择和配置适当的引导模式
2. 准备要通过引导 ROM 中的外设引导加载程序加载到器件上的应用程序
3. 使用引导加载程序将应用程序加载到 RAM 中
4. 使用闪存内核对闪存进行编程

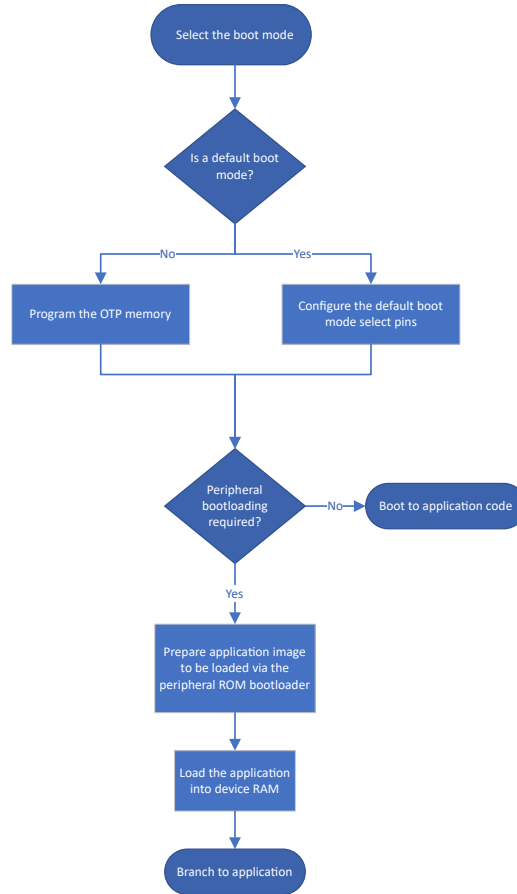


图 1-1. 引导加载程序总体设计流程

2 配置引导模式

在 ROM 引导序列结束时，器件根据是否连接了 JTAG 调试程序来决定是需要进入仿真引导还是独立引导。这是通过读取 JTAG 状态机中的“DCON”位实现的（F2837xD/F2837xS/F2807x 器件除外，该器件轮询“TRSTn”引脚）。仿真引导通过提供位于 RAM 中的寄存器来模拟独立引导流，这些寄存器在结构和配置上与一次性可编程（OTP）存储器中的引导寄存器相同。因此，仿真引导允许根据需要多次写入引导寄存器。

备注

TI 强烈建议尝试对 OTP 编程之前使用仿真模式调试和验证器件引导配置的正确性，因为只能对 OTP 进行一次编程。

本章详细介绍了独立引导（节 2.1）和仿真引导（节 2.2）的配置和使用，但请注意，仿真引导流将在后面的节 4.2 中演示。

2.1 独立引导

备注

本章基于 F280015x 系列器件，但可以适用任何采用 BOOTPIN-CONFIG/BOOTDEF 寄存器的器件，如表 2-3 中所示。器件特定的信息可以在器件技术参考手册 (TRM) 的引导 ROM 一章中找到。

每次 CPU 复位时，器件都会执行引导 ROM 中的预定义引导序列，具体取决于复位类型和引导配置。成功初始化器件后（假设未连接调试程序），将轮询引导模式选择引脚 (BMSP)，以确定要调用的引导模式。BMSP 可以映射到外部 GPIO 引脚，它由器件的默认配置 (表 2-1) 或 OTP 存储器中用户的自定义来定义（有关更多详细信息，请参阅节 2.1.1）。该过程称为独立引导，为通过器件上的外设模块执行固件更新提供了更大的灵活性，而不用直接连接调试程序。

在表 2-2 中按照器件系列提供，所有最新的 C2000 器件都进行了预编程，包含可以使用默认 BMSP 选择的引导模式。通过选择出厂默认引导模式，该器件可以选择直接引导至闪存存储器，或者通过预定的外设通信模块将新的应用代码加载到 RAM，无需对任何引导寄存器进行编程。但是，如果用户需要使用默认引导模式中未包含的外设，或者需要在引导选项中进行更多自定义，该怎么办？

表 2-1. F280015x 器件默认引导模式

引导模式	GPIO24 (默认引导模式选择引脚 1)	GPIO32 (默认引导模式选择引脚 0)
并行 IO	0	0
SCI/等待引导	0	1
CAN	1	0
闪存	1	1

表 2-2. 器件默认引导模式选择引脚

器件系列	GPIO 24 和 GPIO 32	GPIO 72 和 GPIO 84
	F28002x、 F28003x、 F28004x、 F280013x、 F280015x、 F28P55x	F2837xD、 F2837xS、 F2807x、 F2838x、 F28P65x

2.1.1 引导模式选择引脚 (BMSP)

引导模式选择引脚 (BMSP) 由器件解码并用于为引导定义表编制索引，以确定要执行哪种引导模式。除了每个器件上提供的默认引导配置外，用户还可以通过对 OTP 存储器进行编程来选择其他 BMSP 和引导模式。虽然 ROM 的存储器内容是在制造时确定的，但在生产后只能对 OTP 编程一次，从而在要求可靠且可重复地读取数据的应用场合下提供更大的灵活性。

在独立引导加载的背景中，在以下情况需要对 OTP 进行编程：

1. 默认引导选项中不包括引导选项
2. 外设或 BMSP 需要不同的 GPIO
3. 需要使用不同的应用程序入口点
4. 需要灵活使用多种引导选项

根据器件系列 (如 表 2-3 中所示) , 可通过写入用户可配置的双代码安全模块 (DCSM) OTP [7] 中相应的 BOOTPIN-CONFIG 或 BOOTCTRL 存储器位置来修改 BMSP。

备注

在 DCSM 中, 有两个可以分配安全资源的独立安全区 - 区域 1 (Z1) 和区域 2 (Z2)。安全模块限制 CPU 的访问为片上安全存储器和资源, 而且不中断或停止 CPU 执行。当读取安全存储器位置时, 读取返回零值, CPU 继续执行下一条指令。这可以有效阻止通过 JTAG 端口对安全存储器的读取和写入访问。请注意, JTAG 调试程序仍然可以访问不安全的资源。

BOOTPIN-CONFIG 寄存器是一个 32 位宽的位置, 包含四个 8 位宽的分区。为 BMSP 指定了三个分区, 最后一个分区是指定 OTP 配置有效性的密钥。根据正在配置的区域, 该寄存器在 DCSM OTP 中可以是 Z1-OTP-BOOTPIN-CONFIG 和 Z2-OTP-BOOTPIN-CONFIG。

备注

在 Z2 中进行编程的配置优先于 Z1 中的配置。因此, 如果需要更改 OTP 配置, TI 建议先使用 Z1 位置, 然后使用 Z2 位置。

DCSM OTP 中的 BOOTPIN-CONFIG 寄存器可以使用 CCS 中的片上闪存工具或闪存 API 进行编程 (相关步骤, 请分别参阅 节 5.4 或 节 5.5) , 或使用 SysConfig 中的 DCSM 工具以图形方式进行编程 [8]。

表 2-3. 按器件系列划分的引导配置类型

BOOTPIN-CONFIG 和 BOOTDEF 器件	BOOTCTRL 器件
F28002x、 F28003x、 F28004x、 F280013x、 F280015x、 F2838x、 F28P55x、 F28P65x	F2807x、 F2837xD、 F2837xS

BMSP 可以设置为在引导期间使用的几乎任何 GPIO (有关例外情况, 请参阅器件特定 TRM 中的 *配置引导模式引脚一章*) , 其中 GPIO0 为 0x0, GPIO1 为 0x01, 依此类推。尽管大多数情况下 BMSP 需要通过外部 GPIO 引脚手动拉至高电平或低电平, 但软件控制的固件更新可以按照 节 5.1 中所述的方法进行。

只要不违反引导模式引脚的保持时间 (请参阅数据表中的 *复位 - XRSn - 时序要求*) , BMSP 也可以在引导 ROM 和之后的应用程序中使用 (具有替代功能) [29]。

备注

但是, 如果使用 LaunchPad™ 或 controlCard, 则默认 BMSP 通过外部引导开关手动上拉/下拉, 并且之后无法在应用程序中安全地使用。

使用的 BMSP 的数量会以指数级扩展或限制可在引导表选择的潜在可用引导模式。如果使用三个 BMSP, 则最多可以选择 8 个引导选项。减少到两个 BMSP 意味着只有四个引导选项可用。使用零个 BMSP 意味着自动选择单个引导选项, 因而无需对 GPIO 进行外部操作, 并且释放了需要重新用于引导引脚的其他引脚。

可以通过将 0xFF 写入与更改所用的 GPIO 编号时相同的 BOOTPIN-CONFIG 存储器位置来禁用任何特定 BMSP。解码引导模式时, BMSP0 是引导表索引值的最低有效位, BMSP2 是最高有效位。TI 建议在禁用 BMSP 时先禁用 BMSP2。

在独立引导中，如果未使用指定寄存器有效性的正确 **BOOTPIN-CONFIG_KEY** (0x5A) 写入 Z1 或 Z2 OTP 加载的寄存器，则会解码默认 **BMSP**，为默认引导表编制索引。

表 2-4. BOOTPIN-CONFIG 位字段

位	名称	说明
31:24	密钥	将 0x5A 写入这 8 位，以告知引导 ROM 代码此寄存器中的位有效。
23:16	引导模式选择引脚 2 (BMSP2)	请参阅 BMSP0 说明。
15:8	引导模式选择引脚 1 (BMSP1)	请参阅 BMSP0 说明。
7:0	引导模式选择引脚 0 (BMSP0)	设置为在引导期间使用的 GPIO 引脚 (最多 255)。 0x0 = GPIO0, 0x01 = GPIO1, 依此类推。 写入 0xFF 会禁用此 BMSP ，此引脚不再用于选择引导模式。

备注

如果仅使用 **BMSP2** (**BMSP1** 和 **BMSP0** 禁用)，则只能选择引导表索引 0 和 4。如果仅使用 **BMSP0**，则可选引导表索引 0 和 1。在对引导表中的引导选项进行编程时务必记住这一点，因为无论其他 **BMSP** 的状况如何，每个 **BMSP** 仍会保持位置权重。

备注

使用 **BOOTCTRL** 寄存器 (请参阅表 2-3) 的器件与使用 **BOOTPIN-CONFIG** 寄存器的器件系列具有不同的引导流程和配置。

使用有效密钥进行编程时，**BOOTCTRL** 寄存器允许将不同的 GPIO 用作双引导模式选择引脚。**BOOTCTRL** 器件的可自定义 **BMSP** 的总数固定为 2 个，而 **BOOTPIN-CONFIG** 器件的可自定义 **BMSP** 的最大数量为 3 个。但是、可以为两个 **BMSP** 分配相同的 GPIO，允许在所有器件上使用单个引脚。

有关 **BOOTCTRL** 寄存器的更多详细信息，请参阅器件特定的 TRM。

2.1.2 引导定义表 (BOOTDEF)

在 OTP 中配置 **BMSP** 时，还必须通过使用引导选项条目写入引导定义表寄存器 (**BOOTDEF**) 来定义自定义引导模式表。用户定义的 **BOOTDEF** 表替换默认引导模式选择表，并使用 OTP 中的自定义 **BMSP** 对其编制索引。例如，用户现在可以将第一个引导选项设置为任何可用的引导模式，而不是使用连接到默认配置中的引导选项 0 的并行引导，以此类推。

BOOTDEF 表通过配置一个 64 位寄存器来设置 (请参阅表 2-5)，该寄存器在 **DCSM** OTP 中拆分成两个 32 位宽的位置，称为 **Z1-OTP-BOOTDEF-LOW** 和 **Z1-OTP-BOOTDEF-HIGH** (或 **Z2-OTP-BOOTDEF-LOW** 和 **Z2-OTP-BOOTDEF-HIGH**，具体取决于配置的区域)。然后，这些寄存器被划分为 8 位宽的条目，定义要使用的每个引导选项。

BOOTDEF 表中可自定义引导模式的范围取决于正在使用的 **BMSP** 数量。请记住，零个 **BMSP** 允许一个表条目，一个 **BMSP** 允许最多两个表条目，两个 **BMSP** 允许最多四个表条目，三个 **BMSP** 允许最多八个表条目。

要配置 **BOOTDEF** 表，请执行以下操作：

1. 在数据表或技术参考手册的 **GPIO 分配** 部分中选择一种引导选项
2. 在目标 **BOOTDEF-LOW** 或 **BOOTDEF-HIGH** OTP 存储器位置，设置引导选项的相关 **BOOTDEF** 值。

DCSM OTP 中的 **BOOTDEF** 寄存器可以使用 CCS 中的片上闪存工具或闪存 API 进行编程 (相关步骤，请分别参阅节 5.4 或节 5.5)，或使用 SysConfig 中的 **DCSM** 工具以图形方式进行编程 [8]。

使用有效的 **BOOTDEF** 进行编程后，可以使用 **BOOTPIN-CONFIG** 寄存器中配置的 **BMSP** 为引导定义表编制索引，以便选择在复位时在引导 ROM 中执行哪个引导选项。

表 2-5. BOOTDEF 位字段

BOOTDEF 名称	字节位置	名称	说明
BOOT_DEF0	7:0	[3:0] BOOT_DEF0 模式	从表 2-6 中设置引导模式编号。任何不支持的引导模式都会导致器件进入等待引导 (已连接调试程序) 或引导至闪存 (独立)。
		[7:4] BOOT_DEF0 选项	设置备用/附加引导选项。这可能包括更改特定引导外设的 GPIO 或指定不同的闪存入口点。有关表中要设置的有效 BOOTDEF 值, 请参阅 GPIO 分配。
BOOT_DEF1	15:8	BOOT_DEF1 模式/选项	请参阅 BOOT_DEF0 说明。
BOOT_DEF2	23:16	BOOT_DEF2 模式/选项	
BOOT_DEF3	31:24	BOOT_DEF3 模式/选项	
BOOT_DEF4	39:32	BOOT_DEF4 模式/选项	
BOOT_DEF5	47:40	BOOT_DEF5 模式/选项	
BOOT_DEF6	55:48	BOOT_DEF6 模式/选项	
BOOT_DEF7	63:56	BOOT_DEF7 模式/选项	

表 2-6. F280015x 器件引导模式

引导编号	引导模式
0	并行
1	SCI/等待
2	CAN
3	闪存
4	等待
5	RAM
6	SPI
7	I2C
8	CAN-FD
10	安全闪存

备注

引导选择表的配置性存在例外情况, 具体取决于器件系列:

1. 在 F2833x 器件上, 引导表不可自定义并限制为出厂默认值
2. 在 F2802x、F2803x、F2806x、F2837xD、F2837xS 和 F2807x 上, 引导表是半可自定义的, 因为默认引导表中的第 4 个条目 (GET 模式) 可以编程为一个额外的引导模式

这与使用 BOOTDEF 寄存器 (请参阅表 2-3) 的器件不同, 最多允许选择 8 种引导模式。有关 BOOTCTRL 的更多详细信息, 请参阅器件特定的 TRM。

2.1.3 引导 ROM OTP 配置寄存器

引导 ROM 代码包含执行期间使用的许多内存地址和寄存器, 支持从 DCSM 区域 1 (Z1) 和区域 2 (Z2) 寄存器中进行引导配置。独立引导流程中使用的用户可配置 DCSM OTP 位置只能编程一次。这些寄存器的配置在节 2.1 中进行了详细介绍。

在 DCSM 背景下, BOOTPIN-CONFIG 映射到 GPREG1, BOOTDEF-LOW/BOOTDEF-HIGH 分别映射到 GPREG3/GPREG4。表 2-7 提供了这些位置。

节 5.4 和节 5.5 通过示例用例, 详细介绍了如何分别使用片上闪存工具和闪存 API 对 DCSM OTP 进行编程。还可以使用 SysConfig 对 DCSM OTP 进行编程, 它具有直观的图形用户界面(GUI) [8]。

备注

表 2-7 中的寄存器地址适用于 **F280015x** 系列器件。器件特定的信息可以在技术参考手册 (TRM) 引导 ROM 一章的引导 ROM 寄存器表中找到。

表 2-7. F280015x 引导 ROM 寄存器

启动流程	寄存器名称	引导 ROM 名称	寄存器地址	用户 OTP 地址
独立 (使用 Z1)	Z1-GPREG1	Z1-OTP-BOOTPIN-CONFIG	0x0005 F008	0x0007 8008
	Z1-GPREG2	Z1-OTP-BOOT-GPREG2	0x0005 F00A	0x0007 800A
	Z1-GPREG3	Z1-OTP-BOOTDEF-LOW	0x0005 F00C	0x0007 800C
	Z1-GPREG4	Z1-OTP-BOOTDEF-HIGH	0x0005 F00E	0x0007 800E
独立 (使用 Z2)	Z2-GPREG1	Z2-OTP-BOOTPIN-CONFIG	0x0005 F088	0x0007 8208
	Z2-GPREG2	Z2-OTP-BOOT-GPREG2	0x0005 F08A	0x0007 820A
	Z2-GPREG3	Z2-OTP-BOOTDEF-LOW	0x0005 F08C	0x0007 820C
	Z2-GPREG4	Z2-OTP-BOOTDEF-HIGH	0x0005 F08E	0x0007 820E

2.1.4 CPU2 启动流程**备注**

本节基于 F28P65x 系列器件。器件特定信息可以在器件特定的技术参考手册 (TRM) 的引导 ROM 一章中找到。

尽管根据引导引脚配置，CPU1 可以在不同模式下引导，但 CPU2 必须由 CPU1 使用处理器间通信 (IPC) 模块引导 [9]。CPU1 应用程序通过 IPCBOOTMODE 寄存器为 CPU2 配置引导模式，并控制 CPU2 从复位状态释放到引导状态的时间。

无论复位源如何，CPU2 都要求 CPU1 在每次复位时设置 IPC 标志，以确认 IPCBOOTMODE 寄存器的内容有效并继续完成引导过程。CPU2 在引导期间确认并清除该标志。

总而言之，CPU2 引导序列如下所示：

1. CPU2 引导并且：
 - a. 保持在复位状态，或者
 - b. 处于等待引导模式，等待 IPC 标志
2. CPU1 应用程序配置 CPU1TOCPU2IPCBOOTMODE 寄存器
3. CPU1 设置 CPU1TOCPU2IPCFLG0 以确认 CPU1TOCPU2IPCBOOTMODE 的内容有效
4. 如果 CPU2 保持在复位状态，CPU1 应用程序会将释放 CPU2
5. CPU2 在引导期间确认并清除 IPC 标志
6. CPU2 引导 ROM 在 CPU1TOCPU2IPCBOOTMODE 中运行指定的引导模式

F28P65x TRM 的 *IPCBOOTMODE 详细信息* 一节介绍了 IPCBOOTMODE 寄存器中用来引导 CPU2 的位域的配置和要求。表 2-8 详细说明了如何为 F28P65x 器件配置该寄存器。

与 CPU1 BOOTPIN-CONFIG 和 BOOTDEF 寄存器类似，CPU1TOCPU2IPCBOOTMODE 配置如下：

1. 高 8 位包含指示有效性的密钥 (0x5A)
2. 低 8 位设置 CPU2 的引导模式
3. 第 16-19 位指定调用“从 IPC 消息 RAM 复制并引导至 M1RAM”引导模式时，要从 CPU1TOCPU2MSGRAM1 复制到 CPU2 M1RAM 的字数

表 2-8. CPU1TOCPU2IPCBOOTMODE 寄存器详细信息

位	名称	有效值	说明
31:24	密钥	0x5A	必须设置密钥，此寄存器才能视为有效。
23:20	保留	-	保留
19:16	IPC 消息 RAM 复制长度	0x0 = 0 个字 (未使用引导模式) 0x1 = 100 个字 0x2 = 200 个字 ... 0x9 = 900 个字 0xA = 1000 个字	为“从 IPC 消息 RAM 复制并引导至 M1RAM”引导模式设置数据长度 (以字为单位)。这是要从 CPU1TOCPU2MSGRAM1 复制到 CPU2 M1RAM 的字数。 如果不使用该引导模式，请将值设置为 0x0。
15:8	保留	-	保留
7:0	CPU2 引导模式	0x00 = 无/等待引导 0x01 = IPC 消息 RAM 复制并引导至 M1RAM 0x03 = 闪存引导选项 0 (扇区 0) 0x05 = 引导至 M0RAM 0x0A = 安全闪存引导选项 0 (扇区 0) 0x0B = 引导至用户 OTP 0x23 = 闪存引导选项 1 (扇区 4) 0x2A = 安全闪存引导选项 1 (扇区 4) 0x43 = 闪存引导选项 2 (扇区 8) 0x4A = 安全闪存引导选项 2 (扇区 8) 0x63 = 闪存引导选项 3 (扇区 13) 0x6A = 安全闪存引导选项 3 (扇区 13)	设置 CPU2 的引导模式

2.2 仿真引导

如果 JTAG 调试程序连接到器件，则器件进入仿真引导模式。与独立引导模式一样，可以通过对位于 RAM 中的仿真引导寄存器进行编程来访问默认引导表以外的引导选项，使用户能够反复对引导配置进行编程。因此，仿真引导允许用户使用调试程序测试引导配置并查看引导 ROM 的状态（请参阅节 5.3）。

备注

TI 强烈建议尝试对 OTP 编程之前使用仿真模式调试和验证器件引导配置的正确性，因为只能对 OTP 进行一次编程。

- EMU-BOOTPIN-CONFIG 是 Z1-OTP-BOOTPIN-CONFIG/Z2-OTP-BOOTPIN-CONFIG 的仿真等效，可进行编程，从而在不写入 OTP 的情况下使用不同的引导模式进行实验。
- EMU-BOOTDEF-LOW/EMU-BOOTDEF-HIGH 是 Z1-OTP-BOOTDEF-LOW/Z1-OTP-BOOTDEF-HIGH 的仿真等效。

位于 RAM 中的仿真位置可根据需要多次写入，使用 CCS 中的 Memory Browser 写入在表 2-9 中定义的位置：

表 2-9. 仿真引导寄存器位置

引导 ROM 名称	寄存器地址
EMU-BOOTPIN-CONFIG	0x0000 0D00
EMU-GPREG2	0x0000 0D02
EMU-BOOTDEF-LOW	0x0000 0D04
EMU-BOOTDEF-HIGH	0x0000 0D06

在仿真引导模式启动时，检查 EMU-BOOTPIN-CONFIG 和 EMU-BOOTDEF 位置并验证 EMU-BOOTPIN-CONFIG-KEY。

- 如果 EMU-BOOTPIN-CONFIG-KEY 等于 0xA5，则 CPU 模拟独立引导，尤其是在已编程的情况下使用 OTP 定义模拟。
- 如果 EMU-BOOTPIN-CONFIG-KEY 等于 0x5A，则使用指定的 BMSP 解码仿真 BOOTDEF 选项并执行所选的引导模式。

3 对闪存编程

在尝试对器件进行编程之前，需要了解 C2000 器件的非易失性存储器的工作原理。C2000 器件上的闪存存储器允许用户在断电后轻松擦除器件并对其重新编程，而不会丢失数据。擦除操作将给定扇区中的所有位设置为 1，而编程操作则有选择地将位清除为 0。

在开发期间，可以使用 Code Composer Studio™ (CCS) [1] 将应用程序可执行文件编程到闪存存储器中。当 CCS 识别出应用程序代码被映射到闪存存储器时，会自动调用片上闪存插件将可执行文件加载到闪存中。默认情况下，该插件会在编程前擦除闪存，为可执行文件生成 ECC，然后将应用程序编程到闪存中并验证。

连接到目标 CPU 内核时，可以使用闪存插件 GUI，该 GUI 位于 CCS 的以下位置：

- 对于 CCS v12，*Tools > On-Chip Flash*

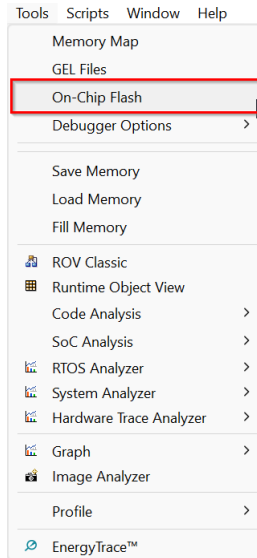


图 3-1. CCS v12 中的片上闪存工具位置

- 对于 CCS v20，在 *Debug* 视图中右键点击目标 CPU，然后导航至 *Properties > Flash Settings (Flash Settings 位于 Categories 下拉菜单中)*

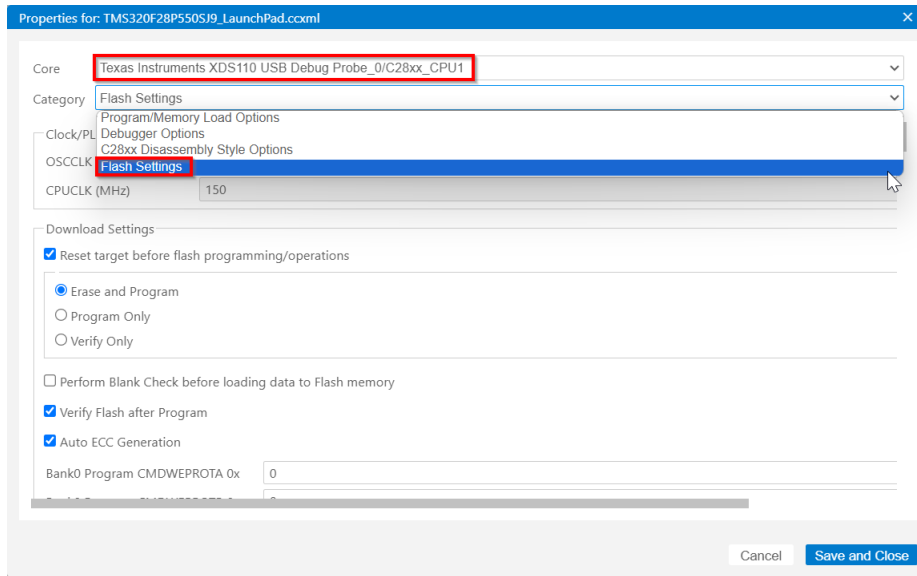


图 3-2. CCS v20 中的片上闪存工具位置

TI 还提供使用 UniFlash [2] 进行的应用程序刷写，UniFlash 是一种独立的基于 JTAG 的闪存编程工具，由于调试支持较少，其占用空间小于 CCS。然而，UniFlash 提供 CCS 片上闪存插件可提供的的所有 GUI 操作。

3.1 闪存 API

所有 C2000 操作中的闪存操作均由 CPU 执行。算法加载到 RAM 中并由 CPU 执行以执行任何闪存操作。例如，使用 CCS 对 C2000 器件的闪存进行擦除或编程，需要通过 JTAG 将闪存算法加载到 RAM 中并让 CPU 执行它们。

所有闪存操作均使用闪存应用程序编程接口 (API) 执行；器件特定的库和参考指南位于 C2000Ware [3] 中的 “libraries/flash_api” 位置。需要在运行时对闪存进行擦除或编程的应用程序可以链接到闪存 API 库来执行闪存编程。

然而，不建议调用闪存 API 的应用程序从同一个闪存组中执行，因为在执行代码的同时对闪存进行擦除或编辑会导致竞态条件和异常行为。因此，闪存 API 需要在 RAM 或另一个闪存组中执行（如果同一内核存在额外组）。

通过将闪存 API 分配到连接器命令文件中的 “.ti.ramfunc” 部分，指定闪存加载地址和 RAM 运行地址，然后在执行之前将函数复制到主函数中的 RAM，可以实现这。有关在 C2000 器件上进行闪存编程的完整详细信息，请参阅 [4] 和 [5]。

备注

在多核器件上，一个 CPU 无法访问另一个 CPU 分配的闪存组。例如，CPU2 的闪存组只能通过从 CPU2 RAM 执行闪存 API 进行编程。

3.2 闪存内核

鉴于引导 ROM 中的引导加载程序只能将代码加载到 RAM 中，闪存内核可充当 ROM 和闪存之间的纽带，它在现场提供通过各种通信协议 (SCI、CAN、I2C 等) 进行的基于闪存的固件升级机制。可以对闪存进行编程，方式如下：使用 ROM 引导加载程序将闪存内核下载到 RAM，然后使用闪存 API 运行 RAM 中的闪存内核以将应用程序下载到闪存。

在接收任何应用程序数据之前，闪存内核会擦除器件的闪存，以为编程做好准备。主机开始发送应用程序代码后，使用一个缓冲区保存接收到的连续应用程序代码块。当缓冲区已满或检测到新的非连续数据块时，将对缓冲区中的代码进行编程。在接收整个应用程序并将其编程到闪存后，闪存内核会转移到应用程序的入口点。

TI 已开发出闪存内核，可根据引导 ROM 源代码将代码从 RAM 加载到闪存，这可在 C2000Ware 的 *examples* 文件夹中找到。用户可以应用类似的开发流程，为特定应用程序实现自定义引导加载程序。可在 C2000Ware 中查看器件引导流程使用的引导 ROM 源代码（包括外设引导加载程序），其位置如下为：

- C2000Ware_x_xx_xx_xx > libraries > boot_rom > DEVICE_FAMILY > REV# > rom_sources > DEVICE_FAMILY_ROM > bootROM > source

在具有多个闪存组的器件上，可以调整闪存内核工程，以便从闪存而不是 RAM 中执行，从而对另一个闪存组进行编程。这样，用户可以简单地跳转到位于闪存中的闪存内核，避免在引导 ROM 中使用的引导加载程序。有关如何实现此功能的更多详细信息，请参阅 节 5.2。

备注

在多核器件（即 F2838x、F2837xD 或 F28P65x）上，CPU1 和 CPU2 内核工程可以利用 CPU1 中的引导加载程序为 CPU2 下载修改后的引导加载程序（即闪存内核），以便下载 CPU2 应用程序映像。

CPU1 操作完成后，CPU1 可以将 CPU2 内核写入共享消息 RAM (CPU1TOCPU2MSGRAM)，并在 CPU2 引导序列期间指示 CPU2 转移到 CPU2 IPC 消息复制目标 RAM (M1RAM) 中的 CPU2 入口点。在转移指令写入 M1RAM 且 CPU2 引导序列完成后，CPU2 从 M1RAM 中开始执行并转移到 CPU2 内核入口点。CPU1 内核等待 CPU2 内核命令完成，然后再继续。

有关 IPC 协议的更多详细信息，请参阅器件特定的技术参考手册。

总之，使用闪存内核的一般流程如下：

1. 复位器件并使用目标引导模式
2. 通过引导 ROM 中的引导加载程序，将闪存内核从主机传输到器件

3. 在 ROM 引导加载程序完成后，闪存内核获得控制权
4. 内核从控制器的闪存存储器中擦除旧的应用程序代码
5. 内核使用目标外设通信协议，配置与主机的连接并接收新的应用程序代码
6. 内核将新接收到的应用程序代码写入闪存存储器，并将控制权转移给应用程序
7. 从主机新接收到的代码执行

4 将代码引导加载到闪存

在器件启动开始时，器件决定是将固件编程到需要执行的闪存中，还是需要使用 ROM 加载程序加载代码。这是通过检查 BMSP 确定的，由用户在 OTP 仿真寄存器中定义，或者遵循出厂默认引导配置。

但是、如果选择从外部主机加载代码的引导加载程序，则需要按照特定的程序对应用程序映像进行格式化并传送，才能成功编程到 RAM 中。然后，可以应用闪存内核以便通过链接闪存 API 来弥合闪存和 RAM 之间的差距，使内核能够擦除和编程闪存。

本部分通过讲解从器件引导配置到闪存内核执行的整个流程，演示了在最常见的引导模式（闪存、SCI、CAN、CAN-FD 和 USB）下在闪存中如何编程和/或执行应用程序的方法。

4.1 C2000 Hex Utility

ROM 加载程序要求以数据流和引导表向其提供数据。该结构对所有 ROM 加载程序都是通用的，器件特定 TRM 中 [引导加载程序数据流结构](#) 一节对此进行了详细介绍。用户可以使用 TI C2000 编译器随附的 hex2000 实用程序，轻松生成这种格式的应用程序。

通过添加项目属性中的编译后处理步骤行，甚至可以在 Code Composer Studio 编译过程中生成此文件格式，如 [图 4-1](#) 中所示。通过在项目属性中启用 *C2000 Hex Utility* 并选择必要的转换选项，也可以在 GUI 中配置 hex2000 实用程序。

对于 C2000 ROM 引导加载程序，需要在 CCS Build（项目属性 > CCS 构建 > 步骤 > 编译后处理步骤）的编译后处理步骤中添加以下行，才可以使固件工程加载到片上闪存并且闪存内核工程加载到 RAM：

SCI、CAN、CAN-FD 引导加载：

```
"${CG_TOOL_HEX}" "${BuildArtifactFileName}" -boot -sci8 -a -o "${BuildArtifactFileBaseName}.txt"
```

USB 引导加载：

```
"${CG_TOOL_HEX}" "${BuildArtifactFileName}" -boot -b -o "${BuildArtifactFileBaseName}.dat"
```

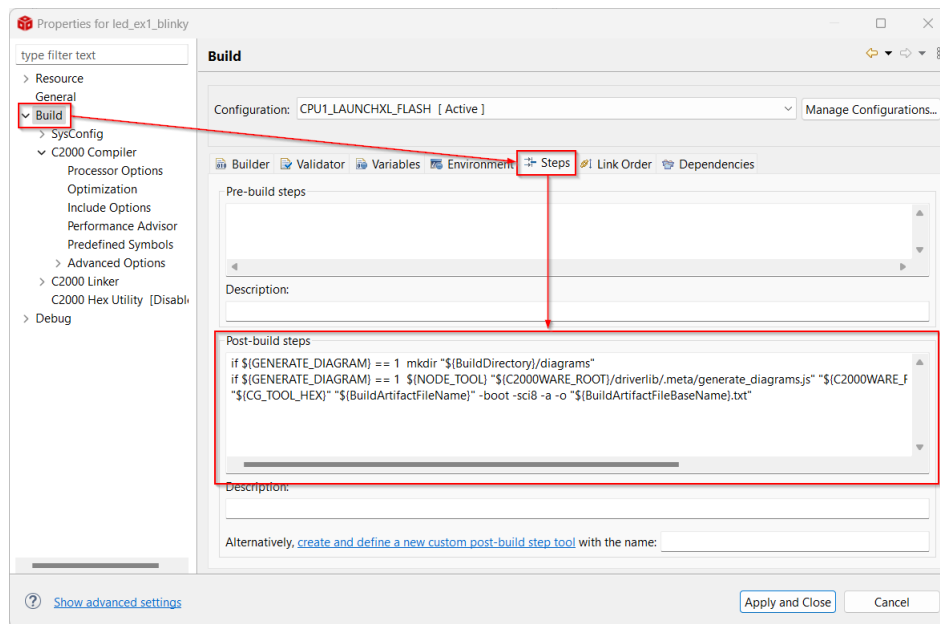


图 4-1. 添加编译后处理步骤以调用 Hex Utility

此十六进制实用程序支持创建 SCI、SPI、I2C、CAN 和并行 I/O 加载程序所需的引导表。十六进制实用程序会向该文件添加必需的信息、例如键值、保留位、入口点、地址、块起始地址、块长度和终止值。取决于运行十六进

制转换实用程序时选择的引导模式和选项，引导表的内容略有不同。主机所需的实际文件格式 (ASCII、二进制、十六进制等) 因具体应用而异，并且可能需要执行一些额外转换。

有关用于生成引导表的 hex2000 选项的详细说明，请参阅 [TMS320C28x 汇编语言工具用户指南 \[11\]](#)。

4.2 常见引导模式

本节全面介绍在最常见的引导模式 (闪存、SCI、CAN、CAN-FD 和 USB) 下，在闪存上对应用程序进行编程和执行的完整过程。

备注

本节重点介绍连接了仿真器 (JTAG) 但未使用 BMSP (零引脚引导) 时的引导执行路径。通过参考 [节 5.4](#) 或 [节 5.5](#)，可以轻松地将仿真过程转换为使用不同 BMSP 配置的独立引导，因为 RAM 中的仿真引导配置寄存器的格式与 OTP 寄存器相同。

4.2.1 引导至闪存

备注

尽管这些步骤在 F2800157 LaunchPad 上执行过，但一般流程可以轻松应用于支持自定义 BMSP 和引导定义表的任何 C2000 器件 ([表 2-3](#) 中提供了所有器件)。有关要引导加载的器件的详细信息，请参阅器件特定 TRM。

如果用户需要引导至已在片上闪存中编程的代码，则用户可以使用默认 BMSP 引导至闪存入口点地址 0x0008 0000，或者配置 BOOTPIN-CONFIG 和 BOOTDEF 寄存器以引导至特定的闪存地址。

备注

一些器件上有一个安全闪存引导选项，在代码实际执行之前，可以使用该选项从闪存中执行应用程序引导，这样可提供引导代码身份验证这一额外安全层。有关启用安全引导的一般程序，请参阅 [C2000 器件上的安全引导 \[13\]](#)。用户可以验证器件特定 TRM 中安全闪存引导功能的可用性。

有关用于配置引导至闪存的默认 BMSP，请参阅 [TMS320F280015x 实时微控制器数据表](#)。如果用户将 GPIO24 和 GPIO32 均设置为 1，引导 ROM 将转移至闪存入口点地址 0x0008 0000，无需配置器件寄存器。

但是，如果用户需要引导至另外的闪存扇区，则需要为特定的引导选项配置 BOOTCONFIG 和 BOOTDEF 寄存器。请参阅数据表中的 [GPIO 分配](#) 一节，确定要配置哪个引导选项以到达目标闪存入口点。这些步骤描述了如何模拟引导至闪存入口点 0x0009 0000。例如，引导选项 0x63。

备注

通过使用连接器命令文件，可以将应用程序加载到不同的闪存位置。有关如何将代码加载到特定闪存地址的详细信息，请参阅 [编译器工具用户手册](#) 中对于连接器命令文件 [\[12\]](#) 的说明。

1. 打开 CCS 的工作区。
2. 选择 View > Target Configurations。

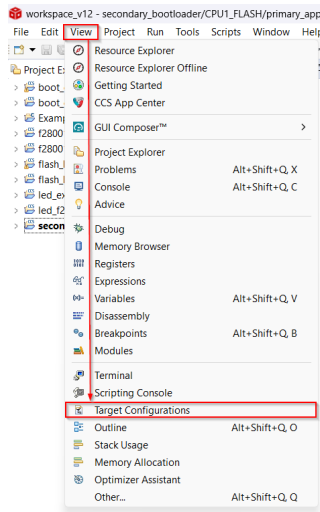


图 4-2. 在 CCS 中打开 “Target Configuration” 菜单

3. 用户可以将该器件的工程导入 CCS 并使用该工程连接器件，或将目标配置文件 (.ccxml) 从 C2000Ware (C2000Ware_x_xx_xx_xx > device_support > DEVICE_FAMILY > common > targetConfigs) 复制到用户定义的目标配置。
 - a. 找到器件目标配置，然后右键单击手动启动。

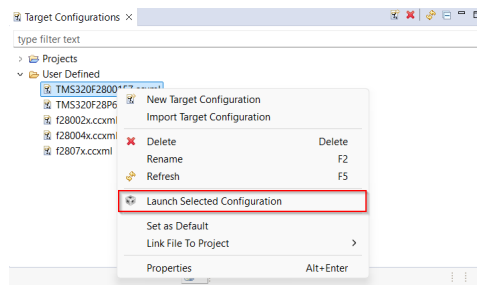


图 4-3. 启动 CCS 中的目标配置

4. CCS 启动调试窗口时，选择目标 CPU 并连接到目标。

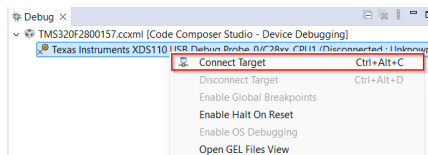


图 4-4. 连接到 CCS 中的目标核心

5. 如果弹出一个窗口，表明引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照节 5.3 中的说明调试引导 ROM。
6. 加载符号后，转到 View > Memory Browser 以打开 Memory Browser。

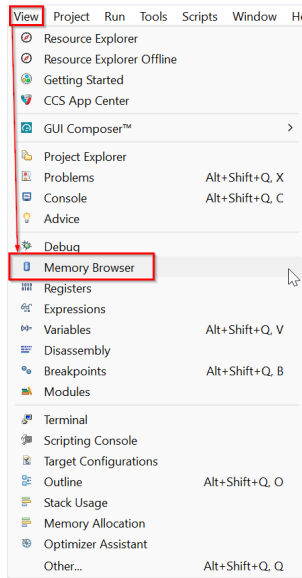


图 4-5. 导航到 CCS 中的 Memory Browser

7. 在“Memory Browser”选项卡中，导航到地址 0xD00。请记住，位置 0xD00 使用有效性密钥 (EMU-BOOTPIN-CONFIG) 指定 BMSP，0xD04-0xD05 指定引导定义 (EMU-BOOTDEF-LOW)。
8. 目标是配置零引脚引导至闪存地址 0x0009 0000，因此需要禁用所有 BMSP 并且 EMU-BOOTDEF-LOW 需要设置为最低索引中的 0x63。如果引导选项编程为 EMU-BOOTDEF-LOW 中的任何其他条目，则不会选择目标引导模式。
 - a. 将 0xD00-0xD01 (EMU-BOOTPIN-CONFIG) 设置为 0x5AFF FFFF。
 - b. 将 0xD04 (EMU-BOOTDEF-LOW) 设置为 0x0063。

备注

零引脚引导意味着器件自动引导至 BOOTDEF 表定义的第一个条目。这是通过禁用所有 BMSP，从而允许器件仅考虑一个引导选项来实现的。

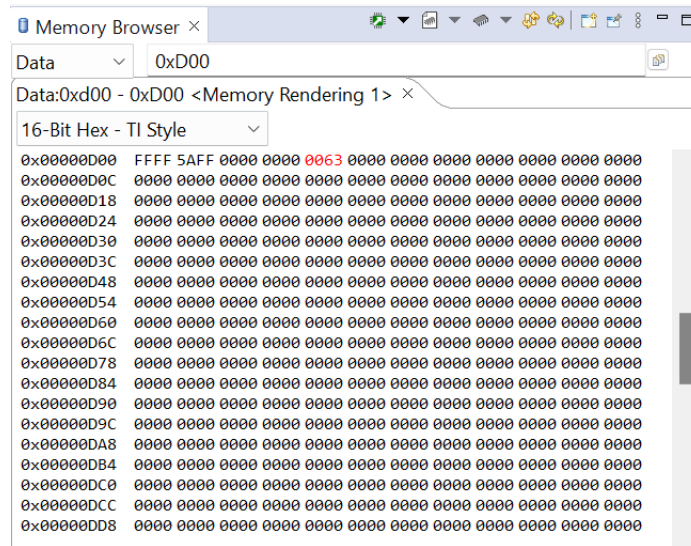


图 4-6. 模拟零引脚引导至闪存 (0x0009 0000)

9. 复位 CPU 并执行外部复位 (XRSn)。然后，单击 *Resume* 以开始引导序列。
10. 现在，按照引导选项 0x63 的指定，器件在复位时引导至闪存地址 0x0009 0000。

4.2.2 SCI 引导

备注

尽管这些步骤在 F2800157 LaunchPad 上执行过，但一般流程可以轻松应用于支持自定义 BMSP 和引导定义表的任何 C2000 器件 (表 2-3 中提供了所有器件)。有关要引导加载的器件，请参阅器件特定的 TRM。

SCI 闪存内核基于 ROM 引导加载程序，它与 C2000Ware 中提供的主机 PC 应用程序 (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > serial_flash_programmer) 进行通信，并就接收数据包和分配给它的命令的完成情况向主机提供反馈。

备注

本节详细介绍了 CPU1 SCI 引导加载。有关 SCI 内核命令和功能的更详细说明，或有关如何使用 CPU2 或连接管理器 (CM) SCI 引导加载程序的步骤，请参阅 [C2000 微控制器的串行闪存编程应用手册 \[12\]](#)。

CCS 的闪存内核源文件和工程文件在 C2000Ware 中提供，位于相应器件的示例目录中。这些工程有一个编译后处理步骤，将编译和链接的 .out 文件转换为 SCI ROM 引导加载程序所需的正确的十六进制格式引导文件，并按扩展名为 .txt 的工程名称进行保存。

1. 在 C2000Ware 中查找适用于目标器件的 SCI 闪存内核工程并导入 CCS。

器件	编译配置	位置
F2802x	RAM	C2000Ware_x_xx_xx_xx > device_support > f2802x > examples > structs > f28027_flash_kernel
F2803x	RAM	C2000Ware_x_xx_xx_xx > device_support > f2803x > examples > c28 > f2803x_flash_kernel
F2805x	RAM	C2000Ware_x_xx_xx_xx > device_support > f2805x > examples > c28 > f28055_flash_kernel
F2806x	RAM	C2000Ware_x_xx_xx_xx > device_support > f2806x > examples > c28 > f28069_sci_flash_kernel
F2807x	RAM	C2000Ware_x_xx_xx_xx > device_support > f2807x > examples > cpu1 > F2807x_sci_flash_kernel
F2833x	RAM	C2000Ware_x_xx_xx_xx > device_support > f2833x > examples > f28335_flash_kernel
F2837xS	RAM	C2000Ware_x_xx_xx_xx > device_support > f2837xs > examples > cpu1 > F2837xS_sci_flash_kernel > cpu01
F2837xD	RAM	C2000Ware_x_xx_xx_xx > device_support > f2837xd > examples > dual > F2837xD_sci_flash_kernels
F28004x	RAM，支持 LDFU 的闪存，不支持 LDFU 的闪存	C2000Ware_x_xx_xx_xx > driverlib > f28004x > examples > flash, select flashapi_ex2_sci_kernel
F2838x	RAM	CPU1-CPU2 C2000Ware_x_x_xx_xx > driverlib > f2838x>examples>c28x_dual>flash_kernel CPU1-CM C2000Ware_x_x_xx_xx > driverlib > f2838x>examples>c28x_cm>flash_kernel
F28002x	RAM，支持 LDFU 的闪存	C2000Ware_x_xx_xx_xx > driverlib > f28002x > examples > flash, select flash_kernel_ex3_sci_flash_kernel
F28003x	RAM，支持 LDFU 的闪存	C2000Ware_x_xx_xx_xx > driverlib > f28003x > examples > flash, select flash_kernel_ex3_sci_flash_kernel
F280013x	RAM	C2000Ware_x_xx_xx_xx > driverlib > f280013x > examples > flash, select flash_kernel_ex3_sci_flash_kernel

器件	编译配置	位置
F280015x	RAM	C2000Ware_x_xx_xx_xx > driverlib > f280015x > examples > flash, select flash_kernel_ex3_sci_flash_kernel
F28P65x	RAM	C2000Ware_x_xx_xx_xx > driverlib > f28p65x > 示例 > c28x_dual > flash_kernel
F28P55x	RAM	C2000Ware_x_xx_xx_xx > driverlib > f28p55x > examples > flash , 选择 t f28p55x_flash_ex3_sci_flash_kernel

2. 确保 SCI 闪存内核工程的 Active Build Target Configuration 设置为 RAM，因为需要链接内核才能在 RAM 中执行。
3. 构建内核工程。这些工程有一个编译后处理步骤，将编译和链接的 .out 文件转换为 SCI ROM 引导加载程序所需的正确的十六进制格式引导文件，并按扩展名为 .txt 的示例名称进行保存。
 - a. 如果未生成 .txt 输出，则按照 节 4.1 操作以确保定义了生成十六进制文件的正确编译后处理步骤。

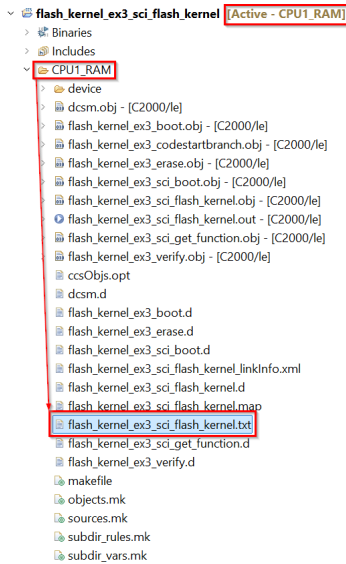


图 4-7. 查找转换后的 SCI 内核输出文件

4. 对内核加载到闪存中的固件代码重复执行构建过程。
 - a. 请确认为闪存设置了 Active Build Target Configuration。
 - b. 确认已定义用于生成 txt 文件的正确编译后处理步骤。
 - c. 构建固件工程。

在 CCS 中构建内核和固件工程后，正确设置器件硬件，以便与运行 C2000Ware 中提供的 serial_flash_programmer 可执行文件的主机 PC 进行通信。首先要确保正确配置引导模式选择引脚，以便将器件引导至 SCI 引导模式。如果用户需要在片上闪存中从外部主机中加载代码，则默认 BMSP 或 BOOTPIN-CONFIG 和 BOOTDEF 寄存器可以配置 SCI 引导。

有关用来启用 SCI 引导的默认 BMSP，请参阅 TMS320F280015x 实时微控制器数据表。如果用户将 GPIO24 设置为 0 并将 GPIO32 设置为 1，则引导 ROM 跳转到 SCI 引导加载程序且 SCIRXDA 设置为 GPIO28，SCITXDA 设置为 GPIO29，并且无需对器件寄存器进行编程。

但是，如果用户希望灵活地使用具有不同 GPIO 分配的 SCI 引导，则需要为特定的引导选项配置 OTP 或仿真 BOOTCONFIG 和 BOOTDEF 寄存器。请参阅数据表中的 GPIO 分配一节，确定哪种 SCI 引导选项符合 GPIO 要求。

使用 serial_flash_programmer 可执行文件时，连接到 Rx 和 Tx 引脚的相应 SCI 引导加载程序 GPIO 引脚需要连接到主机 PC COM 端口。通常需要收发器来将虚拟 COM 端口从 PC 转换为可以连接到器件的 GPIO 引脚。在某些系统（例如 controlCARD 或 LaunchPad）上，使用 FTDI 芯片将用于 SCI 通信的 GPIO 引脚连接到 USB 虚拟 COM 端口。

对于 F2800157 LaunchPad, PC 必须连接到 LaunchPad 上的 USB 并使用器件上的 UART 布线将 GPIO 引脚连接到 XDS110 COM 端口。UART 布线的原理图位于 C2000Ware_x_xx_xx_xx > boards > (LaunchPads or controlCARDs) > DEVICE_NAME > Rev# > documentation。对于 F2800157, 使用默认 SCI_SEL 设置, SCIRXDA 从内部布线到 GPIO28, SCITXDA 从内部布线到 GPIO29, 因此需要配置引导选项 0x01。

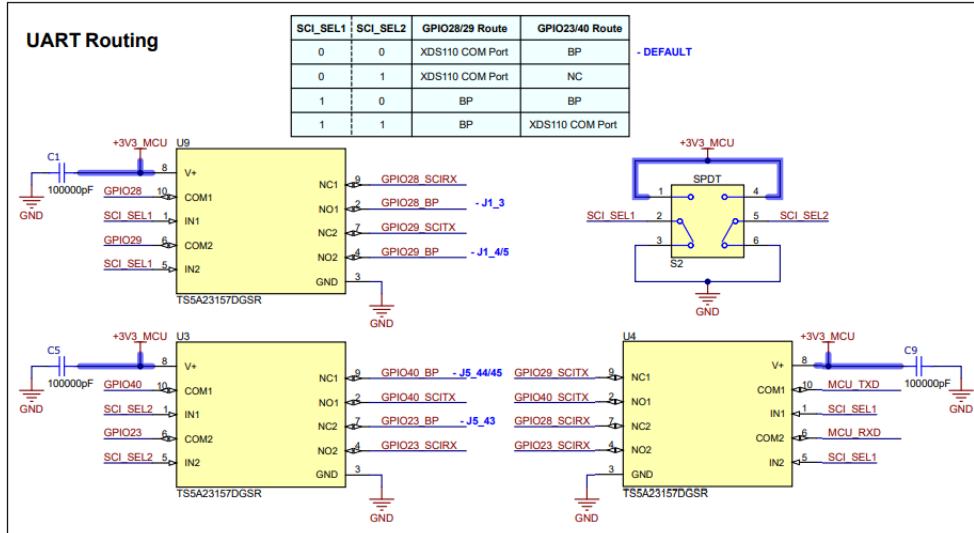


图 4-8. F2800157 LaunchPad UART 布线原理图

但是, 用户也可以选择使用跳线将 XDS110 COM 端口从外部布线到 GPIO BoosterPack™ (BP) 接头。如果选择了备用 SCI GPIO 分配, 并且需要连接到 XDS COM 端口, 这样布线会很有用。以下步骤演示了如何从外部将 XDS110 COM 端口布线到 F2800157 LaunchPad 上的 GPIO28 或 GPIO29。

1. 确保 UART 布线设置为连接到 GPIO28 和 GPIO29 (SCI_Sel1 = 1) 的 BoosterPack™ (BP); 而不是从内部连接到 XDS110 COM 端口。这样, SCI-A 信号可以输出至 BP 接头引脚。

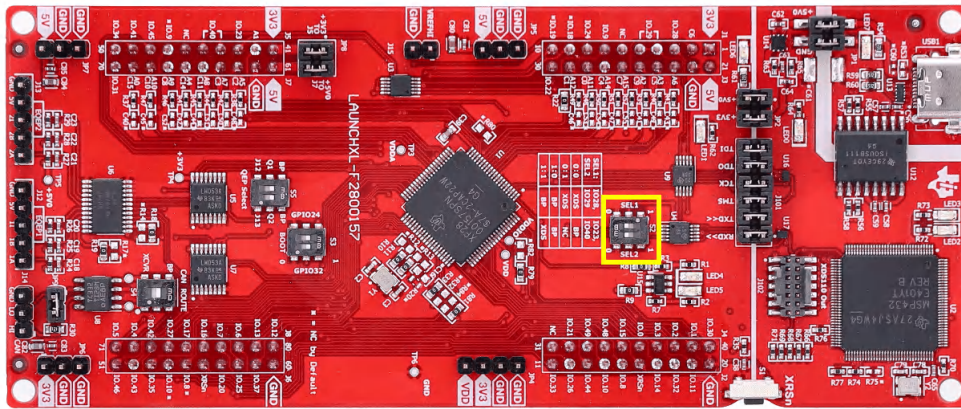


图 4-9. 将 SCI TX 或者 RX (GPIO28 或 GPIO29) 信号路由至 BoosterPack (BP) 引脚

2. 移除 J101 接头上用于 TXD 和 RXD 引脚的跳线。请注意, XDS RXD 引脚 (更靠近 XDS 电路) 连接到 MCU TXD, 而该引脚更靠近 XDS 电路。同样, XDS TXD 连接到 MCU RXD。
3. 根据电路板原理图中的 XDS110 目标接口部分, 将一条跳线从 XDS TXD 连接到 GPIO28 (SCI-A RX), 并将一条跳线从 XDS RXD 连接到 GPIO 29 (SCI-A TXD)。

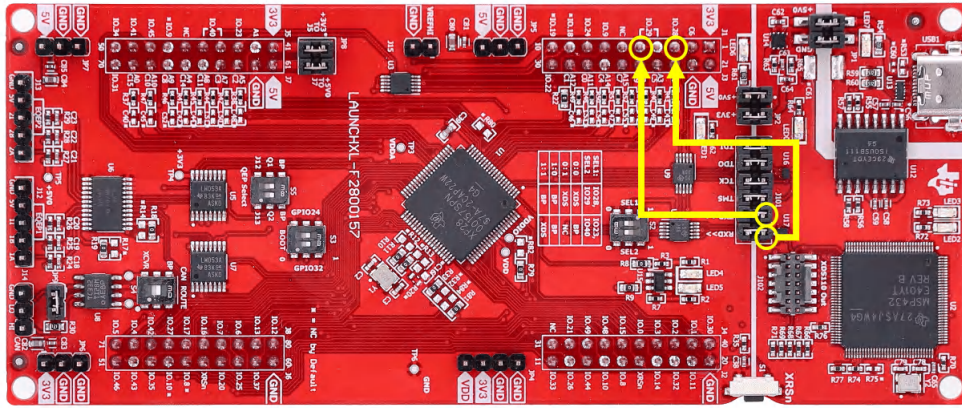


图 4-10. 将 XDS TX、RX 跳转到 SCI TX、RX GPIO

现在需要设置将器件，以模拟使用引导选项 0x01、SCIRXA = GPIO28 且 SCITXA = GPIO29 的零引脚 SCI 引导。

1. 打开 CCS 的工作区。
2. 选择 View > Target Configurations。

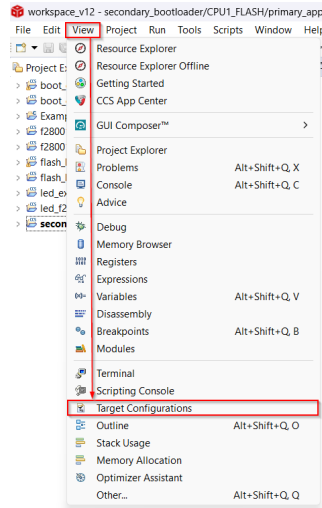


图 4-11. 在 CCS 中打开“Target Configuration”菜单

3. 用户可以将该器件的工程导入 CCS 并使用该工程连接器件，或将目标配置文件 (.ccxml) 从 C2000Ware (C2000Ware_x_xx_xx_xx > device_support > DEVICE_FAMILY > common > targetConfigs) 复制到用户定义的目标配置。
 - a. 找到器件目标配置，然后右键单击手动启动。

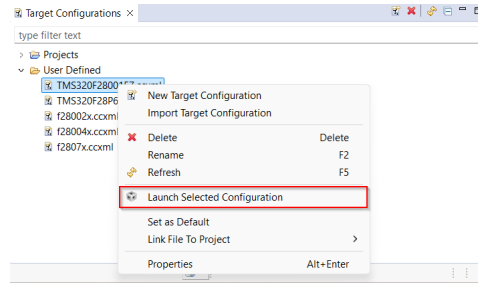


图 4-12. 启动 CCS 中的目标配置

4. CCS 启动调试窗口时，选择目标 CPU 并连接到目标。

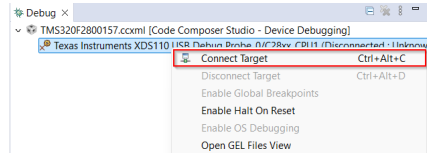


图 4-13. 连接到 CCS 中的目标核心

5. 如果弹出一个窗口，表明引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照节 5.3 中的说明调试引导 ROM。
6. 加载符号后，转到 View > Memory Browser 以打开 Memory Browser。

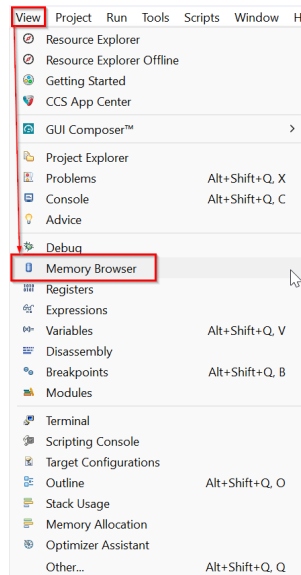


图 4-14. 导航到 CCS 中的 Memory Browser

7. 在“Memory Browser”选项卡中，导航到地址 0xD00。请记住，位置 0xD00 使用有效性密钥 (EMU-BOOTPIN-CONFIG) 指定 BMSP，0xD04-0xD05 指定引导定义 (EMU-BOOTDEF-LOW)。
8. 目标是配置 SCIRXDA 为 GPIO28 且 SCITXDA 为 = GPIO29 的零引脚 SCI 引导，因此需要禁用所有 BMSP 并且 EMU-BOOTDEF-LOW 需要设置为最低索引中的 0x01。如果引导选项编程为 EMU-BOOTDEF-LOW 中的任何其他条目，则不会选择目标引导模式。
 - a. 将 0xD00-0xD01 (EMU-BOOTPIN-CONFIG) 设置为 0x5AFF FFFF。
 - b. 将 0xD04 (EMU-BOOTDEF-LOW) 设置为 0x0001。

备注

零引脚引导意味着器件自动引导至 **BOOTDEF** 表定义的第一个条目。这是通过禁用所有 **BMSP**，从而允许器件仅考虑一个引导选项来实现的。

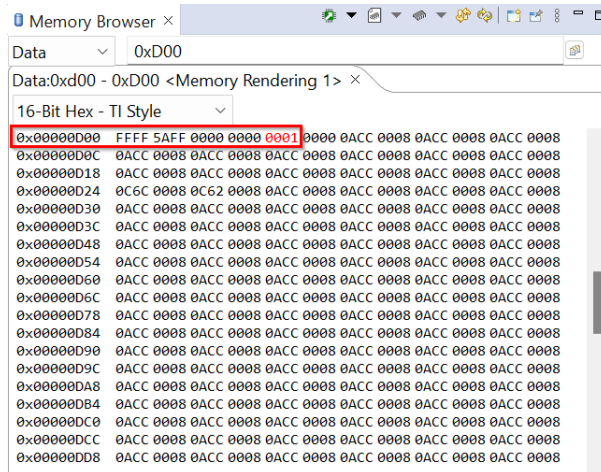


图 4-15. 模拟 **SCIRXDA** 为 **GPIO28** 且 **SCITXDA** 为 **GPIO29** 的零引脚 **SCI** 引导

9. 复位 CPU 并执行外部复位 (**XRSn**)。然后，单击 **Resume** 以开始引导序列。
10. 如果引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照 节 5.3 中的说明加载引导 ROM 符号。然后，确认器件处于 **SCI** 自动波特率锁定状态。

现在，ROM 中的 **SCI** 引导加载程序 (具有引导选项 **0x01** 指定的 **GPIO** 分配) 开始执行并等待与主机进行自动波特率锁定 (以接收 **A** 字符来确定进行通信时的波特率)。此时，器件已准备好接收来自主机的代码。

命令行 PC 实用程序是一种编程设计，可以轻松集成到脚本环境中，用于生产线编程等应用程序。它是使用 **Microsoft Visual Studio®** 用 **C++** 编写的。工程及其源代码可在 **C2000Ware (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > serial_flash_programmer)** 中找到。

若要使用此工具对 **C2000** 器件进行编程，请确保目标板已复位且当前处于上面配置的 **SCI** 引导模式并连接至 **PC COM** 端口。下面介绍了该工具在单核 **SCI** 引导中的命令行用法，其中 **-d**、**-k**、**-a**、**-p** 是必需参数。如果忽略波特率，则波特率默认设置为 **9600**。有关实用程序参数的更多详细信息，请参阅 [12]。

```
serial_flash_programmer.exe -d DEVICE -k KERNEL_FILE -a APPLICATION_FILE -p COM# -b BAUDRATE -v
```

1. 导航至 **Device Manager > Ports (COM & LP)**，找到 **XDS110 UART COM** 端口。

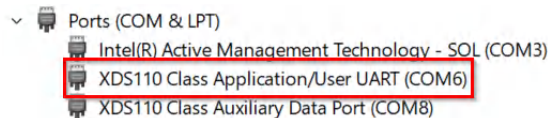


图 4-16. 在器件管理器中查找 **XDS COM** 端口

2. 导航至包含已编译 **serial_flash_programmer** 可执行文件的文件夹 (**C2000Ware_x_xx_xx_xx > utilities > flash_programmers > serial_flash_programmer**)。使用以下命令运行可执行文件 **serial_flash_programmer.exe**。

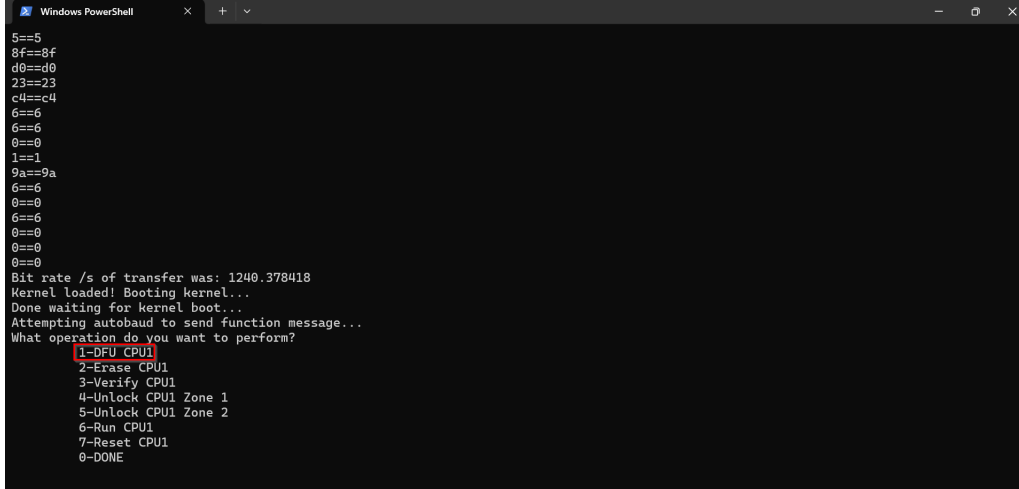
```
serial_flash_programmer.exe -d DEVICE_NAME -k <path_to_kernel_hex> -a <path_to_application_hex> -p COM# -v
```

备注

如 节 4.1 中所述，闪存内核和闪存应用程序都必须采用 **SCI8** 引导格式。

这将自动连接到器件，执行自动波特率锁定，并将 CPU1 内核下载到 RAM 中并执行。现在，CPU1 内核正在运行并等待来自主机的数据包。

1. `serial_flash_programmer` 将选项输出到屏幕上以供选择，而这些选项会发送到器件内核。选择 **1-DFU CPU1** 以将应用程序刷写到 CPU1。这种情况下，原始命令已经指定了应用程序文件，因此此时不需要其他信息。

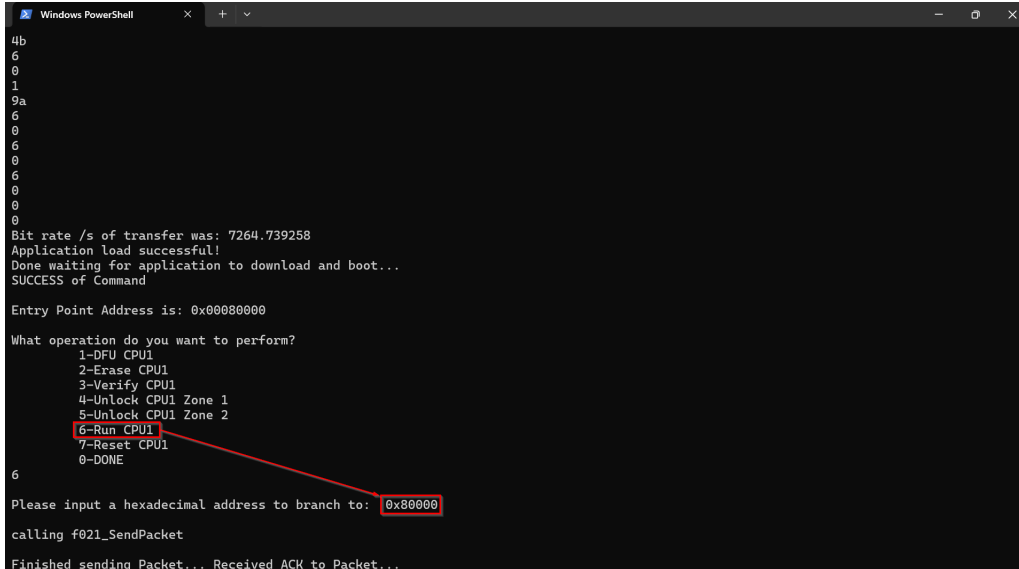


```

Windows PowerShell
5==5
8f==8f
d0==d0
23==23
cd==cd
6==6
6==6
0==0
1==1
9a==9a
6==6
0==0
6==6
0==0
0==0
0==0
0==0
Bit rate /s of transfer was: 1240.378418
Kernel loaded! Booting kernel...
Done waiting for kernel boot...
Attempting autobaud to send function message...
What operation do you want to perform?
1-DFU CPU1
2-Erase CPU1
3-Verify CPU1
4-Unlock CPU1 Zone 1
5-Unlock CPU1 Zone 2
6-Run CPU1
7-Reset CPU1
0-DONE
    
```

图 4-17. 命令串行闪存编程器下载应用程序

2. 执行命令后，需要执行应用程序。要运行应用程序，请选择 **6-Run CPU1** 并指定转移地址 `0x0008 0000` (应用程序的闪存入口点)。SCI 引导成功加载到器件上的应用程序现在执行闪存中加载的应用程序。



```

Windows PowerShell
4b
6
0
1
9a
6
0
6
0
6
0
0
Bit rate /s of transfer was: 7264.739258
Application load successful!
Done waiting for application to download and boot...
SUCCESS of Command

Entry Point Address is: 0x00080000

What operation do you want to perform?
1-DFU CPU1
2-Erase CPU1
3-Verify CPU1
4-Unlock CPU1 Zone 1
5-Unlock CPU1 Zone 2
6-Run CPU1
7-Reset CPU1
0-DONE
6

Please input a hexadecimal address to branch to: 0x80000
calling f021_SendPacket
Finished sending Packet... Received ACK to Packet...
    
```

图 4-18. 将应用程序加载到闪存后运行应用程序

4.2.3 CAN 引导

备注

尽管这些步骤在 F280039C LaunchPad 上执行过，但一般流程可以轻松应用于支持自定义 BMSP 和引导定义表的任何 C2000 器件 (表 2-3 中提供了所有器件)。有关要引导加载的器件的详细信息，请参阅器件特定的 TRM。

DCAN 闪存内核基于 ROM 引导加载程序，它与 C2000Ware 中提供的主机 PC 应用程序 (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > dcan_flash_programmer) 进行通信，并就接收数据包和分配给它的命令的完成情况向主机提供反馈。主机应用程序的源代码和可执行文件可在

dcan_flash_programmer 文件夹中找到。有关内核功能的更详细说明，请参阅 [C2000 微控制器 CAN 闪存编程应用手册 \[16\]](#)。

C2000Ware 中的 dcan_flash_programmer 支持以下器件：

1. F28003x
2. F280015x
3. F28P65x

备注

在 F280015x LaunchPad 中，内置 CAN 收发器和连接器没有映射到任何可能的 CAN GPIO 分配，需要进行焊接以将正确的 GPIO 短接到收发器 [32]。

备注

本报告中使用的术语 DCAN 定义为 DCAN 闪存内核、DCAN 闪存编程器等，指的是由 Bosch 设计的控制器局域网通信接口 (CAN) [13] 版本 D。本文档中描述的 DCAN 闪存编程器是指 CAN 模块。

DCAN 闪存内核基于 DCAN ROM 加载程序源。为了使该代码能够擦除和编程闪存，必须合并闪存 API，这是通过链接闪存 API 来完成的。

在接收到任何应用程序数据之前，F28P65x 和 F280015x DCAN 闪存内核会擦除器件的闪存，为编程做好准备。F28P65x 和 F280015x DCAN 闪存内核工程允许用户指定在进行应用程序编程之前要擦除哪些闪存组和闪存扇区。[C2000 微控制器 CAN 闪存编程应用手册 \[16\]](#) 对此进行了更详细的讨论。

在闪存存储器中的相应位置进行擦除后，应用程序加载开始。缓冲区用于保存接收到的连续应用程序代码块。当缓冲区已满或检测到新的非连续数据块时，将对缓冲区中的代码进行编程。此过程一直持续到收到整个应用程序为止。

在首次写入扇区之前，F28003x DCAN 闪存内核会检查该扇区是否已被擦除。如果尚未擦除该扇区、则 F28003x 闪存内核使闪存 API 执行擦除操作。此后，一个缓冲区将被写入闪存的内容填满，并从闪存 API 发送编程命令。一旦发生写入，闪存内核就会通过闪存 API 验证相应段是否已写入闪存的正确地址。一旦内核将所有内容复制到闪存，工程就会跳转到映像的入口地址。

DCAN 模块在闪存内核中初始化后，该模块等待主机发送固件映像。闪存内核一次从主机接收 8 个字节，并将内容放入中间 RAM 缓冲区中。然后，将该缓冲区以 128 位或 512 位增量写入闪存。

- F28P65x 和 F280015x DCAN 闪存内核工程支持 512 位编程
 - 如果需要，还有一个可用于 F28P65x 器件的 128 位编程工程
 - F280015x 闪存 API 支持 128 位编程，但闪存内核是用 512 位编程实现的
- F28003x DCAN 闪存内核工程支持 128 位编程

存储在闪存中的固件映像的所有段都根据一次性编程的位数对齐。

- **如果一次性编程 128 位 (F28003x)，则应用程序的闪存段与 128 位边界对齐。**在固件镜像的连链接器命令文件中，所有初始化段都需要映射到闪存扇区，并且在每次映射之后，需要添加 ALIGN(8) 指令以验证 128 位对齐。
- **如果一次性编程 512 位 (F280015x 和 F28P65x)，则应用程序的闪存段与 512 位边界对齐。**在固件镜像的连接命令文件中，所有初始化段都需要映射到闪存扇区，并且在每次映射之后，需要添加 ALIGN(32) 指令以验证 512 位对齐。

备注

ALIGN(x) 指令会插入填充字节直到已编程位置在 x 字边界上对齐。对于 C2000，字大小为 16 位，因此 16 位编程需要使用 ALIGN(1)，32 位编程需要使用 ALIGN(2)，依此类推。

CCS 的闪存内核源文件和工程文件在 C2000Ware 中提供，位于相应器件的示例目录中。这些工程有一个编译后处理步骤，将编译和链接的 .out 文件转换为 DCAN ROM 引导加载程序所需的正确的十六进制格式引导文件，并按扩展名为 .txt 的工程名称进行保存。

通过检查 C2000Ware_x_xx_xx_xx > boards > (LaunchPads or controlCARDs) > DEVICE_NAME > Rev# > documentation 中 CAN 收发器和连接器的原理图，可以确认 LaunchPad 中 CAN 布线所需的正确 GPIO 分配。在 LAUNCHXL-F280039C 上，收发器 RXD 从内部布线到 GPIO5，TXD 从内部布线到 GPIO4，因此需要配置引导选项 0x02。

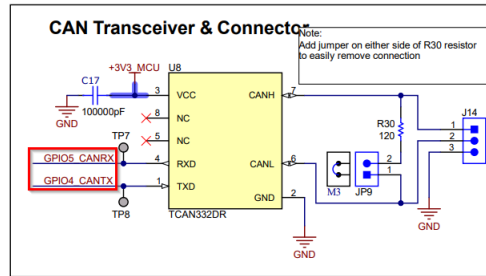


图 4-19. F280039C LaunchPad CAN 收发器和连接器原理图

1. 在 C2000Ware 中查找适用于目标器件的 DCAN 闪存内核工程并导入 CCS。例如，F28003x 的 DCAN 闪存内核位于 C2000Ware_x_x_xx_xx > driverlib > f28003x > examples > flash。
2. 确保 DCAN 闪存内核工程的 Active Build Target Configuration 设置为 RAM，因为需要链接内核才能在 RAM 中执行。
3. 如果使用 LaunchPad，请应用 Project Properties > CCS Build > C2000 Compiler > Predefined Symbols 中的预定义符号 `_LAUNCHXL_F280039C`，以便在 device.h 中为 LaunchPad 使用正确的 GPIO 分配。

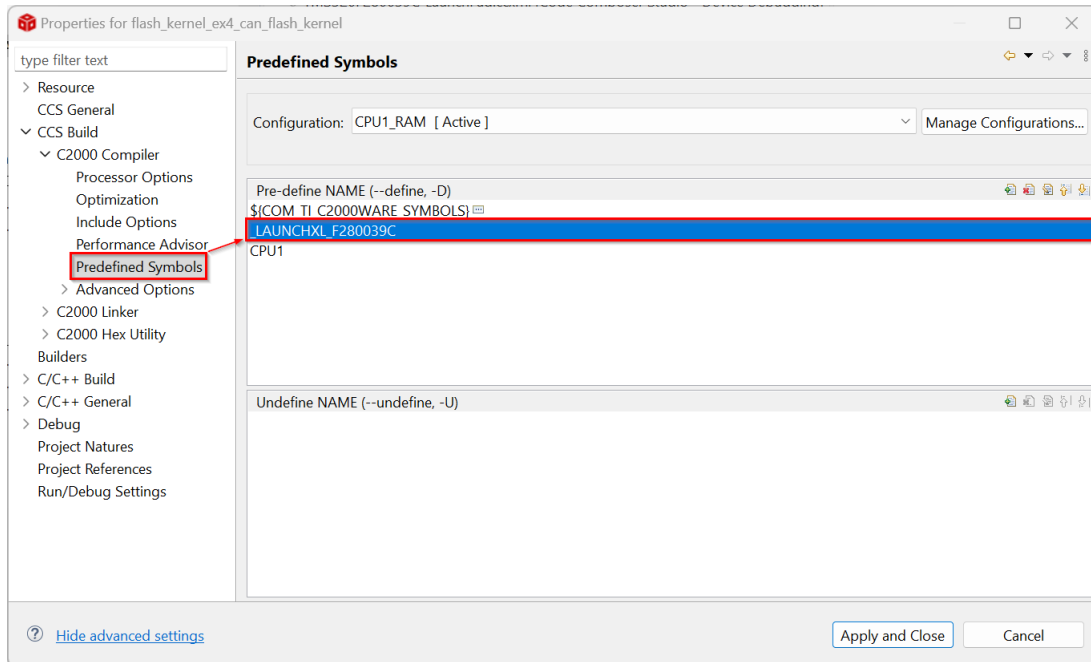


图 4-20. 为正确的 LaunchPad CAN GPIO 分配添加预定义符号

4. 上电时，器件引导 ROM 由片上 10MHz 振荡器 (INTOSC2) 提供时钟。该值需要设置为闪存内核中的主内部时钟源，并且是复位时的默认时钟。在 device.h 中，取消注释并在第 295 行使用 `#define USE_PLL_SRC_INTOSC`。注释掉 `#define USE_PLL_SRC_XTAL`。
5. 在 DCAN_boot.c 中，确认 bootloader_can_timing.h 的本地副本包含在第 64 行，而不是包含在 include 文件夹中自动生成的头文件中。

```

54 //
55 // Included Files
56 //
57 #include "can.h"
58 #include "cpu1bootrom.h"
59 #include "cpu1brom_utils.h"
60 #include "inc/hw_memmap.h"
61 #include "inc/hw_sysctl.h"
62 #include "inc/hw_can.h"
63 #include "inc/hw_gpio.h"
64 #include "bootloader_can_timing.h"
65
66 #include "F021_F28003x_C28x.h"
67 #include "flash_programming_f28003x.h"

```

图 4-21. 包括本地 CAN 时序头文件

6. 在 `bootloader_can_timing.h` 中，确认第 51-53 行定义了以下 CAN 时序设置：
 - a. 1Mbps 比特率
 - b. 20Mhz CAN 时钟
 - c. 20ms 位时间

```

48 //
49 // CAN bit timing settings for running CAN at 100kbps with a 20MHz crystal
50 //
51 #define CAN_CALC_BITRATE 1000000U
52 #define CAN_CALC_CANCLK 20000000U
53 #define CAN_CALC_BITTIME 20U

```

图 4-22. 确认 CAN 时序设置

7. 构建内核工程。这些工程有一个编译后处理步骤，将编译和链接的 `.out` 文件转换为 DCAN ROM 引导加载程序所需的正确的十六进制格式引导文件，并按扩展名为 `.txt` 的示例名称进行保存。
 - a. 如果未生成 `.txt` 输出，则按照 节 4.1 中的步骤操作，以确保定义了用于生成十六进制文件的正确编译后处理步骤。

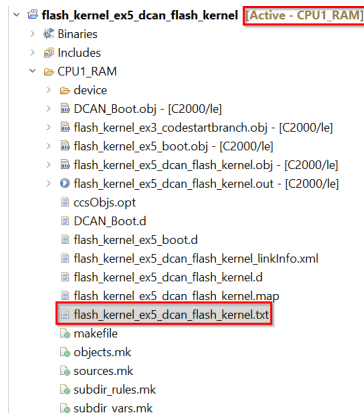


图 4-23. 生成 DCAN 内核 txt 输出

8. 在文本编辑器中打开闪存内核 `txt` 文件并将第 3-4 字节更改为 `C0 7A`。这些字节包含用于配置 CAN 总线比特率的元数据。
9. 对内核加载到闪存中的固件代码重复执行构建过程。
 - a. 请确认为闪存设置了 **Active Build Target Configuration**。
 - b. 确认已定义用于生成十六进制文件的正确编译后处理步骤。
 - c. 构建固件工程。

在 CCS 中构建内核和固件工程后，正确设置器件硬件，使其能与运行 C2000Ware 中提供的 `dcan_flash_programmer` 的主机 PC 进行通信。首先要做的是确保正确配置引导模式选择引脚，以将器件引导至

CAN 引导模式。如果用户需要在片上闪存中从外部主机加载代码，则用户可以使用默认 BMSP (如果支持) 配置 CAN 引导，或者可以配置 BOOTPIN-CONFIG 和 BOOTDEF 寄存器。

用于启用 CAN 引导的默认 BMSP 可以在 [TMS320F28003x 实时微控制器数据表](#) 中找到。如果用户将 GPIO24 设置为 1 并将 GPIO32 设置为 0，则引导 ROM 跳转到 CAN 引导加载程序且 CANRXA 设置为 GPIO5，CANTXA 设置为 GPIO4，并且无需对器件寄存器进行编程。

但是，如果用户希望灵活地使用具有不同 GPIO 分配的 CAN 引导，则需要为特定的引导选项配置 OTP 或仿真 BOOTCONFIG 和 BOOTDEF 寄存器。请参阅 [TMS320F28003x 实时微控制器数据表](#) 中的 [GPIO 分配](#)，以确定哪种 CAN 引导选项符合 GPIO 要求。

运行示例所需的硬件组件是连接到 CAN 收发器的 C2000 器件和 PEAK PCAN-USB Pro FD 分析仪。

1. LaunchPad 器件包含一个板载 CAN 收发器。确认 PEAK PCAN-USB Pro FD 分析仪通过接地、CAN-Lo 和 CAN-Hi 连接来连接到 LaunchPad。
2. 确认板载 CAN 路由由开关需要设置为低电平 (至收发器)，以便收发器使用 GPIO 进行通信。

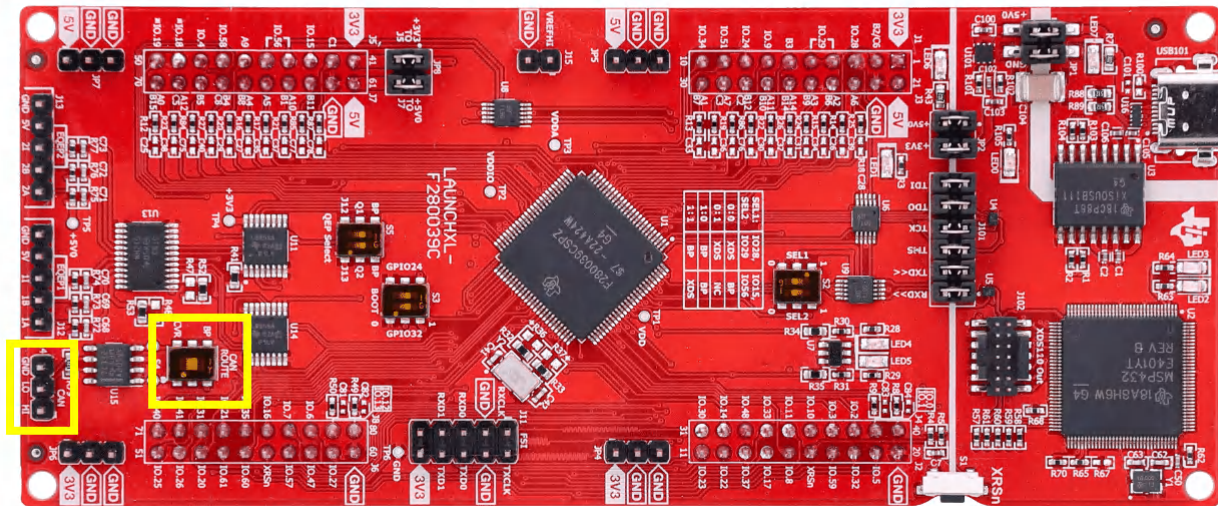


图 4-24. 设置 CAN 路由开关

备注

对于 controlCard，需要使用定制设计的 CAN 收发器板和 HSEC 180 引脚 controlCard 扩展坞。定制设计的收发器板通过四种连接方式连接到 controlCard：接地、3.3V、CANTX 和 CANRX。

现在需要设置将器件，以模拟使用引导选项 0x02、CANRXA = GPIO5 和 CANTXA = GPIO4 的 CAN 引导。只有一个大约 10 秒的超时窗口，用于可将第一个 CAN 帧发送到器件。

1. 打开 CCS 的工作区。
2. 选择 View > Target Configurations。

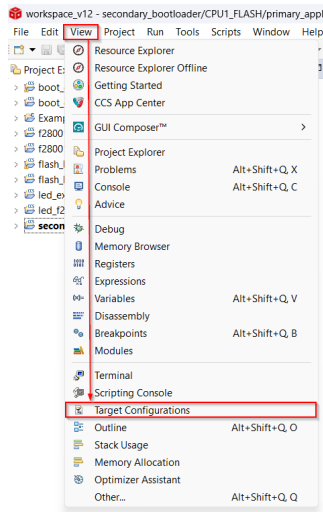


图 4-25. 在 CCS 中打开 “Target Configuration” 菜单

3. 用户可以将该器件的工程导入 CCS 并使用该工程连接器件，或将目标配置文件 (.ccxml) 从 C2000Ware (C2000Ware_x_xx_xx_xx > device_support > DEVICE_FAMILY > common > targetConfigs) 复制到用户定义的目标配置。
 - a. 找到器件目标配置，然后右键单击手动启动。

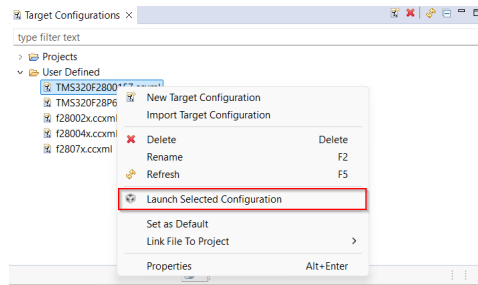


图 4-26. 启动 CCS 中的目标配置

4. CCS 启动调试窗口时，选择目标 CPU 并连接到目标。

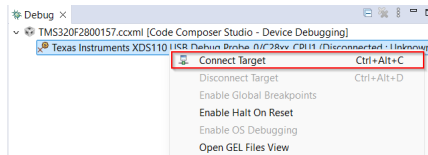


图 4-27. 连接到 CCS 中的目标核心

5. 如果弹出一个窗口，表明引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照节 5.3 中的步骤调试引导 ROM。
6. 加载符号后，转到 View > Memory Browser 以打开 Memory Browser。

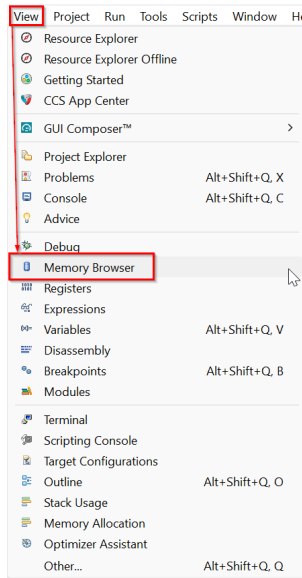


图 4-28. 导航到 CCS 中的 Memory Browser

7. 在“Memory Browser”选项卡中，导航到地址 0xD00。请记住，位置 0xD00 使用有效性密钥 (EMU-BOOTPIN-CONFIG) 指定 BMSP，0xD04-0xD05 指定引导定义 (EMU-BOOTDEF-LOW)。
8. 目标是配置 CANRXA = GPIO5 且 CANTXA = GPIO4 的零引脚 CAN 引导，因此需要禁用所有 BMSP 并且 EMU-BOOTDEF-LOW 需要设置为最低索引中的 0x02。如果引导选项编程为 EMU-BOOTDEF-LOW 中的任何其他条目，则不会选择目标引导模式。
 - a. 将 0xD00-0xD01 (EMU-BOOTPIN-CONFIG) 设置为 0x5AFF FFFF。
 - b. 将 0xD04 (EMU-BOOTDEF-LOW) 设置为 0x0002。

备注

零引脚引导意味着器件自动引导至 BOOTDEF 表定义的第一个条目。这是通过禁用所有 BMSP，从而允许器件仅考虑一个引导选项来实现的。

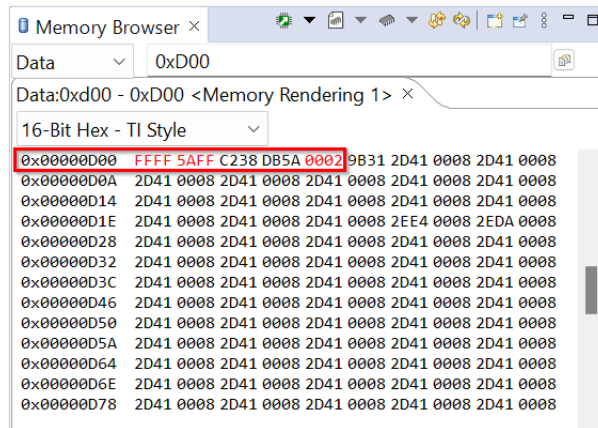


图 4-29. 模拟 CANRXA=GPIO5 且 CANTXA=GPIO4 的零引脚 CAN 引导

9. 复位 CPU 并执行外部复位 (XRSn)。然后，单击 *Resume* 以开始引导序列。
10. 如果引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照 节 5.3 中的说明加载引导 ROM 符号。然后，确认器件处于 CAN 引导状态。

现在，ROM 中的 CAN 引导加载程序 (包含引导选项 0x02 指定的 GPIO 分配) 开始执行并等待从主机发送 CAN 帧。此时，器件已准备好接收来自主机的代码。

备注

CAN 引导加载程序有一个 10 秒的窗口，用于接受 CAN 帧。如果在器件引导选项提供，则 SEND_TEST 模式会消除超时。这些选项使用相同的 GPIO 引脚进行通信，作为等效引导选项。但是，GPIO 引脚会在开始引导加载过程之前传输一个帧，帮助确定模块的功能是否正常。在 F280039x 器件中，可以使用引导选项 0x82 来避免超时，同时使用相同的 GPIO 分配作为引导选项 0x02。

命令行 PC 实用程序是一种编程解决方案，可以轻松集成到脚本环境中，用于生产线编程等应用程序。它是使用 Microsoft Visual Studio 用 C++ 编写的。工程及其源代码可在 C2000Ware (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > dcan_flash_programmer) 中找到。

主机负责将 DCAN 内核映像和闪存 (固件) 映像发送到 MCU。PEAK PCAN-USB Pro FD CAN 总线分析仪用作主机。闪存编程器工程在 Visual Studio 2019 上编译并运行。主机编程器使用 PEAK 的 PCAN_Basic API。[17]。PCAN_Basic API 可用于在 CAN 分析仪上发送和接收 CAN 帧。

备注

PEAK PCAN-USB Pro FD CAN 总线分析仪向后兼容，可以接收 CAN 帧以及 CAN-FD 帧。

在 F28003x 器件上，CAN 模块的时钟由引导 ROM 切换到外部时钟源。LaunchPad 和 controlCard 中的外部时钟为 20MHz。引导 ROM 将标称比特率配置为 100Kbps。主机 CAN 编程器将 PEAK CAN 分析仪配置为具有相同的时钟和标称比特率值。

主机初始化分析仪以使用 CAN，以 2 字节增量发送内核，然后以 8 字节增量发送映像，每帧之间延迟 10ms，以便闪存 API 有时间将接收到的数据编程到闪存中。一旦固件映像被写入，主机 CAN 编程器就会退出。

若要使用此工具对 C2000 器件进行编程，请确保目标板已复位且当前处于前面配置的 CAN 引导模式，并连接至 PEAK PCAN-USB Pro FD CAN 总线分析仪。下面介绍了该工具在单核 CAN 引导中的命令行用法，其中 -d、-k 和 -a 是必需参数。可以使用 -v 启用详细输出。有关该实用程序参数的更多详细信息，请参阅 [4]。

```
dcan_flash_programmer.exe -d DEVICE -k KERNEL_FILE -a APPLICATION_FILE -v
```

1. 导航至包含已编译可执行文件 dcan_flash_programmer 的文件夹 (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > dcan_flash_programmer)。
2. 打开终端并使用以下命令运行可执行文件 dcan_flash_programmer :

```
dcan_flash_programmer.exe -d DEVICE_NAME -k <path_to_kernel_hex> -a <path_to_application_hex> -v
```

首先，这会使用引导加载程序将 DCAN 闪存内核加载到器件的 RAM 中。通过 CAN 总线传输的字节可以在终端中看到。然后，内核会执行并加载，而后再用“-a”命令行参数指定的文件对闪存编程，如图 4-30 和图 4-31 所示。如果成功加载到闪存中，内核会转移到应用程序并开始执行。

```

69
ff
84
fe
6
0
2
fe
42
1e
a9
28
0
8
82
fe
6
0
ff
76
6c
ce
67
3e
6
0
0
0
Kernel Loaded

```

图 4-30. 已加载的 DCAN 闪存编程器内核

```

84
fe
6
0
48
76
3f
2f
6
0
48
76
38
2e
6
0
1
9a
6
0
6
0
6
0
6
0
0
0
Application Load Completed

```

图 4-31. 已加载的 DCAN 闪存编程器应用程序

4.2.4 CAN-FD 引导

备注

尽管这些步骤在 F280039C LaunchPad 上执行过，但一般流程可以轻松应用于支持自定义 BMSP 和引导定义表的任何 C2000 器件（表 2-3 中提供了所有器件）。有关引导加载的器件，请参阅器件特定的 TRM。

如节 3.2 中所述，ROM 引导加载程序只能将代码加载到 RAM 中，这就是使用 ROM 引导加载程序在闪存内核中加载以便代码可以存储到闪存中的原因。CAN 闪存内核基于 ROM 引导加载程序，它与 C2000Ware 中提供的主机 PC 应用程序 (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > can_flash_programmer) 进行通信，并就接收数据包和分配给它的命令的完成情况向主机提供反馈。主机应用程序的源代码和可执行文件可在 can_flash_programmer 文件夹中找到。有关内核功能的更多信息，请参阅 [C2000 微控制器的 CAN 闪存编程应用手册 \[16\]](#)。

C2000Ware 中的 can_flash_programmer 支持以下器件：

1. F28003x
2. F28P55x
3. F28P65x

备注

在本文档中，术语 MCAN、MCAN 闪存内核、CAN 闪存编程器等指的是模块化控制器局域网 (MCAN)。MCAN 是可与控制器局域网灵活数据速率 (CAN-FD) [15] 互换使用的术语。本文档中描述的 CAN 闪存编程器是指 MCAN 模块。

闪存内核工程是根据 MCAN ROM 引导加载程序建模的。它直接进入已修改为写入闪存的 MCAN_Boot 函数。闪存内核的 MCAN 模块初始化与引导加载程序相同。MCAN 模块的时钟源、标称和数据比特率、GPIO 引脚等由内核在初始化时根据引导模式进行设置。

在接收到任何应用程序数据之前，F28P55x 内核会擦除器件的闪存，为编程做好准备。此外，F28P55x MCAN 闪存内核工程允许用户指定在进行应用程序编程之前要擦除哪些闪存组和闪存扇区。C2000 微控制器的 CAN 闪存编程应用手册 [16] 的自定义闪存组和扇区擦除一节中对此进行了更加详细的介绍。

在对应用程序进行编程之前，F28003x 内核会检查每个闪存扇区是否被擦除过。如果之前没有擦除闪存扇区，则会擦除该扇区并写入应用程序数据。

F28P65x 内核会在应用程序下载过程开始时擦除闪存。擦除闪存可能需要几秒时间。请注意，当应用程序加载显示为失败时，说明闪存正在被擦除。

在闪存存储器中的相应位置进行擦除后，应用程序加载开始。闪存内核一次从主机接收 64 个字节，并将内容放入中间 RAM 缓冲区中。然后，将该缓冲区以 128 位或 512 位增量写入闪存。

- F28003x 和 F28P65x MCAN 闪存内核一次对 128 位进行写入
- F28P55x MCAN 闪存内核一次对 512 位进行写入

此后，RAM 缓冲区将被写入闪存的内容填满，并从闪存 API 发送编程命令。一旦发生写入，闪存内核就会通过闪存 API 验证相应段是否已写入闪存的正确地址。一旦内核将所有内容复制到闪存，工程就会跳转到映像的入口地址。

存储在闪存中的固件映像的所有段都必须根据一次性编程的位数对齐。

- 如果一次编程 128 位 (F28003x 和 F28P65x)，则应用程序的这些闪存段会与 128 位边界对齐。在固件映像的连接器命令文件中，所有初始化段都需要映射到闪存扇区。在每次映射之后，需要添加 ALIGN(8) 指令以确保 128 位对齐。
- 如果一次编程 512 位 (F28P55x)，则应用程序的这些闪存段会与 512 位边界对齐。在固件映像的连接器命令文件中，所有初始化段都需要映射到闪存扇区。在每次映射之后，需要添加 ALIGN(32) 指令以确保 512 位对齐。

备注

ALIGN(x) 指令会插入填充字节直到已编程位置在 x 字边界上对齐。对于 C2000，字大小为 16 位，因此 16 位编程需要使用 ALIGN(1)，32 位编程需要使用 ALIGN(2)，依此类推。

用于传输应用程序数据的协议遵循 MCAN ROM 加载程序协议。使用原始 MCAN ROM 加载程序协议，使用的标称比特率为 1Mbps，并且每帧从主机向目标器件传输 64 个字节，以实现标称位时序。该协议使用的数据比特率是 2Mbps，以实现数据位时序。

CCS 的闪存内核源文件和工程文件在 C2000Ware 中提供，位于器件相应的示例目录中。这些工程有一个编译后处理步骤，将编译和链接的 .out 文件转换为 MCAN ROM 引导加载程序所需的正确的十六进制格式引导文件，并按扩展名为 .txt 的工程名称进行保存。

通过检查 C2000Ware_x_xx_xx_xx > boards > (LaunchPads or controlCARDs) > DEVICE_NAME > Rev# > documentation 中 CAN 收发器和连接器的原理图，可以确认 LaunchPad 中 CAN 布线所需的正确 GPIO 分配。在

LAUNCHXL-F280039C 上，收发器 RXD 内部布线到 GPIO5，TXD 内部布线到 GPIO4，因此需要配置 BOOTDEF 0x02。

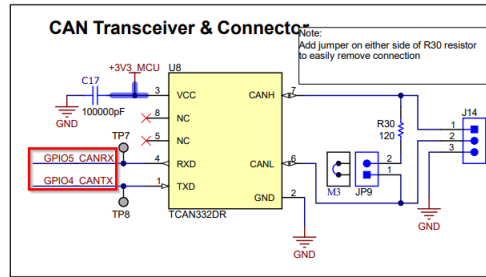


图 4-32. F280039C LaunchPad CAN 收发器和连接器原理图

1. 在 C2000Ware 中查找适用于目标器件的 MCAN 闪存内核工程并导入 CCS。例如，F28003x 的 MCAN 闪存内核位于 C2000Ware_x_x_xx_xx > driverlib > f28003x > examples > flash。
2. 确保 MCAN 闪存内核工程的 Active Build Target Configuration 设置为 RAM，因为需要链接内核才能在 RAM 中执行。
3. 在 Project Properties > CCS Build > C2000 Compiler > Predefined Symbols 中添加预定义符号 `_LAUNCHXL_F280039C`，以便在 device.h 中为 LaunchPad 使用正确的 GPIO 分配。

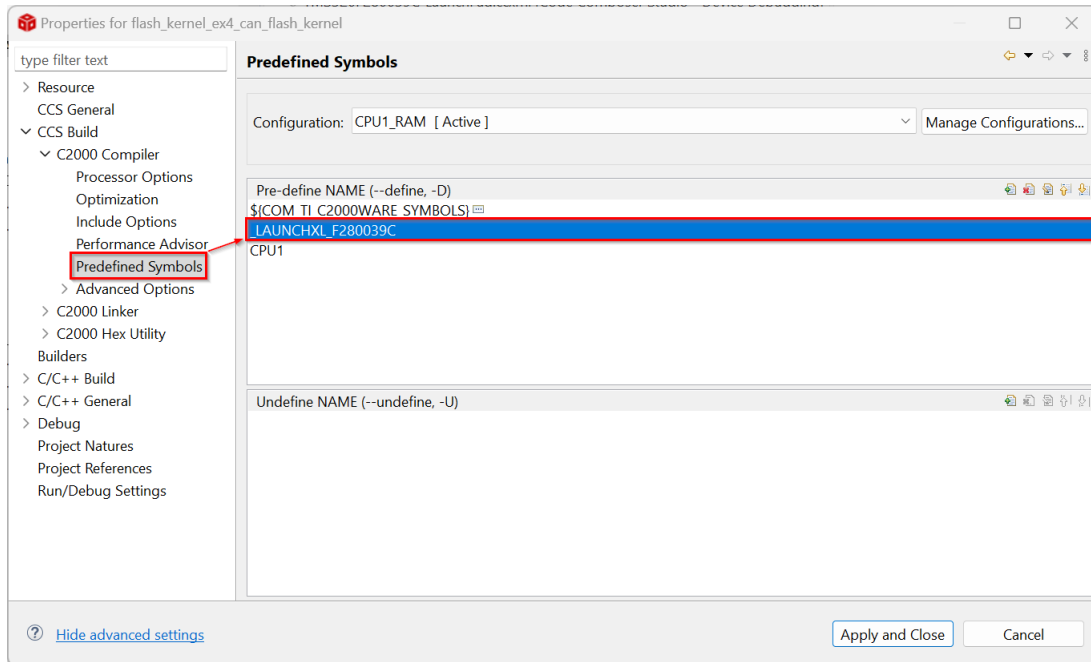


图 4-33. 为正确的 LaunchPad CAN GPIO 分配添加预定义符号

4. 上电时，器件引导 ROM 由片上 10MHz 振荡器 (INTOSC2) 提供时钟。该值需要设置为主内部时钟源，并且是复位时的默认时钟。在 device.h 中，取消注释并在第 295 行使用 `#define USE_PLL_SRC_INTOSC`。注释掉 `#define USE_PLL_SRC_XTAL`。
5. 构建内核工程。这些工程有一个编译后处理步骤，将编译和链接的 .out 文件转换为 MCAN ROM 引导加载程序所需的正确的十六进制格式引导文件，并按扩展名为 .txt 的示例名称进行保存。
 - a. 如果未生成 .txt 输出，则按照 节 4.1 中的步骤操作，以确保定义了用于生成十六进制文件的正确编译后处理步骤。

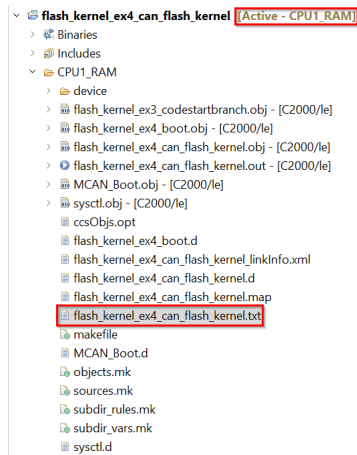


图 4-34. 生成 CAN 内核 txt 输出

6. 对内核加载到闪存中的应用程序代码重复执行构建过程。
 - a. 确认为闪存设置了 Active Build Target Configuration。
 - b. 确认已定义用于生成十六进制文件的正确编译后处理步骤。
 - c. 构建固件工程。

在 CCS 中构建内核和固件工程后，正确设置器件硬件，使其能与运行 C2000Ware 中提供的 can_flash_programmer 的主机 PC 进行通信。首先要做的是确保正确配置引导模式选择引脚，以将器件引导至 CAN-FD 引导模式。如果用户需要将外部主机中的代码加载到片上闪存中，则用户需要配置 BOOTPIN-CONFIG 和 BOOTDEF 寄存器，因为它们不作为默认引导选项提供。请参阅 TMS320F28003x 实时微控制器数据表中的 GPIO 分配，以确定哪种 CAN-FD 引导选项符合 GPIO 要求。

运行示例所需的硬件组件是连接到 CAN 收发器的 C2000 器件和 PEAK PCAN-USB Pro FD 分析仪。

1. LaunchPad 器件包含一个板载 CAN 收发器。确认 PEAK PCAN-USB Pro FD 分析仪通过接地、CAN-Lo 和 CAN-Hi 连接来连接到 LaunchPad。
2. 确认板载 CAN 路由开关需要设置为低电平（至收发器），以便收发器使用 GPIO 进行通信。

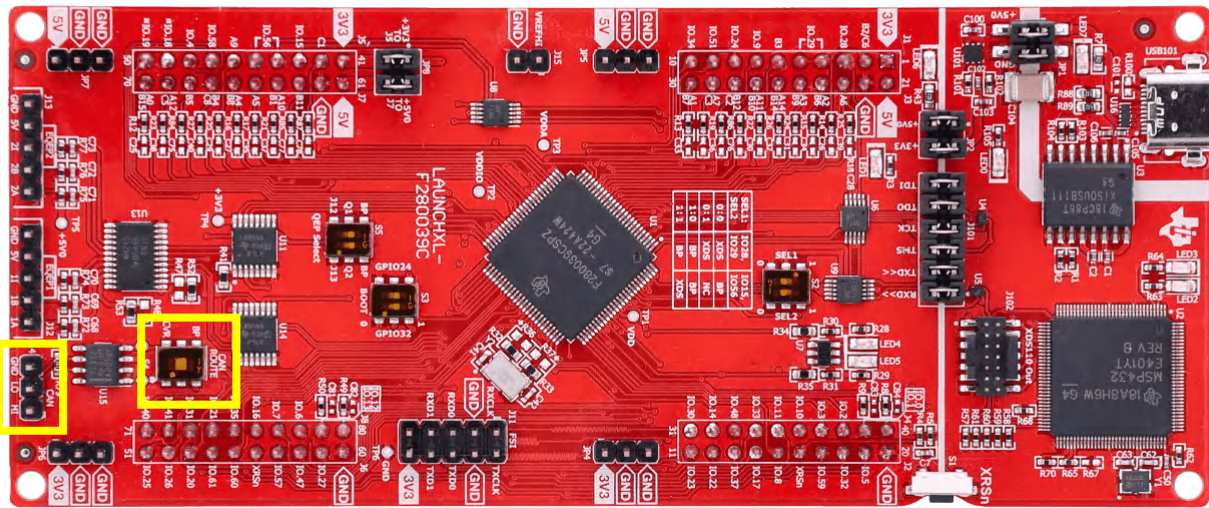


图 4-35. 设置 CAN 路由开关

备注

对于 ControlCard，需要使用定制设计的 CAN 收发器板以及 HSEC 180 引脚 ControlCard 扩展坞。定制设计的收发器板通过四种连接方式连接到 ControlCard：接地、3.3V、CANTX 和 CANRX。

需要设置器件，以模拟使用引导选项 0x02、CANRXA = GPIO5 且 CANTXA = GPIO4 的零引脚 CAN 引导。现在需要设置将器件，以模拟使用引导选项 0x08、CANRXA = GPIO5 且 CANTXA = GPIO4 的零引脚 CAN-FD 引导。请注意，只需大约 10 秒即可将第一个 CAN 帧发送到器件。

1. 打开 CCS 的工作区。
2. 选择 View > Target Configurations。

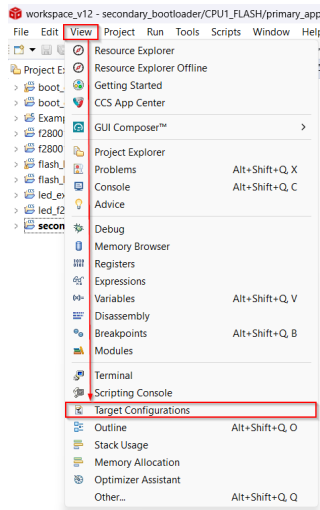


图 4-36. 在 CCS 中打开“Target Configuration”菜单

3. 用户可以将该器件的工程导入 CCS 并使用该工程连接器件，或将目标配置文件 (.ccxml) 从 C2000Ware (C2000Ware_x_xx_xx_xx > device_support > DEVICE_FAMILY > common > targetConfigs) 复制到用户定义的目标配置。

- a. 找到器件目标配置，然后右键单击手动启动：

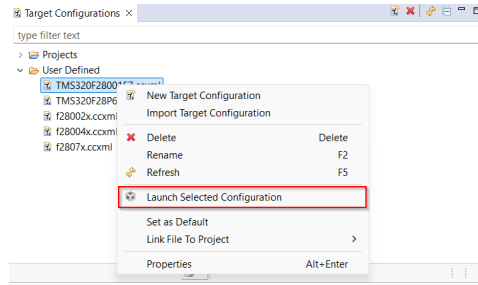


图 4-37. 启动 CCS 中的目标配置

4. CCS 启动调试窗口时，选择目标 CPU 并连接到目标。

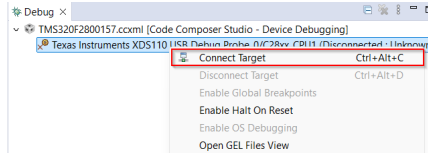


图 4-38. 连接到 CCS 中的目标核心

5. 如果弹出一个窗口，表明引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照节 5.3 中的步骤调试引导 ROM。
6. 加载符号后，转到 View > Memory Browser 以打开 Memory Browser。

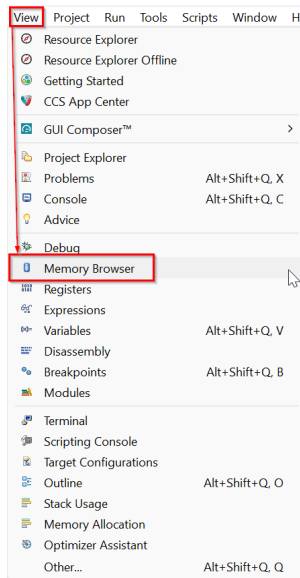


图 4-39. 导航到 CCS 中的 Memory Browser

7. 在“Memory Browser”选项卡中，导航到地址 0xD00。请记住，位置 0xD00 使用有效性密钥 (EMU-BOOTPIN-CONFIG) 指定 BMSP，0xD04-x0D05 指定引导定义 (EMU-BOOTDEF-LOW)。
8. 目标是配置 CANRXA = GPIO5 且 CANTXA = GPIO4 的零引脚 CAN 引导，因此需要禁用所有 BMSP 并且 EMU-BOOTDEF-LOW 需要设置为最低索引中的 0x08。如果引导选项编程为 EMU-BOOTDEF-LOW 中的任何其他条目，则不会选择目标引导模式。
- 将 0xD00-0xD01 (EMU-BOOTPIN-CONFIG) 设置为 0x5AFF FFFF。
 - 将 0xD04 (EMU-BOOTDEF-LOW) 设置为 0x0008。

备注

除了 EMU-BOOTDEF-LOW 中较低的 8 个字节外，EMU-BOOTDEF-LOW 和 EMU-BOOTDEF-HIGH 可以设置为任意值，因为在零引脚引导中会忽略这些值。

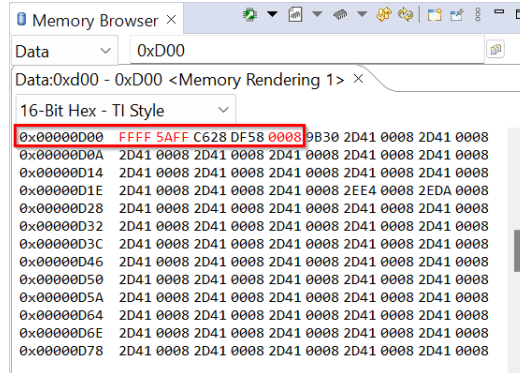


图 4-40. 模拟 CANRXA=GPIO5 且 CANTXA =GPIO4 的零引脚 MCAN 引导

9. 复位 CPU 并执行外部复位 (XRSn)。然后，单击 *Resume* 以开始引导序列。
10. 如果引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照节 5.3 中的步骤加载引导 ROM 符号。然后，确认器件处于 CAN 引导状态。

现在，ROM 中的 MCAN 引导加载程序 (包含引导选项 0x08 指定的 GPIO 分配) 开始执行并等待从主机发送 CAN 帧。此时，器件已准备好接收来自主机的代码。

备注

MCAN 引导加载程序有一个 10 秒的窗口，用于接受 CAN-FD 帧。如果在器件引导选项提供，则 SEND_TEST 模式会消除超时。这些选项使用相同的 GPIO 引脚进行通信，作为等效引导选项。但是，GPIO 引脚会在开始引导加载过程之前传输一个帧，帮助确定模块的功能是否正常。在 F280039x 器件中，可以使用引导选项 0x68 来避免超时，同时使用相同的 GPIO 分配作为引导选项 0x08。

命令行 PC 实用程序是一种编程解决方案，可以轻松集成到脚本环境中，用于生产线编程等应用程序。它是使用 Microsoft Visual Studio 用 C++ 编写的。工程及其源代码可在 C2000Ware (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > can_flash_programmer) 中找到。

主机负责将 MCAN 内核映像和闪存 (固件) 映像发送到 MCU。PEAK PCAN-USB Pro FD CAN 总线分析仪用作主机。闪存编程器工程在 Visual Studio 2019 上编译并运行。主机编程器使用 PEAK 的 PCAN_Basic API。[17]。PCAN_Basic API 可用于在 CAN 分析仪上发送和接收 CAN-FD 帧。

备注

PEAK PCAN-USB Pro FD CAN 总线分析仪向后兼容，可以接收 CAN 帧以及 CAN-FD 帧。

在 F28003x 器件上，MCAN 模块的时钟由引导 ROM 切换到外部时钟源。LaunchPad 和 ControlCard 中的外部时钟为 20MHz。引导 ROM 将标称比特率配置为 1Mbps，将数据比特率配置为 2Mbps。主机 CAN 编程器将 PEAK CAN 分析仪配置为具有相同的时钟、标称和数据比特率值。

主机初始化分析仪以使用 CAN-FD，以 64 字节增量发送内核，以 64 字节增量发送映像，每帧之间延迟 100ms，以便闪存 API 有时间对接收到存闪存中的数据编程。一旦固件映像被写入，主机 CAN 编程器就会退出。

若要使用此工具对 C2000 器件进行编程，请确保目标板已复位且当前处于前面配置的 CAN 引导模式，并连接至 PEAK PCAN-USB Pro FD CAN 总线分析仪。下面介绍了该工具在单核 MCAN 引导中的命令行用法，其中 -d、-k 和 -a 是必需参数。可以使用 -v 启用详细输出。有关该实用程序参数的更多详细信息，请参阅 [14]。

```
can_flash_programmer.exe -d DEVICE -k KERNEL_FILE -a APPLICATION_FILE -v
```

1. 导航至包含已编译 dcan_flash_programmer 可执行文件的文件夹 (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > can_flash_programmer)。
2. 打开终端并使用以下命令运行可执行文件 can_flash_programmer.exe。

```
can_flash_programmer.exe -d DEVICE_NAME -k <path_to_kernel_hex> -a <path_to_application_hex> -v
```

这会使用引导加载程序将 MCAN 闪存内核加载到器件的 RAM 中。通过 MCAN 总线传输的字节可以在终端中看到。然后，内核会执行并加载，而后再用“-a”命令行参数指定的文件对闪存编程，如图 4-41 和图 4-42 所示。如果成功加载到闪存中，内核会转移到应用程序并开始执行。

```
2
0
1a
76
6
0
2
fe
41
96
42
2b
42
92
82
fe
6
0
5
8f
d0
83
c4
6
6
0
0
0
Kernel Loaded
```

图 4-41. 已加载的 MCAN 闪存编程器内核

```
48
0
9c
21
22
76
5
8f
2e
d2
c4
1a
2
0
1a
76
6
0
1
9a
6
0
6
0
6
0
0
0
Application Load Completed
```

图 4-42. 已加载的 MCAN 闪存编程器应用程序

4.2.5 USB 引导

备注

由于 C2000 LaunchPad 没有允许 USB 数据传输的 USB 外设，因此与 controlCard 不同，这些步骤基于 F28379D controlCARD Rev 1.3。更重要的是，早期器件版本在 USB 引导加载程序中存在已知错误，阻止器件与主机 PC 通信，因此必须至少使用 F28379D Rev C 器件版本。

USB 闪存内核基于 ROM 引导加载程序，与 C2000Ware (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > usb_flash_programmer) 中提供的主机 PC 应用程序进行通信，本质上执行与 ROM 引导加载程序相同的操作。由于 ROM 引导加载程序具有闪存 API，因此引导加载程序能够擦除闪存并对其进行编程，以执行固件更新。

备注

本节演示 CPU1 USB 引导加载。有关 USB 内核功能和 CPU2 用法的更详细说明，请参阅 [C2000 微控制器的 USB 闪存编程应用报告 \[20\]](#)。

CCS 的闪存内核源文件和工程文件在 C2000Ware 中提供，位于相应器件的示例目录中。这些工程有一个编译后处理步骤，将编译和链接的 .out 文件转换为 SCI ROM 引导加载程序所需的正确的十六进制格式引导文件，并按扩展名为 .dat 的工程名称进行保存。

- 在 C2000Ware 中查找适用于目标器件的 USB 闪存内核工程并导入 CCS。
 - 对于 F2837xD 器件，内核可以在 C2000Ware_x_xx_xx_ > device_support > f2837xD > dual > F2837xD_usb_flash_kernels 中找到。
- 确保 USB 闪存内核工程的 Active Build Target Configuration 设置为 RAM，因为需要链接内核才能在 RAM 中执行。

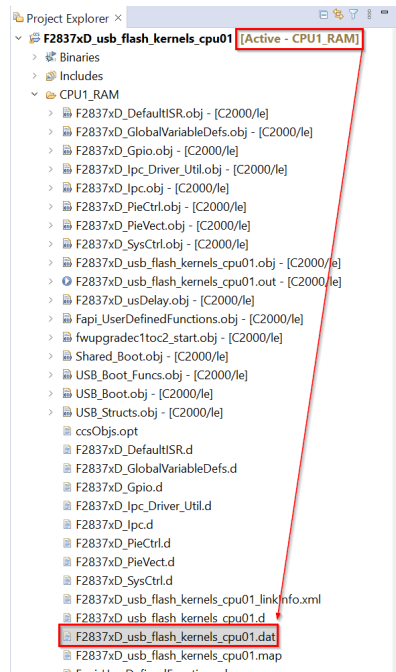


图 4-43. 生成正确的 USB 内核 Bin 输出

- 构建内核工程。这些工程有一个编译后处理步骤，将编译和链接的 .out 文件转换为 USB ROM 引导加载程序所需的正确的二进制格式，并按扩展名为 .dat 的示例名称进行保存。

备注

如果未生成 .dat 输出，则按照 [节 4.1](#) 中的步骤操作，验证是否定义了用于生成二进制文件的正确编译后处理步骤。

- 对内核加载到闪存中的固件代码重复执行构建过程。

- a. 确认为闪存设置了 Active Build Target Configuration。
- b. 按照 节 4.1 中的步骤操作，验证是否定义了用于生成二进制文件的正确编译后处理步骤。
- c. 构建固件工程。

在 CCS 中构建内核后，正确设置器件，使其与运行 `usb_flash_programmer` 的主机 PC 进行通信。

1. 在 controlCARD 和主机 PC 之间连接 mini-USB 和 micro-USB。
2. 设置 controlCARD GPIO 以使用 CCS 和卡上 XDS100v2 仿真器进行调试 [20]。有关连接选项，请参阅 C2000™ 微控制器 USB 闪存编程应用报告中的 ROM 引导加载程序一节。
 - a. 将 A:SW1 位置 1 设置为 On (上侧)，将位置 2 设置为 Off (下侧)。

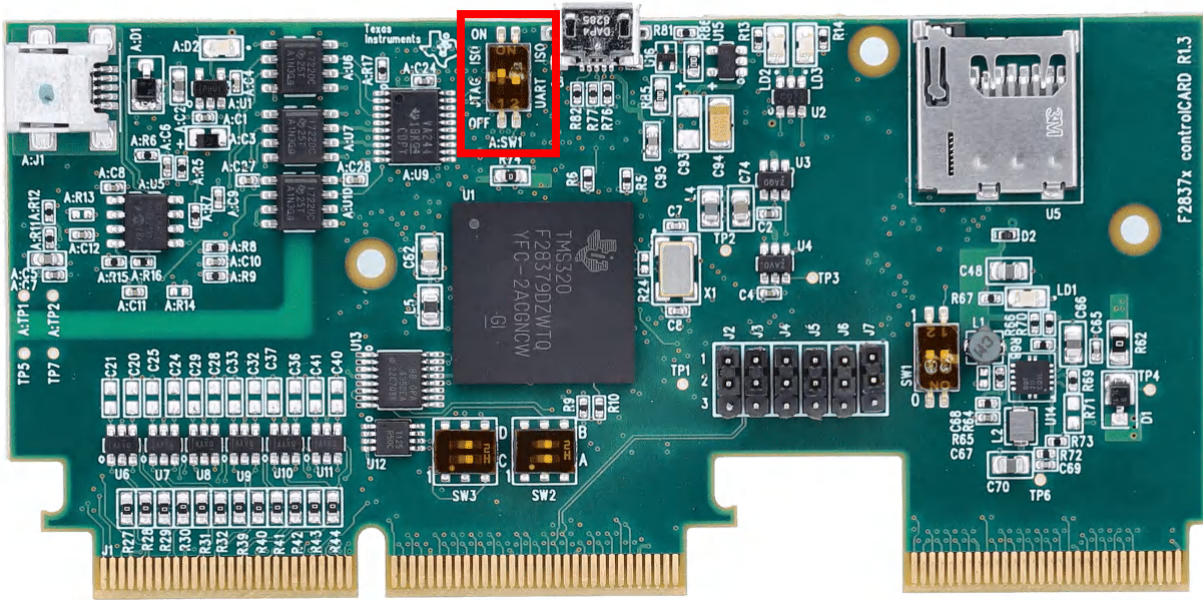


图 4-44. 配置 F2837xD controlCard 以使用 CCS 和板载 XDS 仿真器进行调试

3. 打开 CCS 的工作区。
4. 选择 View > Target Configurations。

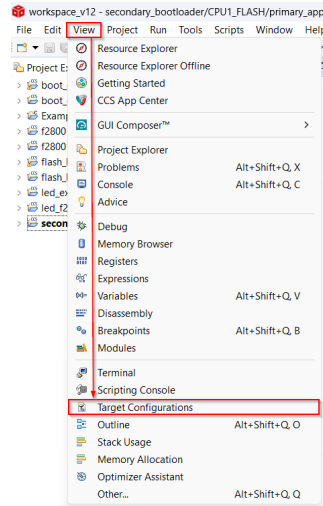


图 4-45. 在 CCS 中打开“Target Configuration”菜单

5. 用户可以将该器件的工程导入 CCS 并使用该工程连接器件，或将目标配置文件 (.ccxml) 从 C2000Ware (C2000Ware_x_xx_xx_xx > device_support > DEVICE_FAMILY > common > targetConfigs) 复制到用户定义的目标配置。
 - a. 找到器件目标配置，然后右键单击手动启动。

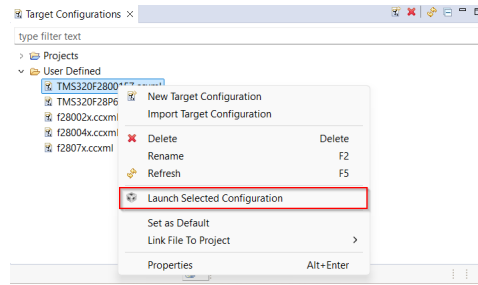


图 4-46. 启动 CCS 中的目标配置

6. CCS 启动调试窗口时，选择目标 CPU 并连接到目标。

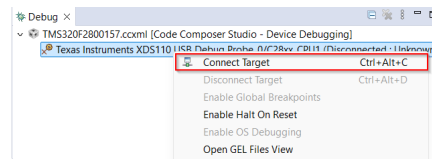


图 4-47. 连接到 CCS 中的目标核心

7. 如果弹出一个窗口，表明引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照节 5.3 中的步骤调试引导 ROM。
8. 加载符号后，转到 View > Memory Browser 以打开 Memory Browser。

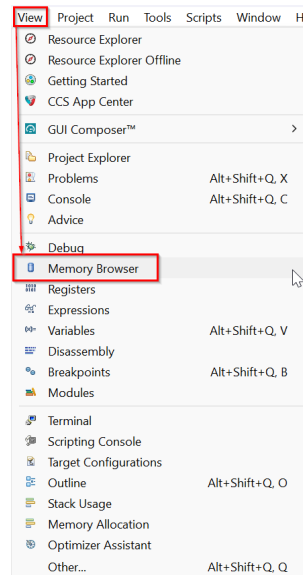


图 4-48. 导航到 CCS 中的 Memory Browser

9. 在“Memory Browser”选项卡中，导航到地址 0xD00。
10. 设置器件以进行仿真 USB 引导。在 [TMS320F28003xD 实时微控制器技术参考手册 \[23\]](#) 的配置 Get 引导选项一节中，BMODE 值定义为 0x0C。将 0xD00 编程为 0x0C5A。

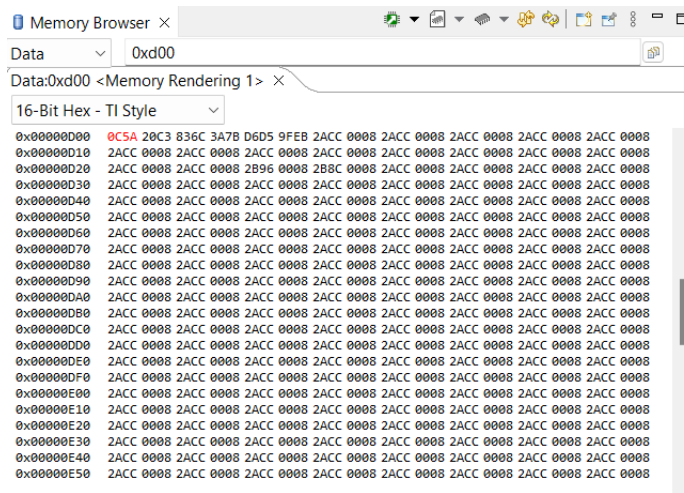


图 4-49. 配置 F2837xD controlCard 以模拟 USB 引导

11. 复位 CPU，然后单击 *Resume* 以启用 USB 引导序列。
12. 如果引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外，则按照节 5.3 中的步骤加载引导 ROM 符号。然后，确认器件在引导 ROM 中处于 USB 引导模式。

命令行选项和文件 IO 可以通过 C 标准库完成，但 USB 操作只能通过操作系统的器件驱动程序框架完成。有两个广泛使用的库提供此功能。

第一个是 libusb，它是一个开源 (LGPL) 库，采用 Unix 样式的 API。第二个是 WinUSB，它是 Windows 驱动程序开发套件的一部分。这两个库都在用户模式下运行并提供对 USB 器件的通用访问，无需使用客户驱动程序。Libusb 非常简便易用，也可以在 Linux 上使用，但它有点慢并且，任何分发都因许可证而变得复杂。WinUSB 使用起来要难一些，但速度更快并且生成的软件更易于分发。

C2000Ware 随附的 usb_flash_programmer.exe 预编译版本使用 WinUSB，但为这两个库提供了源代码。需要在器件上安装 WinUSB 驱动程序才能运行 USB 引导加载程序。

1. 转至“设备管理器”，右键单击通用串行总线控制器下无法识别的设备。F28379D 显示为 *Stellaris Device Firmware Upgrade*。
2. 选择更新驱动程序 > 浏览我的电脑以查找驱动程序 > 让我从计算机上的可用驱动程序列表中选择 > 从磁盘安装。
 - a. 将其输入到从以下位置的制造商文件中复制：
C2000Ware_x_xx_xx_xx\utilities\flash_programmers\usb_flash_programmer\windows_driver\x86

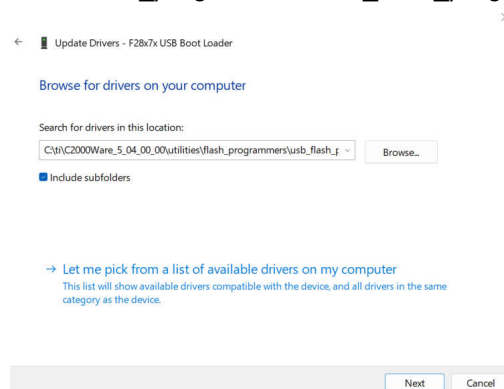


图 4-50. 浏览 C2000Ware 以查找 USB Windows 驱动程序

3. 现在，如果安装成功，WinUSB 驱动程序安装到器件上，并在 *Texas Instruments Microcontrollers* 类别下显示为 *F28x7x USB Boot Loader*。

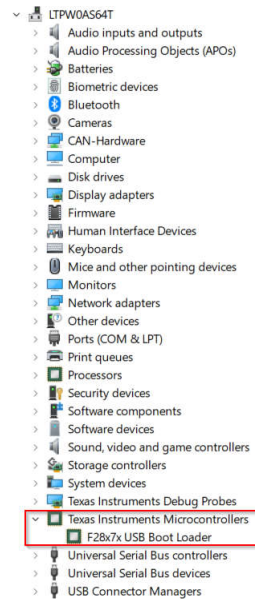


图 4-51. 设备管理器中显示的 F28x7x USB 引导加载程序器件

4. 复位 CPU 并执行外部复位 (XRSn)。然后，单击 *Resume* 以开始引导序列。

命令行 PC 实用程序是一种轻量级 (可执行文件大小约为 64KB) 编程解决方案，可以轻松集成到脚本环境中，用于生产线编程等应用程序。它是使用 Microsoft Visual Studio 用 C++ 编写的。工程及其源代码可在 C2000Ware (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > usb_flash_programmer > src > VS2010_USBLoader2000) 中找到。

输入文件路径在命令行上按升序加载，因此必须首先提供并列列出闪存内核文件路径才能加载到 RAM 中。闪存内核加载完毕后，ROM 转移控制权并且内核开始执行以将应用程序编程到闪存。闪存内核必须准备好器件以便进行闪存编程，然后器件才可以开始通信，因此需要一个较短的延迟。在此期间，闪存内核配置 PLL 和闪存等待状态。

下载过程开始时，在实际闪存应用程序代码传输之前先传输一些初始数据，包括密钥、一些保留字段和应用程序入口点。接收到入口点后，内核开始擦除闪存。擦除闪存可能需要几秒钟时间。请注意，当应用程序加载看似失败时，说明闪存正在被擦除。闪存被擦除后，应用程序加载将通过传输每个应用程序代码块并将其编程到闪存而继续。

1. 导航至包含已编译可执行文件 `usb_flash_programmer` 的文件夹 (C2000Ware_x_xx_xx_xx > utilities > flash_programmers > usb_flash_programmer)。
2. 打开终端并使用以下命令运行可执行文件 `usb_flash_programmer.exe` :

```
usb_flash_programmer.exe <path_to_kernel_dat> <path_to_application_dat>
```

备注

闪存内核和闪存应用程序都必须采用二进制引导格式，如上面在 节 4.1 中所述。

应用程序编程到闪存中后，闪存内核会转到其在应用程序加载过程开始时的入口点来尝试运行应用程序。

5 常见问题解答

本节详细介绍了用户尝试在各种外设引导模式下引导加载时遇到的常见障碍的相关建议。

5.1 通过基于软件的实现选择 BMSPI GPIO

问题: 是否有无需手动干预即可实现基于软件的引导加载过程的方法？

回答: **C2000™ 软件控制的固件更新过程应用报告 [7]** 介绍了一种在 C2000 器件上进行的软件控制的固件更新过程，该过程使用现有引导模式且无需手动选择引导模式。本文档中描述的方法直接适用于 F28004x 器件，并且经必要修改后可应用于传统器件。

5.2 从闪存而非 RAM 运行闪存内核

问题: 如何修改闪存内核以便从闪存（而非 RAM）运行？

回答: 在多组器件上，用户可以在闪存中实现自定义引导加载程序，以便直接将应用程序代码加载到其他闪存组中。在单组器件上，闪存中的自定义引导加载程序可以在同一闪存组内的单个扇区上执行固件升级，但闪存 API 必须从 RAM 执行。

备注

闪存 API 函数和调用闪存 API 的应用程序函数不得从同一组中执行。在进行数据编程和擦除时，在同一组执行闪存 API 会导致竞态条件。闪存 API 需要从 RAM 或另一个闪存组执行（如果同一核心存在另一个组）。[6]

使用自定义引导加载程序可在引导流中为用户提供更大的灵活性，允许使用不同的外设 GPIO 而不是仅可以访问引导 ROM 中有限的预定义 GPIO 分配集。使用自定义引导加载程序还可以有效减少引导加载过程中的步骤数，因为无需将中间闪存内核加载到 RAM 中以对闪存编程。

此外，如果固件升级过程失败，具有基于闪存的引导加载程序也可提供失效防护。引导至未升级的代码（即基于闪存的引导加载程序）可以最大限度地减少器件尝试引导至已损坏代码的几率，因为引导加载程序可以使用多种方法验证现有固件（键值、校验和等）。基于 RAM 的闪存内核也可以实现相同的功能，但在每次器件启动时加载它并不实用。

实现自定义引导加载程序的一种方法是修改 C2000Ware 中的现有闪存内核工程，以便从闪存而非 RAM 执行。这需要修改内核的连接器命令文件，后者指定在存储器中加载应用程序的位置并添加编译器要使用的预定义符号。

备注

以下步骤用于修改 F28P55x SCI 闪存内核以便从闪存执行，但相同流程可以应用于在 C2000Ware 中有现有闪存内核工程的任何 C2000 器件。

1. 右键点击工程，然后选择 **Add Files/Folders**。

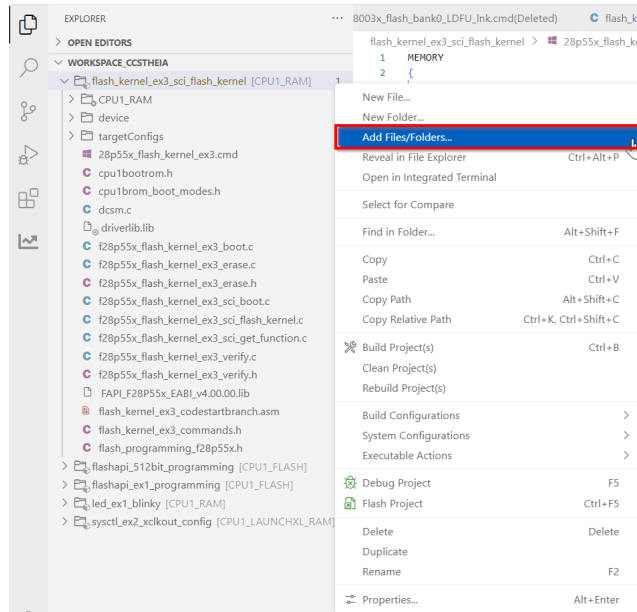


图 5-1. 将工程文件添加到 CCS 中

2. 在 C2000Ware 中导航到以下路径，并为目标器件选择通用闪存连接器命令文件。
 - a. `C2000Ware_X_XX_XX_XX\device_support\DEVICE_NAME\common\cmd`
3. 右键单击 RAM 连接器命令文件并选择 **Exclude from Build**，这样该闪存连接器命令文件将用于编译器构建。

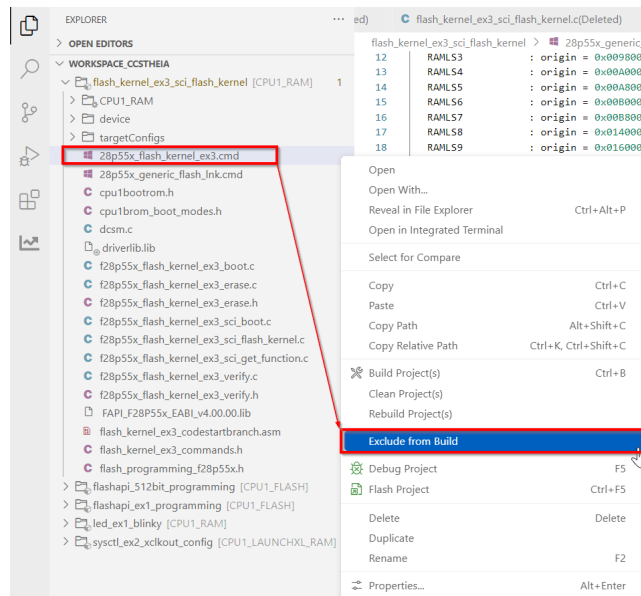


图 5-2. 从编译器构建中排除 RAM 连接器命令文件

备注

通过通用闪存连接器命令文件，内核加载到闪存组 0（进入点在 0x80000）。如果用户希望将内核放置到不同的闪存组，请将连接器命令文件中分配给闪存组 0 的 **SECTIONS** 修改到目标闪存组。

此外，在连接器命令文件的 **MEMORY** 部分，**BEGIN** 也必须根据需要更新为目标闪存入口点位置（**codestart**），包括闪存组原点和长度规范，以考虑入口点的 **codestart** 分配。

有关连接器命令文件指令的更多详细信息，请参阅 [33]。

4. 通过右键点击工程并导航至 **Build Configurations > Manage...** 创建新的构建配置

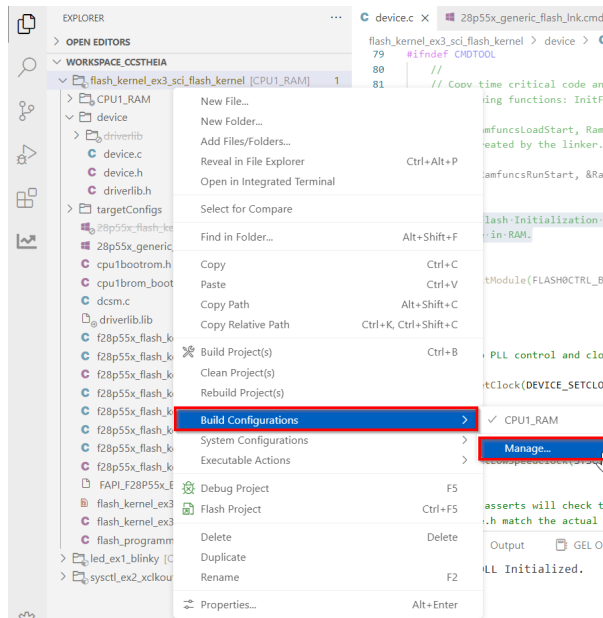


图 5-3. 为工程创建新构建配置

- a. 将构建配置命名为 `CPU1_FLASH` 并选择从 `CPU1_RAM` 配置复制设置。

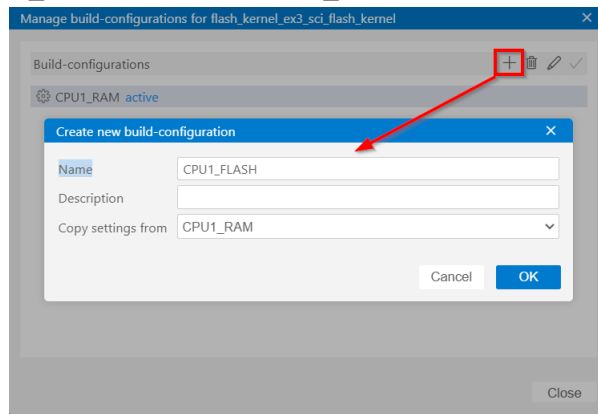


图 5-4. 创建 `CPU1_FLASH` 构建配置

5. 将有效构建配置设置为 `CPU1_FLASH`。
6. 在 `device.c` 中，时间关键型代码和闪存设置代码（称为 `Ramfuncs`）被复制到 RAM。此外，用于设置闪存等待状态的闪存初始化函数必须驻留在 RAM 中。由于内核正在闪存流中执行，必须定义符号 `_FLASH` 才能出现这些条件。
 - a. 右键单击并导航至 `Properties > Tools > C2000 Compiler > Predefined Symbols`。定义符号 `_FLASH`。

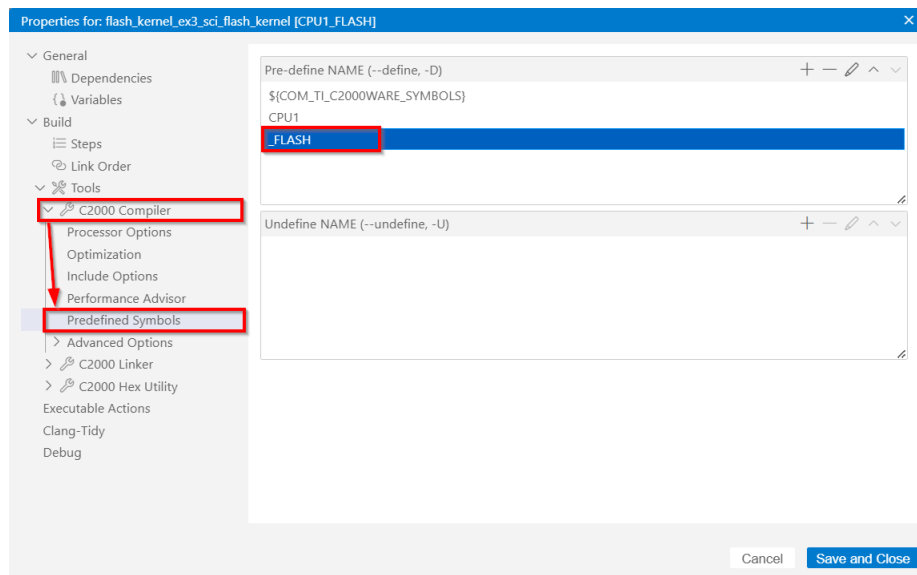


图 5-5. 添加预定义符号 `_FLASH` 以来初始化闪存函数

- 使用 `CPU1_FLASH` 构建配置重新构建闪存内核工程。现在，闪存内核已构建为从闪存执行并可以编程到器件闪存中。

备注

由于内核已加载到器件中，因此必须使用 C2000Ware 中的 `serial_flash_programmer_appln.exe`。可使用与常规串行闪存编程器相同的命令行参数调用此可执行文件，但不需要使用内核参数 (`-k`)。

5.3 在调试引导 ROM 时没有定义符号

问题: 当我调试引导 ROM 时，ROM 声明引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外。缺少什么？

回答: 用户可以通过将引导 ROM 符号 (`.out` 文件) 加载到器件中，逐步调试器件引导 ROM。加载符号是一种重要的调试方法。此选项添加生成的工程 `.out` 文件中提供的符号以用于调试，而不是通过 CCS 将实际的 `.out` 程序加载到内核中。这也是用户可以使用用于引导 ROM 的此方法或使用内置引导加载程序进行调试并获得增强的可见性的原因。

- 打开 CCS 的工作区。

2. 选择 View > Target Configurations。

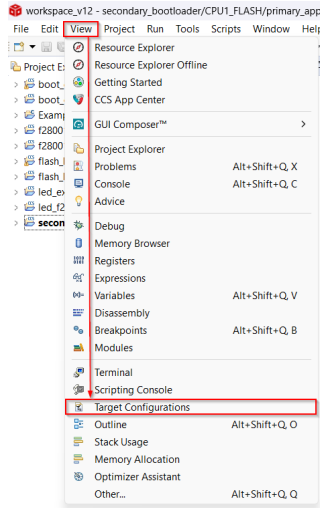


图 5-6. 在 CCS 中打开“Target Configuration”菜单

3. 用户可以将该器件的工程导入 CCS 并使用该工程连接器件，或将目标配置文件 (.ccxml) 从 C2000Ware (C2000Ware_x_xx_xx_xx > device_support > DEVICE_FAMILY > common > targetConfigs) 复制到此窗口中的用户定义的目标配置 (View > Target Configurations)。无论使用哪种方法，找到器件目标配置，然后右键单击手动启动：

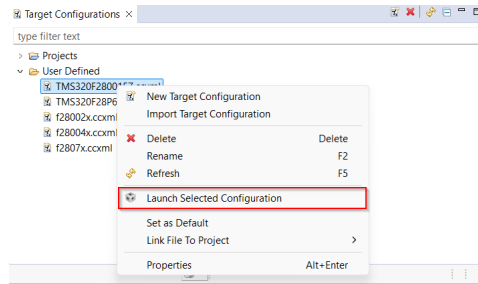


图 5-7. 启动 CCS 中的目标配置

4. CCS 启动调试窗口时，选择目标 CPU 并连接到目标。

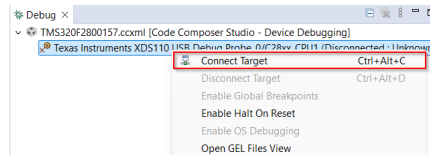


图 5-8. 连接到 CCS 中的目标核心

5. 此时会弹出一个窗口，声明引导 ROM 中存在中断且没有提供调试信息，或者中断处于程序代码之外。

Break at address "0x3fd2a" with no debug information available, or outside of program code.

Configure when this editor is shown

图 5-9. 没有加载引导 ROM 符号时的 CCS 视图

6. 导航至工具栏，然后单击按钮以显示 **Load Symbols**。

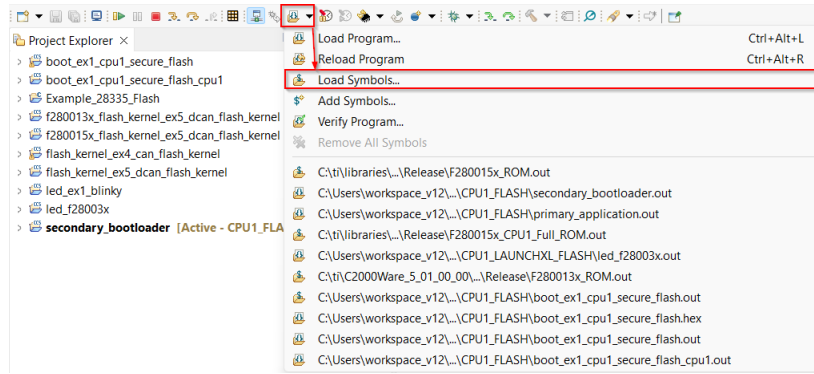


图 5-10. 在 CCS 中导航至 Load Symbols

7. 加载引导 ROM .out 文件。该文件可以在 C200Ware 中找到，位置如下：

a. `C2000Ware_x_xx_xx_xx > libraries > libraries > boot_rom > DEVICE_FAMILY > REV# > rom_sources > CPU# > ccs_files > Release`

8. 如果弹出一个窗口，声明找不到源文件，则用户可以选择 **Locate File**，并在 C2000Ware 中的以下位置查找：

a. `C2000Ware_x_xx_xx_xx > libraries > boot_rom > DEVICE_FAMILY > REV# > rom_sources > CPU# > DEVICE_FAMILY_ROM > bootROM > source`

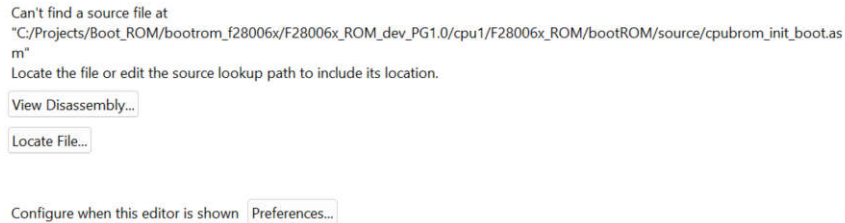


图 5-11. 定位引导 ROM 源文件

该文件打开，以显示引导 ROM 中的当前指令，而且用户可以浏览它了解如何调试、复位和重新启动器件。

5.4 使用片上闪存工具向 OTP 中写入值

本节通过两个示例用例演示了如何使用片上闪存工具对 OTP 进行编程。

备注

尽管本节基于 F280015x 器件，但相同流程可以应用于支持自定义 BMSP 和引导模式表的任何 C2000 器件。

器件特定的信息可以在器件技术参考手册 (TRM) 的 **引导 ROM** 一章中找到。

1. 使用目标 CPU 内核启动调试会话后，打开片上闪存工具（请参阅 [节 3](#) 了解如何在 CCS 中查找）。
2. 找到区域 1 或 2 下的 GPREG (BOOTCTRL)（记住区域 2 优先于区域 1）用户可以写入 OTP-BOOTPIN-CONFIG 和 OTP-BOOTDEF-LOW/OTP-BOOTDEF-HIGH 寄存器。
 - a. [表 2-7](#) 显示了 F280015x 器件上引导配置寄存器的位置，可以在器件特定 TRM 的 **引导 ROM 配置寄存器** 一节中找到。寄存器名称表示如何是在片上闪存工具中引用引导 ROM 寄存器的方式。

示例 1：零引导模式选择引脚

本用例展示了一个不希望使用任何 BMSP 并始终将器件引导至闪存入口点 0x88000 的应用场景。

有关表中要设置的值，请参阅器件 TRM 中的 *GPIO 分配* 一节。对于闪存入口点，请参阅 TRM 中的 *入口点* 一节，了解有关各种引导模式的入口点地址的详细信息。这些入口点指示引导 ROM 在引导结束时根据所选引导模式转移到什么地址。

1. 按如下方式对 OTP 中的 BOOTPIN_CONFIG 位置进行编程：
 - a. 将 BOOTPIN_CONFIG.BMSP0 设置为 0xFF (禁用)
 - b. 将 BOOTPIN_CONFIG.BMSP1 设置为 0xFF (禁用)
 - c. 将 BOOTPIN_CONFIG.BMSP2 设置为 0xFF (禁用)
 - d. 将 BOOTPIN_CONFIG.KEY 设置为 0x5A，以便引导 ROM 将这些寄存器位视为有效并使用自定义引导表
2. 对器件的 BOOTDEF 位置选项进行编程。这本质上是设置一个器件特定的引导模式表。
 - a. 将 BOOTDEF.BOOTDEF0 设置为 0x23 以引导至闪存 (入口地址选项 1)。这将闪存引导设置为引导表索引 0。

图 5-12 显示了对此示例进行编程的片上闪存工具中已完成的输入字段。



图 5-12. 示例 1：已编程的闪存插件引导配置

表 5-1. 生成的零引脚引导配置

BMSP 索引	BOOTDEF
0	0x23 (闪存引导至地址 0x88000)

示例 2：双引导模式选择引脚

本用例演示了一种更常见的应用场景，即使用两种引导模式选择引脚在自定义引导表中的 CAN、安全闪存和 SCI 引导之间进行选择。

有关表中要设置的值，请参阅器件 TRM 中的 *GPIO 分配* 一节。对于闪存入口点，请参阅 TRM 中的 *入口点* 一节，了解有关各种引导模式的入口点地址的详细信息。这些入口点指示引导 ROM 在引导结束时根据所选引导模式转移到什么地址。

1. 按如下方式对 OTP 中的 BOOTPIN_CONFIG 位置进行编程：
 - a. 将 BOOTPIN_CONFIG.BMSP0 设置为用户指定的 GPIO，例如适用于 GPIO42 的 0x2A
 - b. 将 BOOTPIN_CONFIG.BMSP1 设置为用户指定的 GPIO，例如适用于 GPIO88 的 0x58
 - c. 将 BOOTPIN_CONFIG.BMSP2 设置为 0xFF (禁用)
 - d. 将 BOOTPIN_CONFIG.KEY 设置为 0x5A，以便引导 ROM 将这些寄存器位视为有效并使用自定义引导表。

2. 对器件的 **BOOTDEF** 位置选项进行编程。这本质上是设置一个器件特定的引导模式表。
 - a. 将 **BOOTDEF.BOOTDEF0** 设置为 **0x6A** 以便进行安全闪存引导 (入口地址选项 3)。这将安全闪存引导设置为引导表索引 0。
 - b. 将 **BOOTDEF.BOOTDEF1** 设置为 **0x22** 以便进行 CAN 引导选项 1。这将 CAN 引导设置为引导表索引 1。
 - c. 将 **BOOTDEF.BOOTDEF2** 设置为 **0x41** 以便进行 SCI 引导选项 2。这将 CAN 引导设置为引导表索引 2。

图 5-13 显示了对此示例进行编程的片上闪存工具中已完成的输入字段。



图 5-13. 示例 2 : 已编程的闪存插件引导配置

表 5-2. 双引导模式选择引脚配置

BMSP 索引	BOOTDEF
0	0x6A (安全闪存引导至地址 0x90000)
1	0x22 (使用其他 GPIO 的 CAN 引导 1)
2	0x41 (使用其他 GPIO 的 SCI 引导 2)
3	无关紧要 (未使用)

5.5 使用闪存 API 插件向 OTP 中写入值

备注

尽管本节基于 F280015x 器件，但相同流程可以应用于支持自定义 BMSP 和引导模式表的任何 C2000 器件。

器件特定的信息可以在器件技术参考手册 (TRM) 的 *引导 ROM* 一章中找到。

也可以使用 [31] 中概述的编译器的 **RETAIN** 和 **DATA_SECTION** pragma 将 节 5.5 中的自定义引导配置写入 DCSM OTP。以下步骤遵循中 示例 2 : 双引导模式选择引脚 的示例。

1. 在 CCS 中打开任何 C2000Ware 示例，并在主函数上方添加以下代码片段 (可以是文件中函数以外的任何位置)。这些行分别对器件的 **BOOTPIN-CONFIG** 和 **BOOTDEF-LOW** 寄存器进行编程。

```
#pragma RETAIN(otp_z1_data_1)
#pragma DATA_SECTION(otp_z1_data_1,"dcsm_zsel_z1");
const long otp_z1_data_1 = 0x5AFF582A;

#pragma RETAIN(otp_z1_data_2)
#pragma DATA_SECTION(otp_z1_data_2,"dcsm_zsel_z1_2");
const long otp_z1_data_2 = 0xFF41226A;
```

- 在工程的连接器命令文件中，添加以下行来定义器件特定 TRM 的 **引导 ROM 配置寄存器**一节中所定义的 DCSM OTP BOOTPIN-CONFIG 和 BOOTDEF-LOW 存储器映射位置。以下地址适用于 F280015x 器件上的 DCSM 用户 OTP 引导配置寄存器。

```
MEMORY {
  PAGE 0:
    DCSM_ZSEL_Z1_P0: origin = 0x078008, length = 0x000002 // Z1-OTP-BOOTPIN-CONFIG
    DCSM_ZSEL_Z1_P1: origin = 0x07800C, length = 0x000002 // Z1-OTP-BOOTDEF-LOW
}
SECTIONS {
  dcsm_zsel_z1_1 : > DCSM_ZSEL_Z1_P0, PAGE = 0
  dcsm_zsel_z1_2 : > DCSM_ZSEL_Z1_P1, PAGE = 0
}
```

- 重新编译示例并在 CCS 中通过 JTAG 加载到目标。CCS 中的程序加载程序和闪存 API 插件负责将这些值写入 OTP 位置。由于无法对 OTP 位置重新编程，因此必须仔细选择和写入这些值。

6 总结

当无法可靠地使用 JTAG 调试探针时，嵌入式处理器通常需要灵活的编程方法，尤其是在可现场部署的系统中。C2000 微控制器通过在引导 ROM 中提供引导加载实用程序来支持器件固件升级，从而满足了这一需求。这些引导加载程序使用户可以使用各种通信接口将应用程序代码从外部主机加载到 RAM 中，并且能够使用闪存内核随后对片上闪存存储器进行编程。本文档介绍了基本引导加载配置，并详细介绍了如何利用常见引导模式将应用程序代码加载到 C2000 器件中。

7 参考资料

1. 德州仪器 (TI), [Code Composer Studio™ 集成开发环境 \(IDE\)](#), 网页
2. 德州仪器 (TI), [UniFlash 闪存编程工具](#), 网页
3. 德州仪器 (TI), [面向 C2000 MCU 的 C2000Ware](#), 网页
4. 德州仪器 (TI), [\[E2E\] 有关 C2000 MCU 闪存软件的常见问题解答](#), 网页
5. 德州仪器 (TI), [\[E2E\] 有关 C2000 CCS 闪存插件和 UniFlash 的常见问题解答](#), 网页
6. 德州仪器 (TI), [\[E2E\] 有关 C2000 器件闪存 API 使用的常见问题解答](#), 网页
7. 德州仪器 (TI), [C2000™ 软件控制的固件更新过程](#), 应用报告
8. 德州仪器 (TI), [C28x Academy, 双代码安全模块 \(DCSM\)](#), 网页
9. 德州仪器 (TI), [C2000™ DCSM 安全工具](#), 应用报告
10. 德州仪器 (TI), [C2000 多核开发用户指南](#)
11. 德州仪器 (TI), [TMS320C28x 汇编语言工具](#), 用户指南
12. 德州仪器 (TI), [TI Arm Clang 编译器工具用户指南 - 连接器说明](#)
13. 德州仪器 (TI), [C2000 器件上的安全引导](#), 应用报告
14. 德州仪器 (TI), [C2000™ 微控制器串行闪存编程](#), 应用手册
15. 德州仪器 (TI), [控制器局域网 \(CAN\) 简介](#), 应用报告
16. 德州仪器 (TI), [C2000 微控制器 CAN 闪存编程](#), 应用手册
17. PEAK 系统, [PCAN-Basic API](#)
18. 德州仪器 (TI), [MCAN \(CAN FD\) 模块入门](#), 应用手册
19. 德州仪器 (TI), [\[E2E\] TMS320F28377D : 28377D USB 引导加载程序](#), 网页
20. 德州仪器 (TI), [C2000™ 微控制器 USB 闪存编程](#), 应用报告册
21. 德州仪器 (TI), [XDS100 调试探针](#), 产品页面
22. 德州仪器 (TI), [Delfino™ TMS320F28379D controlCARD R1.3](#), 用户指南
23. 德州仪器 (TI), [TMS320F28003xD 实时微控制器](#), 技术参考手册
24. 德州仪器 (TI), [TMS320F280015x 实时微控制器](#), 技术参考手册
25. 德州仪器 (TI), [TMS320F28P65x 实时微控制器](#), 技术参考手册
26. 德州仪器 (TI), [TMS320F2837xD 实时微控制器](#), 技术参考手册
27. 德州仪器 (TI), [TMS320F28P55x 实时微控制器](#), 技术参考手册
28. 德州仪器 (TI), [具有连接管理器的 TMS320F2838xD 实时微控制器技术参考手册 \(SPRU110\)](#)
29. 德州仪器 (TI), [TMS320F280015x 实时微控制器](#), 数据表
30. 德州仪器 (TI), [TMS320F280039x 实时微控制器](#), 数据表
31. 德州仪器 (TI), [F021 闪存 API 版本 2.01.01](#), 参考指南
32. 德州仪器 (TI), [TMS320F280013x/15x 闪存 API 版本 2.00.10.00](#), 参考指南
33. 德州仪器 (TI), [TI 连接器命令文件入门](#), 网页
34. 德州仪器 (TI), [TMS320C28x 优化 C/C++ 编译器](#), 用户指南
35. 德州仪器 (TI), [\[E2E\] LAUNCHXL-F2800157 CAN 闪存编程](#), 网页

重要通知和免责声明

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 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。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
版权所有 © 2025，德州仪器 (TI) 公司