Application Note

F28P65x 器件上的双核调试

Ira Thete, Mithra Kandasamy

摘要

C28x 系列微控制器包含许多的双核器件。这些内核可以是两个相同或不同内核。此类器件的主要应用领域包括工业驱动器、光伏逆变器、储能等。每个内核处理应用中的不同功能。例如，在工业驱动器中，第一个内核可用于处理内部电流/速度控制环路，第二个内核可用于处理 I/O 通信和故障协调。本应用手册探讨了如何对双核 C28x 器件进行编程和调试，以及需要避免的常见错误。

内容

商标

所有商标均为其各自所有者的财产。

1 简介

在实时控制应用方面，始终有一个根本的问题。确定性硬实时控制回路（通常在 100kHz 左右的频率下运行）不能与通信、保护逻辑、遥测、故障管理等较慢的监控任务共享计算时间。第二个 CPU 通过划分整体工作负载并允许单个 CPU 的专用带宽用于实时控制相关任务来缓解这种担忧。本应用手册探讨了如何有效调试可能在双核环境中出现意外行为的应用。其中讨论了启动序列及存储器或外设所有权冲突如何导致双核应用的性能不稳定。

2 C28x 系列中的双核架构

C2000 双核系列涵盖两种架构类型：

2.1 同构架构 (C28x + C28x)

两个 CPU 都是相同的 C28x DSP 内核，运行相同的 ISA，具有相同的编译器工具链以及相同的指令集扩展（例如 FPU 和 TMU）。这两个内核的能力是对称的，任一内核都可以运行任何任务。此类器件的一些示例包括

- TMS320F2837x
- TMS320F28P65xx

即使硬件是对称的，内核在启动期间的运行方式也不同。CPU1 硬接线为系统主 CPU。

2.2 异构架构 (C28x + C28X + ARM Cortex M4)

这样，双核 C28x 子系统就像以前一样处理实时控制。ARM Cortex-M4“连接管理器”是在 125MHz 上运行的通用 32 位 RISC 内核，专用于运行通信中间件和基于 RTOS 的任务。

2.2.1 TMS320F2838x（F28388D、F28386D 等）

F2838xD 系列器件包含 2 个 C28x CPU 子系统及一个 CM（连接管理器）子系统。每个 C28x CPU 子系统包含的内容

1. 一个 C28x CPU
2. 一个 CLA（控制律加速器）协处理器
3. 一个 DMA（直接存储器存取单元）

请注意，C28x1 和 C28x2 内核也分别称为 CPU1 和 CPU2。多核调试的一个重要组成部分是确定哪个 CPU 具有哪些硬件外设。F2838xD CPU 包含多个存储器块、外设及 GPIO。其中一些只能由某个内核拥有，而其中一些允许在多个内核之间共享。在执行任何功能之前，需要为“可由多个内核共享”类别下的所有资源分配一个显式所有者。

3 双核应用的一般流程

- 多核调试的一条重要规则是，此所有权始终由 CPU1 决定。
- 另一个重要方面是实现 2 个 CPU 之间的有效通信及数据共享。这由处理器间通信模块处理。除此之外，此器件还具有额外的专用消息 RAM，可用于共享数据。
- 尽管两个内核具有相同的硬件 — 但对于多核系统，它们具有不同级别的自主能力。CPU1 用作主内核。器件的基本初始化（包括时钟和所有 GPIO 的初始化）由 CPU1 的应用程序完成。根据引导引脚配置，可以在不同模式下引导 CPU1。CPU2 及 CM（对于 F2838Xd）内核必须由使用 IPC 模块的 CPU1 应用程序引导。

4 引导序列

当器件退出复位状态时，仅执行 CPU1 的引导 ROM。CPU2 由硬件保持在复位状态，并且在 CPU1 显式释放之前无法运行任何代码。这意味着：

- **系统时钟及 PLL** 专门由 CPU1 初始化。在 CPU1 配置 PLL 并等待其锁定之前，器件会以较低的速度在内部振荡器上运行。CPU2 继承 CPU1 已建立的任何时钟状态。
- **外设所有权**由 CPU1 分配。该器件具有一组可分配给任一内核的外设，但分配寄存器受到 CPU2 的写保护。只有 CPU1 可以决定哪个内核拥有哪个外设。
- **GPIO 初始化**由 CPU1 完成。所有 GPIO 多路复用器、方向和上拉设置都必须通过 CPU1 进行配置，然后 CPU2 才能安全使用任何 GPIO。
- 整个系统的**看门狗**由 CPU1 管理。CPU2 有自己的看门狗，但系统级保护从 CPU1 开始。

每次复位时，都会发生以下序列：

1. **CPU1 引导 ROM 执行。**引导 ROM 读取引导模式引脚（GPIO84、GPIO72 等，具体取决于器件）并确定如何引导 — JTAG、SCI、SPI、CAN、I2C、并行，或者直接从闪存运行。这是引导加载程序阶段，完全在 CPU1 上运行。
2. **CPU1 应用程序启动。**引导加载程序结束后，CPU1 开始执行应用。编写完善的 CPU1 应用程序的第一件事是：
   - 初始化器件（PLL，时钟，闪存等待状态）
   - 分配外设和内存所有权
   - 初始化 CPU2 将需要的任何共享存储器区域
3. **CPU1 将 CPU2 从复位之中释放。**CPU2 无法自行启动。CPU1 必须调用 CPU2 的引导序列 — 在 driverlib 中，这是 Device_bootCPU2()。此函数写入 IPC 寄存器以向 CPU2 的引导 ROM 发出执行信号，并为 CPU2 指定引导模式（通常从闪存引导或由 CPU1 加载的特定 RAM 地址引导）。
4. **CPU2 引导 ROM 执行。**CPU2 的引导 ROM 使用 CPU1 经由 IPC 指定的引导模式运行自己的引导序列。如果 CPU1 将 CPU2 引导至闪存引导，则 CPU2 会跳转到其闪存入口点。如果 CPU1 指示 CPU2 从 SARAM 引导（在 CPU1 已复制 CPU2 的映像的 RAM 调试构建中很有用），CPU2 会跳转到加载的代码。
5. **两个内核均独立运行。**此后，两个 CPU 同时执行各自的应用程序。两者之间的协调完全通过 IPC 标志和共享/消息 RAM 来处理 — 在正常执行期间它们之间不存在进一步的硬件依赖关系。

5 运行一个简单的双核示例

要开始了解与双核环境相关的概念，一个好方法是在双核 c28x 器件上运行一个简单的 led 闪烁示例。对于这个例子，我们将使用 F28P65x launchpad。您将需要最新版本的 c2000ware 和 Code Composer Studio 才能运行此示例。

如何使用 CPU1 及 CPU2 使 LED 闪烁？

步骤 1 — 导入两个工程

导航至 C:/ti/C2000Ware_x_x_x_x/driverlib/f28p65x/examples/c28x_dual/led/CCS/，然后单击特定文件夹。应该会出现 2 个不同的工程。确保导入 **led_ex1_c28x_dual_blinky.projectspec**。此操作会将 cpu1 和 cpu2 工程导入您的 CCS 工作区。

第 2 步 — 选择您的构建配置

在 RAM 配置中调试

1. 将两个工程的构建配置更改为 CPUx_LAUNCHXL_RAM
2. 首先构建 CPU1 工程，然后构建 CPU2。两者都应该简洁构建，并分别生成 led_ex1_c28x_dual_blinky_cpu1\CPU1_LAUNCHXL_RAM\led_ex1_c28x_dual_blinky_cpu1.out and led_ex1_c28x_dual_blinky_cpu2\CPU2_LAUNCHXL_RAM\led_ex1_c28x_dual_blinky_cpu2.out。
3. 然后右键单击目标配置文件（.ccxml 文件），然后单击启动无项目调试。
4. 完成此操作后，将启动 CCS 调试会话，但任何内核上都不会加载程序。您应该会在调试视图的“线程”部分中看到所有可用内核。
5. 连接 C28xx_CPU1 和 C28xx_CPU2（按该顺序），方法是在“THREADS”部分右键单击它们并选择“Connect Target”选项。
6. 连接两个内核后，选择 C28xx_CPU1，然后转至 Run → Load → Load Program → Navigate to 并选择 led_ex1_c28x_dual_blinky_cpu1.out
7. 执行此操作后，您将看到 CPU1 程序执行在 main() 处停止。让程序运行，您应该会在 launchpad 上看到 LED4 闪烁。
8. 在运行 CPU2 代码之前，CPU1 代码必须通过 Device_bootCPU2() 函数运行，这一点非常重要。
9. 您可以使 CPU1 代码保持运行（或在 Device_bootCPU2() 函数之后的某个点停止它），然后将 led_ex1_c28x_dual_blinky_cpu2.out 映像加载到 C28xx_CPU2。
10. 如果正确执行了所有步骤，它也应该在 main() 停止，然后您应该能够正常运行它。launchpad 上的 LED5 将闪烁。

闪存配置中的调试

1. 将两个工程的构建配置更改为 CPUx_LAUNCHXL_FLASH
2. 在 CPU1 目标配置 .ccxml 文件中开始无工程调试。
3. 将目标连接到 CPU1 及 CPU2 线程上。
4. 转到 CPU1 的属性，然后在下拉列表中打开闪存组闪存设置。向下滚动到 Flash Bank Map Settings，并为组 3 和 4 设置 1（0 表示 CPU1，1 表示 CPU2）。
5. 单击配置时钟。
6. 向下滚动至擦除设置并选中 Selected banks only -> 然后选择组 0,1,2（它们适用于 CPU1）。
7. 现在单击 CPU2 属性。如果启用了“闪存组映射设置”，请再次为组 3 和 4 设置 1。否则，如果该引脚被禁用，则保持原样。
8. 再次向下滚动至擦除设置并选择组 3 和 4。如果调试控制台显示“Device is in reset”，请拔下并再次插入 USB，或打开新会话。如果错误仍然弹出，程序无论如何也将运行。
9. 现在，单击 CPU1 线程并转到 Run -> Load -> Load program。
10. 现在浏览相应的 .out 文件（来自任何先前构建）并运行程序以执行该文件。（请注意，请确保 CPU1 程序至少运行到函数 Device_bootCPU2 ()。CPU2 线程也执行相同操作。）

运行示例时遇到的常见问题

hex2000 实用程序无法识别？

有时必须使用其原始项目路径来引用它，特别是在使用 Windows powershell 时。不是仅提及 hex2000，而是提供 hex2000 的完整路径。

将 combined.hex 加载到 cpu1 后，存储器是否未更新到正确的地址？

引导 ROM 序列有时不同，CPU2 可能仍处于默认引导地址，检查“Disassembly”视图并手动更改指针位置，以检查存储器中填充的内容。如果更新了存储器，请断开 launchpad（触发复位）并再次连接以进行检查。如果未在存储器中更新该文件，请检查 combined.map 文件以查看组合输出文件是否构建正确。如果没有，请全新导入工程并重试。

两个 CPU 都运行，但指示灯不闪烁？

设置断点以找出程序正在停止的位置。如果它停止在 IPCsync 处，请重新编译工程并重新启动无工程调试，并在连接 CPU2 的目标后按正确的顺序加载程序，还要确保传递到两个 CPU IPCsync 函数的参数相同。否则，还要确保 CPU1 工程文件夹中的 sysconfig 文件将 CPU2 引导模式设置为从闪存组 3 引导，然后保存文件。

6 关键陷阱

1. **引导顺序所有权：**

   CPU1 始终是主核心，必须先启动。CPU1 控制 CPU2 的复位并负责所有系统初始化（PLL、时钟、看门狗）。不能直接将 CCS 连接到 CPU2 cold - CPU1 必须正在运行并已经将 CPU2 从复位状态释放。
2. **CCS 中的加载顺序很重要：**

   首先连接并加载 CPU1，运行足够远的位置以初始化系统和引导 CPU2（或至少将 CPU2 从复位中释放），然后连接并加载 CPU2。如果在 CPU1 初始化系统时钟之前加载 CPU2，则将获得不稳定的行为。
3. **单独的 CCS 项目：**

   双核应用程序是两个完全独立的映像（输出文件）。它们仅通过共享存储器和 IPC 进行链接 — 没有单个“双核工程”。（除非使用第 5 节中的编译后处理步骤）
4. **内存所有权/访问权限：**

   共享 RAM 块具有所有权分配 — 如果 CPU1 拥有 GS RAM 块，CPU2 将遇到尝试写入它的存储器访问违规。CPU1 和 CPU2 的链接器命令文件 (.cmd) 必须仔细协调，以免重叠，且共享部分必须使用专用的 IPC/MSG RAM 区域或正确拥有的 GS RAM。
5. **停止一个内核会影响另外一个内核：**

   当您将 CPU1 暂停在断点处时，CPU2 会继续运行，除非您也明确停止它（或在 CCS 中使用同步组）。如果暂停的内核是预期响应的内核，则正在运行的内核上的 IPC 标志轮询循环可能会无限期旋转。这是双核调试期间令人困惑的行为的最常见来源。
6. **全局断点/同步组：**

   CCS 支持对内核进行分组，因此一个断点同时停止两个内核。对于 IPC 重代码，这几乎总是您想要的 — 否则，正在运行的核心竞争将会消耗 IPC 消息，并且状态将在您查看时变得不一致。
7. **看门狗：**

   每个内核都有自己的看门狗。如果您暂停 CPU1 以进行调试，但 CPU2 仍在运行并为共享外设提供服务，或者相反，未监视内核的看门狗可能会触发并使器件复位。在调试构建的早期禁用看门狗。
8. **闪存与 RAM：**

   在闪存构建中，两个内核共享闪存控制器，并且在同一流水线阶段从两个内核进行并发闪存读取可能会导致停滞。在调试期间，大多数团队都从 RAM 中工作以避免这种情况，并能够在没有擦除周期的情况下快速重新加载。闪存编程需要协调两个内核（CPU1 通常会初始化所有闪存）。
9. **IPC 竞态条件：**

   基于 IPC 标志的协议本质上为异步。常见错误：CPU1 设置标志，然后立即将数据写入共享缓冲区，但 CPU2 在 CPU1 完成写入之前读取缓冲区。仅在数据写入完成后使用 IPC 标志来指示信号，并在需要时使用存储器屏障（C28x 上的 __asm(" RPT #7 || NOP")）。
10. **CLA 是第三个可调试内核：**

    在存在 CLA 的器件上，该 CLA 在 CCS 中显示为单独的调试目标。从 CPU2 .out 加载 CLA 符号（它们是一起编译的）。CLA 没有 RTOS，没有栈，并且调试可见性非常有限 — 您可分步查看寄存器，但没有调用栈。
11. **异构调试（F2838x CM 内核）：**

ARM CM 内核使用完全不同的工具链。在 CCS 中，它显示为单独的 ARM Cortex-M4 目标。您需要一个 ARM 编译器工程与 C28x 工程一起使用。CM 内核无法直接访问 C28x 本地 RAM — 所有数据交换都要通过 MSG RAM 进行。

7 适用于双核刷写的一键式解决方案

本节探讨如何将两个二进制文件（一个用于 CPU1、一个用于 CPU2）组合在一起，并仅需一步即可将它们直接加载到闪存中所需的区域。我们只需将组合的二进制文件加载到 CPU1，而不是手动将每个二进制文件加载到其各自的内核中。这是通过使用编译后处理步骤来实现的。

生成组合输出文件以加载到一个 CPU 上，并在 F28P65x 中直接控制另一个 CPU（适用于尚未下载最新 c2000ware 版本的用户）

- 从 C2000 f28p65x 中导入 led_ex2_blinky_sysconfig_multi，这会自动将 cpu1 和 cpu2 工程导入到您的工作区中。
- 右键单击多个文件夹，然后选择所需的构建配置。
- 打开 cpu1 工程文件夹中的 **.sysconfig** 文件，将 CPU2 引导模式设置为从闪存组 3 引导，然后保存文件。
- 右键单击 **sysconfig_multi** 工程，然后选择 Build Projects。
- 现在，在右键单击 CPU2 → Properties → Build → Steps → 编译后处理步骤下手动输入以下命令作为 postBuildStep 并保存该命令。命令为：

  ${CG_TOOL_ROOT}/bin/hex2000.exe --memwidth=16 --romwidth=16 -ii --map="..\..\combined.map" "..\..\led_ex2_blinky_sysconfig_cpu1\"FLASH_LAUNCHXL or FLASH"\led_ex2_blinky_sysconfig_cpu1.out" "..\..\led_ex2_blinky_sysconfig_cpu2\"FLASH_LAUNCHXL or FLASH"\led_ex2_blinky_sysconfig_cpu2.out" -o "..\..\combined.hex"
- 打开 cpu1 **targetConfigs** 文件夹中的目标配置文件，然后单击 Start Project-less Debug。
- 现在，右键单击 CPU1 和 CPU2 以连接到目标。
- 右键单击 CPU1 -> Flash settings -> 向下滚动到 Flash Bank Map Settings ->确保都设置为 0（我们正在从 CPU1 本身加载整个数据，因此需要访问所有闪存组）
- 进一步向下滚动以选中 Erase settings 下的 select Entire flash。
- 向上滚动以单击配置时钟，然后单击 Save and Close。
- 现在单击 Run -> Load program -> Browse -> combined.hex
- 打开 View → Memory Browser ，并验证数据是否已正确加载到 CPU1 的 0x8000 和 CPU2 的 0xE0000。
- 单击 CPU1 和 CPU2 上的 Run 以开始执行，然后使用 Launchpad LED 或 Output Console 来验证输出。

如果用户已下载最新的 c2000ware 版本（将在下次发布时提供）

为了运行示例，直接编译多工程将在多工程文件夹中创建 combined.hex 和 combined.map 文件，可以执行上述无工程调试。

在 projectspec 中合并 postBuildStep：

对于

“examples_driverlib\c28x_dual\led\sysconfig_led_ex_f28p65x\CCS\led_ex2_blinky_sysconfig.projectspec”进行了以下更改，

其位于 c2000-device-support 存储库中，用于合并到 SDK。

为了在 launchpad 上进行验证，必须在

*"C:\ti\c2000\C2000Ware_26_01_00_00\driverlib\f28p65x\examples\c28x_dual\led\CCS\led_ex2_blinky_sysconfig.projectspec" 文件*进行相同的更改，该文件位于 c2000ware 文件夹中。

- <configuration name="FLASH" compilerBuildOptions="--opt_level=off -I${C2000WARE_ROOT} -I${PROJECT_ROOT}/device -I${C2000WARE_DLIB_ROOT} --define=_FLASH -v28 -ml -mt --cla_support=cla2 --float_support=fpu64 --tmu_support=tmu1 --define=DEBUG --define=CPU2 --gen_func_subsections=on --diag_warning=225 --diag_suppress=10063" linkerBuildOptions="--entry_point code_start --stack_size=0x3F8 --heap_size=0x200 --define=_FLASH "

  postBuildStep="${CG_TOOL_ROOT}/bin/hex2000.exe --memwidth=16 --romwidth=16 -ii --map='..\..\led_ex2_blinky_sysconfig_multi\combined.map' '..\..\led_ex2_blinky_sysconfig_cpu1\FLASH\led_ex2_blinky_sysconfig_cpu1.out'

'..\..\led_ex2_blinky_sysconfig_cpu2\FLASH\led_ex2_blinky_sysconfig_cpu2.out' -o
'..\..\led_ex2_blinky_sysconfig_multi\combined.hex'"

/>

<configuration name="FLASH_LAUNCHXL" compilerBuildOptions="--opt_level=off -I${C2000WARE_ROOT} -I${PROJECT_ROOT}/device -I${C2000WARE_DLIB_ROOT} --define=_FLASH -v28 -ml -mt --cla_support=cla2 --float_support=fpu64 --tmu_support=tmu1 --define=DEBUG --define=CPU2 --gen_func_subsections=on --diag_warning=225 --diag_suppress=10063" linkerBuildOptions="--entry_point code_start --stack_size=0x3F8 --heap_size=0x200 --define=_FLASH "

postBuildStep="${CG_TOOL_ROOT}/bin/hex2000.exe --memwidth=16 --romwidth=16 -ii --map='..\..\led_ex2_blinky_sysconfig_multi\combined.map'
'..\..\led_ex2_blinky_sysconfig_cpu1\FLASH_LAUNCHXL\led_ex2_blinky_sysconfig_cpu1.out'
'..\..\led_ex2_blinky_sysconfig_cpu2\FLASH_LAUNCHXL\led_ex2_blinky_sysconfig_cpu2.out' -o
'..\..\led_ex2_blinky_sysconfig_multi\combined.hex'"

/>

上述更改已作为 cpu2 工程的构建后处理步骤进行。

如何在 f28p65 中加载单个输出文件？

- hex2000 合并两个 CPU 输出文件，并自动从第一个文件中去除 EOF 记录，因此第二个文件可以在一个 Intel-HEX 映像中无缝执行。
- 生成的 HEX 包含 0x08000 处的 CPU 1 代码（闪存组 0）和 0xE0000 处的 CPU 2 代码（闪存组 3）；一个闪存操作对两个组一起进行编程。
- 加载后，起始矢量被重新指向：CPU 1 从其闪存地址开始执行，然后释放从闪存组 3 引导的 CPU 2 的引导功能。
- CPU 1 被授予对所有五个闪存存储体的写访问权限，从而使其有完全控制权，可对自己的区域和 CPU 2 使用的区域进行编程。
- 加载通过单个 **Run→Load Program** 命令完成；加载后，您可以打开 Memory Browser，以在 0x08000 for CPU 1 和 0xE0000 for CPU 2 处确认数据。
- 执行开始时，CPU 1 停止，触发 CPU 2 的引导，并仅在 CPU 2 运行后继续，从而保证协调的双-CPU 运行。

为什么这种十六进制组合方法不适用于其他双核器件？

由于存储器所有权不同，合并 CPU1 和 CPU2 二进制文件的相同方法可能无法正常工作。例如，以 F2838x 器件为例

f2838x 闪存：

CPU	物理/全局闪存地址	尺寸	本地地址
CPU1	0x200000	512KB	0x080000（CPU2 也会看到相同地址）
CPU2	0x300000	512KB	0x080000（CPU1 也会看到相同地址）

- 在这里，两个 CPU 都有自己的*本地*内存映射，,每个 CPU 都可以看到自己的闪存起始地址为 **0x080000**。因此通过单个-CPU 版本生成的链接器映射会显示两个 CPU 看起来都是从同一地址开始。
- **hex2000** 根据 .out 文件中出现的*本地*地址合并文件。由于两个 CPU 都将 0x08000 报告为闪存开端，因此在没有地址重映射的情况下进行合并会导致一个映像覆盖另一个映像。
- 为了正确对它们进行编程，在合并之前，必须将 CPU2 段重新映射到全局库。

我们目前正在努力为其他双核器件实施类似解决方案。

8 处理器间通信

IPC 是内核发出信号并将数据传递给彼此的硬件机制。IPC 的主要组件包括：-

- **IPC** 标志寄存器 — 每个内核对 32 个硬件标志位（CPU1↔CPU2、CPU1↔CM、CPU2↔CM）。一个内核设置标志；另一个内核获得中断或者可以轮询标志。这用于轻量级信令（“数据准备就绪”、“收到命令”）。
- **消息 RAM (MSG RAM)** — 这是一个专用的 SRAM，可由两个内核物理访问，并用于传递数据。每个方向都有自己的 MSG RAM（CPU1 至 CPU2 的 MSG RAM 从 CPU2 侧受到写保护以避免损坏）。
- **全局共享 RAM (GS RAM)** – 这些是更大的共享存储器块，可由多个内核访问。可动态分配此内存的所有权。在 C28x 及 CM 内核之间不可用（此处仅使用 MSG RAM）。
- **IPC 驱动程序/ driverlib** - C2000Ware 提供了一个软件 IPC 驱动程序，可将消息传递协议（命令+ 地址 + 数据）层置于硬件标志之上。

有关每个器件特定 IPC 寄存器的更多信息，请参阅器件 TRM。

9 有用链接

C28x 器件的一般调试提示 — C2000 调试提示及技巧

调试多个内核 — 6.调试多个内核 — C2000™ 多核开发指南

相关常见问题解答

(4) TMS320F28388D：使用具有 2 个内核的调试器 — C2000 微控制器论坛 — C2000™ 微控制器 — TI E2E 支持论坛

(4) TMS320F28384S：调试多个内核问题 — C2000 微控制器论坛 — C2000™ 微控制器 — TI E2E 支持论坛

(4) TMS320C28345：C28p65 双核 CPU2 调试问题 — C2000 微控制器论坛 — C2000™ 微控制器 — TI E2E 支持论坛

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出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

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。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 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。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月