User's Guide

CLB 工具

Nima Eskandari, Richard Poley and Omer Amir

摘要

本用户指南介绍了可配置逻辑块 (CLB) 工具的结构和使用方法，并假定读者已经熟悉 CLB 的架构和 CCS IDE。有关 CLB 架构的信息，请参阅器件特定技术参考手册 (TRM)。

内容

插图清单

表格清单

商标

1 引言

CLB 是集成到某些 C2000™ 器件中的硬件模块。CLB 包含一组可配置的块和内部连接，使用户能够按照可以使用 FPGA 实现的方式创建自己的自定义数字逻辑。CLB 可以增强现有器件外设的功能，也可以创建新的外设功能。之前已针对器件发布了不同版本的 CLB。有关可用于特定器件的 CLB 类型的更多信息，请参阅 *C2000 实时控制 MCU 外设*。使用软件实用程序（本文中称为“CLB 工具”）来配置 CLB。

1.1 CLB 工具概述

用户可以使用 CLB 工具配置和连接每个 CLB 逻辑块中的子模块。该工具包含在 SysConfig 图形用户界面 (GUI) 中，名称为 Tile Design；该 GUI 是 Code Composer Studio™ (CCS) 的一部分。该工具提供一些示例，旨在帮助用户了解该工具的功能并自行创建工程。

该工具会生成一个 C 头文件，其中包含一组与用户在 GUI 中定义的配置设置相对应的常数。该工具还生成一个 C 源文件，该文件使用 C 头文件中的常数，通过一系列寄存器加载操作将这些常数加载到 CLB 寄存器中，从而初始化 CLB 模块。器件初始化期间必须调用该 C 源文件中的函数。要配置 CLB 逻辑块与其他器件外设（包括交叉开关和其他 CLB 逻辑块）之间的输入和输出连接，请使用 SysConfig 中的 CLB 模块。这些连接的配置必须与 CLB 工具分开完成。CLB 工具包括以下功能：对这些输入进行仿真以及测试逻辑块配置的功能。

1.2 CLB 配置过程概述

CLB 工具基于 CCS 中的 SysConfig 工具。生成的 C 头文件和源文件与 C2000Ware SDK 相结合，可以配置 CLB。要对设计进行仿真，需要安装许多第三方工具，包括编译器和波形查看器。有关 CLB 仿真器的更多信息，请参阅节 3.5。

CLB 工具生成一个“.dot”文件，该文件以方框图的形式显示子模块互连，有助于验证设计。此文件是使用 CCS 编译后处理步骤以 HTML 和 SVG 格式创建的。这些步骤使用了提供的示例中的 node.js 和 JavaScript 库。该工具还生成“clb_sim.cpp”文件。使用 GCC 编译器来编译 CPP 文件以及其他 CLB 仿真模型。编译的输出是一个“.exe”文件，必须在本地计算机上执行该文件以生成“.vcd”文件。该“.vcd”文件可用于通过外部图形查看器进行时序分析。所有这些步骤都通过该工具生成的“clb_simulation”文件完成。此文件为批处理 (.bat) 或 shell (.sh) 文件，具体取决于所使用的操作系统（本用户指南中使用的图像基于 Windows 操作系统）。必须执行此文件以生成适当的“.vcd”文件。

在生成的 C 头文件“clb_config.h”中对 CLB 配置进行编码。CLB 工具生成的“clb_config.c”文件使用生成的头文件将配置加载到 CLB 模块的寄存器中。SysConfig 中的 CLB 模块必须连接适当的 Tile Design 才能实际进行配置，如图 1-1 中所示。因为 SysConfig 中 Tile Design 与 CLB 模块互不相关，因此可以创建任意数量的逻辑块配置，而器件上的 CLB 实例数量有限。这些 Tile Design 有选择地加载到 CLB 中。

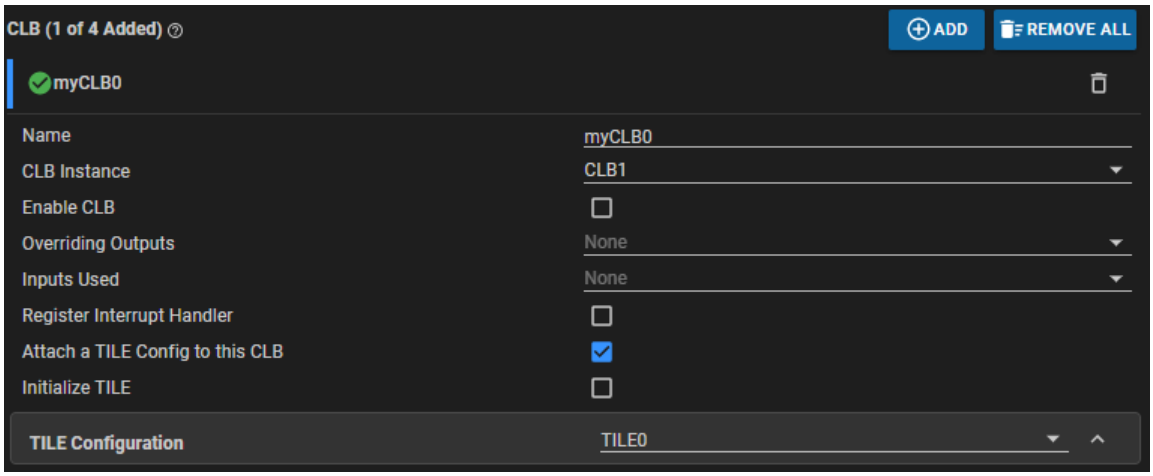

图 1-1. 将 Tile Design 连接到 CLB 实例

请务必注意，在 C28x 器件的应用代码中，必须在执行器件初始化步骤期间调用“clb_config.c”文件中的函数。图 1-2 显示了 CLB 工具的输出以及 post-build 步骤。

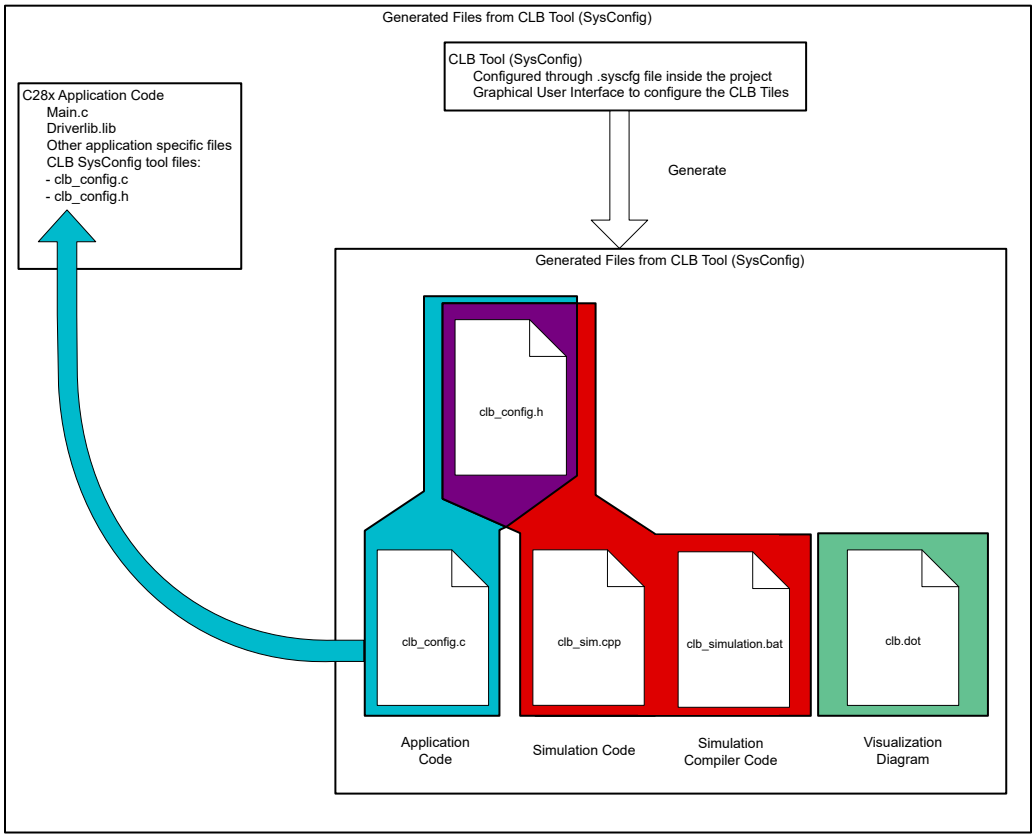

图 1-2. CLB 工具工程结构

在典型情况下，用户从指定 CLB 逻辑功能开始。这可以采用逻辑电路图、时序信息、书面说明、VHDL 代码或其他形式。安装必需的工具后，第一步是连接逻辑块子模块以实现所需的逻辑。

指定内容可能包含一组时序图，在这种情况下，用户可以选择执行 CLB 配置的仿真，以确保行为符合预期。此步骤包括定义一组输入测试激励，以及构建一个仿真工程以生成可在图形查看器中打开的仿真波形。如果结果不符合预期，用户可以修改 CLB 工具设置并重复仿真。

在仿真产生正确的波形后，用户可以使用 CCS 将设计下载到器件中以运行或调试代码。

在为 C28x 器件启用了 SysConfig 的 CCS 工程中，创建 Tile Design 的 HTML 和 SVG 方框图的步骤是在编译后处理步骤中完成的。在用户构建 CCS 工程时，使用 C28x 编译器编译用户应用代码以及生成的“clb_config.h”和“clb_config.c”，并生成一个“.out”文件。

“clb_simulation”文件使用 GCC 编译器编译生成的仿真文件“clb_sim.cpp”和“clb_config.h”以及 CLB 仿真模式。此步骤的输出是“.exe”文件 (“simulation_output.exe”)，该文件会自动运行以生成“CLB.vcd”文件。可以使用外部图形查看器来查看该文件。

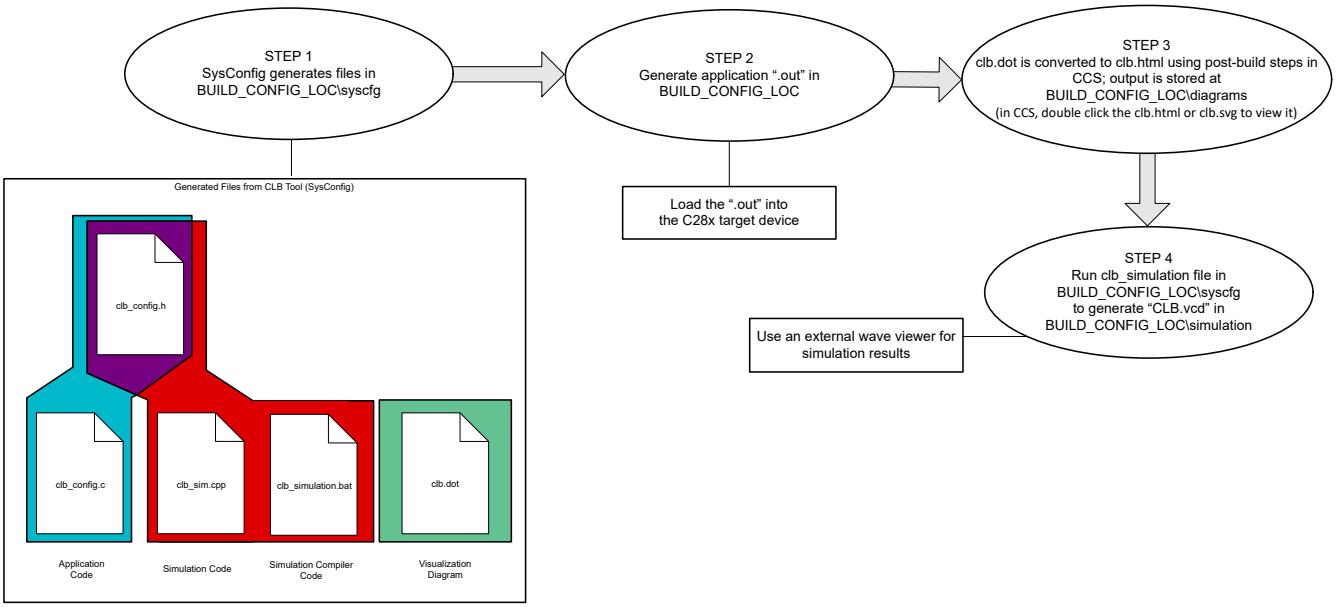

图 1-3. CLB 工具构建过程

在 CLB 工具构建过程中，BUILD_CONFIG_LOC 目录是用作工程的构建配置输出的目录（这可以是“Debug”、“CPU1_RAM”等），并共用活动构建配置的名称。

2 开始使用

该部分旨在帮助新用户快速开始使用 CLB 工具。

2.1 CLB 相关配套资料

2.1.1 基础资料

- C2000 Academy - CLB
- C2000™ 可配置逻辑块 (CLB) 系列（视频）
- 定制片上外设颠覆传统逻辑
- *在各种应用中通过 CLB 实现差异化并脱颖而出*
- 在各种汽车应用中通过可配置逻辑实现差异化（视频）

2.1.2 入门资源

- *C2000™ Position Manager PTO API 参考指南应用报告*
- *CLB 工具用户手册*
  - 基本示例为 7-15（从这些示例开始）。1-6 是更复杂的示例。
- *使用 C2000 可配置逻辑块进行设计*
- 如何向现有的 driverlib 工程添加 SYSCONFIG 支持（Pinmux 和外设初始化）？

- *如何将自定义逻辑从 FPGA/CPLD 迁移到 C2000 微控制器应用报告*
  - 第 1-3 章对开始使用和学习 CLB 非常有用。通过学习后面各章，可让您成为从 FPGA/CPLD 到 C2000 CLB 迁移方面的专家。

2.1.3 专家资料

- *利用 CLB 实现三电平逆变器延时保护*
- *使用 C2000™ 可配置逻辑块应用报告诊断 Δ-Σ 调制器位流*
- *如何使用可配置逻辑块 (CLB) 应用报告实现定制串行接口*
- *基于 C2000™ MCU 的 Tamagawa T-Format 绝对编码器主接口参考设计*

2.2 引言

为了使用该工具，必须安装 CCS 版本 9.0 或更高版本。早期版本的 CCS 不包含 CLB 配置所需的 SysConfig 实用程序。有关 CCS 的更多信息和下载选项，请访问 CCS 下载页面。

用户可以使用上述工具来配置 CLB。为了对设计进行仿真，必须安装以下附加的外部（非 TI）工具：

- GNU 编译器 (TDM-GCC)
- 仿真查看器 (GTKWave)

2.3 安装

2.3.1 用于编译 SystemC 的安装

为了允许仿真源文件 clb_sim.cpp 针对 Windows 进行编译，请执行以下操作：

1. 从 SourceForge 下载“tdm-gcc”版本 5.1.0-2。
2. 打开已下载的文件。
3. 取消选中“Check for updated files on the TDM-GCC server”选项。
4. 从安装向导中选择“Create”。

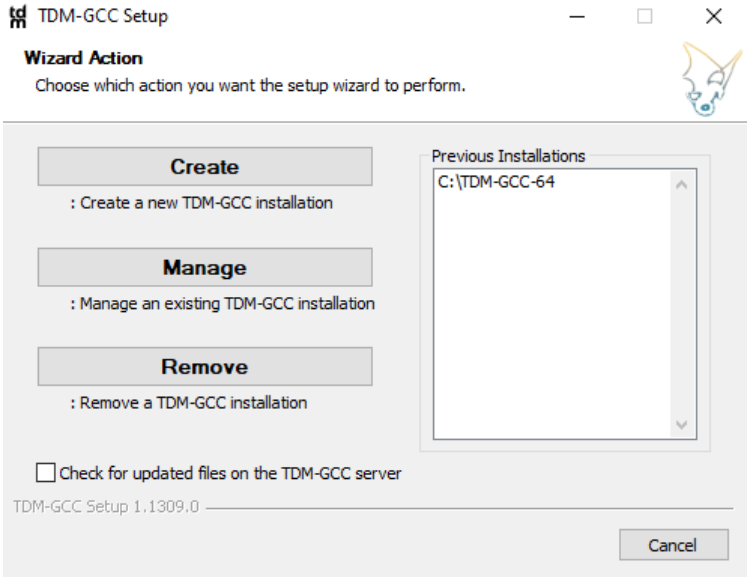

图 2-1. TDM 编译器安装向导

5. 如果向导显示有关许可证更改的信息，请选择“Next”。

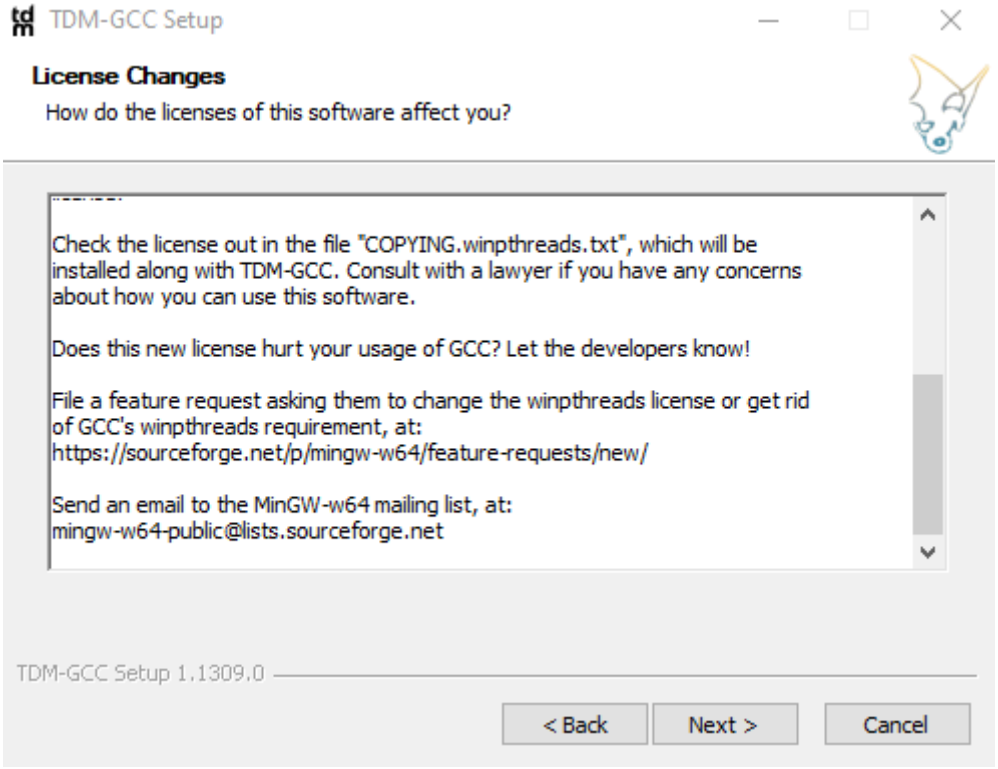

图 2-2. TDM 许可证更改

6. 选择 C:\TDM-GCC-64 作为安装目录并点击“Next”。

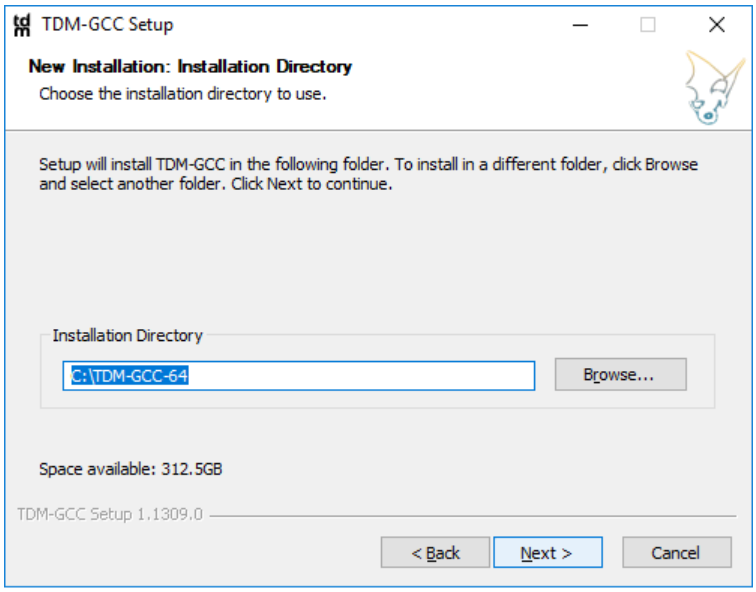

图 2-3. TDM 编译器路径

7. 在点击“Install”之前，请确认选择了正确的组件。

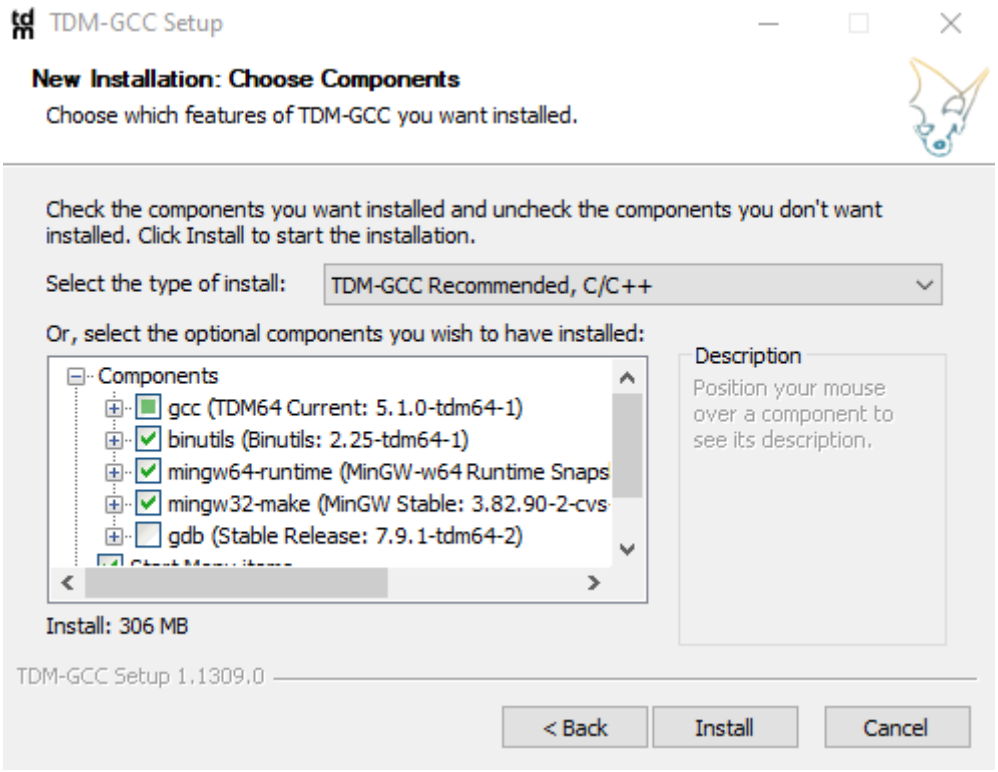

图 2-4. TDM 组件

8. 安装完成后，依次选择“Next”和“Finish”。

对于 Mac 或 Linux，需要安装 SystemC 库，但 G++ 编译器不需要。在继续操作之前，请验证 G++ 编译器是否为最新版本。要为 Mac 或 Linux 安装 SystemC，请执行以下操作：

1. 打开终端。
2. 运行 `sudo apt-get install build-Essential`。
3. 从 Accelera 中安装 SystemC 2.3.3 并通过在终端中运行 `tar -xvf systemc-2.3.3.tar.gz` 来对其进行解压缩。
4. 通过执行 `sudo cp -r systemc-2.3.3 /usr/bin` 将解压的文件夹复制到“/usr/bin”目录中。转到 tar 命令创建的目录 (不是在“/usr/bin”中)，然后创建一个名为“objdir”的目录。

```
~/Downloads$ sudo cp -r systemc-2.3.3 /usr/bin
~/Downloads$ cd systemc-2.3.3
~/Downloads/systemc-2.3.3$ mkdir objdir
~/Downloads/systemc-2.3.3$ cd objdir
```

图 2-5. SystemC 目录创建

5. 运行 `sudo ../configure --prefix=/usr/bin/systemc-2.3.3/`。

```
----------------------------------------------------------------------
Configuration summary of SystemC 2.3.3 for x86_64-unknown-linux-gnu
----------------------------------------------------------------------

Directory setup (based on classic layout):
  Installation prefix (aka SYSTEMC_HOME):
     /usr/bin/systemc-2.3.3
  Header files  : <SYSTEMC_HOME>/include
  Libraries     : <SYSTEMC_HOME>/lib-linux64
  Documentation : <SYSTEMC_HOME>/docs
  Examples      : <SYSTEMC_HOME>/examples

Architecture    : linux64
Compiler        : g++ (C/C++)

Build settings:
  Enable compiler optimizations  : yes
  Include debugging symbols      : no
  Coroutine package for processes: QuickThreads
  Enable VCD scopes by default   : yes
  Disable async_request_update   : no
  Phase callbacks (experimental) : no

----------------------------------------------------------------------
```

图 2-6. SystemC 配置输出

6. 运行 `sudo make`。

```
  CXX      tracing/sc_wif_trace.lo
  CXX      utils/sc_hash.lo
  CXX      utils/sc_list.lo
  CXX      utils/sc_mempool.lo
  CXX      utils/sc_pq.lo
  CXX      utils/sc_report.lo
  CXX      utils/sc_report_handler.lo
  CXX      utils/sc_stop_here.lo
  CXX      utils/sc_string.lo
  CXX      utils/sc_utils_ids.lo
  CXX      utils/sc_vector.lo
  CXXLD    libsysc.la
make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/sysc'
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/sysc'
Making all in tlm_core
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_core'
  CXX      tlm_2/tlm_generic_payload/tlm_gp.lo
  CXX      tlm_2/tlm_generic_payload/tlm_phase.lo
  CXX      tlm_2/tlm_quantum/tlm_global_quantum.lo
  CXXLD    libtlm_core.la
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_core'
Making all in tlm_utils
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_utils'
  CXX      convenience_socket_bases.lo
  CXX      instance_specific_extensions.lo
  CXXLD    libtlm_utils.la
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src/tlm_utils'
Making all in .
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src'
  CXXLD    libsystemc.la
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/src'
Making all in examples
make[1]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
Making all in sysc
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/sysc'
  GEN      copy-check-data
To compile and run the examples type
   make check
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/sysc'
Making all in tlm
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
Making all in common
make[3]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm/common'
  GEN      copy-check-data
To compile the TLM examples library type
   make check
make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm/common'
Making all in .
make[3]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
  GEN      copy-check-data
make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
To compile and run the examples type
   make check
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples/tlm'
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
  GEN      copy-check-data
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[1]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[1]: Nothing to be done for 'all-am'.
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
                ~/Downloads/systemc-2.3.3/objdir$
```

图 2-7. Make 输出

7. 运行 `sudo make install`。

```
make[3]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir/examples'
make[1]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[2]: Entering directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[2]: Nothing to be done for 'install-exec-am'.
 /bin/mkdir -p '/usr/bin/systemc-2.3.3'
 /usr/bin/install -c -m 644 ../AUTHORS ../NOTICE ../ChangeLog ../INSTALL ../LICENSE ../NEWS ../README ../RELEASENOTES ../cmake/INSTALL_USING_CMAKE '/usr/bin/systemc-2.3.3'
make[2]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
make[1]: Leaving directory '/home/a0498187/Downloads/systemc-2.3.3/objdir'
                ~/Downloads/systemc-2.3.3/objdir$
```

图 2-8. Make Install 输出

2.3.2 安装仿真查看器

要在 Windows 上安装 GTKWave，请执行以下操作：

1. 从 SourceForge 下载波形查看器 GTKWave
2. 下载可正确安装在 Windows 上的本地二进制文件（例如，对于 64 位 Windows，选择“gtkwave-3.3.100-bin-win64”），并将下载的 zip 文件解压缩到目录 c:\gtkwave 中。

要在 Mac 或 Linux 上安装 GTKWave，请执行以下操作：

1. 从终端中，运行命令 `sudo apt install gtkwave`

3 使用 CLB 工具

该部分介绍如何使用 CLB 工具来配置 CLB 逻辑块。CLB 工具需要 CCS 版本 9.0 或更高版本。

3.1 导入空 CLB 工程

启用了 CLB 的基于 Driverlib 的示例工程可在 <C2000WARE_INSTALL>/driverlib/<device>/examples 中找到。

例如，对于 F2837xD 器件，CLB 示例工程的路径为 <C2000WARE_INSTALL>/driverlib/f2837xd/examples/cpu1/clb。

1. 在 CCS 菜单中，依次点击“Project”>“Import CCS Projects…”。
2. 在“Select search-directory”中输入 CLB 示例工程的路径，然后点击“Refresh”，或者通过选择“Browse”按钮直接选择合适的文件夹。
3. 选择“clb_empty”工程。
4. 选中“Copy projects into workspace”。
5. 点击“Finish”。

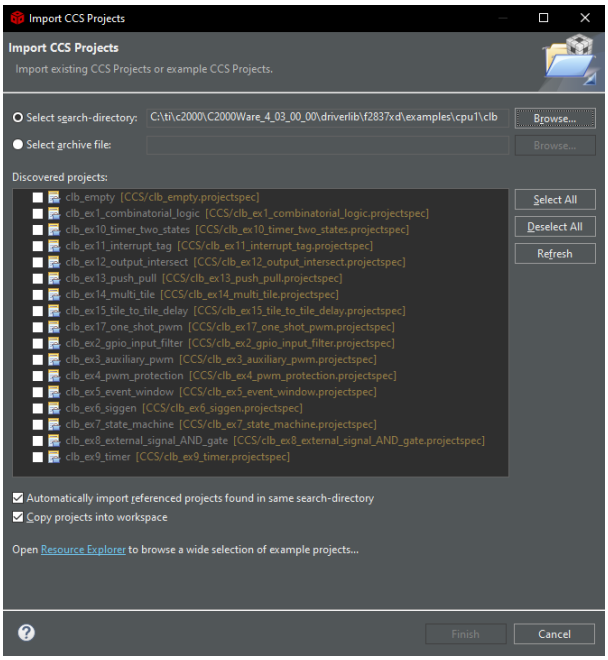

图 3-1. 导入 CCS Eclipse 工程

3.2 更新变量路径

上面导入的空 CLB 工程不仅能够为 C28x 目标器件生成“.out”文件，而且能够生成设计的仿真文件和 HTML/SVG 方框图。要使用编译后处理步骤创建方框图，必须为 C2000Ware 根和节点工具的位置设置正确的路径。要仔细检查这些路径是否正确，请执行以下操作：

1. 右键点击工程并选择“Project Properties”。
2. 在“Resources”下，选择“Linked Resources”。
3. 检查以确保下面的所有路径均正确：
   a. C2000WARE_ROOT（此路径用于 CLB 方框图和其他包含路径）

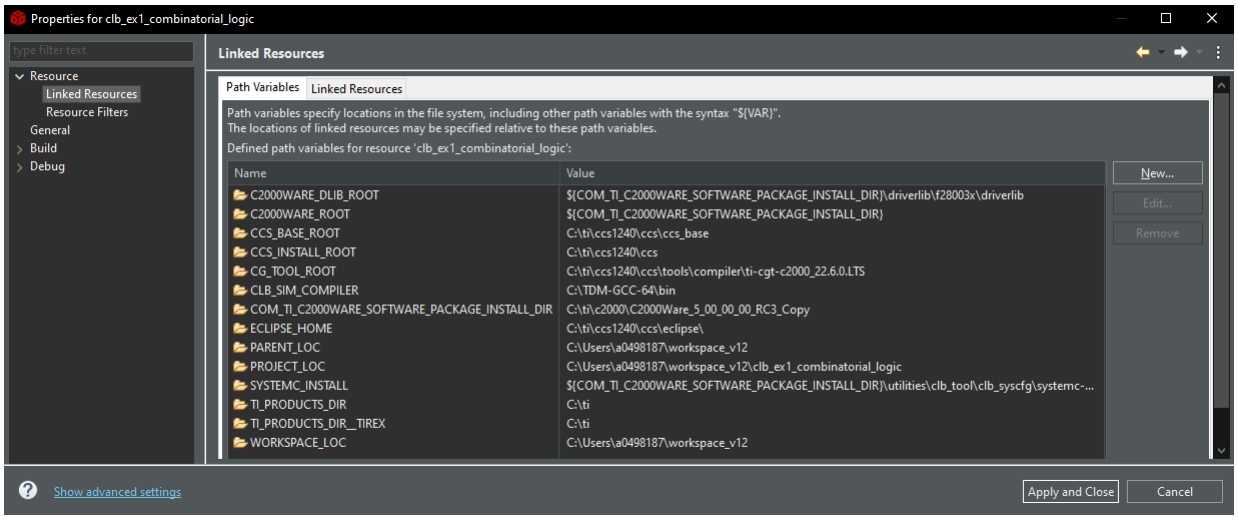

图 3-2. 链接资源

4. 如果名称左侧的图标不是文件夹，而是一个感叹号，则表示您的系统上不存在该路径，您必须手动选择正确的路径
5. 检查以确保以下系统变量的路径正确：
   a. NODE_TOOL

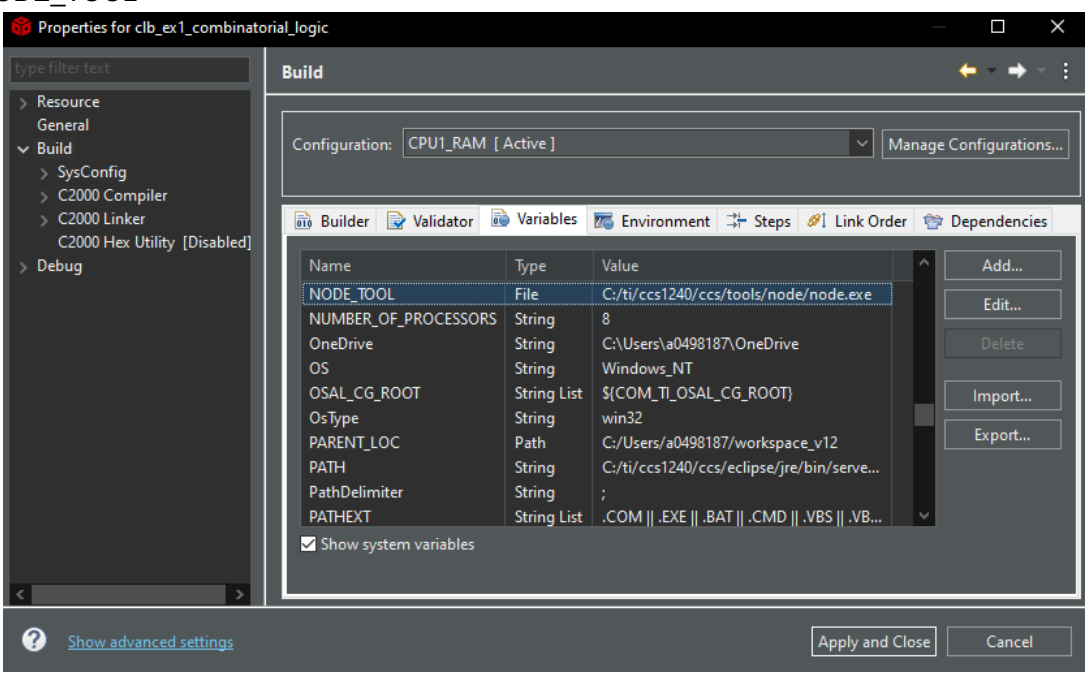

图 3-3. 构建变量

要生成方框图，必须将构建变量 GENERATE_DIAGRAM 设置为 1。通过转到“Project Properties”>“Build”>“Variables”，可以找到工程的构建变量，如图 3-4 中所示。此变量允许在构建工程后运行“Steps”下面列出的编译后处理步骤。这些方框图可以在工程相关构建配置下的“diagrams”目录中找到，如图 5-5 中所示。

备注

对于 Mac 和 Linux，编译后处理步骤中使用的条件将无法正确执行。要生成方框图，必须删除测试 `if ${GENERATE_DIAGRAM} == 1`。

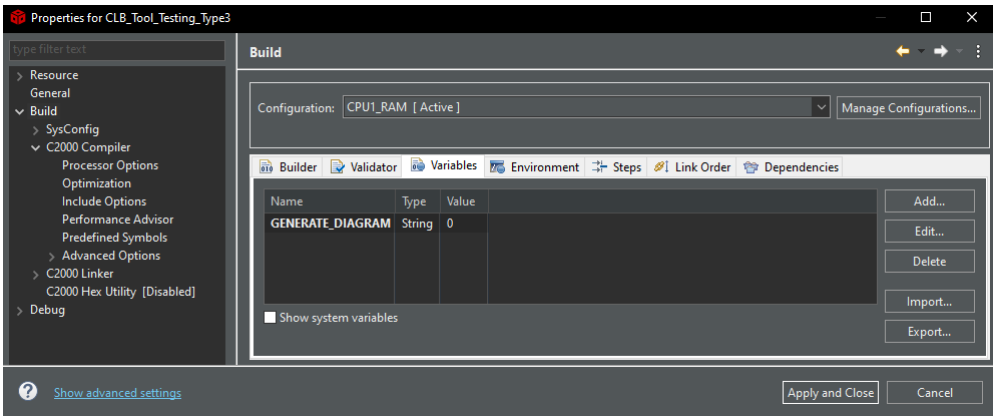

图 3-4. 用于生成方框图的构建变量

3.3 配置 CLB 逻辑块

要打开 SysConfig 工具，请在 CCS Project Explorer 窗口中双击您要编辑的“.syscfg”文件。图 3-5 显示了相关屏幕。

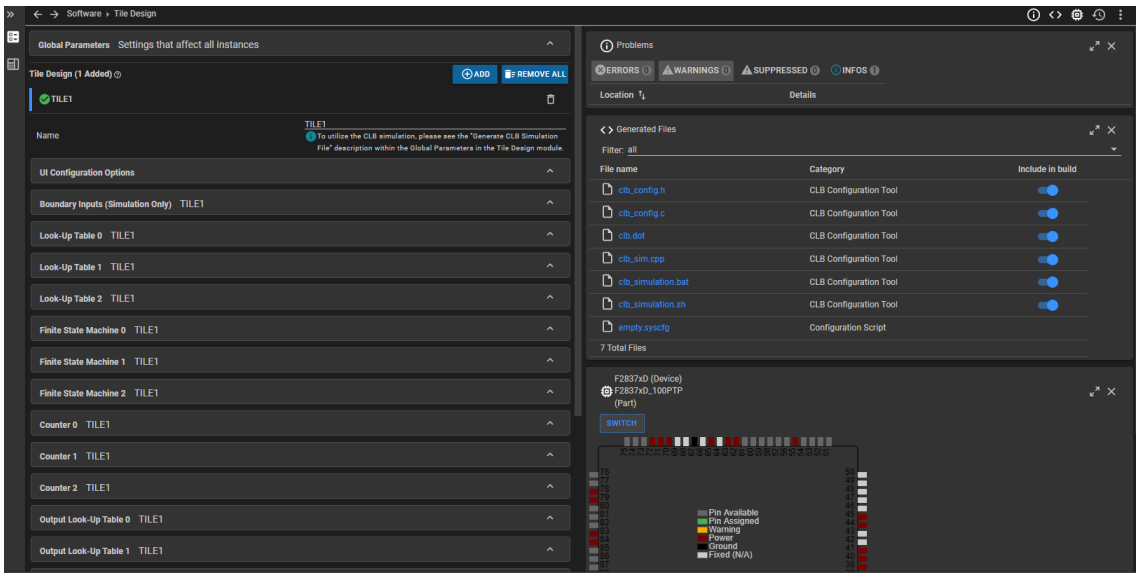

图 3-5. CLB 工具 SysConfig 屏幕

如果未打开该屏幕，请确保您已正确完成之前的步骤。

每个 .syscfg 文件中都包含 CLB 逻辑块的配置。您可以根据需要更改逻辑块的名称。对于突出显示的逻辑块，在右侧窗格中显示了子模块列表。通过选择子模块，可以检查和编辑每个子系统的参数。

“BOUNDARY”项目是一种特殊情况。该组允许用户选择仅用于仿真的逻辑块输入。生成工具配置时，CLB 输入始终来自 SysConfig 中的 CLB 模块，不过，出于仿真目的，用户可以指定方波信号源、周期和占空比（均以时钟周期为单位）、同步以及输入流水线条件，如图 3-6 中所示。还支持用于仿真目的的自定义波形生成。有关仿真器的更多信息，请参阅节 3.5。这些选项仅用于仿真，不会影响实际的 CLB 配置或其在器件上的实现。

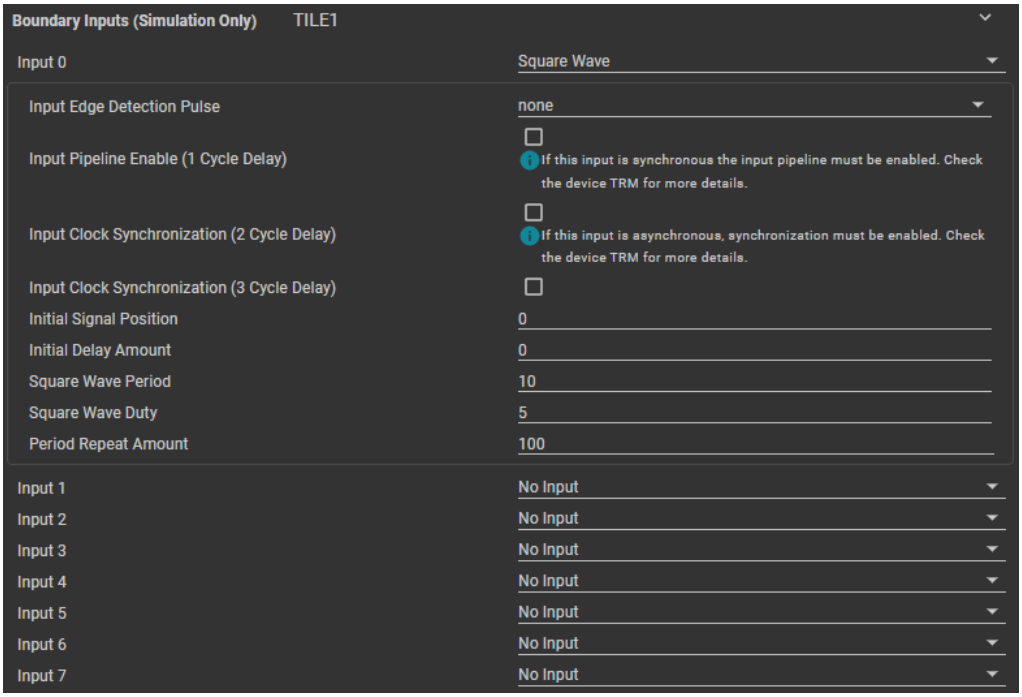

图 3-6. 边界输入选项

用户使用工具中的复选框和下拉选项配置和连接每个逻辑块中的子模块。除“BOUNDARY”外的所有子模块也都有“User Description”。此说明是一个多行文本框，用户可以在其中输入注释以帮助提供 CLB 每个部分的上下文。

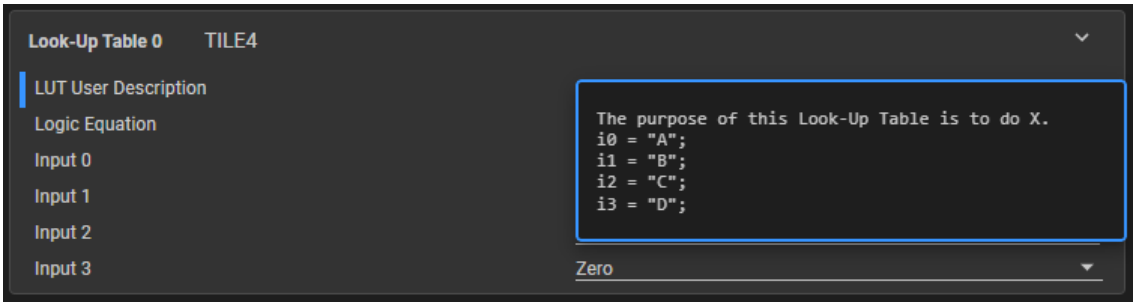

图 3-7. “User Description”文本框

当鼠标光标悬停在配置工具中的每个项目上时，会显示上下文相关帮助。图 3-8 显示了“Counter 0”子模块中“Match Reference 1”字段的示例。

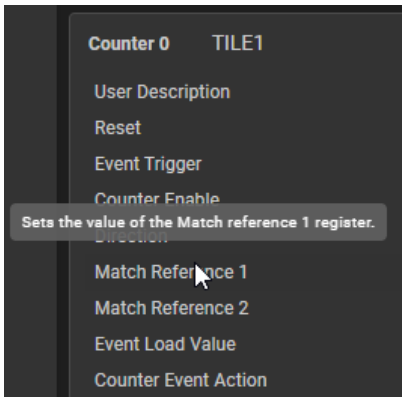

图 3-8. 计数器选项

通过使用 C 格式的文本输入来配置 LUT 和 FSM 的逻辑方程。表 3-1 显示了布尔方程中允许使用的符号。

表 3-1. 支持的逻辑运算

逻辑运算	符号
与	&
或	|
异或	^
非	!

支持使用括号：例如，用户可以编写 i1 | !(i2 & i3)。输入方程后，该工具会对方程进行语法检查。无效的方程通过输入行下方的错误消息指示。

一些不太可能的逻辑组合会向用户生成警告。图 3-9 显示了一个示例，其中用户尝试在布尔方程的“LUT 0”中使用 i2 输入。但是，i2 被配置为一个常数，不太符合用户期望。该警告同时显示在方程下方和输入选择下方。

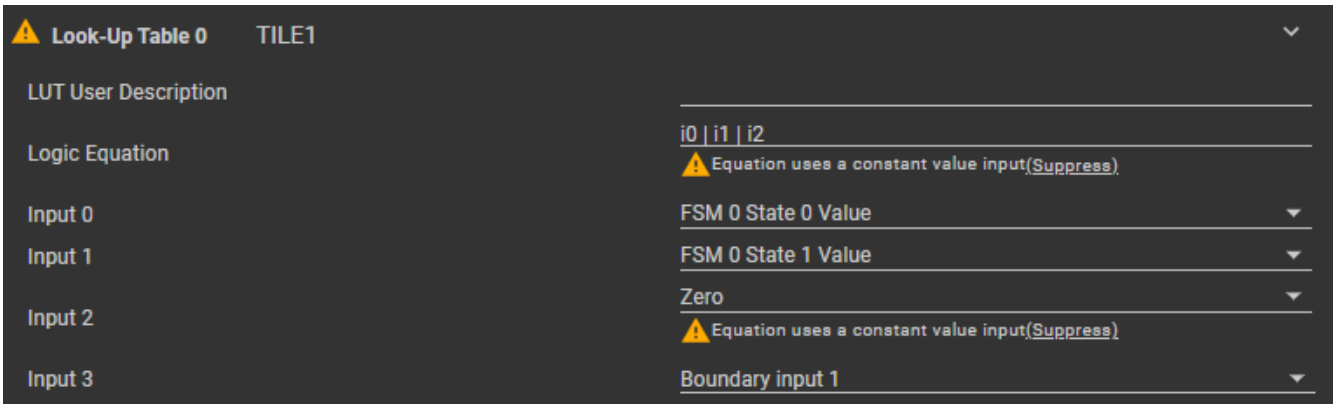

图 3-9. 方程警告

对于某些字段，该工具会对数字输入执行范围检查，以确保它们落在允许的范围内。例如，尝试加载大于 2^{32} 的值的计数器子模块将产生警告，因为计数器只有 32 位宽。

当用户输入配置数据时，该工具会自动生成若干个文件。要查看生成的文件，请点击该工具右上角的“< >”符号，然后选择文件名以打开它。

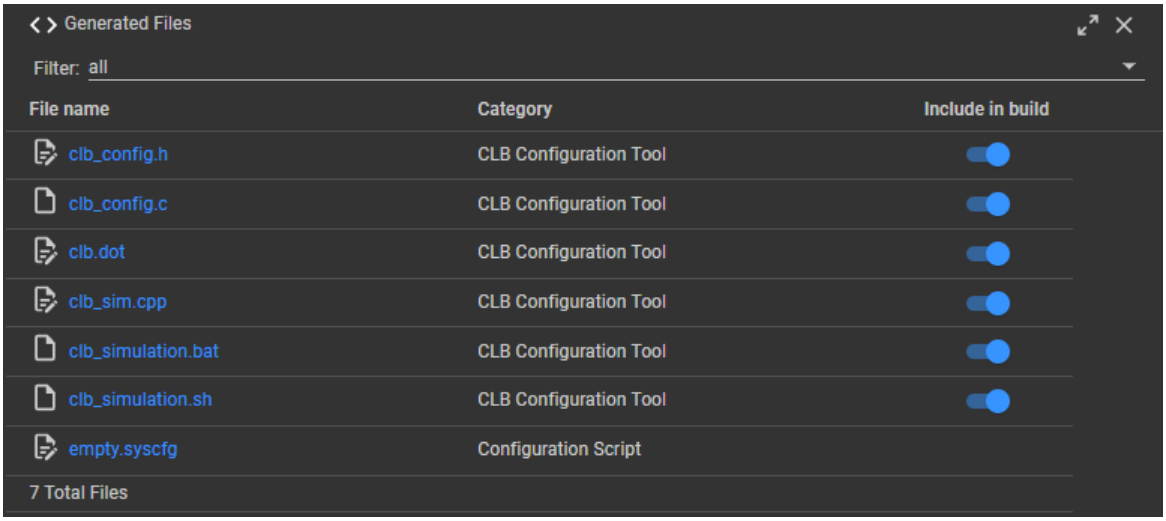

图 3-10. CLB 工具生成的文件

CLB 寄存器设置包含在头文件“clb_config.h”中，用户可以通过点击文件名来打开该文件。图 3-11 显示了一个示例。请务必记住，每次用户更改任何 CLB Tile Design 设置时，该工具都会更新此文件。因此，该工具将覆盖对所生成文件内容的手动更改。如果在更改 CLB 设置时该文件保持打开状态，则在选择“Unified Diff”选项时可以查看文件中受影响的寄存器数据的变化。

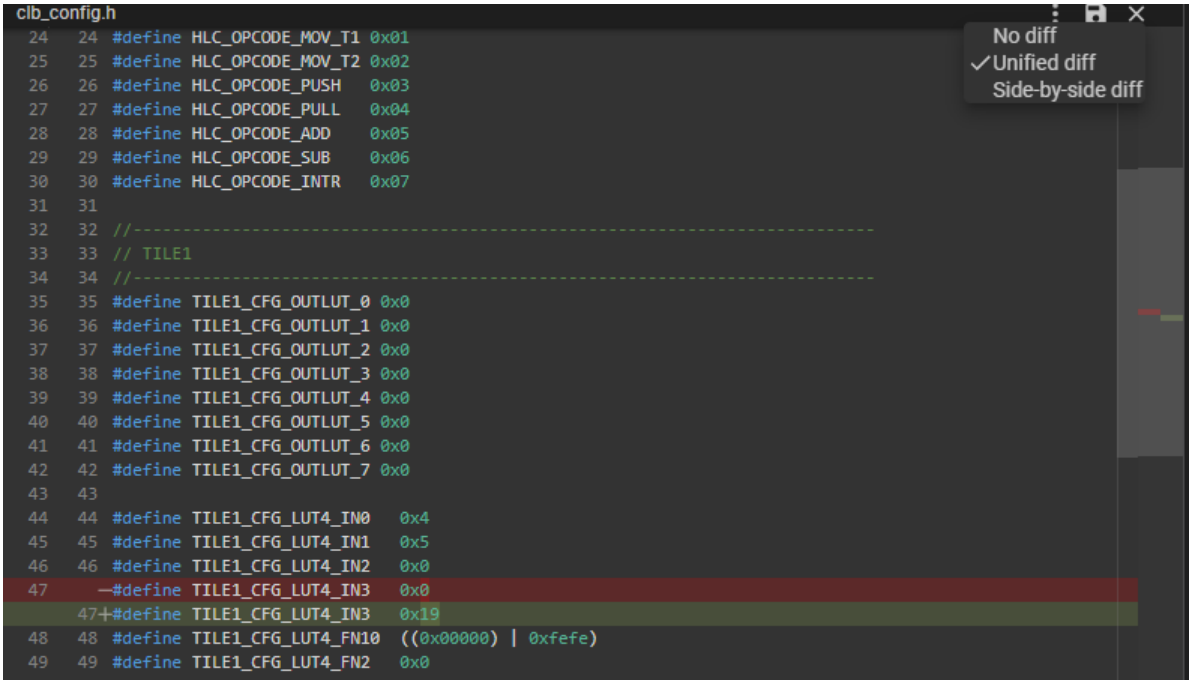

图 3-11. “clb.h”头文件示例

利用文件“clb.dot”，用户可以检查子模块互连的外观。此方框图的 HTML 和 SVG 版本在编译后处理步骤中生成，可在 CCS 中打开和查看这些步骤。

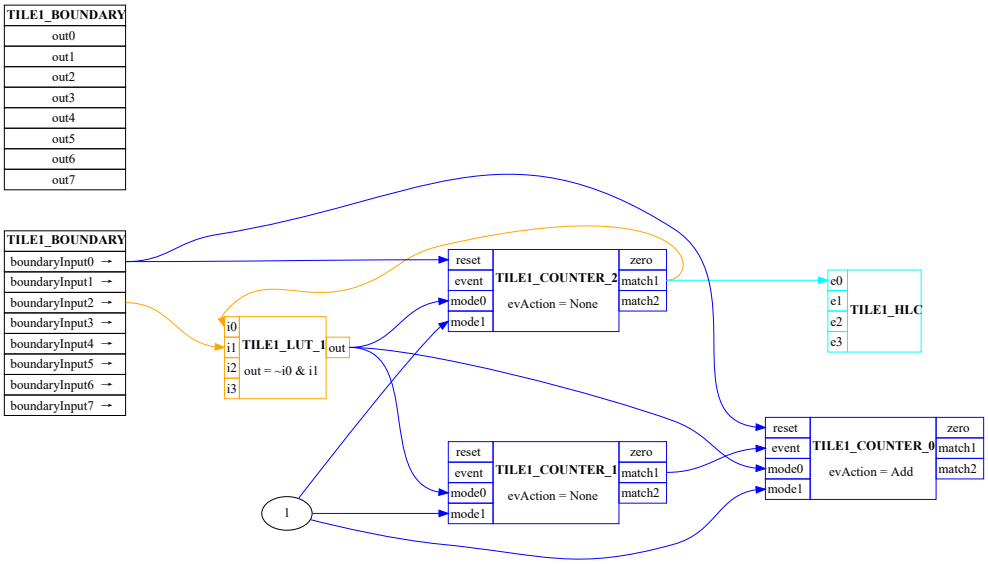

图 3-12. CLB 方框图

HLC 子模块的字段包含用于配置事件和初始值的字段。四个事件中的每个事件都可能触发由最多八条指令构成的简短程序的执行。有关 HLC 的更多信息，请参阅器件特定 TRM。

可在“Other Dependencies”的“HLC Program”下拉列表中输入 HLC 指令。在使用完全部八条指令之前，会始终显示一个空白行。在图 3-13 中，用户选择了一个 HLC 触发事件并键入了一个由三条指令构成的简短程序。

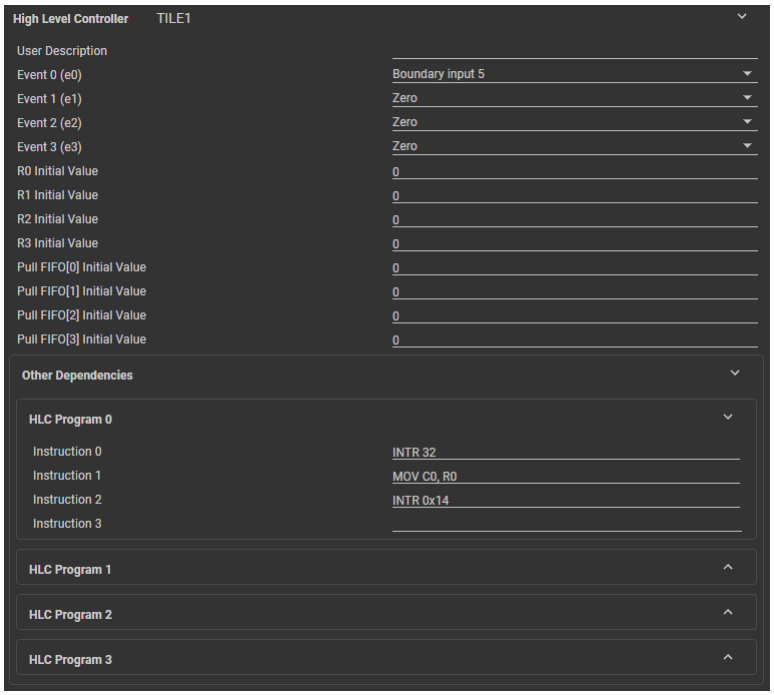

图 3-13. HLC 配置示例

3.4 创建 CLB 方框图

要创建 CLB 方框图，请执行以下步骤：

1. 设置“Build”选项中的“GENERATE_DIAGRAM”变量（必须设置为 1）。

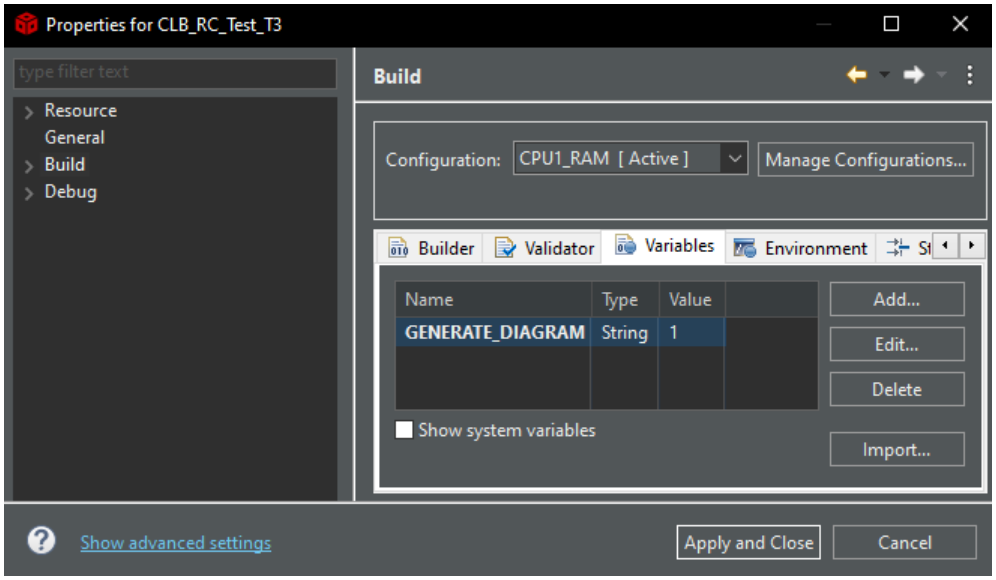

图 3-14. GENERATE_DIAGRAM Build 变量

备注

对于 Linux 或 Mac 等操作系统，需要从编译后处理步骤中删除条件性 `if ${GENERATE_DIAGRAM} == 1` 才能正确执行。请参阅图 5-2，了解 CLB 工程的编译后处理步骤。

2. 构建工程。
3. 查看”diagrams“目录中的 HTML 或 SVG 方框图，如图 5-5 所示。

3.5 使用仿真器

3.5.1 “Statics”面板

配置工具的顶部面板包含用于仿真的“Global Parameters”设置。点击“?”图标可查看简短说明。

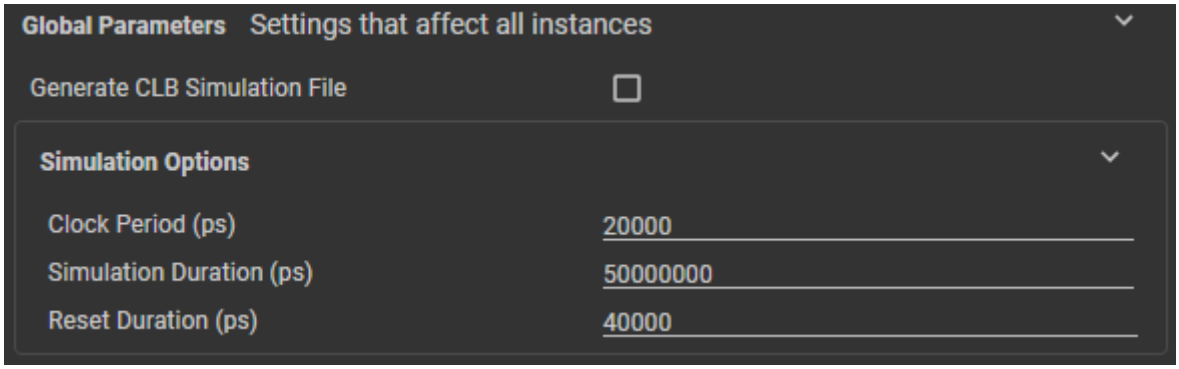

图 3-15. 静态选项

使用“Generate CLB Simulation File”来生成“clb_simulation”文件的内容。当执行此文件时，将为所有 CLB Tile Design 生成 .vcd。此 .vcd 文件将出现在上面执行批处理文件的目录中，并位于新创建的“simulation”目录内。

“Clock Period (ps)”是用于仿真的 CLB 时钟周期（以皮秒为单位）。利用“Simulation Duration (ps)”字段，用户可以控制仿真运行的持续时间（以皮秒为单位）。利用“Reset Duration (ps)”字段，用户可以插入延迟（以皮秒为单位），以模拟器件复位的效果。

3.5.2 创建输入激励

通过在 CCS Project Explorer 窗口中双击文件名来打开 .syscfg 文件。选中“Boundary”类别，将其展开。

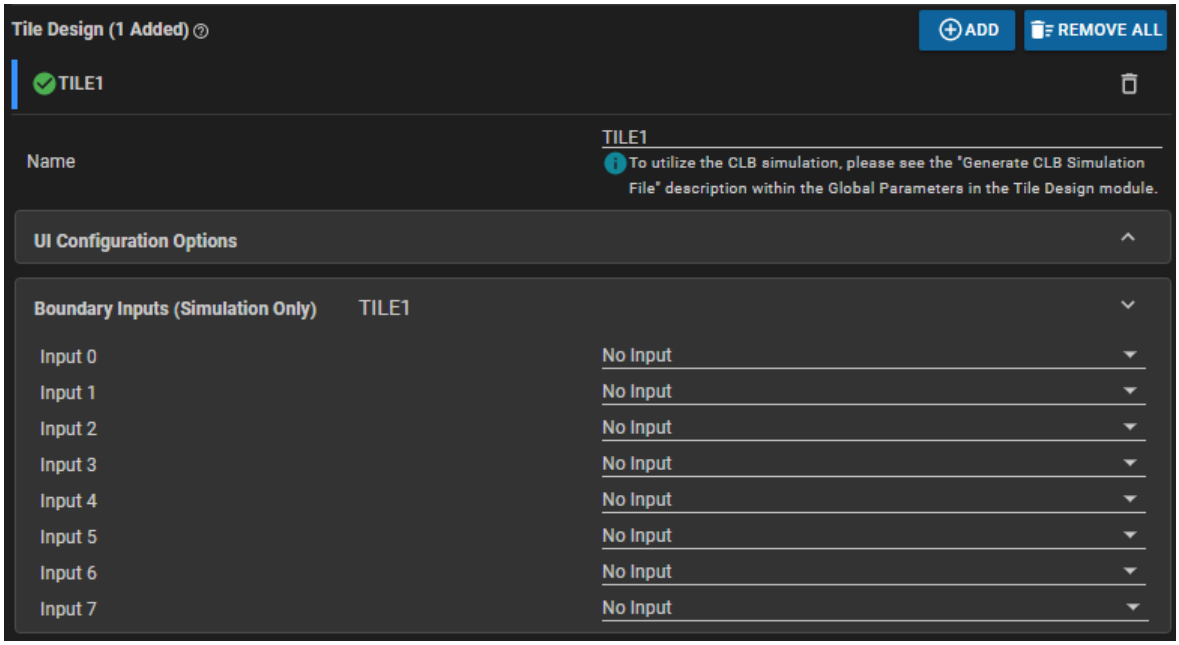

图 3-16. 边界输入 0 到 7

可以使用下拉菜单为八个 CLB 输入中的每一个定义单独的输入激励。点击右侧的向下箭头可显示选项：

- No Input – 默认选项，不生成激励。
- Square Wave – 通过可配置初始信号位置、初始延迟、周期、占空比和周期重复量定义周期性 PWM 输入。

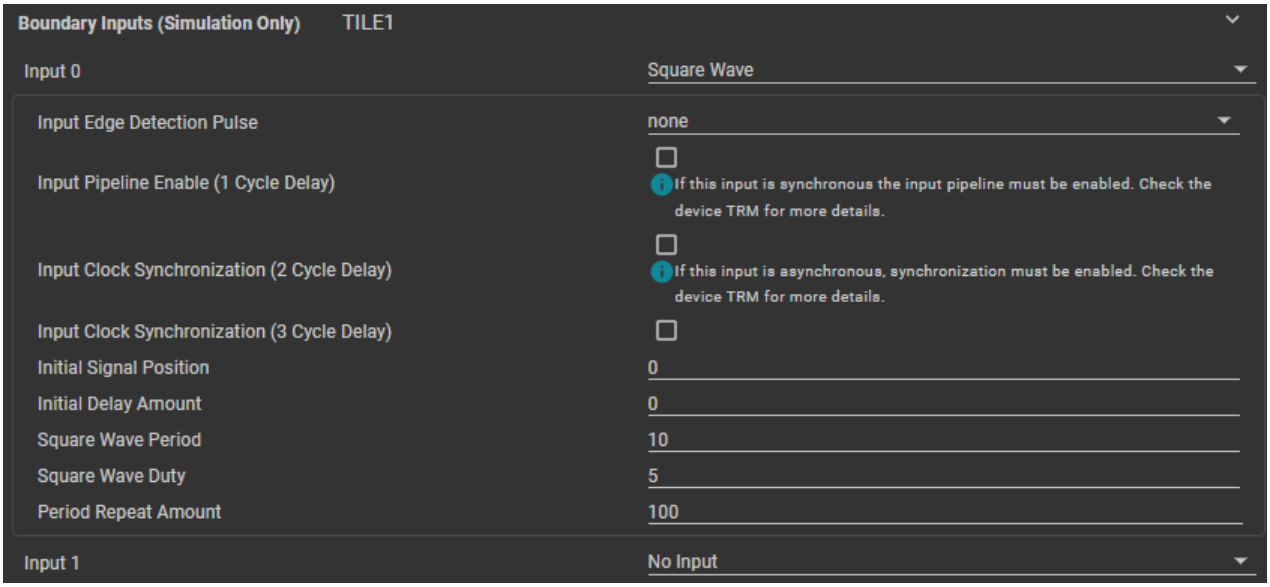

图 3-17. 边界输入“Square Wave”

“Input Edge Detection Pulse”选项为用户提供了通过 PWM 波的上升沿和/或下降沿生成脉冲的选择，在图 3-17 中，其周期和占空比分别设置为 10 个和 5 个 CLB 时钟脉冲。

“Input Pipeline Enable”复选框会向输入信号增加一个单周期延迟；输入信号用作路由到 CLB 作为输入的同步信号。请注意，流水线滤波器仅在某些 CLB 类型上可用。有关更多详细信息，请参阅器件特定 TRM 中的“CLB 输入多路复用器”部分。

“Input Clock Synchronization”复选框会强制输入波形与 CLB 时钟同步（同步器会产生 2-3 个周期的延迟，因此两个时序都有相应的复选框，因为无法预测确切的延迟）。对于来自相对于 CLB 的异步源的信号，此选项是必需的。有关更多信息，请参阅器件特定 TRM 中的“CLB 输入多路复用器”部分。

- Low (0) 或 High (1) - 分别将激励设置为恒定的低电平或高电平。
- Custom Wave Input - 使用伪代码生成自定义激励。

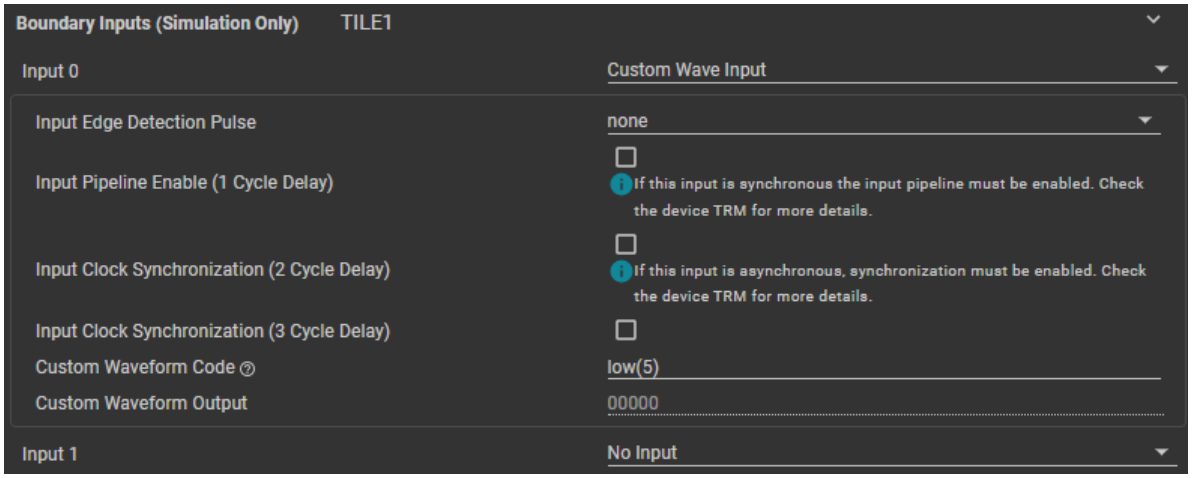

图 3-18. 边界输入定制

“Input Edge Detection Pulse”、“Input Pipeline Enable”和“Input Clock Synchronization”与 Square Wave 激励的工作方式相同，但增加了定制波形伪代码。“high”、“low”和“rpt”指令的数值参数可以是十六进制 (0x1A) 或十进制 (26)。

表 3-2. 定制波形代码指令

指令	说明
#define	用于定义宏的模式替换器
high(N)	将“N”个 CLB 周期的波形设置为高电平
low(N)	将“N”个 CLB 周期的波形设置为低电平
rpt(N)	启动重复块；用 rpt(N) 和 rpt_end 封装的代码将一共重复“N”次
rpt_end	表示重复块结束

- Tile Output - 使用选定的逻辑块输出作为当前逻辑块的输入激励。

图 3-19. 边界输入“Tile Output”

- “Tile Name”必须是 CLB 工具工程中有效逻辑块的名称。由于 CLB 逻辑块的输出是同步的，因此应启用“Input Pipeline Enable”。有关更多信息，请参阅器件特定 TRM 中的 *CLB 输入多路复用器* 部分。

3.5.3 运行仿真

定义 CLB 配置和输入激励之后，用户可以编译工程。下面概述了生成“CLB.vcd”的完整步骤。

1. 启用 Tile Design 模块顶部的“Global Parameters”下拉列表中的“Generate CLB Simulation File”复选框。
2. 构建工程以验证 Tile Design 设置是否正确无误
3. 在文件浏览器或命令行中，执行“clb_simulation”文件。此文件位于保存生成的 SysConfig 文件的位置。对于 CCS，这是工程构建配置目录中的 syscfg 目录（例如“CPU1_RAM/syscfg”）
4. 打开“simulation”目录（位于执行“clb_simulation”的目录上层）
5. 双击“CLB.vcd”文件

假设已经完成了波形查看器配置，双击“CLB.vcd”文件应打开查看器并允许检查波形。图 3-20 显示了设置为显示输入波形样本的 GTKWave 查看器。有关如何添加和查看信号的信息，请参阅查看器文档。

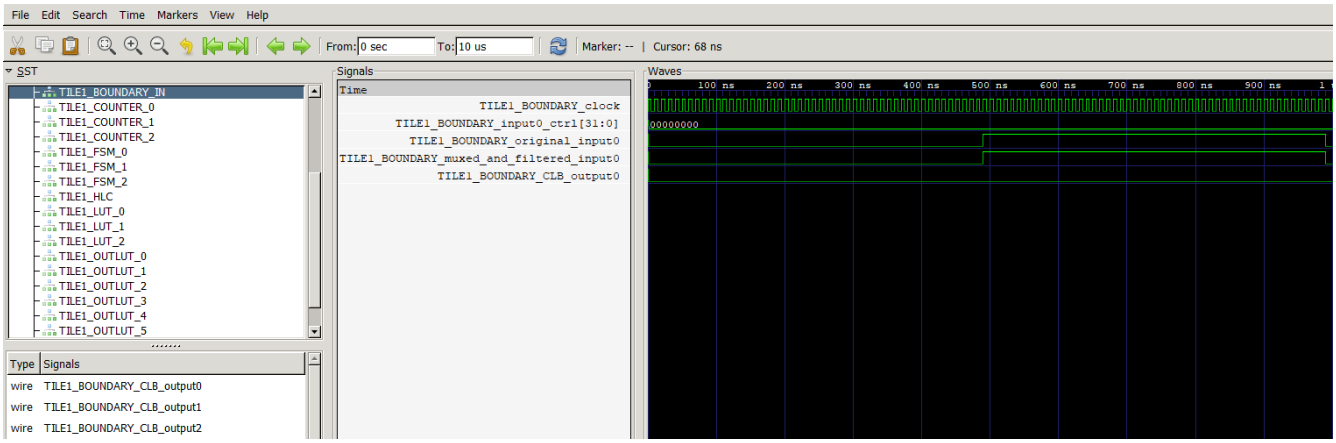

图 3-20. CLB 仿真示例

如果仿真波形与期望的波形不相符，请修改 .syscfg 文件中的配置并重复仿真。

3.5.4 布线信号说明

本节简要说明了在“CLB.vcd”文件中创建的不同跟踪信号。请注意，某些信号或组不可用，具体取决于为其创建仿真的器件上可用的 CLB 类型。

备注

以下规则用于缩写和缩短跟踪信号列表：

1. “TILE#”是 Tile Design 实例的名称
2. “N”代表 0 到 7，用于所指示信号的多个实例
3. “X”表示 0 到 2，用于所指示信号的多个实例
4. “Y”表示 0 到 3，用于所指示信号的多个实例

表 3-3. SystemC 顶层跟踪信号

跟踪信号名称	说明
sc_top_clock	CLB 的时钟信号
sc_top_reset	CLB 的复位信号
sc_top_enable	CLB 使能信号；用于启用 CLB 子模块

表 3-4. 异步输出调节块跟踪信号（CLB 类型 2）

跟踪信号名称	说明
TILE#_AOC_N_clb_output	AOC 子模块的输出
TILE#_AOC_N_mux_ctrl [15:0]	AOC 多路复用器控制值；值与 CLB_OUTPUT_COND_CTRL_N 寄存器的格式相同
TILE#_AOC_N_release_signal	释放信号；根据在 TILE#_AOC_N_mux_ctrl 中选择的释放选项确定何时设置、清除或延迟 CLB 输出（默认释放信号不会改变 CLB 输出）
TILE#_AOC_N_gate_signal	栅极信号；该信号通过 AND、OR 或 XOR 运算与 CLB 输出进行逻辑组合，具体取决于在 TILE#_AOC_N_mux_ctrl 中选择的选项（默认栅极信号不会改变输出）
TILE#_AOC_N_mux_input_clb_tile_output	CLB 输出信号对应于该信号的同一 AOC 编号；例如，CLB 输出 0 是 AOC 0 的输入选项
TILE#_AOC_N_mux_input_boundary_input	CLB 边界输入信号对应于该信号的同一 AOC 编号；例如，CLB 边界输入 0 是 AOC 0 的输入选项

表 3-5. 边界跟踪信号

跟踪信号名称	说明
TILE#_BOUNDARY_CLB_outputN	CLB 边界输出信号；该信号从 CLB 外设路由到器件的其余部分；如果器件具有此类 CLB，则该信号已通过 AOC 子模块（有关更多详细信息，请参阅器件技术参考手册中的“CLB 逻辑块”部分）
TILE#_BOUNDARY_muxed_and_filtered_inputN	在通过任何同步、启用输入流水线滤波器或边缘滤波器后，CLB 模块的输入
TILE#_BOUNDARY_inputN_ctrl [31:0]	用于启用同步、输入流水线滤波器或边缘滤波器的控制值
TILE#_BOUNDARY_original_inputN	在进行同步、流水线或边沿滤波等任何修改之前的 CLB 模块输入
TILE#_BOUNDARY_clock	CLB 的时钟信号

表 3-6. 计数器块跟踪信号

跟踪信号名称	说明
TILE#_COUNTER_X_reconfig_pipeline_en	可重新配置的流水线的使能信号，用于对 HLC 和计数器子模块的操作进行流水线处理（这与流水线输入滤波器启用不同）
TILE#_COUNTER_X_counter_equals_match2	当计数器值等于匹配基准 2 值时，此信号为高电平；如果在串行器模式下启用匹配基准 2 抽头输出、则当所选的计数器位位置为高电平时，此信号为高电平
TILE#_COUNTER_X_counter_equals_match1	当计数器值等于匹配基准 1 值时，此信号为高电平；如果在串行器模式下启用匹配基准 1 抽头输出，则当所选的计数器位位置为高电平时，此信号为高电平
TILE#_COUNTER_X_counter_equals_zero	当计数器值等于零时，此信号为高电平
TILE#_COUNTER_X_match2_val [31:0]	匹配基准 2 的值；可通过 HLC 进行修改
TILE#_COUNTER_X_match1_val [31:0]	匹配参考 1 的值；可通过 HLC 进行修改

表 3-6. 计数器块跟踪信号 (continued)

跟踪信号名称	说明
TILE#_COUNTER_X_counter_output [31:0]	计数器本身的值
TILE#_COUNTER_X_hlc_match2_load_en	加载使能信号，用于确定何时从 HLC 子模块加载匹配基准 2 值；这与 TILE#_HLC_hlc_counterX_match2_load_en 信号匹配
TILE#_COUNTER_X_hlc_match1_load_en	加载使能信号，用于确定何时从 HLC 子模块加载匹配基准 1 值；这与 TILE#_HLC_hlc_counterX_match1_load_en 信号匹配
TILE#_COUNTER_X_hlc_counter_load_en	加载使能信号，用于确定何时使用来自 HLC 子模块的值加载计数器；这与 TILE#_HLC_hlc_counterX_load_en 信号匹配
TILE#_COUNTER_X_hlc_counter_load_val [31:0]	从 HLC 子模块加载的计数器值；此值加载到匹配基准 1、匹配基准 2 或计数器值，具体取决于是否设置了适当的 HLC 加载使能（即取决于 HLC 正在执行的指令）。这与 TILE#_HLC_counter_hlc_load_value 信号匹配
TILE#_COUNTER_X_match2_tap [4:0]	指定计数器的哪个位要针对匹配基准 2 进行抽取
TILE#_COUNTER_X_match2_tap_en	使能信号，它允许计数器抽取由 TILE#_COUNTER_X_match2_tap 指定的位；TILE#_COUNTER_X_counter_equals_match2 在设置适当的抽取位时为高电平，这会有效地将位位置从计数器抽取到匹配基准 2 输出
TILE#_COUNTER_X_match1_tap [4:0]	指定要为匹配基准 1 抽取计数器的哪个位
TILE#_COUNTER_X_match1_tap_en	使能信号，它允许计数器抽取由 TILE#_COUNTER_X_match1_tap 指定的位；在设置了适当的抽取位时，TILE#_COUNTER_X_counter_equals_match1 为高电平，这会有效地将位位置从计数器抽取到匹配基准 2 输出
TILE#_COUNTER_X_lfsr_en	线性反馈移位寄存器的使能信号；这将允许计数器子模块计算串行位流上的 CRC
TILE#_COUNTER_X_global_serializer_en	串行器的使能信号；启用时，计数器加载下一个 LFSR 串行值或相应的串行值（有关更多详细信息，请参阅器件技术参考手册）
TILE#_COUNTER_X_mode1	控制计数器的方向；设置为 1 时递增计数，设置为 0 时递减计数
TILE#_COUNTER_X_mode0	控制是否停止计数器；设置为 1 时启用计数
TILE#_COUNTER_X_global_reset	CLB 的复位信号
TILE#_COUNTER_X_add_or_shift_dir	如果计数器事件已启用且未针对加载事件进行配置，则当事件增加计数器值或将计数器值左移动时，该信号为高电平，否则设置为 0
TILE#_COUNTER_X_add_or_shift_on_event_en	如果计数器配置为在事件发生时增加或移位，则此信号设置为高电平，否则设置为 0
TILE#_COUNTER_X_add_or_shift_mode	如果计数器增加或减少，则此信号设置为高电平，否则设置为 0
TILE#_COUNTER_X_global_en	CLB 使能信号
TILE#_COUNTER_X_event_load_val [31:0]	当事件发生且事件操作配置为“load”时，会将此值加载到计数器
TILE#_COUNTER_X_event	当一个事件已经发生或正在发生时，此信号为高电平
TILE#_COUNTER_X_counter_reset	计数器子模块的复位信号
TILE#_COUNTER_X_clock	CLB 的时钟信号

表 3-7. 有限状态机阻止跟踪信号

跟踪信号名称	说明
TILE#_FSM_X_fsm_lut_output	FSM 查找表输出方程的输出
TILE#_FSM_X_fsm_s1_output	FSM 状态 1 方程的输出
TILE#_FSM_X_fsm_s0_output	FSM 状态 0 方程的输出
TILE#_FSM_X_LUT_output_equation [15:0]	表示 FSM 查找表输出方程的值
TILE#_FSM_X_state1_equation_output [15:0]	表示 FSM 状态 1 方程的值
TILE#_FSM_X_state0_equation_output [15:0]	表示 FSM 状态 0 方程的值
TILE#_FSM_X_extra_external_input_select1	选择外部输入 3 (e3) 的值来自何处；何时使用高电平 e3，何时使用低电平状态 1 (s1)

表 3-7. 有限状态机阻止跟踪信号 (continued)

跟踪信号名称	说明
TILE#_FSM_X_extra_external_input_select0	选择外部输入 2 (e2) 的值的来源；何时使用高电平 e2，何时使用低电平状态 0 (s0)
TILE#_FSM_X_extra_external_input1	FSM 子模块的额外外部输入 1 (xe1)；选择作为 xe1 的信号来自 CLB 内部并且只能用于查找表输出方程中
TILE#_FSM_X_extra_external_input0	FSM 子模块的额外外部输入 0 (xe0)；选择作为 xe0 的信号来自 CLB 内部并且只能用于查找表输出方程中
TILE#_FSM_X_external_input1	FSM 子模块的外部输入 1 (e1)；选择作为 e1 的信号来自 CLB 内部并且可用于状态 0、状态 1 和查找表输出方程中
TILE#_FSM_X_external_input0	FSM 子模块的外部输入 0 (e0)；选择作为 e0 的信号来自 CLB 内部并且可用于状态 0、状态 1 和查找表输出方程中
TILE#_FSM_X_global_reset	CLB 的复位信号
TILE#_FSM_X_global_en	CLB 使能信号
TILE#_FSM_X_clock	CLB 的时钟信号

表 3-8. 高级控制器块跟踪信号

跟踪信号名称	说明
TILE#_HLC_spi_export_receive_buffer [15:0]	表示存储在 SPI RX 缓冲区中的数据；这有助于使用 HLC 子模块通过 SPI 缓冲区验证数据导出
TILE#_HLC_fifo_overflow_signal	当为高电平时，此信号指示发生了 FIFO 上溢（在 FIFO 满时推）
TILE#_HLC_fifo_underflow_signal	当为高电平时，此信号指示发生了 FIFO 下溢（在 FIFO 为空时拉）
TILE#_HLC_fifo_write_pointer [1:0]	推 FIFO 写指针的当前值（零索引）
TILE#_HLC_fifo_read_pointer [1:0]	拉 FIFO 读取指针的当前值（零索引）
TILE#_HLC_push_fifo(Y) [31:0]	代表推 FIFO 中的当前值
TILE#_HLC_pull_fifo(Y) [31:0]	代表拉 FIFO 中的当前值
TILE#_HLC_program_current_instruction [11:0]	显示 HLC 正在执行当前指令的操作码的信号
TILE#_HLC_register(Y) [31:0]	表示 HLC 寄存器 R0 至 R3 的当前值
TILE#_HLC_program_interrupt_number [31:0]	指示 HLC 触发了哪个中断的信号；默认值为 0xFFFF，否则该值代表来自中断操作码最后 6 位的中断值
TILE#_HLC_program_interrupt_flag	当 HLC 执行的当前指令是中断时，此信号为高电平
TILE#_HLC_hlc_counterX_match2_load_en	使能信号；当 HLC 使用 MOV_T2 指令加载相应的计数器匹配基准 2 值时，此信号会变为高电平；这与相应的 TILE#_COUNTER_X_hlc_match2_load_en 信号匹配
TILE#_HLC_hlc_counterX_match1_load_en	使能信号；当 HLC 使用 MOV_T1 指令加载相应的计数器匹配基准 1 值时，此信号会变为高电平；这与相应的 TILE#_COUNTER_X_hlc_match1_load_en 信号匹配
TILE#_HLC_hlc_counterX_load_en	使能信号；当 HLC 使用 MOV 指令加载相应的计数器值时，此信号会变为高电平；这与相应的 TILE#_COUNTER_X_hlc_counter_load_en 信号匹配
TILE#_HLC_counter_hlc_load_value [31:0]	加载到匹配基准 1、匹配基准 2 的计数器值或取决于 HLC 正在执行的指令的计数器值；这与相应的 TILE#_COUNTER_X_hlc_counter_load_val 信号匹配
TILE#_HLC_spi_export_enable	此信号指示已启用通过 SPI RX 缓冲区导出数据
TILE#_HLC_reconfig_pipeline_enable	此信号指示启用了可重新配置的流水线模式（这会影响计数器和 HLC 子模块）
TILE#_HLC_alternate_event_clb_async_output(N)	此组信号表示来自 AOC 子模块的 CLB 的异步输出；这是用于 HLC 的替代事件之一
TILE#_HLC_alternate_event_clb_output(N)	此组信号表示来自 OUTLUT 子模块的 CLB 的输出；这是用于 HLC 的备用事件之一
TILE#_HLC_hlc_event_trigger(31..0)	表示 HLC 事件触发信号的一组信号；要查找哪个事件触发信号与哪个 HLC 事件触发信号值相对应，请查看器件技术参考手册（请注意，备用事件不是此组信号的一部分）

表 3-8. 高级控制器块跟踪信号 (continued)

跟踪信号名称	说明
TILE#_HLC_alternate_event_input_selectY	此选择指示备用事件组是否用于相应的 HLC 事件
TILE#_HLC_spi_shift_value [4:0]	用于确定 32 位 R0 寄存器中哪 16 位导出到 SPI RX 缓冲区的值（例如，值 1 选择 R0 寄存器的位 16:1）
TILE#_HLC_spi_event_trigger [4:0]	此信号确定哪个 HLC 事件导致将数据导出到 SPI RX 缓冲区（请注意，事件选项来自静态开关模块，如器件技术参考手册中所述）
TILE#_HLC_programY_event_source [31:0]	该值指示哪个事件触发信号用于相应的 HLC 事件；例如，TILE#_HLC_program0_event_source 的值为 1（备用事件输入选择设置为 0）会使用计数器 0 匹配参考 2 作为事件 0 的源，该事件会触发 HLC 程序 0
TILE#_HLC_counterX_value [31:0]	相应计数器的当前值；这指示执行 HLC 指令时正在使用的值，其中 C0、C1 或 C2 作为操作数
TILE#_HLC_program_global_load_en	CLB 使能信号
TILE#_HLC_program_reset	CLB 的复位信号
TILE#_HLC_program_clock	CLB 的时钟信号

表 3-9. 查找表块跟踪信号

跟踪信号名称	说明
TILE#_LUT_X_output	查找表子模块的输出
TILE#_LUT_X_output_equation [15:0]	代表查找表逻辑方程的值
TILE#_LUT_X_inputY	查找表子模块的输入

表 3-10. 输出查找表块跟踪信号

跟踪信号名称	说明
TILE#_OUTLUT_N_output	输出查找表子模块的输出
TILE#_OUTLUT_N_output_equation [7:0]	表示输出查找表逻辑方程的值
TILE#_OUTLUT_N_inputX	查找表子模块的输入

4 示例

CLB 工具在 C2000Ware 中附带提供了一些示例，用于展示如何配置 CLB 以实现各种用例。所有具有 CLB 的器件都提供了示例，但并非所有示例都适用于所有器件，因为新 CLB 类型中添加了旧版本上不提供的功能，例如异步输出调节块。下面的大多数示例都是针对 F28003x 器件介绍的，但可用于 F2838x 器件，示例 15 除外。

4.1 基础示例

这些示例的目的是展示每个 CLB Tile Design 内子模块的基本功能。每个示例都描述了少数的子模块以及如何使用它们的组合来实现简单的逻辑。如需更多示例，请访问节 2.1 中列出的链接。

为了更好地了解 Tile Design 配置以及子模块的连接方式，请使用节 3.4 中概述的步骤生成 CLB 方框图。

4.1.1 空 CLB 工程

该示例是一个空 CLB 工程，其中包含用于生成“.OUT”目标二进制文件的 post-build 步骤、仿真“.VCD”和 HTML 方框图。

4.1.2 示例 3 – PWM 生成

此示例将 CLB 逻辑块配置为辅助 PWM 发生器。该示例利用组合逻辑 (LUT)、状态机 (FSM)、计数器和高级控制器 (HLC) 来演示使用 CLB 的 PWM 输出生成功能。

PWM 发生器以 CLBCLK 频率运行。使用 FSM 来设置/清除 PWM。在发生 CMP 匹配事件时设置 PWM，该事件与 COUNTER_0 的 match2 相关。在发生零匹配事件 (Z) 时清除 PWM。此事件与 COUNTER_0 match1 输出相关。

PWM 寄存器配置为使用活动和影子寄存器，这是使用 HLC 块完成的。使用 HLC 在发生周期匹配事件 match1 时生成一个中断。在发生中断时，会将新的计数器匹配值加载到 HLC 寄存器 (R0) 中。然后将新计数器匹配值移动到 COUNTER_0 的 match2 寄存器中。这会更新 CMP 匹配值，进而更新正占空比的值。在此示例中，用户在正占空比的两个值之间进行交替。图 4-1 大体上显示了 PWM 发生器的功能。请注意下一个周期中的占空比是如何变化的。

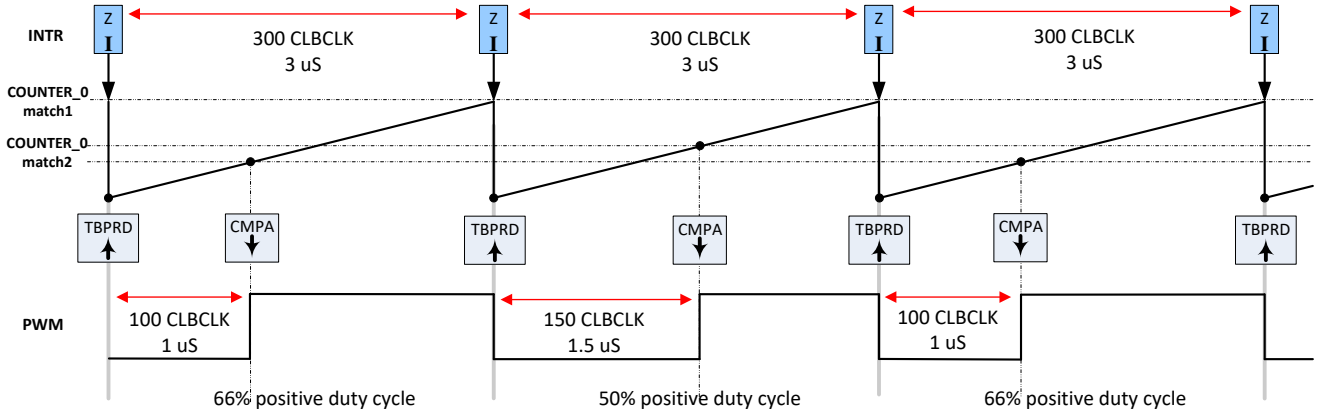

图 4-1. 示例 3：生成的 PWM 波形

CLB 逻辑块采用 PWM 使能信号作为输入，并向 CPU 生成一个中断。CLB 逻辑块配置为使用计数器进行向上计数，直到达到所需的周期和比较事件值。在输出“匹配 2”处计数器达到比较事件匹配值时，该输出被驱动为高电平并保持高电平，直到在输出“匹配 1”处达到周期匹配的计数器值或触发计数器复位。当发生周期事件或复位时，计数器重置为 0，输出被驱动为低电平，并且计数器开始向上计数。使用在 FSM 中输入的逻辑方程来配置该输出逻辑。在此示例中，周期为 300 个 CLBCLK 周期 (3μs)。比较事件在 100 个 CLBCLK 周期 (1 μ s) 或 150 个 CLBCLK 周期 (1.5 μ s) 时发生。

通过将 FSM 的输出馈送到 OUTLUT_4 中，可以查看 PWM 信号。为了在示波器上查看该输出，必须通过输出交叉开关将其传输到 GPIO 多路复用器。

要运行该示例，请执行以下过程：

1. 在 CCS v9.0 或更高版本中，依次点击“Project”-> “Import CCS Projects…”。
2. 导航到 CLB 工具示例目录。路径为：
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs 或
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs 或
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   在后续描述中，假设使用上述 C2000Ware 目录。
3. 选择工程“clb_ex3_auxiliary_pwm”并点击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开工程并打开文件“clb_ex3_aux_pwm.syscfg”。
5. 检查逻辑块的配置并观察 LUT4_0、COUNTER_0、FSM_0 中的逻辑表达式以及 HLC 和输出 LUT 的配置。
6. 在 CCS 菜单中，依次选择“Project”→“Build Project”。
7. 通过打开“Debug/syscfg/clb.html”文件，查看 CLB 逻辑块方框图
8. [可选] – 有关如何运行 CLB 仿真的说明，请参阅节 3.5.3。
9. 要查看 PWM 和中断信号，请设置示波器并当程序正在运行时监控以下引脚。下表显示了每个相应电路板要监测的引脚。

信号	F28379D LaunchPad	F280049 controlCARD	F28388D controlCARD
Interrupt	引脚 J4/40 上的 GPIO0	引脚 49 (GPIO0)	引脚 49 (GPIO0)
辅助 PWM	引脚 J4/34 上的 OutputXBAR1 信号	引脚 53 (OutputXBAR1)	引脚 53 (OutputXBAR1)

10. 打开 CCS“Expressions”窗口并添加程序变量 *dutyValue*。当程序正在运行时，您将注意到每次提供 CLB 中断服务信号时，CMPA 值都会在 100 和 150 个 CLBCLK 周期之间交替。信号应如上面的时序图所示。请注意，PWM 周期保持不变，但正占空比在 50% 和 66% 之间进行交替。可以在中断服务例程中修改 *dutyValue* 变量。

4.1.3 示例 7 – 状态机

使用 C2000 可配置逻辑模块进行设计 通过逐步介绍设计过程，说明了如何使用 CLB 来设计应用。本示例使用了 CLB 逻辑块内部的所有子模块，以实现完整的系统。

4.1.4 示例 13 – 推挽接口

在该示例中，显示了推挽接口的使用。使用了多个计数器子模块、HLC 子模块、FSM 子模块和输出子模块。推挽接口与 GP 寄存器一起使用，以更新计数器子模块的事件频率。

4.1.5 示例 14 – 多逻辑块

在该示例中，CLB 逻辑块的输出传递到另一个 CLB 逻辑块的输入。然后，第二个 CLB 逻辑块的输出被导出到一个 GPIO 中，从而展示如何串联使用两个 CLB 逻辑块。

4.1.6 示例 15 – 逻辑块间延迟

在该示例中，通过输入交叉开关和 CLB 交叉开关将一个 GPIO 的输出置于 CLB 逻辑块中。该逻辑块将此信号转发到下一个逻辑块。这次信号仅通过 CLB 交叉开关，而不通过输入交叉开关。这样做是为了展示当信号在逻辑块之间传递时，增加了延迟，并且延迟并不准确。用户应始终避免在逻辑块之间传递具有时序要求的信号。CLB 内部的计数器模块将以周期为单位对延迟量进行计数。

4.1.7 示例 16 - 胶合逻辑

在此示例中，将向用户分步说明如何将定制逻辑从 FPGA/CPLD 迁移到 C2000™ 微控制器。

4.1.8 示例 18 - AOC

在此示例中，使用异步输出调节块为 CLB 的输入信号添加异步与门。此模块仅适用于 CLB 类型 2 及更高版本。

4.1.9 示例 19 - AOC 释放控制

在此示例中，使用异步输出调节块对 CLB 的输入信号执行异步设置/释放。此模块仅适用于 CLB 类型 2 及更高版本。

4.1.10 示例 20 - CLB XBAR

在此示例中，使用 CLB INPUTXBAR 和 CLB OUTPUTXBAR 将 GPIO 的输入信号获取到 CLB 逻辑块中，并将逻辑块的输出信号获取到 GPIO 中。这些 XBAR 的可用性取决于器件。

4.2 入门示例

这些示例的复杂性稍高一些，可创建更有用的示例和 CLB 外设的实现。如需更多示例，请访问节 2.1 中列出的链接。

为了更好地了解 Tile Design 配置以及子模块的连接方式，请使用节 3.4 中概述的步骤生成 CLB 方框图。

4.2.1 示例 1 – 组合逻辑

该示例的目的是防止 同一对 PWM 同时出现高电平或低电平输出。PWM 模块 1 和 2 配置为基于固定频率向上计数模式生成相同的波形。PWM2 的时基与 PWM1 的时基同步，如图 4-2 所示。

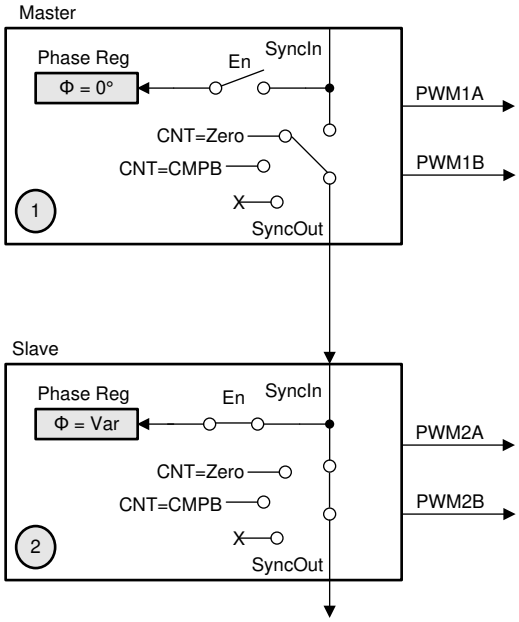

图 4-2. 示例 1：EPWM 同步

生成 PWM 波形，以故意迫使每个模块中的两个输出在不同的时间变为高电平和低电平，如图 4-3 所示。

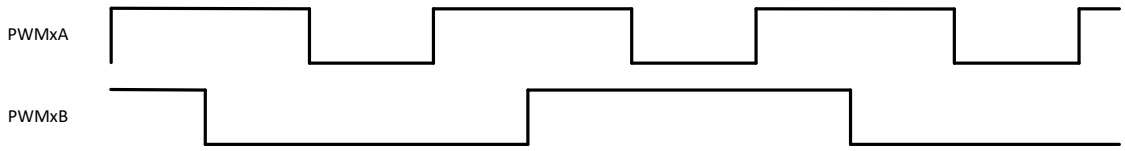

图 4-3. 示例 1：PWM 测试模式

目的是使用 CLB 修改这些波形，以避免同时为高电平或同时为低电平的情况。这展示了一个简单的组合逻辑示例。该逻辑以三种模式工作：正常、高电平有效和低电平有效。在正常模式下，PWM 信号未经修改地通过 CLB。在高电平有效模式下，该逻辑防止在 PWM 引脚上同时出现逻辑“1”输出。类似地，在低电平有效模式下，两个 PWM 引脚上一定不会同时出现逻辑“0”输出。例如，如果逻辑处于低电平有效模式，并且两个 PWM 信号都为低电平，则**两个** PWM 的输出将被强制为高电平。有关更多详细信息，请参阅表 4-2。可以使用一个 2 位字段来选择模式，如表 4-1 所示。

表 4-1. 示例 1：操作模式

模式名称	类型	[模式 1]	[模式 0]
M0	正常	0	0
M1	低电平有效	0	1
M2	高电平有效	1	0
M3	保留	1	1

图 4-4 显示了实现这些信号的逻辑电路。输出信号的名称后附加了“_m”，表示它们可能已由 CLB 修改。

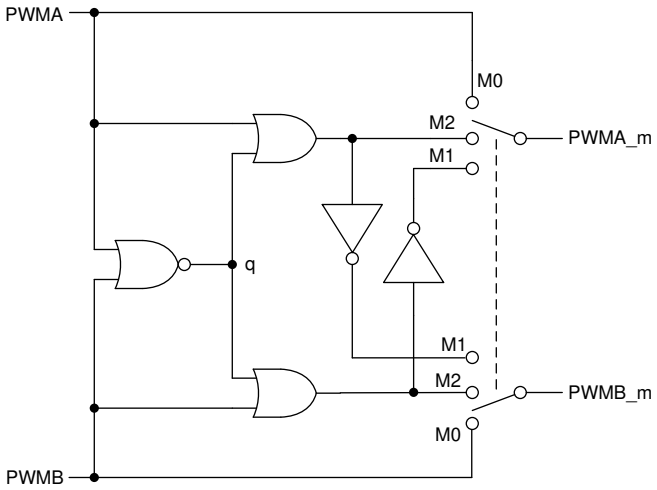

图 4-4. 示例 1：逻辑图

可以使用两个 4 输入 LUT 来实现上述逻辑：每个输出信号一个。因此，仅涉及一个 CLB 逻辑块的一小部分。在该示例中，CLB 仅修改了来自 PWM1 模块的信号。来自 PWM2 的信号直接传送到器件引脚以进行比较。图 4-5 显示了 PWM1 的输入和输出波形（模式 1 和模式 2）。此图像以绿色突出显示 CLB 逻辑强制输出为高电平或低电平的区域。

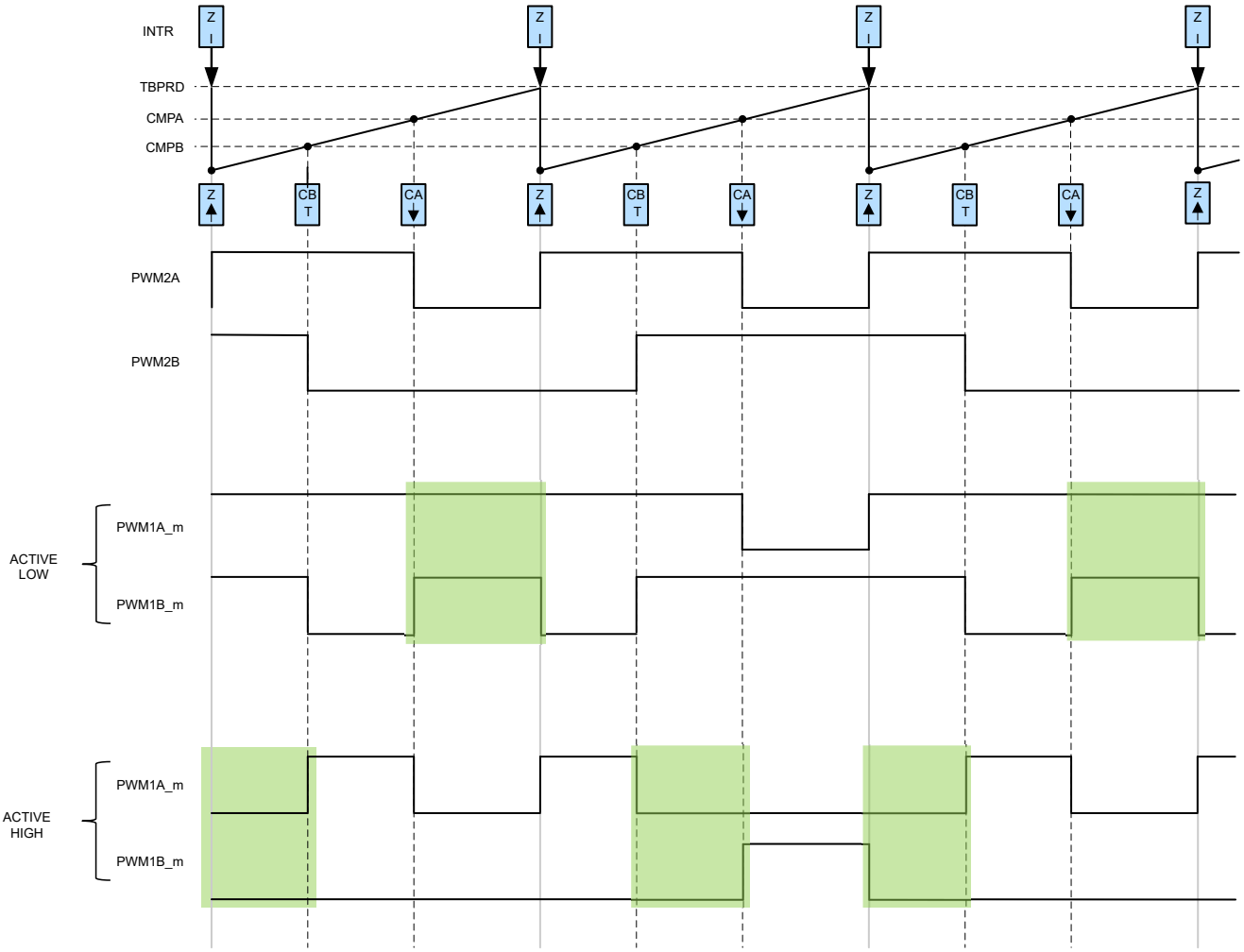

图 4-5. 示例 1：生成的 PWM

表 4-2. PWM 输出

工作模式	原始 PWM A 输出	原始 PWM B 输出	PWM A 输出	PWM B 输出
正常	0	0	0	0
	0	1	0	1
	1	0	1	0
	1	1	1	1
低电平有效	0	0	1	1
	0	1	0	1
	1	0	1	0
	1	1	1	1
高电平有效	0	0	0	0
	0	1	0	1
	1	0	1	0
	1	1	0	0

使用 4 路输入 LUT 0 和 1 来实现所需的逻辑。它们中的每一个都连接到两个 PWM 信号以及软件“mode”变量的两个 LSB，这两个 LSB 被写入到 GPREG 寄存器中。CLB 输出连接到 PWM1A 和 PWM1B 信号，然后这些信号分别传递到 GPIO 引脚 0 和 1。图 4-6 从概念上显示了与 CLB 逻辑块之间的往返连接。

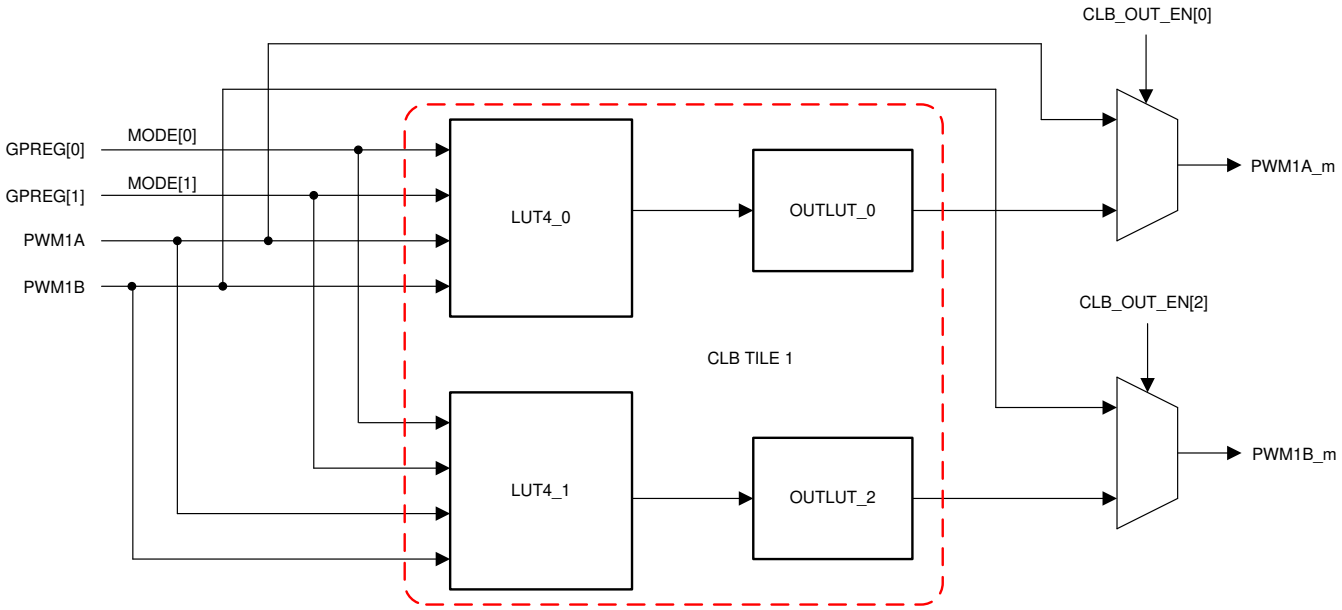

图 4-6. 示例 1：CLB 配置

要运行该示例，请执行以下过程：

1. 依次点击“Project”→“Import CCS Projects…”。
2. 导航到 CLB 工具示例目录。路径为：
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs 或
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs 或
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   在后续描述中，假设使用上述 C2000Ware 目录。
3. 选择工程“clb_ex1_combinatorial_logic”并点击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开工程“clb_ex1_combinatorial_logic”并打开文件“clb_ex1_combinatorial_logic.syscfg”。
5. 检查逻辑块的配置并观察 LUT4_0 和 LUT4_1 中的逻辑表达式以及输出 LUT 的配置。
6. 在 CCS 菜单中，依次选择“Project”→“Build Project”。
7. 监测引脚。
   a. 列出了用于观察 F28379D 的 PWM 的 Launchpad 引脚和 F28388D 的 Experimenter 套件引脚，但未列出其他器件的 Launchpad 引脚。有关可用引脚及其配置的更多信息，请参阅器件数据表
8. 打开 CCS“Expressions”窗口。
9. [可选] – 有关如何运行 CLB 仿真的说明，请参阅节 3.5.3。

如果在 F28379D LaunchPad 板上运行该程序，则可以分别在引脚 J4/40 和 J4/39 上监视 PWM 信号 1A 和 1B。设置示波器，以在该程序运行时在这些引脚上监视信号。

如果在安装了 F28388D controlCARD 的实验套件上运行该程序，则可以分别在引脚 49 和 51 上找到这些信号。

打开一个“CCS Expressions”窗口并添加程序变量“mode”。如果将模式设置为默认值 0，PWM 信号会在不进行任何修改的情况下通过 CLB。停止该程序并将模式更改为 1，然后重新启动该程序。信号应如上面的时序图所示。重复该过程将模式更改为 2，并验证信号是否如先前的时序图所示。

4.2.2 示例 2 – GPIO 输入滤波器

此示例演示了如何使用有限状态机 (FSM) 和计数器来实现简单的“干扰”滤波器，该滤波器可以 (例如) 应用于传入 GPIO 信号以消除不需要的短时脉冲。

图 4-7 大体上显示了干扰滤波器的功能。输入的数字信号以 CLB 时钟速率进行采样，计数器对输入为高电平或低电平的连续采样的数量进行计数。如果该数量等于或大于指定的采样窗口，则该滤波器输出的值与输入的值相同；否则该滤波器的输出保持不变。图 4-7 大体上显示了该滤波器的功能。

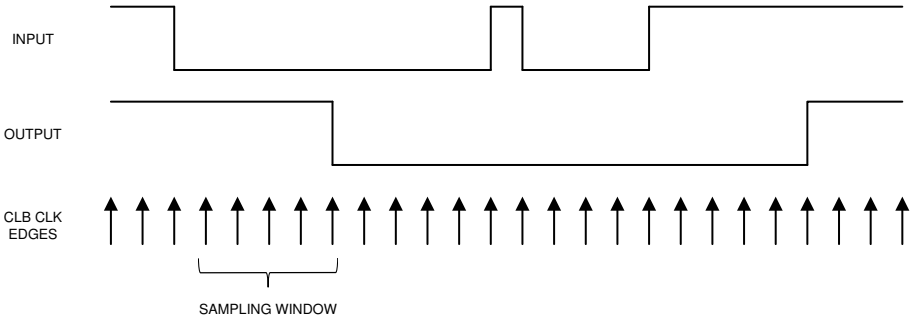

图 4-7. 示例 2：GPIO 干扰示例

CLB 配置使用一个 LUT4 对输入信号进行反相，并使用两个计数器对脉冲数进行计数：一个计数器用于高电平脉冲，另一个计数器用于低电平脉冲。当任何一个计数器达到采样窗口长度时，其“匹配 1”输出端都会出现一个脉冲。在该示例中，滤波器采样窗口长度设置为八。FSM 锁存该脉冲并实现一个简单的逻辑方程，以确定其“S0”状态输出所需的电平。使用一个输出 LUT 将 FWM 输出传送至外设信号多路复用器，以连接至 GPIO0。图 4-8 显示了 CLB 配置。

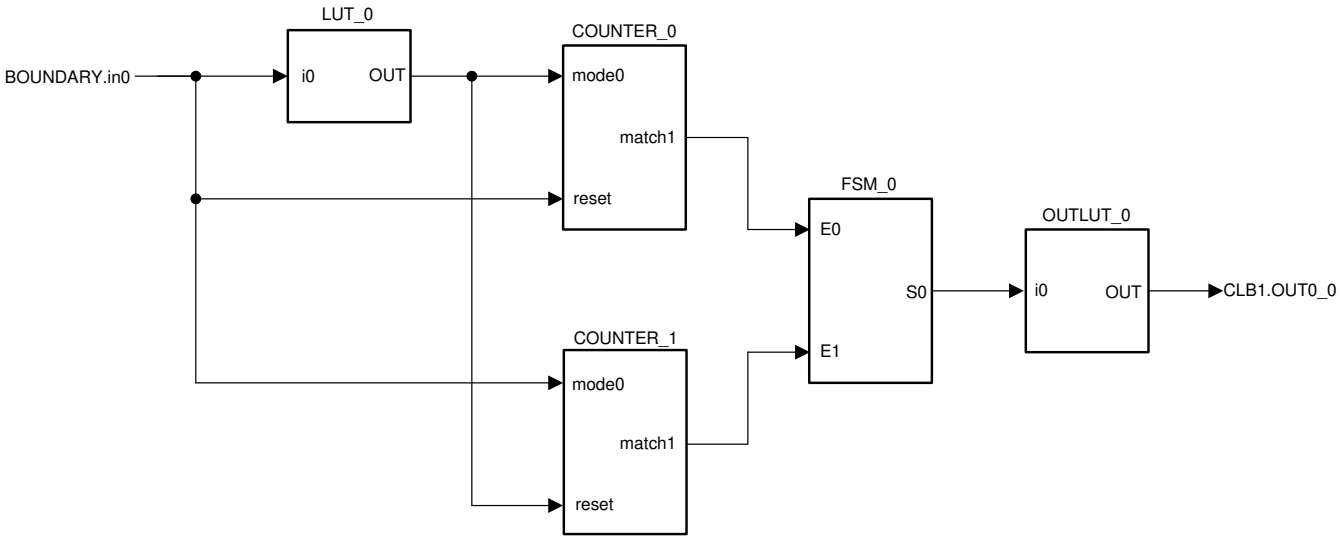

图 4-8. 示例 2：CLB 配置

示例代码对 ePWM1 模块进行配置，以生成测试激励。

要运行该示例，请执行以下过程：

1. 在 CCS v9.0 或更高版本中，依次点击“Project”->“Import CCS Projects…”。
2. 导航到 CLB 工具示例目录。路径为：
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs 或
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs 或
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   在后续描述中，假设使用上述 C2000Ware 目录。
3. 选择工程“glitch_filter”并点击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开工程“glitch_filter”并打开文件“tile.syscfg”。
5. 检查逻辑块的配置并查看子模块 LUT4_0、COUNTER_0、COUNTER_1 和 FSM_0 的设置。验证配置与上面示例说明中的配置是否相符。
6. 在 CCS 菜单中，依次选择“Project”→“Build Project”。

7. [可选] – 有关如何运行 CLB 仿真的说明，请参阅节 3.5.3。

如果在 F28379D LaunchPad 板上运行该程序，则可以分别在引脚 J4/40 和 J4/39 上监视 PWM 信号 1A 和 1B。设置示波器，以在该程序运行时在这些引脚上监视信号。如果在安装了 F280049 或 F28388D controlCARD 的实验套件上运行该程序，则可以分别在引脚 49 和 51 上找到这些信号。

打开一个“CCS Expressions”窗口并添加程序变量“cglitch”。运行该程序，同时观察 PWM 信号 1A 和 1B。暂停该程序并更改“cglitch”的值，然后重新启动该程序（如果将表达式窗口设置为“continuous refresh”，则执行该过程会更容易）。如果值为 7 或更小，则滤波器应消除干扰，因为其宽度小于采样窗口的宽度。当“cglitch”高于 7 时，两个输出上都应出现干扰。另请注意，与 PWM1B 上的边沿相比，PWM1A 上的边沿具有较小的延迟。这是所使用的滤波方法导致的结果。

图 4-9 显示了在干扰宽度低于和高于采样窗口设置 8 时输出引脚处的预期波形。

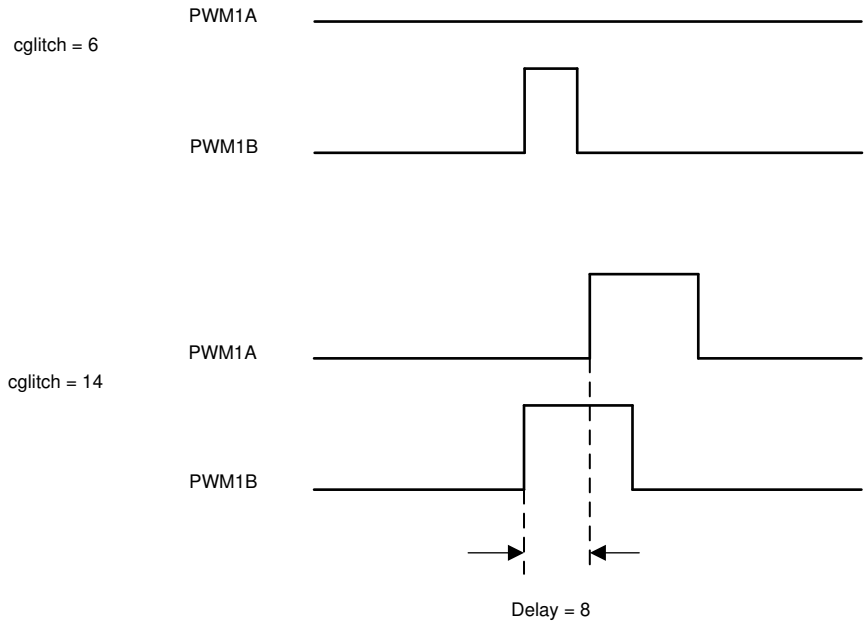

图 4-9. 示例 2：GPIO 干扰宽度

4.2.3 示例 4 – PWM 保护

此示例扩展示例 1 的功能，以确保高电平有效互补对 PWM 配置始终以最小死区值工作，而与生成 PWM 模块的配置方式无关。该示例说明了用于在四个 PWM 模块上实现 PWM 保护的四个单独的 PWM 逻辑块配置。PWM 模块 1 至 4 的输出分别由 CLB 逻辑块 1 至 4 进行操作。

通过程序变量“mode”来启用保护功能。当设置为 0（默认条件）时，PWM 信号未经修改地传递到输出引脚。当设置为 1 时，PWM 输出由 CLB 进行修改，以确保死区。

要运行该示例，请执行以下过程：

1. 在 CCS v9.0 或更高版本中，依次点击“Project”->“Import CCS Projects…”。
2. 导航到 CLB 工具示例目录。路径为：
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs 或
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs 或
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   在后续描述中，假设使用上述 C2000Ware 目录。
3. 选择工程“clb_ex4_pwm_protection”并点击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开工程“clb_ex4_pwm_protection”并打开文件“clb_ex_pwm_protection.syscfg”。
5. 在 CCS 菜单中，依次选择“Project”→“Build Project”。
6. 使用示波器观察每个 LaunchPad/集线站板上以下引脚上的 PWM 信号对。

表 4-3. 示例 4：信号连接

PWM	逻辑块	F28379D LaunchPad	F280049 LaunchPad	F28388 集线站
1A	1	J4/40	J8/60	49
1B	1	J4/39	J8/59	51
2A	2	J4/38	J8/56	53
2B	2	J4/37	J8/55	55
3A	3	J4/36	J4/36	50
3B	3	J4/35	J4/35	52
4A	4	J8/80	J8/58	54
4B	4	J8/79	J8/57	56

7. 打开一个“CCS Expressions”窗口并添加程序变量“mode”。
8. 运行该程序并验证每个 PWM 对之间是否没有死区。
9. 停止该程序，将模式更改为 1，然后再次运行该程序。您现在应该观察到每个 PWM 对之间的上升沿死区。死区时间由加载到两个 CLB 计数器中的 match_1 值设置，在本示例中已将其任意设置为 10。

4.2.4 示例 5 – 事件窗口

该示例使用 CLB 的计数器、FSM 和 HLC 子模块来实现事件计时功能，该功能可以检测中断服务程序是否花费过长的时间来响应中断。该示例将四个 PWM 模块配置为在向上计数模式下运行，并在发生计时器零匹配事件时生成从低到高变化的边沿。零匹配事件还触发 PWM ISR，就该示例而言，该 ISR 包含长度可变的虚拟有效负载。在 ISR 结束时，对 CLB GP 寄存器执行写操作，以指示 ISR 已结束。

CLB 模块在检测到 PWM 计时器零事件后启动一个计时器。计时器“匹配 2”计数被设置为相应 PWM ISR 的最大预期持续时间。如果在达到匹配 2 计数之前未对 GP 寄存器进行写入操作，则 HLC 会触发 CLB 中断。四个 PWM 模块和 CLB 逻辑块的配置类似。

图 4-10 概述了一个逻辑块的工作方式。上半部分显示了 PWM 模块的配置，该配置用于生成固定频率波形，每个计数器零匹配处具有上升沿，比较 A 匹配处具有下降沿。零匹配事件会生成一个 CPU 中断，目的是当 PWM ISR 未在指定时间内完成时触发 CLB 中断。

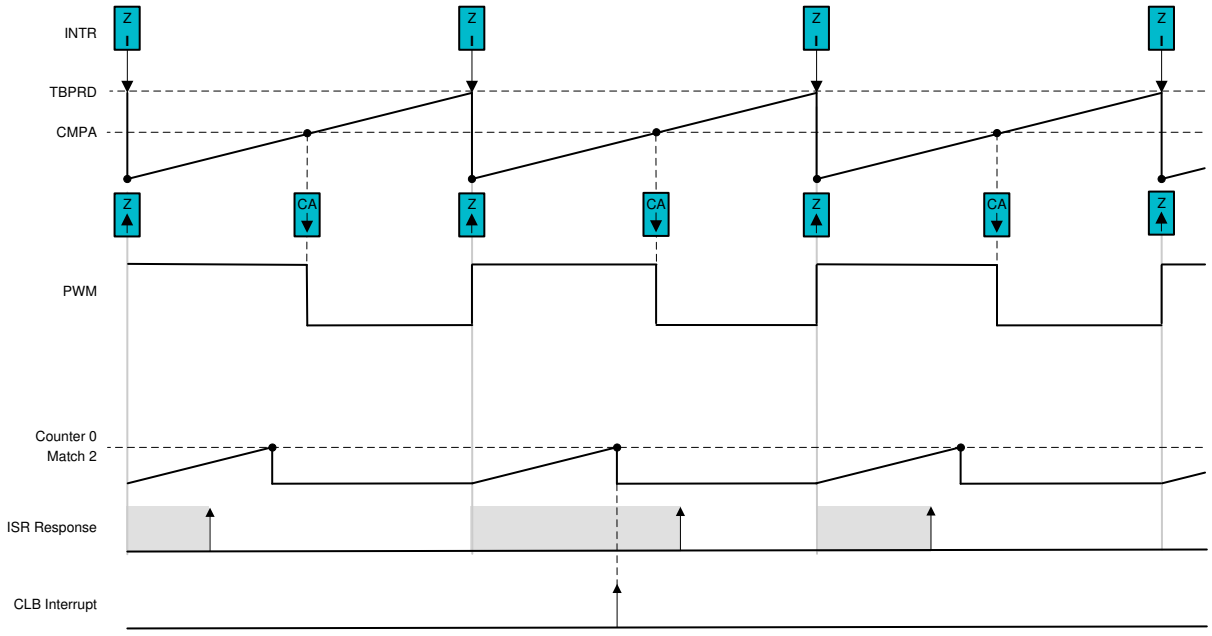

图 4-10. 示例 5：事件窗口配置

下半部分显示 CLB 计数器，该计数器在 PWM ISR 开始时开始计数。如果 ISR 在达到匹配 2 值之前没有响应，则会生成一个中断。CLB ISR 包含一条“ESTOP”指令，其作用类似于程序中的软件断点。

要运行该示例，请执行以下过程：

1. 在 CCS v9.0 或更高版本中，依次点击“Project”->“Import CCS Projects…”。
2. 导航到 CLB 工具示例目录。路径为：
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs 或
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs 或
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   在后续描述中，假设使用上述 C2000Ware 目录。
3. 选择工程“clb_ex5_event_window”并点击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开工程“clb_ex5_event_window”并打开文件“clb_ex5_event_window.syscfg”。
5. 在 CCS 菜单中，依次选择“Project”→“Build Project”。

打开 CCS“Expressions”窗口并添加四个程序变量：“payload_x”，其中“x”为 1 到 4。请注意，在程序开始时，所有有效负载变量均已设置为 45。有效负载在每个 PWM ISR 中实现为“for”循环，每次迭代需要 12 个周期，因此有效负载 45 对应于大约 540 个周期。

打开 .syscfg 文件并检查四个 CLB 模块的计数器 0 中的匹配 2 设置。请注意，所有计时器限制都设置为 3200。

使用默认有效负载运行程序并验证 CLB 中断是否不会触发。然后，停止该程序并增加任何有效负载。重新运行该程序并确定是否超出了任何 ISR 限制。请记住，由于 PWM 未同步，因此最坏情况的 ISR 延迟是所有有效负载的总和加上中断开销。

4.2.5 示例 6 – 信号生成和检查

该示例使用 CLB1 生成矩形波，并使用 CLB2 检查 CLB1 生成的矩形波是否未超过定义的占空比和周期限制。

CLB1：此示例使用 CLB 的计数器和 FSM 子模块来实现矩形脉冲发生器。计数器 0 根据用户编程的匹配 1 和匹配 2 值生成事件。匹配 2 值定义了所生成波形的周期，而（匹配 2 – 匹配 1）值用于确定开启时间。状态机使用

这些来自计数器的事件生成波形 – 在发生匹配 1 事件时设置输出，在发生匹配 2 事件时清除输出。因此状态位 S0 反映了生成的输出波形。该输出又从 CLB1 输出 4 引出，以通过 CLB 交叉开关将输出传递到 CLB2。In0 用作软件的波形生成使能。它也通过 CLB1 输出 5 传递到 CLB2。

CLB2：此示例使用 CLB 的 LUT、计数器、FSM、HLC 子模块在 CLB1 生成的输出上实现校验器。以下是到 CLB2 的信号连接。

CLB1 输出 4 → CLB 交叉开关 AUXSIG0 → CLB2 输入 1（通过全局多路复用器）

CLB1 输出 5 → CLB 交叉开关 AUXSIG1 → CLB2 输入 2（通过全局多路复用器）

计数器 0 在输入 1 上接收的信号开启时间期间进行计数。计数器 0 匹配 1 值设置为占空比的限制值。如果发生匹配 1 事件，则意味着开启时间已超过所需的值。

计数器 1 复位并在输入 1 上接收的信号的上升沿开始计数。计数器 1 匹配 1 值设置为周期的限制值。如果发生匹配 1 事件，则意味着周期已超过所需的值。

状态机 (FSM1 S0) 用于检测输入 1 上接收的信号的上升沿，并进而用作计数器 1 的复位。

每当发生上述任何计数器匹配 1 事件时，就会使用 HLC 向 CPU 生成一个中断 – 作为错误指示器。

图 4-11 概述了逻辑块的工作方式。匹配 1 事件会生成一个 CPU 中断，目的是在 CLB2 内部检测到错误条件时触发 CLB 中断。

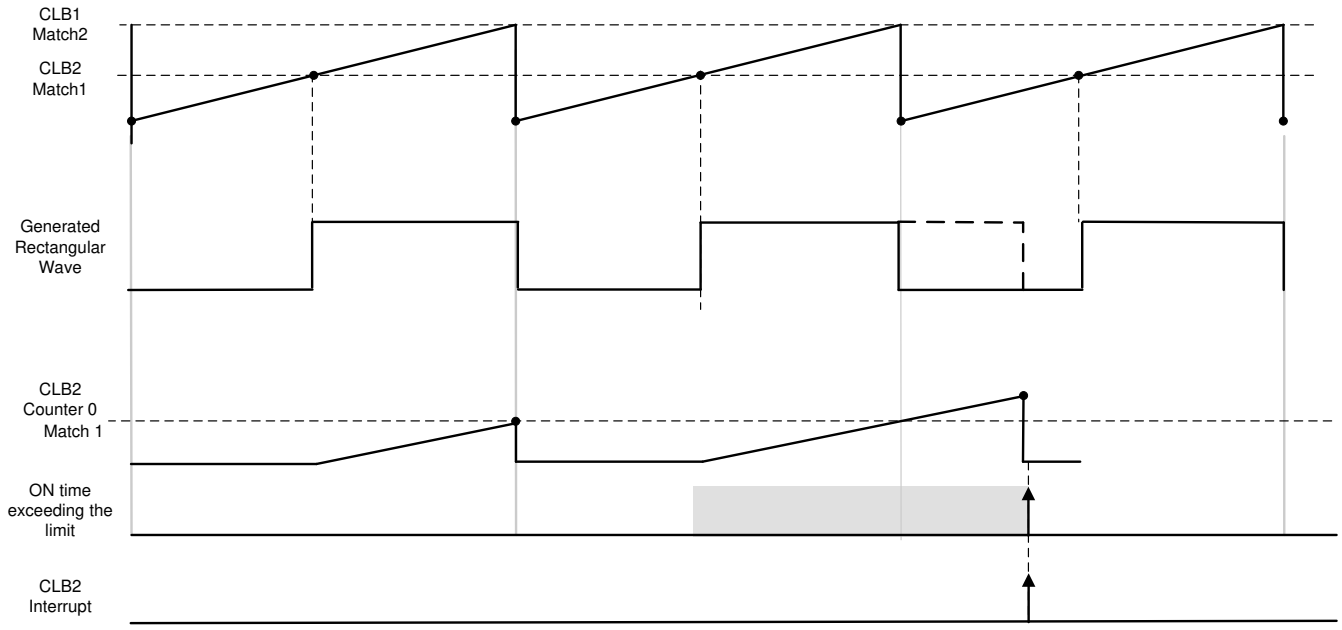

图 4-11. 示例 6：占空比超过预设值

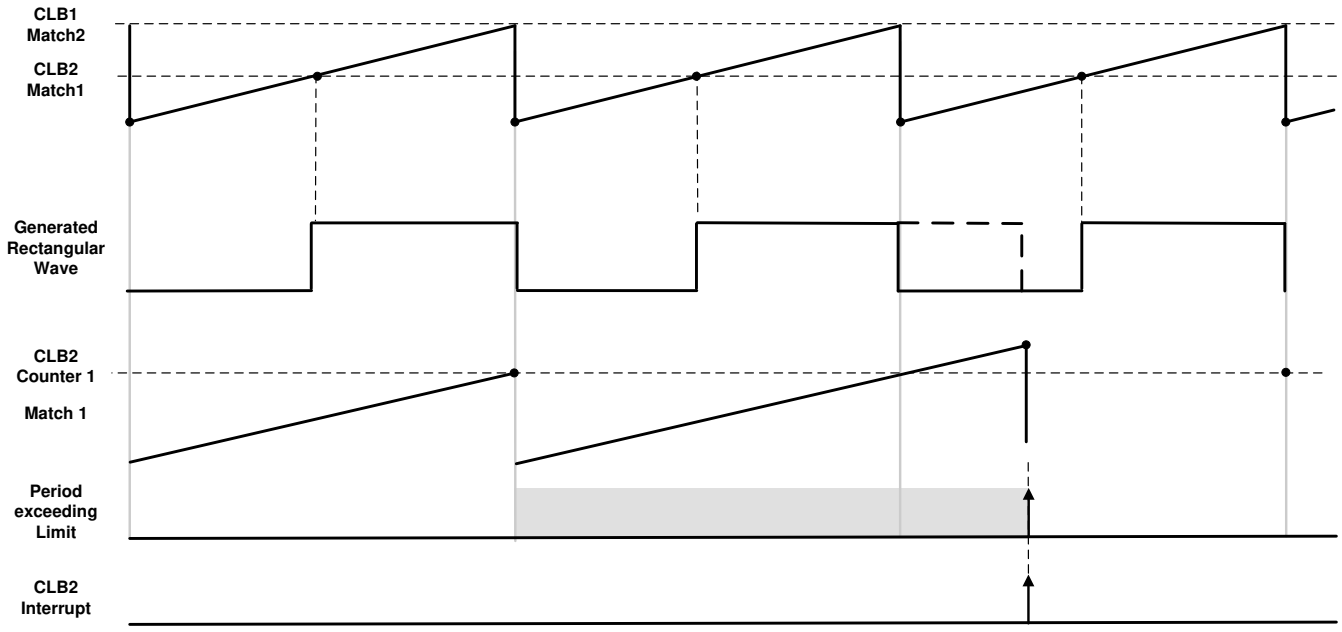

图 4-12. 示例 6：周期超过预设值

下半部分显示 CLB 计数器，该计数器在开启时间开始时开始计数。在第一个图描述了占空比检查，第二个图描述了周期检查。如果达到匹配 1 值，则在任一情况下都会生成一个中断。CLB ISR 包含一条“ESTOP”指令，其作用类似于程序中的软件断点。

要运行该示例，请执行以下过程：

1. 在 CCS v9.0 或更高版本中，依次点击“Project”->“Import CCS Projects…”。
2. 导航到 CLB 工具示例目录。路径为：
   a. [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs 或
   b. [C2000Ware]\driverlib\f28004x\examples\clb\ccs 或
   c. [C2000Ware]\driverlib\f2838x\examples\c28x\clb\ccs

   在后续描述中，假设使用上述 C2000Ware 目录。
3. 13.选择工程“clb_ex6_siggen”并点击“Finish”。
4. 14.在 CCS Project Explorer 窗口中，展开工程“clb_ex6_siggen”并打开文件“clb_ex6_siggen.syscfg”。
5. 在 CCS 菜单中，依次选择“Project”→“Build Project”。

在 CCS 窗口中打开 SysCfg 文件 (clb_ex6_siggen.syscfg) 并检查 CLB1 模块的计数器 0 中的匹配 1/2 设置。更改这些值，以更新生成的输出的占空比和周期。

检查 CLB2 模块中计数器 0/1 的匹配 1 设置。更改这些值，以更新在生成的输出上所检查的占空比和周期。

使用默认值运行程序并验证 CLB 中断是否不会触发。然后，更改值以生成错误（例如：将 CLB2 计数器 1 匹配 1 更改为 400）。重新构建并运行程序，以查看 CLB2 中断服务程序中的代码停止。

4.2.6 示例 8 – 外部与门

在该示例中，来自两个 GPIO 的两个外部信号通过输入交叉开关和 CLB 交叉开关传递到 CLB 逻辑块。在 CLB 模块内部，这两个信号进行与操作。然后，使用输出交叉开关将与门的输出导出到一个 GPIO 中。

4.2.7 示例 9 – 计时器

在该示例中，使用一个计数器模块来创建计时事件。显示了 GP 寄存器的使用。通过设置/清除 GP 寄存器中的位，可以启动、停止计时器或改变其方向。计时器事件（1 个时钟周期）的输出导出到一个 GPIO 中。使用 HLC 模块通过计时器事件生成中断。还会在 CLB ISR 内部反转一个 GPIO。CLB 寄存器间接访问来更新计时器的事件匹配值，使用有效计数器寄存器来修改计时器的频率。

4.2.8 示例 10 – 具有两种状态的计时器

在该示例中，按照与前一个示例相同的方式对计时器进行设置。区别在于使用了 FSM 子模块来反转 CLB 的输出，然后将其导出到一个 GPIO 中。FSM 模块用作 single-bit 存储块。按照与前一个示例相同的格式设置中断。通过比较 CLB 的输出和 ISR 中反转的 GPIO，可以看到 CLB 的中断延迟。

4.2.9 示例 11 – 中断标签

在该示例中，使用两个不同的匹配值来设置计时器。HLC 子模块使用这两个事件来生成中断。使用中断标签来区分由于 CLB 计数器的匹配 1 事件和 CLB 计数器的匹配 2 事件而生成的中断。

4.2.10 示例 12 – 输出相交

在该示例中，按照与 external_AND_gate 示例相同的方式对 CLB 模块进行设置。不过，由 ePWM1 的输出代替 X-BAR 的输出，导出到 GPIO 的输出。这是通过把 GPIO 配置成 EPWM1A 输出，然后启用内部交叉连接来实现的。

4.2.11 示例 17 - 单次 PWM 生成

此示例演示如何配置 CLB 逻辑块以充当单次 PWM 发生器。该示例使用组合逻辑 (LUT)、状态机 (FSM)、计数器和 HLC 来演示收到外部/软件触发时的单次 PWM 输出生成功能。

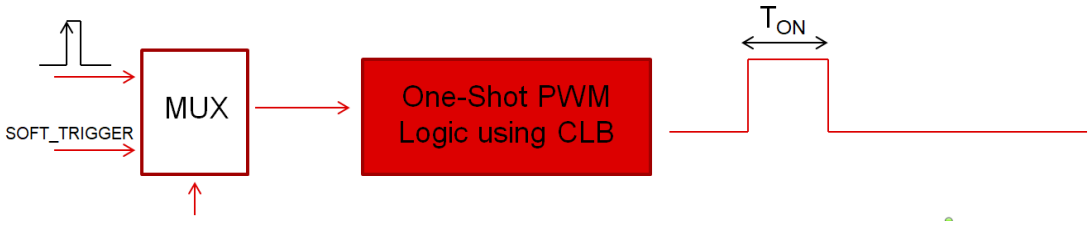

图 4-13. 示例 17：单次 PWM 输出

CLB 逻辑块配置为在接收到触发信号时模拟单次触发计时器。计时器从零开始计数，达到匹配值、然后停止计数，直到接收到下一个触发信号。输出在计数器正在计数时被驱动为高电平，在计数器达到最大值并停止计数时被驱动为低电平。上述逻辑是使用 LUT、FSM 和计数器实现的。另一个计数器用于确保后续系统仅响应上升沿事件，不响应输入电平。该示例还使用 HLC 子模块和 CLB 中断机制支持可变脉冲宽度。HLC 用于在每个第 3 个触发信号事件（由另一个计数器跟踪）之后生成中断，并且脉冲宽度由应用更新。示例中配置的输出脉冲范围是 0.2 µs - 0.8 µs，每个中断 ISR 的阶跃增加 50ns。PWM 寄存器配置为使用活动和影子寄存器，这也是使用 HLC 块完成的。

整体 CLB 配置如图 4-14 所示。

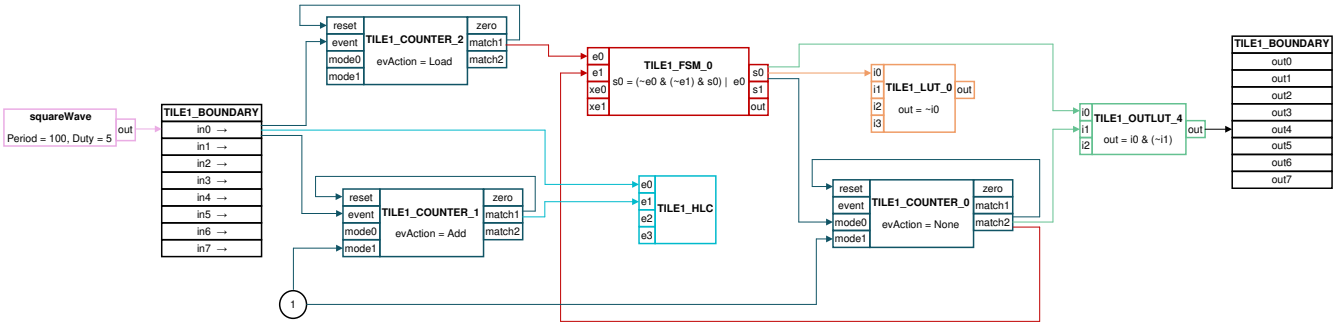

图 4-14. 示例 17：整体 CLB 配置

该示例支持两种配置模式：基于软件的触发和基于外部信号的触发。可以通过将 EXAMPLE_MODE 设置为 0/1 来选择所需的模式。对于基于软件的触发，您可以在 CCS“Expressions”窗口中手动将 SOFT_TRIGGER 从 0 更新为 1，并观察示波器上的单次脉冲输出。请注意，在将该变量设置为“1”之前，该变量已设置为“0”，因为 CLB 系统仅响应上升沿。而对于基于外部信号的触发，EPWM 模块配置为生成 1MHz 的触发信号，并且导通时间非常短（10% 占空比）。GPIO0 上生成的此 EPWM 信号在内部路由为 CLB 的触发信号输入，因此无需外部连接。

要运行该示例，请执行以下过程：

1. 在 CCS v9.0 或更高版本中，依次点击“Project”->“Import CCS Projects…”。
2. 导航到 CLB 工具示例目录。路径为：
   a. [C2000Ware]\driverlib\f28004x\examples\clb\ccs，在后续描述中，假设使用上述 C2000Ware 目录。
3. 选择工程“clb_ex17_one_shot_pwm”并点击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开工程“clb_ex17_one_shot_pwm”并打开文件“clb_ex17_one_shot_pwm.syscfg”。
5. 检查逻辑块的配置并观察 LUT_0 和 FSM_0、COUNTER_0、COUNTER_1、COUNTER_2、HLC 和输出 LUT 中的逻辑表达式。
6. 将 EXAMPLE_MODE 配置为 0/1，以在软件/外部触发信号模式下运行。
7. 在 CCS 菜单中依次选择“Project”->“Build Project”。
8. [可选] – 有关如何运行 CLB 仿真的说明，请参阅节 3.5.3。
9. 在 F28004x 控制卡上加载示例。
10. 如果选择了软件触发信号模式：
    a. 向 CCS“Expressions”窗口添加 SOFT_TRIGGER 变量。
    b. 将 GPIO2 连接到示波器并确保示波器处于“One-shot”模式而不是“FREE_RUN”模式。

c. 运行示例并在窗口中设置 SOFT_TRIGGER = 1，您应该能够在示波器上观察到单个脉冲。
d. 首先设置 SOFT_TRIGGER = 0，然后设置 SOFT_TRIGGER = 1 以生成下一个脉冲。
e. 每次重复上述步骤以提供上升沿触发信号。
f. 每三次触发后，脉冲将增加 50ns。

11. 如果选择了外部触发信号模式：
    a. 将 GPIO0（触发信号）和 GPIO2（输出）连接到示波器并将示波器配置为 FREE_RUN 模式。
    b. 在触发信号的每三个上升沿之后，您应该观察到输出脉冲宽度的线性变化。

4.2.12 示例 21 - 时钟预分频器和 NMI

在此示例中，使用 CLB 模块的时钟预分频器对 CLB 时钟进行分频，并将其用作 TILE 逻辑的输入。此外，还使用了 HLC 模块来生成 NMI 中断。此模块仅适用于 CLB 类型 2 及更高版本。

4.2.13 示例 22 - 串行器

在此示例中，在串行器模式下将 CLB 计数器用作移位寄存器。此模块仅适用于 CLB 类型 2 及更高版本。

4.2.14 示例 23 - LFSR

在此示例中，在线性反馈移位寄存器 (LFSR) 模式下使用 CLB 计数器模块。此模块仅适用于 CLB 类型 2 及更高版本。

4.2.15 示例 24 - 锁定输出屏蔽

在此示例中，使用 CLB 的锁定输出屏蔽来锁定选定的输出信号覆盖设置。此模块仅适用于 CLB 类型 3 及更高版本。

4.2.16 示例 25 - 输入流水线模式

在此示例中，启用 CLB 输入流水线模式以将输入信号延迟一个时钟周期。此模块仅适用于 CLB 类型 3 及更高版本。

4.2.17 示例 26 - 计时流水线模式

在此示例中，启用 CLB 流水线模式并影响 CLB 计数器和 HLC 的行为。此模块仅适用于 CLB 类型 3 及更高版本。

4.3 专家示例

这些示例的目的是提供 CLB 的更深入的用例。每个示例都介绍了如何配置子模块以达成特定的实现。如需更多示例，请访问节 2.1 中列出的链接。

为了更好地了解 Tile Design 配置以及子模块的连接方式，请使用节 3.4 中概述的步骤生成 CLB 方框图。

4.3.1 示例 27 - SPI 数据导出

在此示例中，使用了 CLB 的高速数据导出功能，并使用 SPI RX 缓冲区将其中一个 HLC 寄存器从 CLB 模块中导出。此模块仅适用于 CLB 类型 3 及更高版本。

4.3.2 示例 28 - SPI 数据导出 DMA

在此示例中，使用了 CLB 的高速数据导出功能，并使用 SPI RX 缓冲区将其中一个 HLC 寄存器从 CLB 模块中导出。在 SPI RX 缓冲区中接收到的数据通过 DMA 传输到存储器。此模块仅适用于 CLB 类型 3 及更高版本。

4.3.3 示例 29 - 时间戳

此示例显示如何为由 CLB 生成的中断添加时间戳。当 ePWM1 跳闸时生成中断。ePWM1 配置为被 TZ1 和 TZ2 中断，这两个信号都是单次跳闸源。

4.3.4 示例 30 - 循环冗余校验

此示例演示如何配置 CLB 逻辑块以充当循环冗余校验 (CRC) 模块。它模拟 CRC 过程以检查由于意外更改原始数据而导致的错误检测。此模块仅适用于 CLB 类型 2 及更高版本。

注：CRC 实现的位数限制为 31 位或更少。

4.3.5 CLB TDM 串行端口

文件：clb_ex31_tdm_serial_port.c

有关此示例的详细说明，请参阅：如何使用可配置逻辑块 (CLB) 实施定制串行接口应用手册 (SPRAD62)。

在此示例中，使用单个 CLB 逻辑块输入 TDM 流并生成 TDM 输出流。当接收到四个 32 位字时，CLB 会生成一个 CPU 中断。CPU 可以将四个 32 位值加载到 CLB FIFO 中以进行发送。CLB 和 CPU 配置为以其最大速度运行。

此示例仅在 CLB 类型 2 及更高版本的 C2000 MCU 器件上可用。

外部 连接
TDM 输入信号 GPIO 引脚 FSYNC_IN GPIO00 BCLK_IN GPIO01 DATA1_IN GPIO02

TDM 输出信号 GPIO 引脚 FSYNC_OUT GPIO04 BCLK_OUT GPIO05 DATA1_OUT GPIO06

4.3.6 CLB LED 驱动器

文件：clb_ex32_led_driver.c

有关此示例的详细说明，请参阅：如何使用可配置逻辑块 (CLB) 实施定制串行接口应用手册 (SPRAD62)。

在此示例中，使用了两个 CLB 逻辑块与 LP5891-Q1 LED 驱动器通信。使用一条 CCSI 总线通过 CCSI 总线协议发送数据，使用另一个逻辑块接收来自 CCSI 总线的数据。C28x CPU 通过硬件抽象层 (HAL) 与 CLB 逻辑通信。此示例还利用 PWM 生成所需的 CCSI 时钟，并使用计时器生成发送给 LED 驱动器的周期性同步事件。

此示例仅在 CLB 类型 2 及更高版本的 C2000 MCU 器件上可用。

外部 连接
CCSI 输入信号 GPIO 引脚 LED 驱动器 CLB_SIN_1 GPIO17 SOUT

CCSI 输出信号 GPIO 引脚 LED 驱动器 CLB_SOUT_1 GPIO08 SIN CLB_SCLK_1 GPIO01 SCLK

4.3.7 FPGA/CPLD 到 C2000 示例

如何将基于 FPGA 或 CPLD 的自定义逻辑转换为 C2000 器件上的 CLB 外设的其他示例，请参阅*如何将定制逻辑从 FPGA/CPLD 迁移到 C2000 微控制器应用报告*。

5 在现有 DriverLib 工程中启用 CLB 工具

执行以下步骤将 CLB 支持添加到现有的 C2000WARE DriverLib 工程：

1. 将 CLB 示例文件夹中的“empty.syscfg”文件（对于 F2837xD，路径为 <C2000WARE_INSTALL>\driverlib\f2837xd\examples\cpu1\clb\empty.syscfg）添加到工程中（通过将该文件复制到工程的目录中）。
2. CCS 会询问用户是否启用 SysConfig。接受并选择“Yes”（是）。

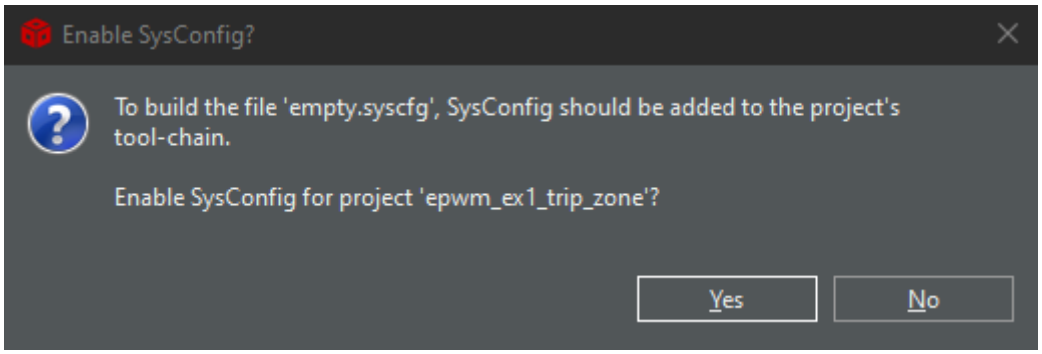

图 5-1. 启用 SysConfig

3. 打开“Project Properties”，然后依次打开“Resources”→“Linked Resources”。添加以下变量路径：
   a. C2000WARE_ROOT [PATH_TO_C2000WARE]
4. 在“Project Properties”窗口中，依次选择“Build”→“Steps”。
5. 将以下行添加到编译后处理步骤中，如图 5-2 中所示。
   a. ${NODE_TOOL} "${C2000WARE_ROOT}/dot_file_libraries/clbDotUtility.js" "${C2000WARE_ROOT}" "${BuildDirectory}/syscfg" "${BuildDirectory}/syscfg/clb.dot"

图 5-2. 编译后处理步骤

6. 接下来，选择“Resources”→“Linked Resources”，验证使用的 CLB_SYSCFG_ROOT 路径是否正确。然后依次选择“Build”→“SysConfig”→“Basic Options”并将正确的路径添加到根系统配置元数据列表中；仅在工程不包含 SysConfig 的情况下执行此操作：
   a. 确保“Linked Resources”中具有正确的 CLB_SYSCFG_ROOT 路径

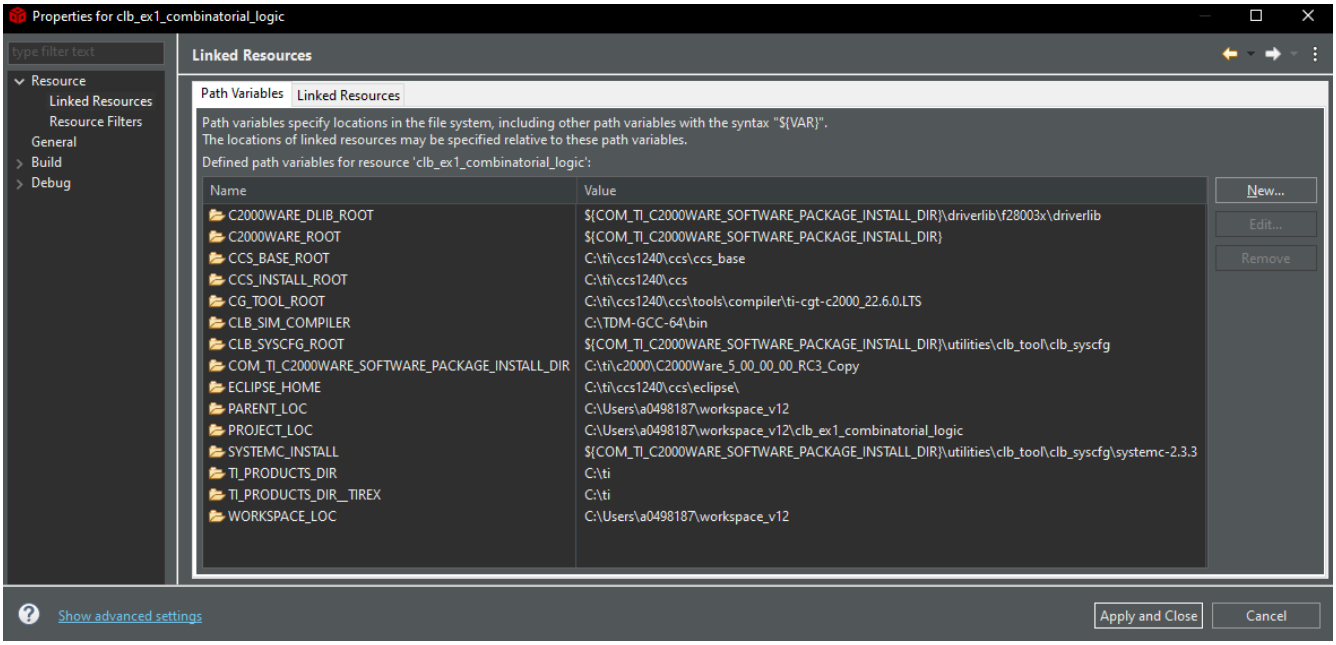

图 5-3. 用于启用 CLB 工具的链接资源

   b. ${CLB_SYSCFG_ROOT}/.metadata/product.json

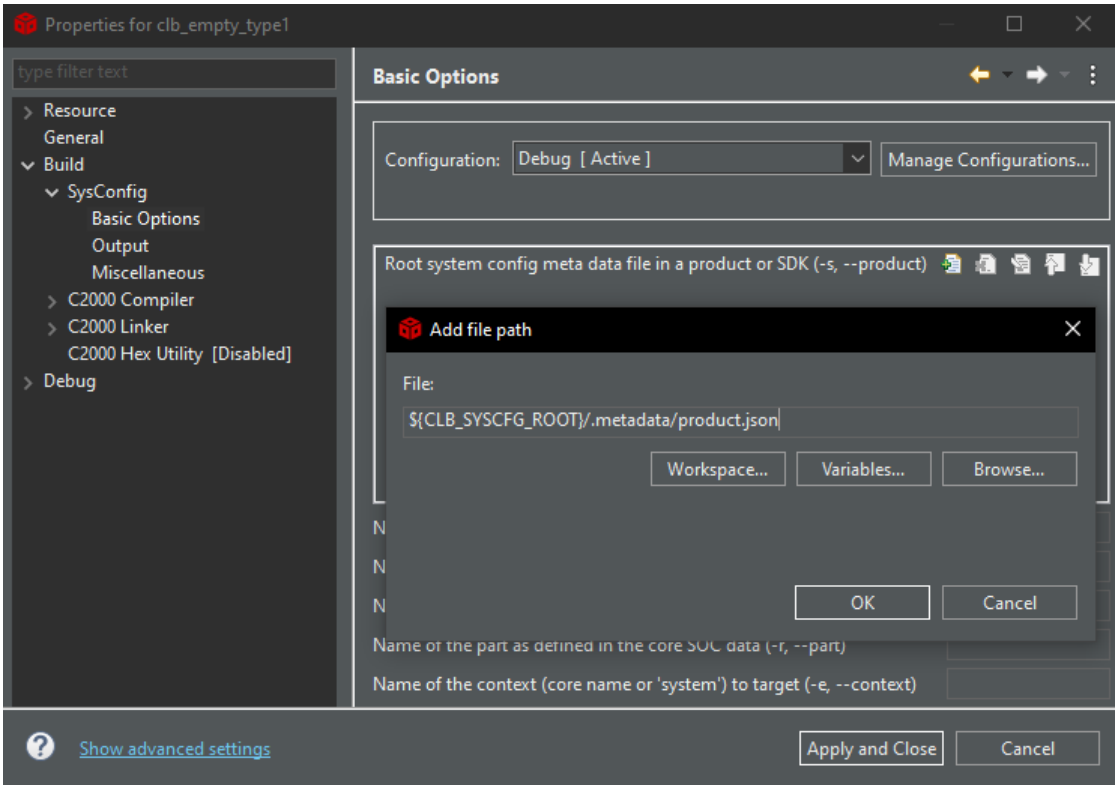

图 5-4. SysConfig SDK 路径

7. 最后，依次点击“Apply”和“Close”。

8. 构建工程后，CLB 工具生成的内容将显示在构建配置目录中。图 5-5 显示了在 epwm_ex1_trip_zone driverlib 示例中添加 CLB 支持后显示的示例内容。

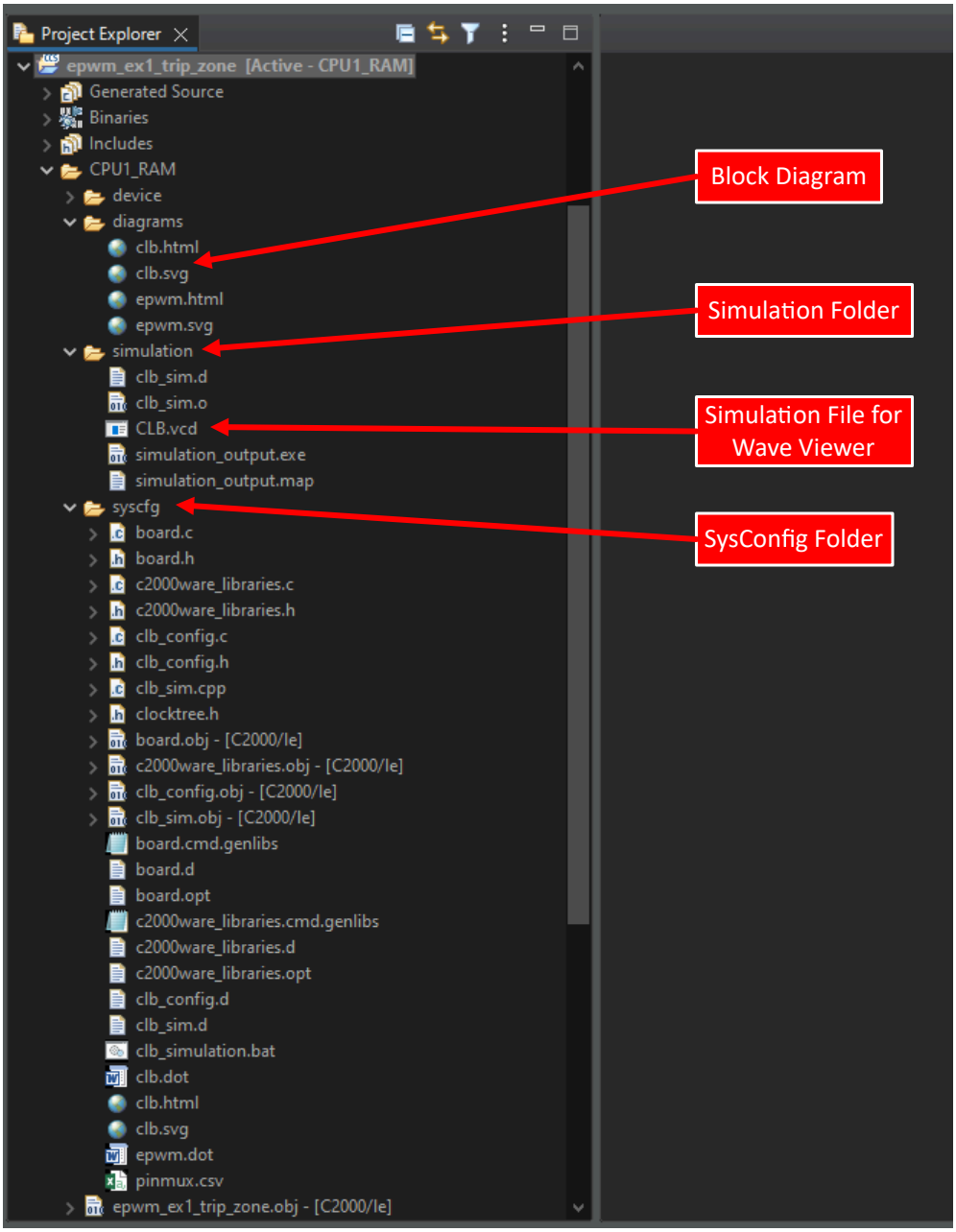

图 5-5. 具有 CLB 工具支持的 epwm_ex1_trip_zone

6 常见问题解答 (FAQ)

问题: 我的现有 CLB 工程与最新的 CLB 软件包不兼容。我在 CCS 问题窗口中看到以下构建错误：“No such resource: /TILE.syscfg.js”。

答案：SysConfig 中的最新更改现在需要修改和更新工程的 .syscfg 文件，以反映 TILE 资源在 CLB 包中的新位置。

备注

为了更新该文件，您将需要使用文本编辑器修改 .syscfg 文件。

更新以下代码行：

var TILE = scripting.addModule("/TILE");

替换为：

var TILE = scripting.addModule("/utilities/clb_tool/clb_syscfg/source/TILE");

问题: 在计算机上使用 CLB 工具有哪些要求？

答案：下面列出了最低要求

- C2000Ware 2.00.00.03
- CCSv9.1.0.00010
- Windows 64 位、Linux 或 Mac

问题: 控制台输出上有许多有关为 CLB 仿真生成 .vcd 文件的警告，我是否需要修复一些问题？

答案：可以忽略这些警告。它们是由于编译指令中使用的调试消息标志而生成的，因此它们不会影响功能。只需要注意错误。

```
../syscfg/clb_sim.cpp: In constructor 'Top::Top(const sc_core::sc_module_name&, sc_core::sc_trace_file*)':
../syscfg/clb_sim.cpp:462:25: warning: 'Top::TILE1_AOC_7_release_customWave' will be initialized after [-Wreorder]
     customWave          TILE1_AOC_7_release_customWave;
                         ^
../syscfg/clb_sim.cpp:554:50: warning:   base 'sc_core::sc_module' [-Wreorder]
         sc_top_clock("sc_top_clock", 20000, SC_PS)
                                                  ^
../syscfg/clb_sim.cpp:469:5: warning:   when initialized here [-Wreorder]
     Top(const sc_module_name &name, sc_trace_file *_tf):
     ^
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
         sc_top_clock("sc_top_clock", 20000, SC_PS)
                                                  ^
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../syscfg/clb_sim.cpp:554:50: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
```

图 **6-1.** 仿真警告示例

7 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision A (April 2020) to Revision B (July 2023)	Page
• 更新了整个文档中的表、图和交叉参考的编号格式。	3
• 基于 CLB 工具的主要更新对整本用户指南进行了更改	3

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