

## CLB 工具

本用户指南介绍可配置逻辑块 (CLB) 工具的结构和使用。可以在特定器件的技术参考手册 (TRM) 中找到有关 CLB 架构的信息。

假设读者已经熟悉 CLB 的架构和 CCS IDE。有关 CLB 的详细信息，请参阅特定器件的 TRM。

### 内容

|   |                                  |    |
|---|----------------------------------|----|
| 1 | 简介 .....                         | 2  |
| 2 | 开始使用 .....                       | 4  |
| 3 | 使用 CLB 工具 .....                  | 7  |
| 4 | CLB 仿真器 .....                    | 13 |
| 5 | 示例 .....                         | 15 |
| 6 | 在现有 DriverLib 项目中启用 CLB 工具 ..... | 28 |
| 7 | 常见问题解答 (FAQ) .....               | 33 |

### 附图目录

|    |                                |    |
|----|--------------------------------|----|
| 1  | CLB 工具项目结构 .....               | 3  |
| 2  | CLB 工具构建过程 .....               | 4  |
| 3  | TDM 编译器安装向导 .....              | 5  |
| 4  | TDM 编译器 64 位安装 .....           | 5  |
| 5  | TDM 编译器路径 .....                | 6  |
| 6  | 导入 CCS Eclipse 项目 .....        | 7  |
| 7  | 链接资源 .....                     | 8  |
| 8  | CLB 工具 SysConfig 屏幕 .....      | 9  |
| 9  | “BOUNDARY”输入选项 .....           | 10 |
| 10 | 计数器选项 .....                    | 10 |
| 11 | 方程警告 .....                     | 11 |
| 12 | CLB 工具生成的文件 .....              | 11 |
| 13 | “clb.h”头文件示例 .....             | 12 |
| 14 | HLC 配置示例 .....                 | 13 |
| 15 | 静态选项 .....                     | 13 |
| 16 | “BOUNDARY”输入 IN0 至 IN7 .....   | 14 |
| 17 | “BOUNDARY”输入“squareWave” ..... | 14 |
| 18 | “BOUNDARY”输入“Custom” .....     | 14 |
| 19 | CLB 仿真示例 .....                 | 15 |
| 20 | 示例 9: EPWM 同步 .....            | 17 |
| 21 | 示例 10: PWM 测试信号 .....          | 17 |
| 22 | 示例 1: 逻辑图 .....                | 18 |
| 23 | 示例 1: 生成的 PWM .....            | 19 |
| 24 | 示例 1: CLB 配置 .....             | 20 |
| 25 | 示例 2: GPIO 干扰示例 .....          | 21 |
| 26 | 示例 2: CLB 配置 .....             | 21 |
| 27 | 示例 2: GPIO 干扰宽度 .....          | 22 |

|    |                                      |    |
|----|--------------------------------------|----|
| 28 | 示例 3: 生成的 PWM 波形.....                | 23 |
| 29 | 示例 5: 事件窗口配置 .....                   | 25 |
| 30 | 示例 6: 有效信号时间超过预设值.....               | 27 |
| 31 | 示例 6: 周期超过预设值 .....                  | 27 |
| 32 | 启用 SysConfig .....                   | 28 |
| 33 | post-build 步骤 .....                  | 30 |
| 34 | SysConfig SDK 路径 .....               | 31 |
| 35 | 具有 CLB 工具支持的 epwm_ex1_trip_zone..... | 32 |

## 附表目录

|   |                  |    |
|---|------------------|----|
| 1 | 支持的逻辑运算 .....    | 11 |
| 2 | 示例 1: 工作模式 ..... | 17 |
| 3 | 示例 4: 信号连接.....  | 24 |

## 商标

C2000, Code Composer Studio are trademarks of Texas Instruments.  
All other trademarks are the property of their respective owners.

## 1 简介

### 1.1 CLB 工具概述

CLB 是集成到某些 C2000™器件中的硬件模块。CLB 包含一组可配置的模块并且能够内部互连，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。例如，可以对 CLB 进行配置，以增强现有器件外设的功能或创建新的外设功能。使用软件实用程序（此处称为“CLB 工具”）来配置 CLB。

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。

该工具使用 Code Composer Studio™(CCS) 的“SysConfig”图形用户界面 (GUI)。该工具包含少量示例，旨在帮助用户 探索 工具的功能并创建自己的项目。

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。该工具不对 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进行配置。这些寄存器的配置必须单独完成，这是用户的责任。

### 1.2 CLB 配置过程概述

CLB 工具基于 CCS 中的“SysConfig”工具。该工具与作为 C2000Ware 下载的一部分所提供的文件结合使用，足以配置 CLB。为了进行设计仿真，必须安装大量的第三方工具，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4 节。

CLB 工具生成一个“.dot”文件，该文件以图表的形式显示子模块互连，可用于验证设计。在提供的示例中，使用 node.js 和 JavaScript 库通过执行 CCS post-build 步骤将该文件转换为 HTML 格式。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。所有这些步骤都是使用 CCS 的 post-build 步骤自动完成的。

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。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”文件中的功能。图 1 显示了 CLB 工具的输出以及 post-build 步骤。

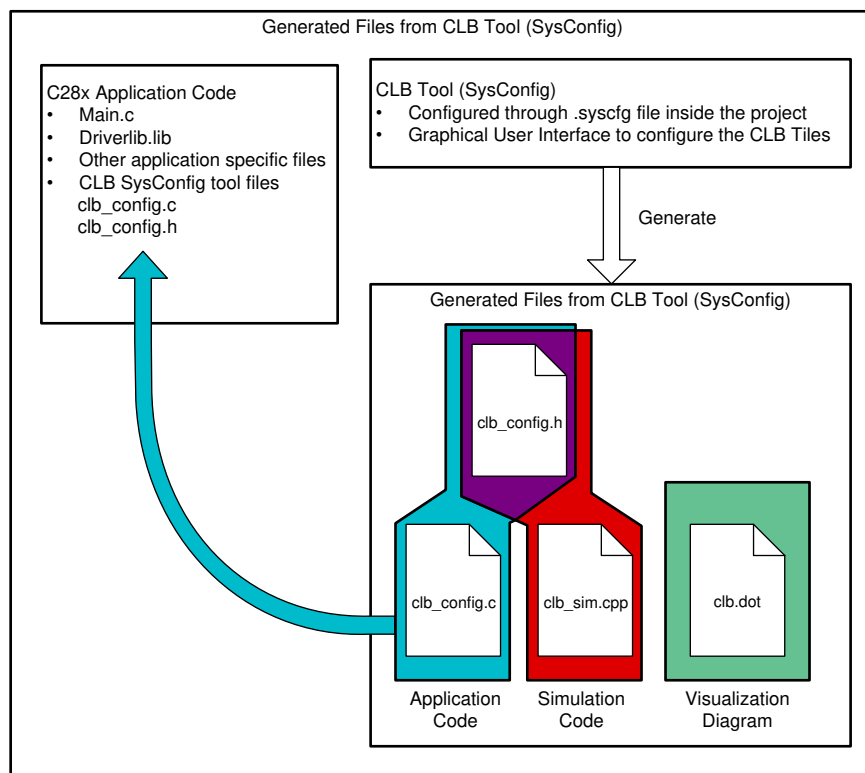


图 1. CLB 工具项目结构

在典型情况下，用户从所需 CLB 逻辑功能的说明开始。这可以采用逻辑电路图、时序信息、书面说明、VHDL 代码或其他某种形式。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。

该说明可能包含一组时序图，在这种情况下，用户可以（可选）决定执行 CLB 配置的仿真，以确保行为符合预期。该步骤包括定义一组输入测试激励，以及构建一个仿真项目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，用户将修改 SysConfig 设置并重复仿真。

从仿真中获得正确的波形之后，用户可以按照常规方式将设计下载到器件中。

在 C28x 器件支持 CLB SysConfig 的 CCS 项目中，创建 CLB 逻辑块配置的 HTML 方框图的步骤和“.vcd”仿真波形的生成是自动执行的。在用户构建 CCS 项目时，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。post-build 步骤使用 GCC 编译器编译生成的仿真文件“clb\_sim.cpp”和“clb\_config.h”以及 CLB 仿真模型。该步骤的输出是“.exe”文件（“simulation\_output.exe”）。接下来，在 post-build 步骤中执行“.exe”文件，从而生成“CLB.vcd”。可以使用外部图形查看器来查看该文件。

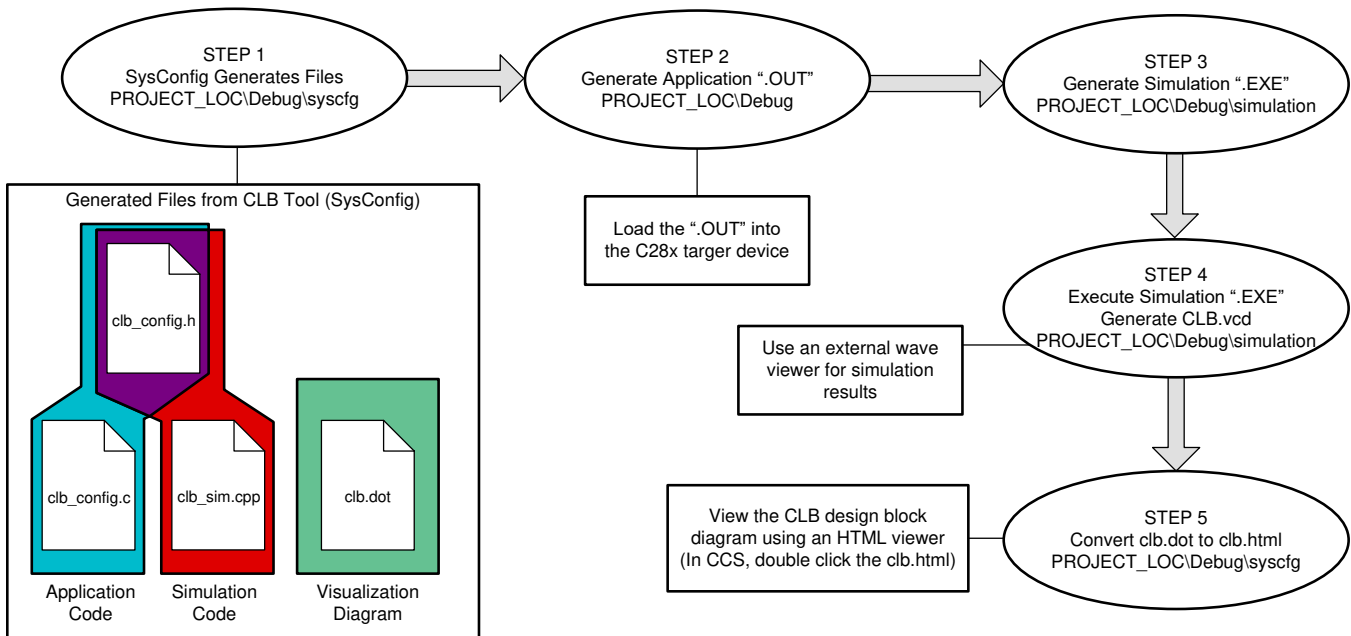


图 2. CLB 工具构建过程

## 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 简介

为了使用该工具，必须安装 Code Composer Studio (CCS) 版本 9.0 或更高版本。早期版本的 CCS 不包含 CLB 配置所必需的 SysConfig 实用程序。如需更多信息和下载“Code Composer Studio”，请访问：<http://www.ti.com.cn/tool/cn/ccstudio-c2000>。

用户可以使用上述工具来配置 CLB。不过，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)

## 2.2 安装

### 2.2.1 GNU 编译器

1. 通过以下链接下载“tdm-gcc”：<http://sourceforge.net/projects/tdm-gcc/files/TDM-GCC%20Installer/tdm-gcc-webdl.exe/download>。
2. 打开下载的文件以安装编译器。
3. 从安装向导中选择“Create”。

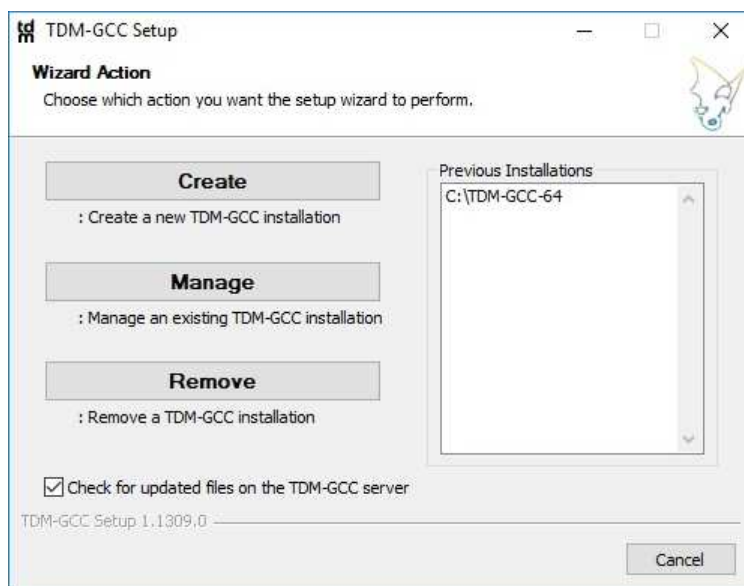


图 3. TDM 编译器安装向导

4. 选择 64 位安装并单击“Next”。

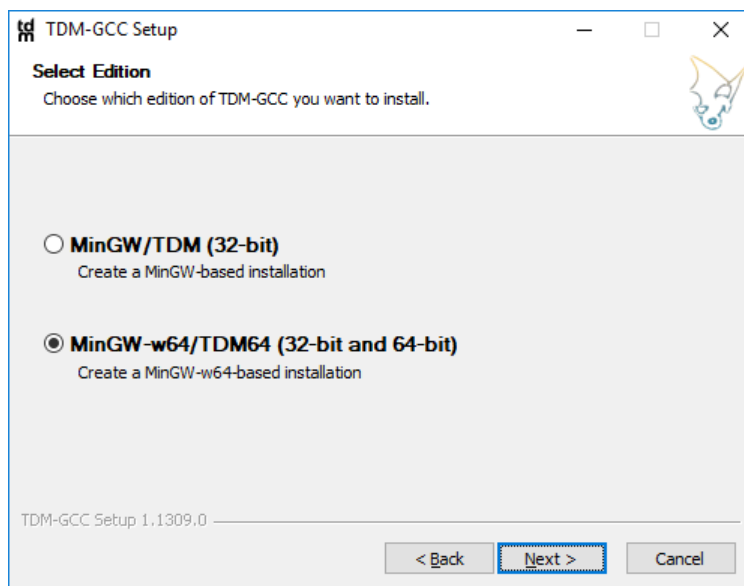


图 4. TDM 编译器 64 位安装

5. 选择 C:\TDM-GCC-64 作为安装目录并单击“Next”。

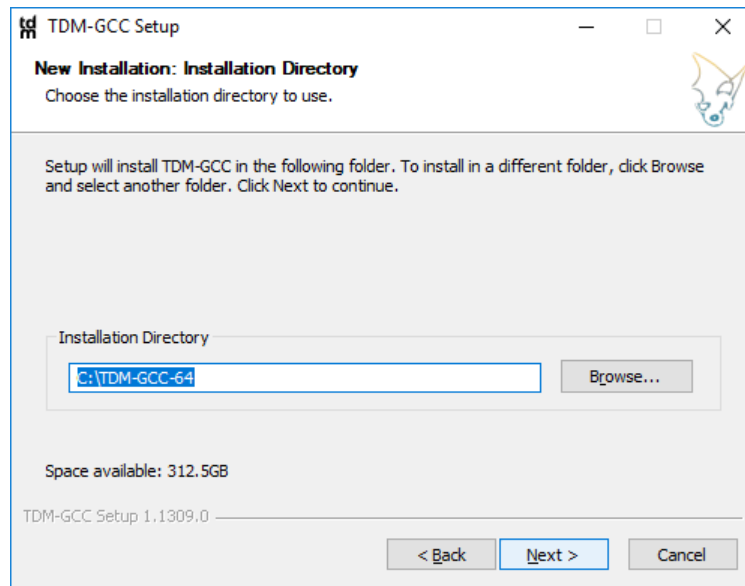


图 5. TDM 编译器路径

6. 根据您的地理位置完成安装过程的其余步骤。

### 2.2.2 安装仿真查看器

1. 通过以下链接下载波形查看器 GTKWave: <https://sourceforge.net/projects/gtkwave/files/>。
2. 下载安装 window 正确版本的本地二进制文件（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。

### 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 项目

<C2000WARE\_INSTALL>/driverlib/<device>/examples 中提供了基于 Driverlib 且支持 CLB 的示例项目。

例如，对于 F2837xD 器件，空 CLB 项目（以及其他 CLB 示例项目）的路径是 <C2000WARE\_INSTALL>/driverlib/f2837xd/examples/cpu1/clb。

1. 在 CCS 菜单中，依次单击“Project”->“Import CCS Projects...”。
2. 在“Select search-directory”中输入 CLB 示例项目的路径。
3. 单击“Refresh”。
4. 选择“clb\_empty”项目。
5. 选中“Copy projects into workspace”。
6. 单击“Finish”。

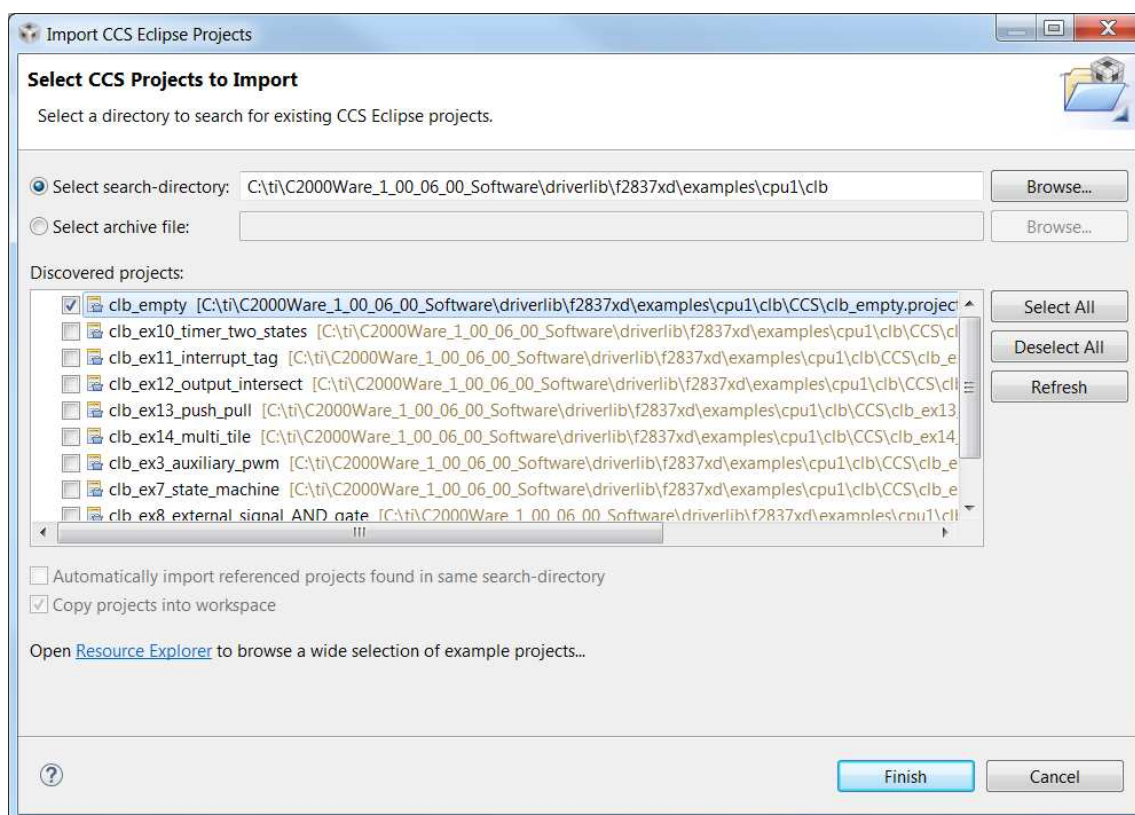


图 6. 导入 CCS Eclipse 项目

## 3.2 更新变量路径

上面导入的空 CLB 项目不仅能够为 C28x 目标生成“.OUT”文件，而且能够生成设计的仿真文件和 HTML 方框图。

上面下载的 GCC 编译器的路径可能与项目中指定的路径不同。要对此进行仔细检查，请执行以下操作：

1. 右键单击项目并选择“Project Properties”。
2. 在“Resources”下，选择“Linked Resources”。
3. 进行检查，以确保下面的所有路径都是正确的：
  - a. CLB\_SYSCFG\_ROOT（所有 CLB 组件都相对于该路径）
  - b. CLB\_SIM\_COMPILER（对仿真很重要）
  - c. SYSTEMC\_INSTALL（对仿真很重要）

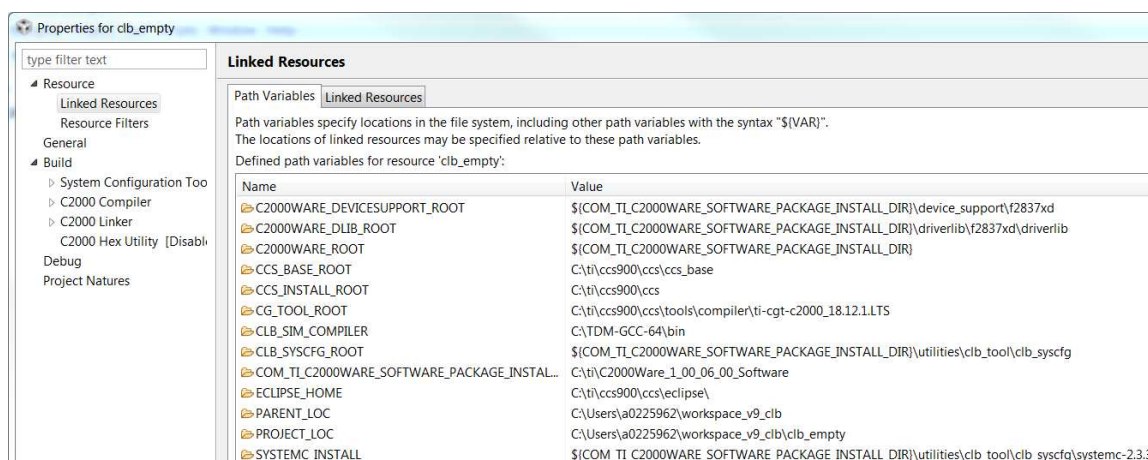


图 7. 链接资源

4. 如果名称左侧的图标不是文件夹，而是一个感叹号，则您的系统上不存在该路径，您必须手动选择正确的路径。



### 3.3 配置 CLB 逻辑块

要打开配置工具，请在 CCS Project Explorer 窗口中双击您要编辑的“.syscfg”文件。图 8 显示了相关屏幕。

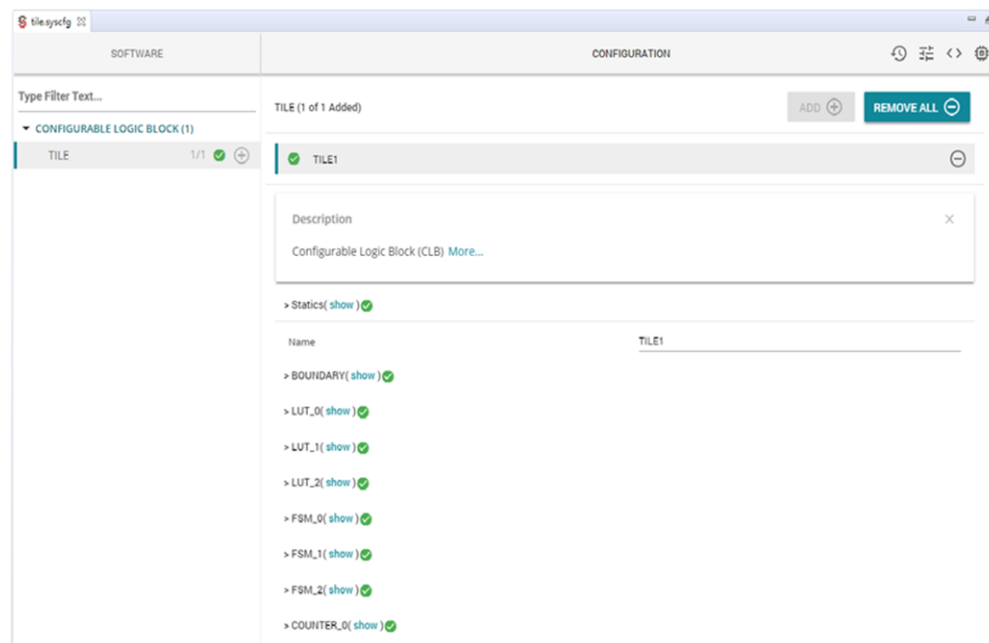


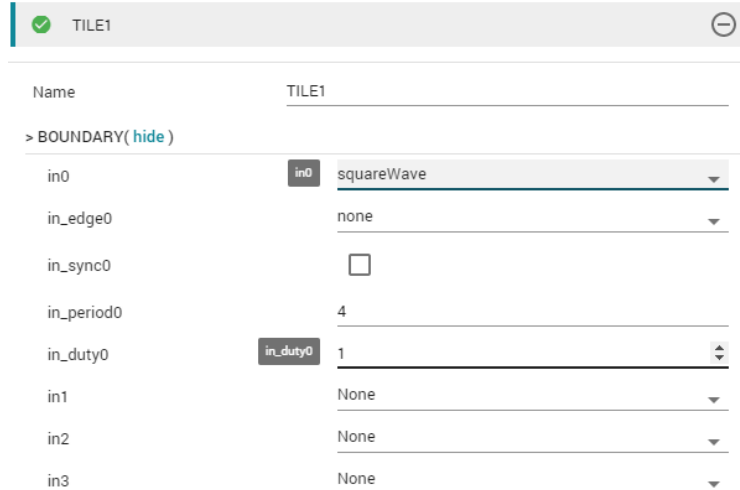
图 8. CLB 工具 SysConfig 屏幕

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 逻辑块配置。您可以根据需要更改逻辑块的名称。可以将多个 .syscfg 文件添加到同一个项目中。

对于突出显示的逻辑块，在右侧窗格中显示了子模块列表。通过单击名称右边的“Show”字样，可以检查和编辑每个子系统的参数。

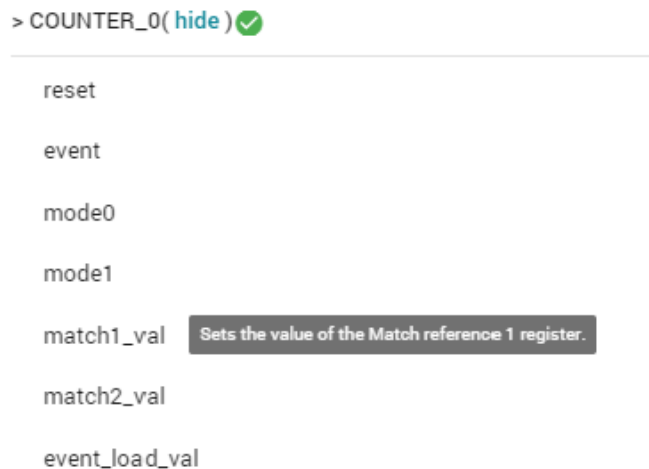
“BOUNDARY”项目是一种特殊情况。该组允许用户选择仅用于仿真的逻辑块输入。生成工具配置时，CLB 输入始终来自 TRM 中所述的全局和本地多路复用器模块，不过，出于仿真目的，用户可以指定方波信号源、周期和占空比（都以时钟周期为单位）以及同步条件，如图 9 所示。还支持用于仿真目的的自定义波形生成。有关仿真器的更多信息，请参阅 4 节。这些选项仅用于仿真，不会影响实际的 CLB 配置。



| TILE1                              |                          |
|------------------------------------|--------------------------|
| > BOUNDARY( <a href="#">hide</a> ) |                          |
| in0                                | in0 squareWave           |
| in_edge0                           | none                     |
| in_sync0                           | <input type="checkbox"/> |
| in_period0                         | 4                        |
| in_duty0                           | in_duty0 1               |
| in1                                | None                     |
| in2                                | None                     |
| in3                                | None                     |

图 9. “BOUNDARY”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。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 10 显示了 COUNTER\_0 子模块中 match1\_val 字段的示例。



```

> COUNTER_0( hide ) ✓
reset
event
mode0
mode1
match1_val  Sets the value of the Match reference 1 register.
match2_val
event_load_val
  
```

图 10. 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 1 显示了布尔方程中允许使用的符号。

表 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，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。输入行下方的错误消息指示方程无效。

一些不太可能的逻辑组合会向用户生成警告。图 11 显示了一个示例，其中用户尝试在布尔方程中使用 LUT\_0 中的 i2 输入。但是，i2 被配置为一个常数，用户不太可能希望这样。该警告同时显示在方程下方和输入选择下方。

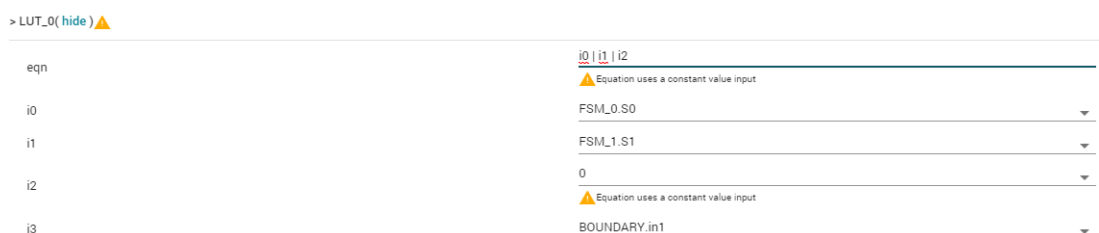


图 11. 方程警告

对于某些字段，该工具会对数字条目执行范围检查，以确保它们处于允许范围之内。例如，尝试加载值大于  $2^{32}$  的计数器模块将产生警告。

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，请单击该工具右上角的“<>”符号。双击文件名可打开相应的文件。

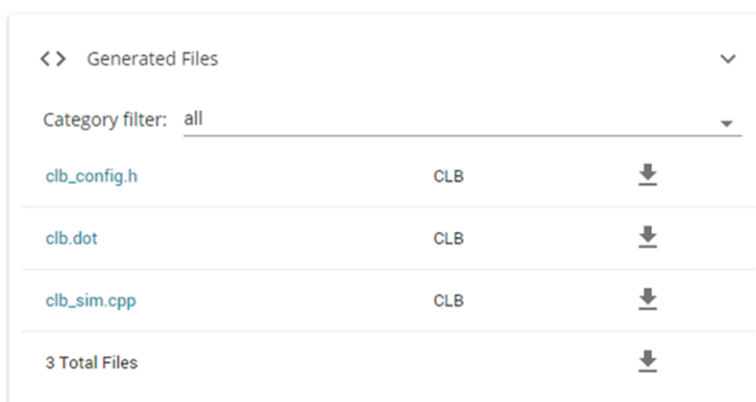


图 12. CLB 工具生成的文件

CLB 配置寄存器设置包含在头文件“clb\_config.h”中，用户可以通过单击文件名来打开和检查该文件。图 13 显示了一个示例。务必记住，每次用户更改任何 CLB 设置时，该工具都会更新此文件。因此，用户不应対文件内容进行任何手动更改，因为该工具将覆盖这些更改。如果在更改 CLB 设置时文件保持打开状态，则用户可能会观察到文件中受影响的寄存器数据发生变化。

```

clb_config.h
1  /*
2   * ===== clb.h =====
3   * DO NOT EDIT - This file is generated by the SysConfig tool.
4   */
5   #ifndef ti_clb_h
6   #define ti_clb_h
7
8   #include <stdint.h>
9
10  #ifdef __cplusplus
11  extern "C" {          // support C++ sources
12  #endif
13
14  // HLC Instruction Register Field definitions
15  #define HLC_OPCODE_R0 0x0
16  #define HLC_OPCODE_R1 0x1
17  #define HLC_OPCODE_R2 0x2
18  #define HLC_OPCODE_R3 0x3
19  #define HLC_OPCODE_C0 0x4
20  #define HLC_OPCODE_C1 0x5
21  #define HLC_OPCODE_C2 0x6
22
23  #define HLC_OPCODE_MOV    0x00
24  #define HLC_OPCODE_MOV_T1 0x01
25  #define HLC_OPCODE_MOV_T2 0x02
26  #define HLC_OPCODE_PUSH   0x03
27  #define HLC_OPCODE_PULL   0x04
28  #define HLC_OPCODE_ADD    0x05
29  #define HLC_OPCODE_SUB    0x06
30  #define HLC_OPCODE_INTR   0x07
31
32  //-----
33  // TILE1
34  //-----
35  #define TILE1_CFG_OUTLUT_0 0x7702c7
36  #define TILE1_CFG_OUTLUT_1 0x0
37  #define TILE1_CFG_OUTLUT_2 0x7702cf
38  #define TILE1_CFG_OUTLUT_3 0x0
39  #define TILE1_CFG_OUTLUT_4 0x770257
40  #define TILE1_CFG_OUTLUT_5 0x0
41  #define TILE1_CFG_OUTLUT_6 0x0
42  #define TILE1_CFG_OUTLUT_7 0x0
43
44  #define TILE1_CFG_LUT4_IN0 0x60a4
45  #define TILE1_CFG_LUT4_IN1 0x60a4
46

```

图 13. “clb.h”头文件示例

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，请参阅特定器件的 TRM。

选择有效的事件触发器后，该工具将显示几行，用户可以在其中键入 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 14 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。

> HLC( [hide](#) ) ✓

|              |              |
|--------------|--------------|
| Event 0 (e0) | BOUNDARY.in5 |
| Event 1 (e1) | 0            |
| Event 2 (e2) | 0            |
| Event 3 (e3) | 0            |
| R0_init      | 0            |
| R1_init      | 0            |
| R2_init      | 0            |
| R3_init      | 0            |

>> program0( [hide](#) ) ✓

|           |            |
|-----------|------------|
| instruct0 | intr 32    |
| instruct1 | mov c0, r0 |
| instruct2 | intr 0x14  |
| instruct3 |            |

图 14. HLC 配置示例

利用文件“clb.dot”，用户可以检查子模块互连的外观。该方框图的 HTML 版本在 post-build 步骤中生成，可以在 CCS 中打开和查看这些步骤。

## 4 CLB 仿真器

### 4.1 使用仿真器

#### 4.1.1 “Statics”面板

配置工具的顶部面板包含三个用于仿真的“静态”设置。将鼠标悬停在每个字段上可查看简短的说明。

> Statics( [hide](#) ) ✓

|                |       |
|----------------|-------|
| clock_period   | 20    |
| sim_duration   | 50000 |
| reset_duration | 40    |

图 15. 静态选项

“clock\_period”是用于仿真的 CLB 时钟周期（以纳秒为单位）。利用“sim\_duration”字段，用户可以控制仿真运行的持续时间（仍以纳秒为单位）。利用“reset\_duration”字段，用户可以插入仿真开始之前的延迟（以纳秒为单位），以模仿器件复位的效果。

### 4.1.2 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。通过单击“show”展开“BOUNDARY”类别。

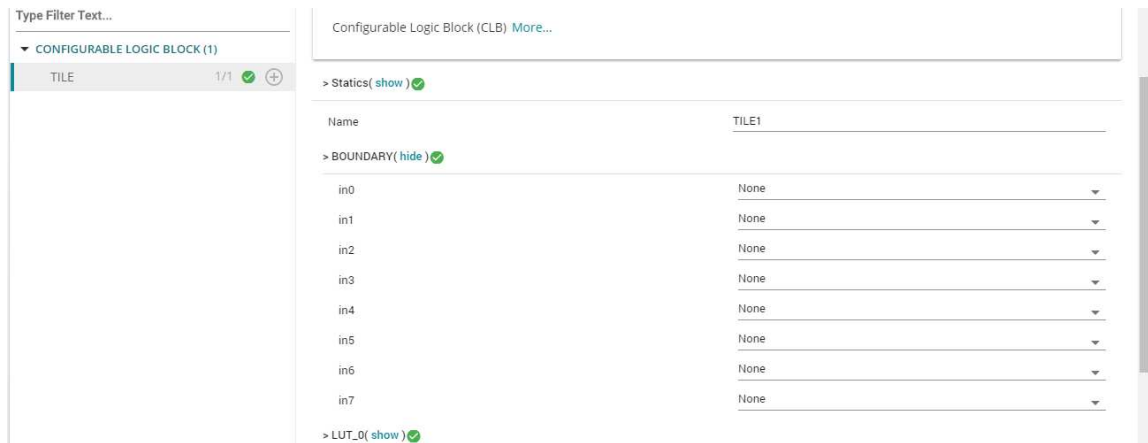


图 16. “BOUNDARY”输入 IN0 至 IN7

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。单击右侧的向下箭头可显示选项：

- None – 默认选项，不会生成任何激励。
- squareWave – 使用户能够定义具有可配置占空比和相位的周期性 PWM 输入。



图 17. “BOUNDARY”输入“squareWave”

“in-edge”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 17 中，其周期和有效信号时间分别设置为 10 个和 5 个 CLB 时钟脉冲。选中“in\_sync”复选框可以强制输入波形与 CLB 时钟同步。有关更多信息，请参阅特定器件的 TRM 中的 CLB 输入多路复用器 部分。

- Custom – 使用户能够生成自己的自定义激励。

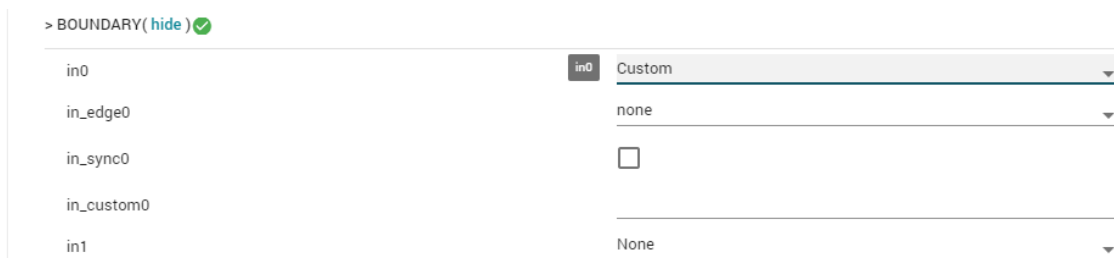


图 18. “BOUNDARY”输入“Custom”

为了使用该选项，用户必须在标记为“in\_custom”的行中输入 `systemC` 命令。这是该工具的高级功能，当前版本中未提供示例。

### 4.1.3 运行仿真

定义 CLB 配置和输入激励之后，用户可以编译项目。完成 `post-build` 步骤之后，将生成“CLB.vcd”文件。

假设已经完成了波形查看器配置，双击该文件应打开查看器并允许检查波形。图 19 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。

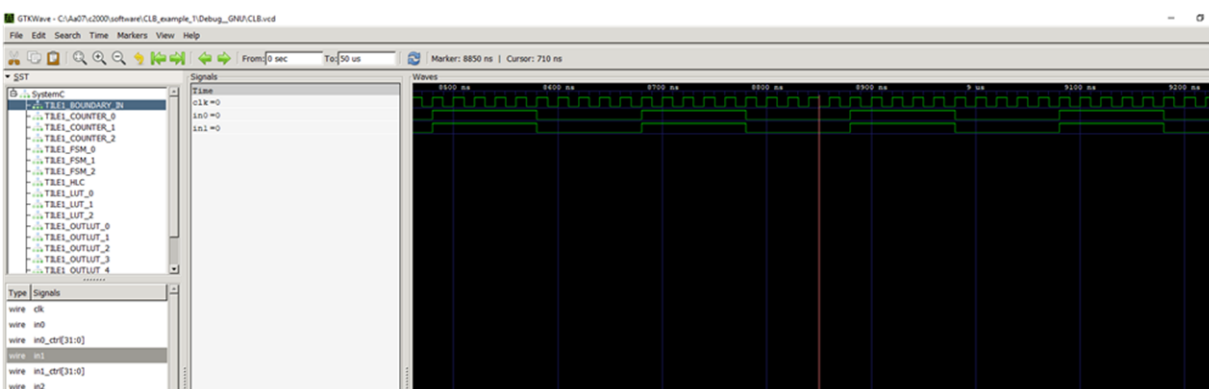


图 19. CLB 仿真示例

如果仿真波形与期望的波形不相符，则用户通常会修改 `.syscfg` 文件中的配置并重复仿真。

## 5 示例

该版本的 CLB 工具随附了一些示例，以显示如何配置 CLB 以实现简单的组合和时序逻辑电路。提供了两组示例：分别针对 F28379D 和 F280049。这两组示例之间的具有微小的代码差异，但运行示例的说明是相似的。

### 5.1 基本示例

这些示例的目的是展示每个 CLB 逻辑块内部子模块的功能。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。

#### 5.1.1 CLB 空项目

该示例是一个空 CLB 项目，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。

#### 5.1.2 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。

#### 5.1.3 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。

#### 5.1.4 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。

#### 5.1.5 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。

#### 5.1.6 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。

#### 5.1.7 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。

#### 5.1.8 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。

#### 5.1.9 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。

#### 5.1.10 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计算延迟量。



## 5.2 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 20 所示。

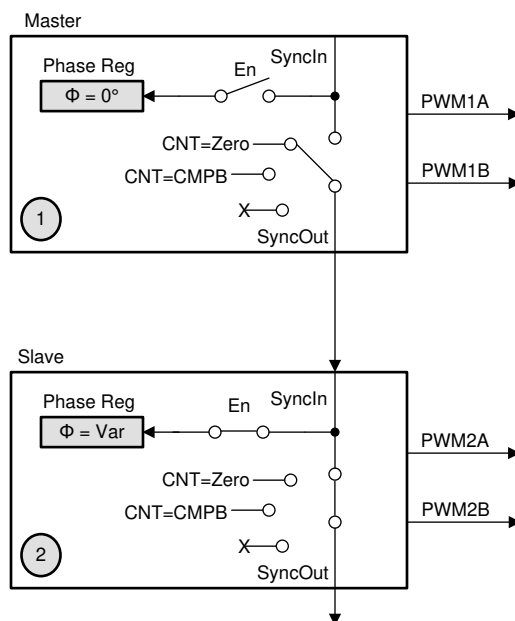


图 20. 示例 9: EPWM 同步

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 21 所示。

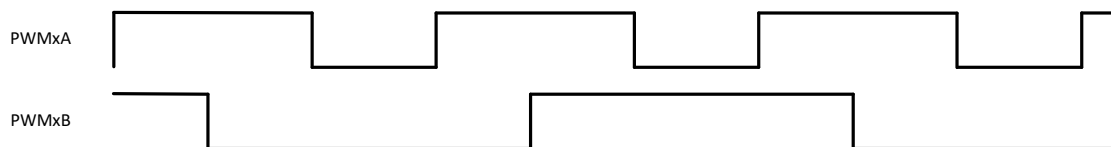


图 21. 示例 10: PWM 测试信号

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，在逻辑低电平模式下，两个 PWM 引脚上一定不会出现逻辑“0”输出。可以使用一个 2 位字段来选择模式，如表 2 所示。

表 2. 示例 1: 工作模式

| 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 22. 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 23 显示了 PWM1 的输入和输出波形（模式 1 和模式 2）。

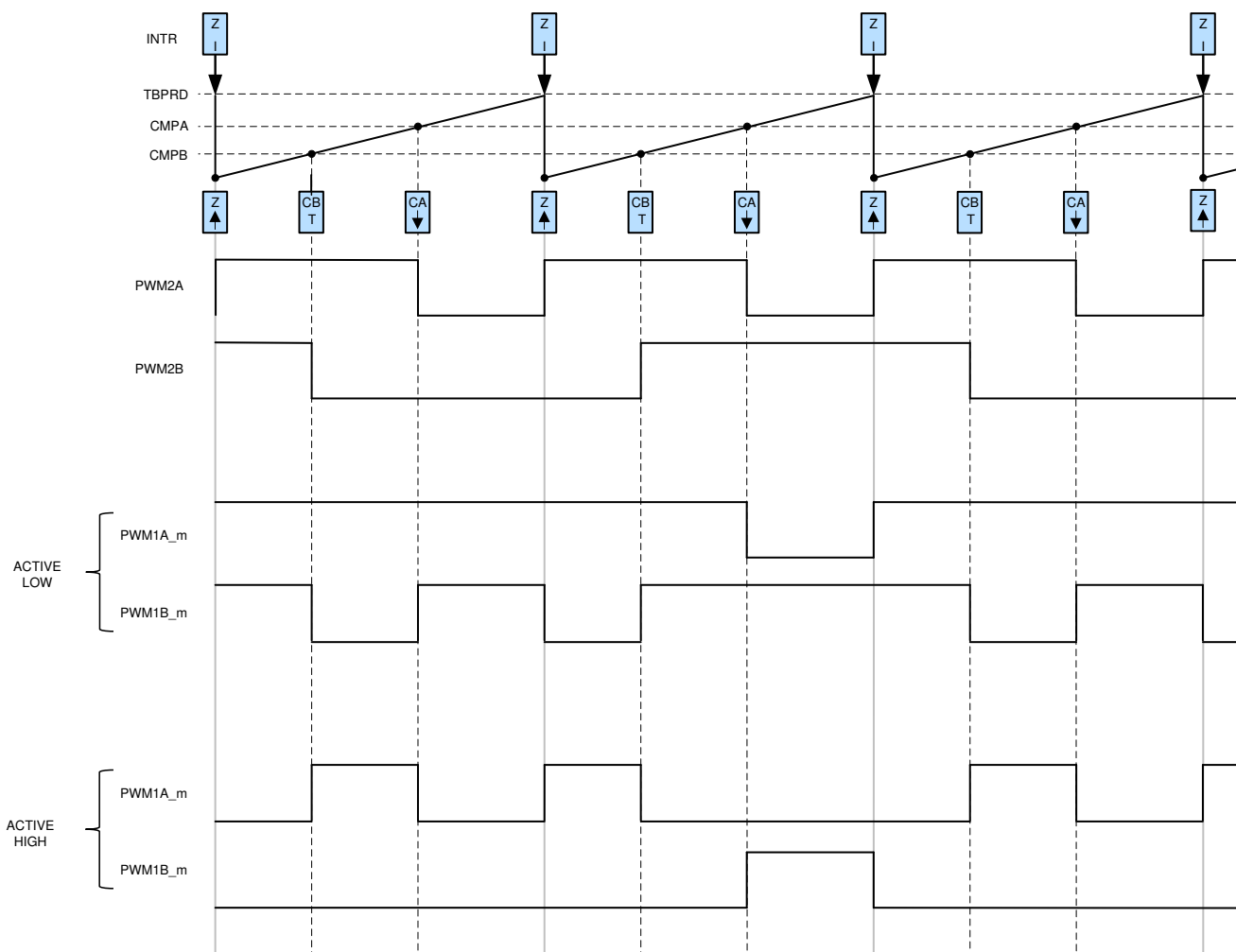


图 23. 示例 1：生成的 PWM

使用 4 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 24 从概念上显示了与 CLB 逻辑块之间的往返连接。

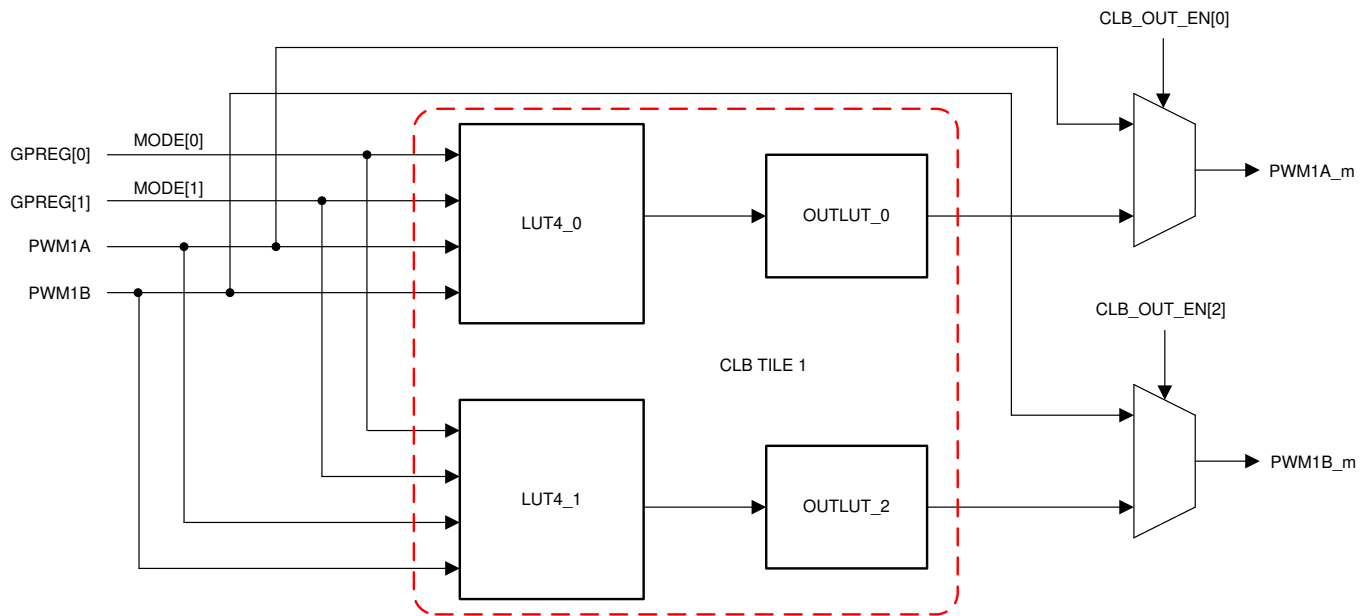


图 24. 示例 1: CLB 配置

要运行该示例，请执行以下过程：

1. 依次单击“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

在后面的说明中，假设使用上面的 C2000Ware 目录。

3. 选择项目“clb\_ex1\_combinatorial\_logic”并单击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开项目“clb\_ex1\_combinatorial\_logic”并打开文件“tile.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. [可选] – 有关如何运行 CLB 仿真的说明，请参阅节 4.1.3。

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。

### 5.3 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 25 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 25 大体上显示了该滤波器的功能。

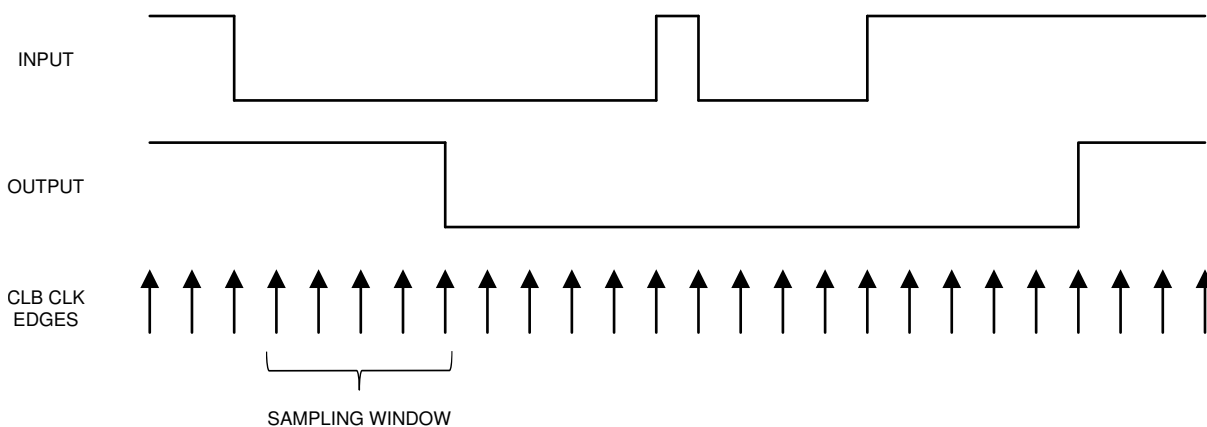


图 25. 示例 2: GPIO 干扰示例

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 26 显示了 CLB 配置。

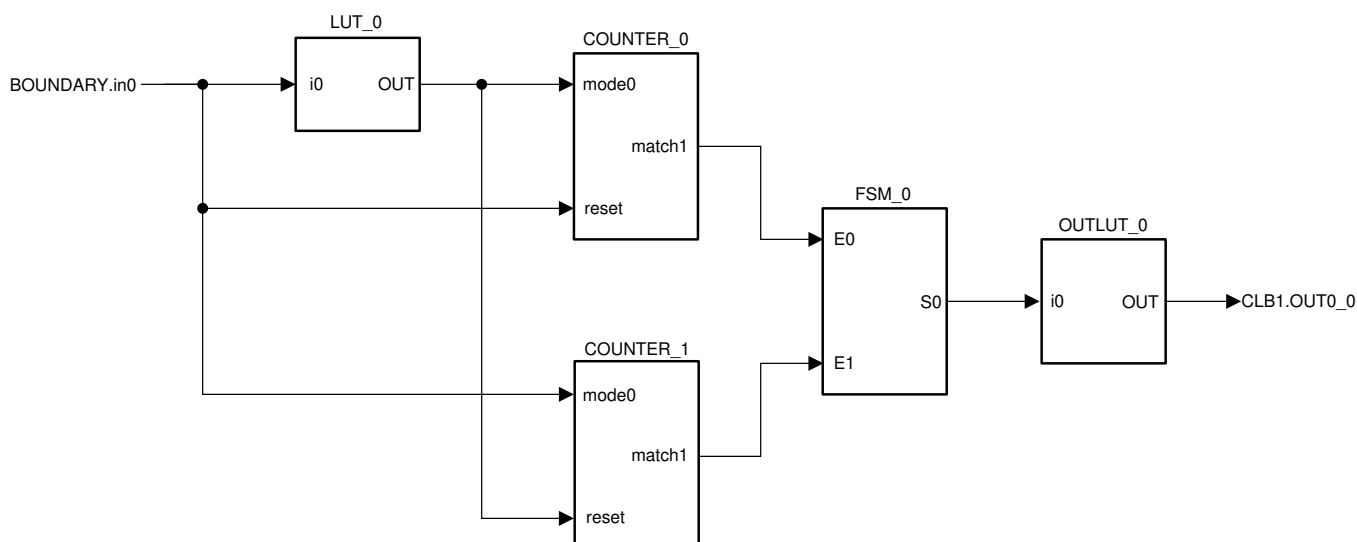


图 26. 示例 2: CLB 配置

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，依次单击“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
- 在后面的说明中，假设使用上面的 C2000Ware 目录。
3. 选择项目“glitch\_filter”并单击“Finish”。
  4. 在 CCS Project Explorer 窗口中，展开项目“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。验证配置是否与上述示例说明中的配置相匹配。
  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 4.1.3。

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 27 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。

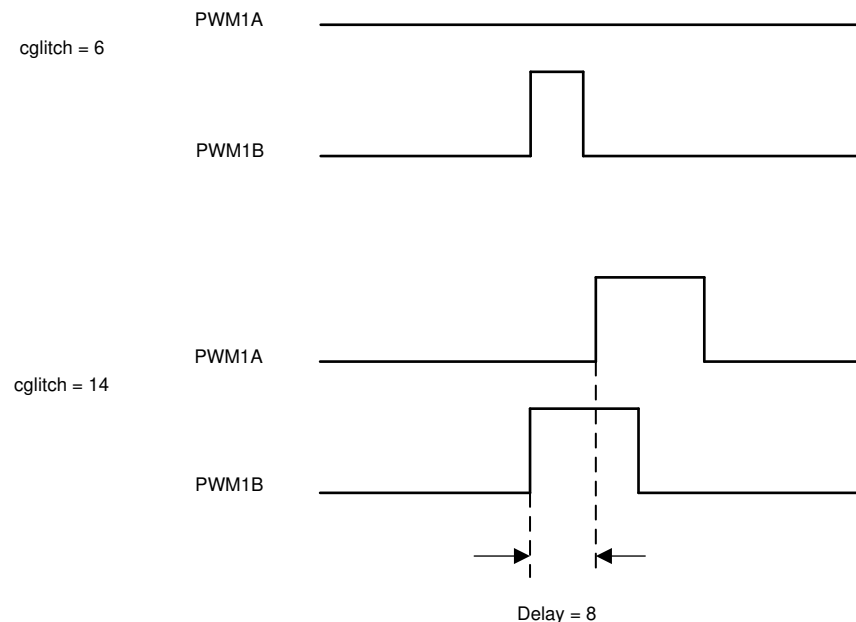


图 27. 示例 2: GPIO 干扰宽度

## 5.4 示例 3 – PWM 生成

该示例演示了如何将 CLB 逻辑块配置为充当辅助 PWM 发生器。该示例利用组合逻辑 (LUT)、状态机 (FSM)、计数器和 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，该事件与计数器的匹配 2 相关。在发生周期匹配事件时清除 PWM。该事件与计数器匹配 1 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匹配 1 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到计数器 C0 的匹配 2 寄存器中。这会更新 CMP 匹配值，进而更新正占空比的值。在该示例中，用户在正占空比的两个值之间进行交替。图 28 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。

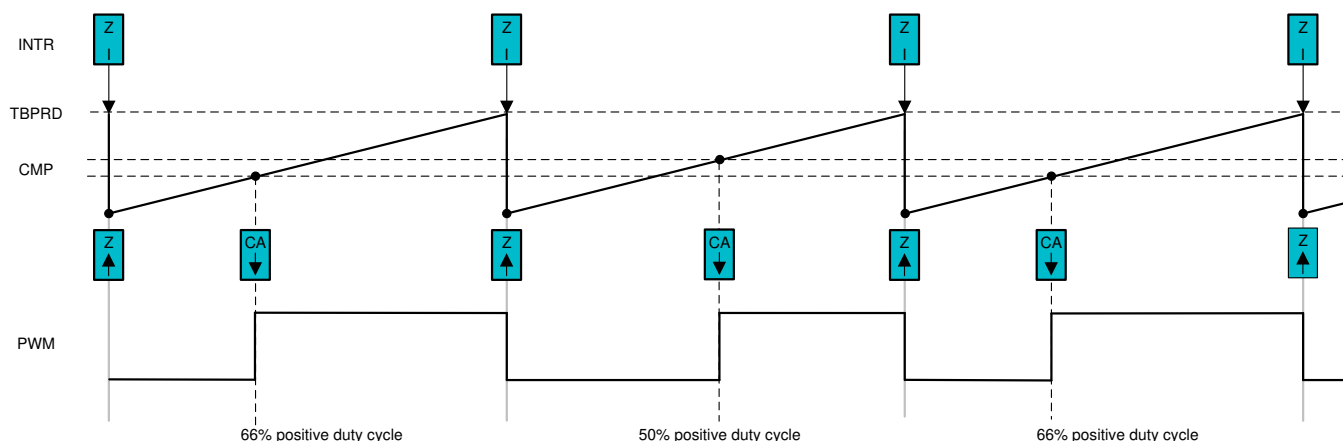


图 28. 示例 3: 生成的 PWM 波形

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。在该示例中，周期为 3μs。在 1μs 或 1.5μs 时发生比较事件。

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，依次单击“Project”->“Import CCS Projects...”。
2. 导航到 CLB 工具示例目录。路径为：
  - a. [C2000Ware]\driverlib\2837xd\examples\cpu1\clb\ccs 或
  - b. [C2000Ware]\driverlib\28004x\examples\clb\ccs 或
  - c. [C2000Ware]\driverlib\2838x\examples\c28x\clb\ccs
 在后面的说明中，假设使用上面的 C2000Ware 目录。
3. 选择项目“clb\_ex3\_auxiliary\_pwm”并单击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开项目“clb\_ex3\_auxiliary\_pwm”并打开文件“clb\_ex3\_aux\_pwm.syscfg”。
5. 检查逻辑块的配置并观察 LUT4\_0 和 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. [可选] – 有关如何运行 CLB 仿真的说明，请参阅节 4.1.3。

如果在 F28379D LaunchPad 板上运行该程序，则可以通过在引脚 J4/40 上监视 GPIO0 的反转来查看中断。可以在引脚 J4/34 上的 OutputXBAR1 信号上查看辅助 PWM。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 (GPIO0) 和 53 (OutputXBAR1) 上观察中断和 PWM 信号。

打开一个“CCS Expressions”窗口并添加程序变量“dutyValue”。在程序运行时，您将注意到每次为 CLB 中断提供服务时，正占空比值都会在 100 和 150 之间交替变化。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。可以在中断服务程序中修改 dutyValue 变量。

## 5.5 示例 4 – PWM 保护

该示例扩展了 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，依次单击“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
在后面的说明中，假设使用上面的 C2000Ware 目录。
3. 选择项目“clb\_ex4\_pwm\_protection”并单击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开项目“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 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。



## 5.6 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 29 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。

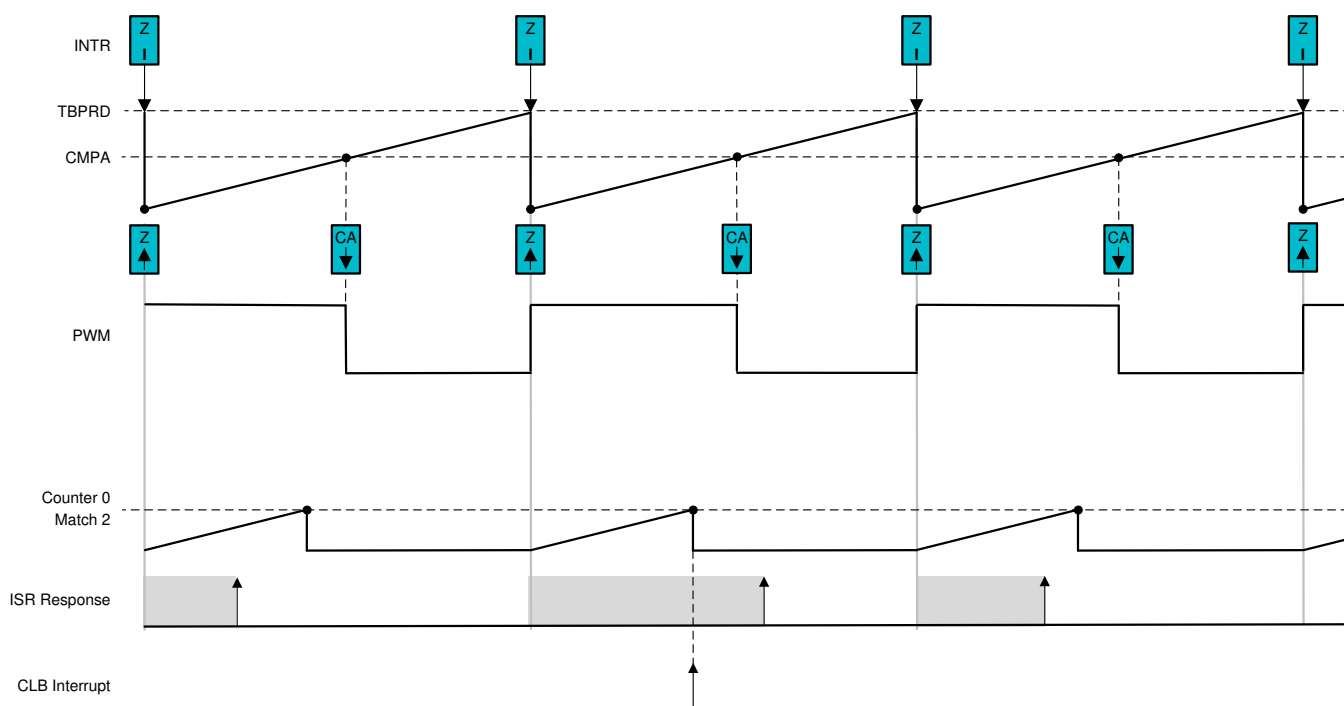


图 29. 示例 5: 事件窗口配置

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，依次单击“Project”->“Import CCS Projects...”。
2. 导航到 CLB 工具示例目录。路径为：
  - a. [C2000Ware]\driverlib\2837xd\examples\cpu1\clb\ccs 或
  - b. [C2000Ware]\driverlib\28004x\examples\clb\ccs 或
  - c. [C2000Ware]\driverlib\2838x\examples\c28x\clb\ccs

在后面的说明中，假设使用上面的 C2000Ware 目录。

3. 选择项目“clb\_ex5\_event\_window”并单击“Finish”。
4. 在 CCS Project Explorer 窗口中，展开项目“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”。

打开一个“CCS Expressions”窗口并添加四个程序变量“payload\_x”，其中“x”是 1 至 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。

## 5.7 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:** 该示例使用 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:** 该示例使用 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 30 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。

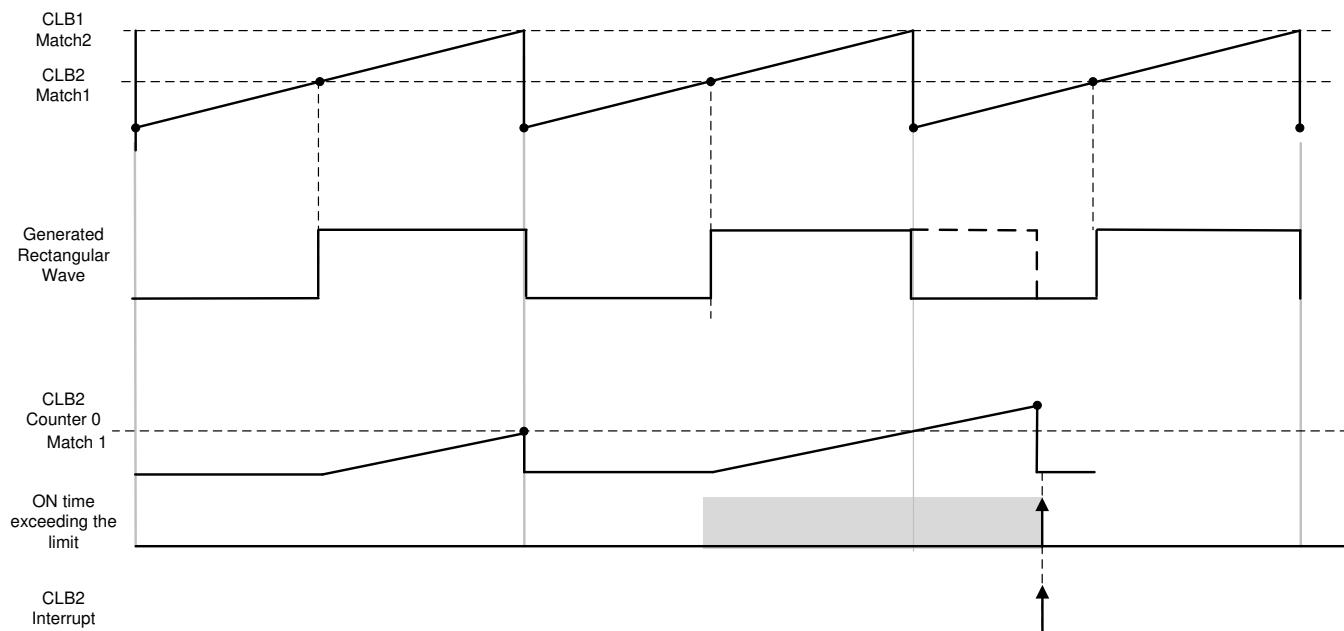


图 30. 示例 6: 有效信号时间超过预设值

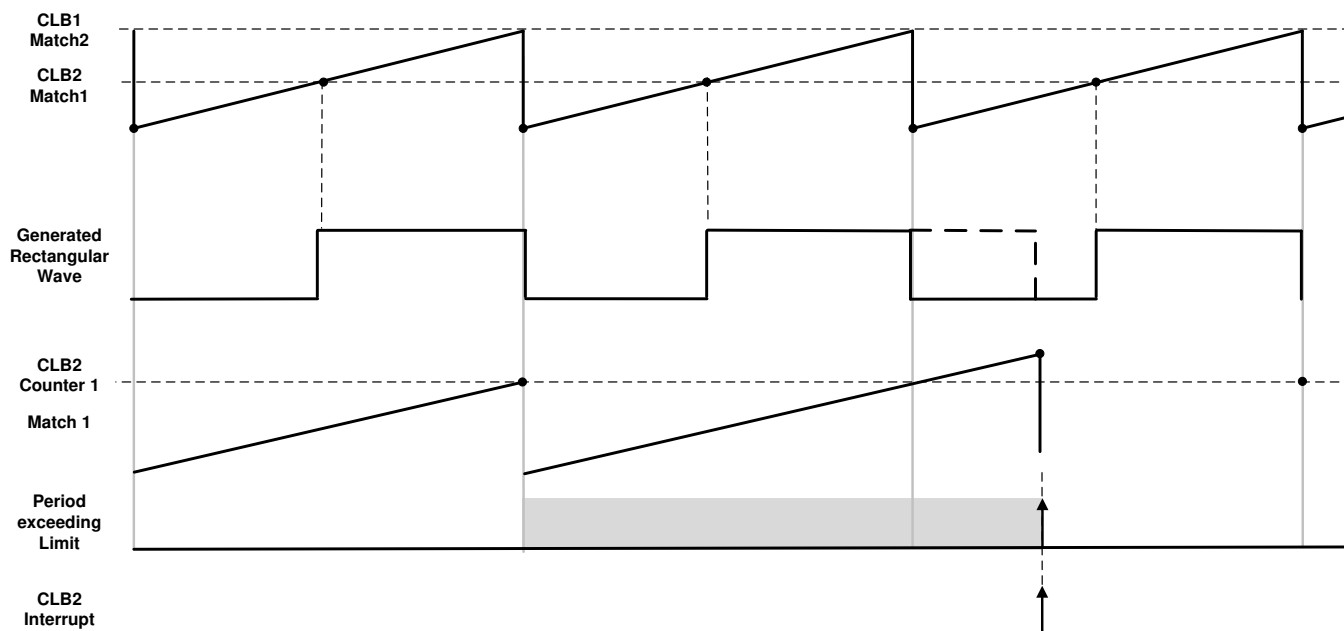


图 31. 示例 6: 周期超过预设值

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，依次单击“**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
- 在后面的说明中，假设使用上面的 **C2000Ware** 目录。
3. 13. 选择项目“**clb\_ex6\_siggen**”并单击“**Finish**”。
  4. 14. 在 **CCS Project Explorer** 窗口中，展开项目“**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。

## 6 在现有 **DriverLib** 项目中启用 **CLB** 工具

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

1. 将 **CLB** 示例文件夹中的“**empty.syscfg**”文件（对于 **F2837xD**，路径为 [C2000Ware]\driverlib\f2837xd\examples\cpu1\clb\ccs\empty.syscfg）添加到该项目中（通过将该文件复制到此项目中）。
2. **CCS** 将询问用户是否启用 **SysConfig**。接受并选择“**Yes**”。

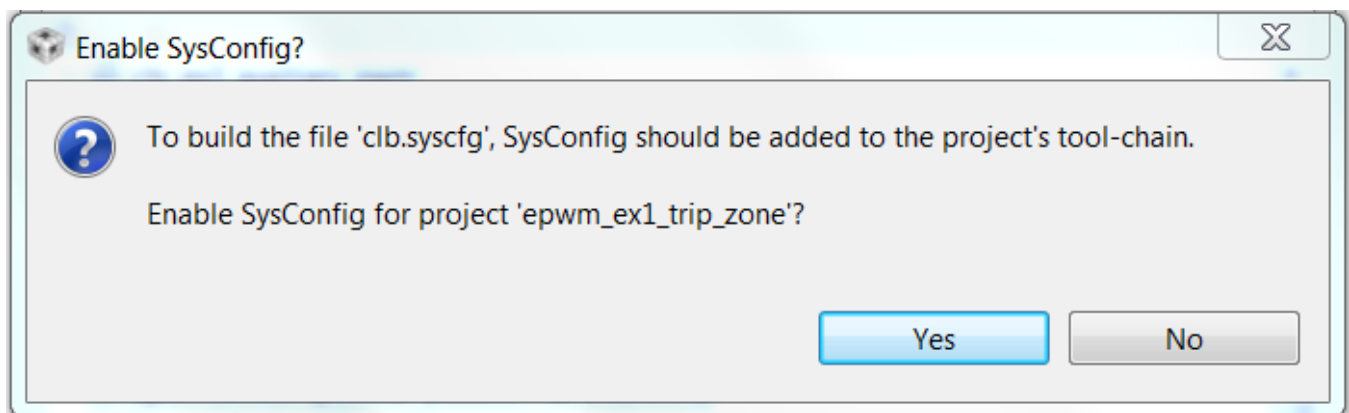


图 32. 启用 **SysConfig**

3. 打开“Project Properties”，然后依次打开“资源”→“Linked Resources”。添加以下变量路径：
  - a. CLB\_SYSCFG\_ROOT  
[PATH\_TO\_C2000WARE]\utilities\clb\_tool\clb\_syscfg
  - b. CLB\_SIM\_COMPILER  
C:\TDM-GCC-64\bin
  - c. SYSTEMC\_INSTALL  
[PATH\_TO\_C2000WARE]\utilities\clb\_tool\clb\_syscfg\systemc-2.3.3
  - d. 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 post-build 步骤中。
  - a. mkdir "\${BuildDirectory}\simulation"
  - b. \${CLB\_SIM\_COMPILER}\g++ -c -DCLB\_SIM -I\${SYSTEMC\_INSTALL}\src -I\${C2000WARE\_ROOT}\utilities\clb\_tool\clb\_syscfg\systemc\include -I\${PROJECT\_ROOT} -I\${CLB\_SIM\_COMPILER}\include -Og -g -gdwarf-3 -gstrict-dwarf -Wall -MMD -MP -MF\${BuildDirectory}\simulation\clb\_sim.d -MT\${BuildDirectory}\simulation\clb\_sim.o -I\${BuildDirectory}\syscfg -fno-threadsafe-statics -o\${BuildDirectory}\simulation\clb\_sim.o \${BuildDirectory}\syscfg\clb\_sim.cpp
  - c. \${CLB\_SIM\_COMPILER}\g++ -DCLB\_SIM -Og -g -gdwarf-3 -gstrict-dwarf -Wall -Wl,-Map,\${BuildDirectory}\simulation\simulation\_output.map -L\${SYSTEMC\_INSTALL}\build\src -o\${BuildDirectory}\simulation\simulation\_output.exe \${BuildDirectory}\simulation\clb\_sim.o \${C2000WARE\_ROOT}\utilities\clb\_tool\clb\_syscfg\systemc\src\CLB\_FSM\_SC\_model.o \${C2000WARE\_ROOT}\utilities\clb\_tool\clb\_syscfg\systemc\src\CLB\_HLC\_SC\_model.o \${C2000WARE\_ROOT}\utilities\clb\_tool\clb\_syscfg\systemc\src\CLB\_LUT4\_SC\_model.o \${C2000WARE\_ROOT}\utilities\clb\_tool\clb\_syscfg\systemc\src\CLB\_OutputLUT\_SC\_model.o \${C2000WARE\_ROOT}\utilities\clb\_tool\clb\_syscfg\systemc\src\CLB\_counter\_SC\_model.o -Wl,--start-group -lsystemc -Wl,--end-group
  - d. .\simulation\simulation\_output.exe
  - e. \${NODE\_TOOL} "\${CLB\_SYSCFG\_ROOT}/dot\_file\_libraries/clbDotUtility.js" "\${CLB\_SYSCFG\_ROOT}" "\${BuildDirectory}/syscfg" "\${BuildDirectory}/syscfg/clb.dot"

6. 最终的 post-build 步骤应类似于图 33 中的步骤。

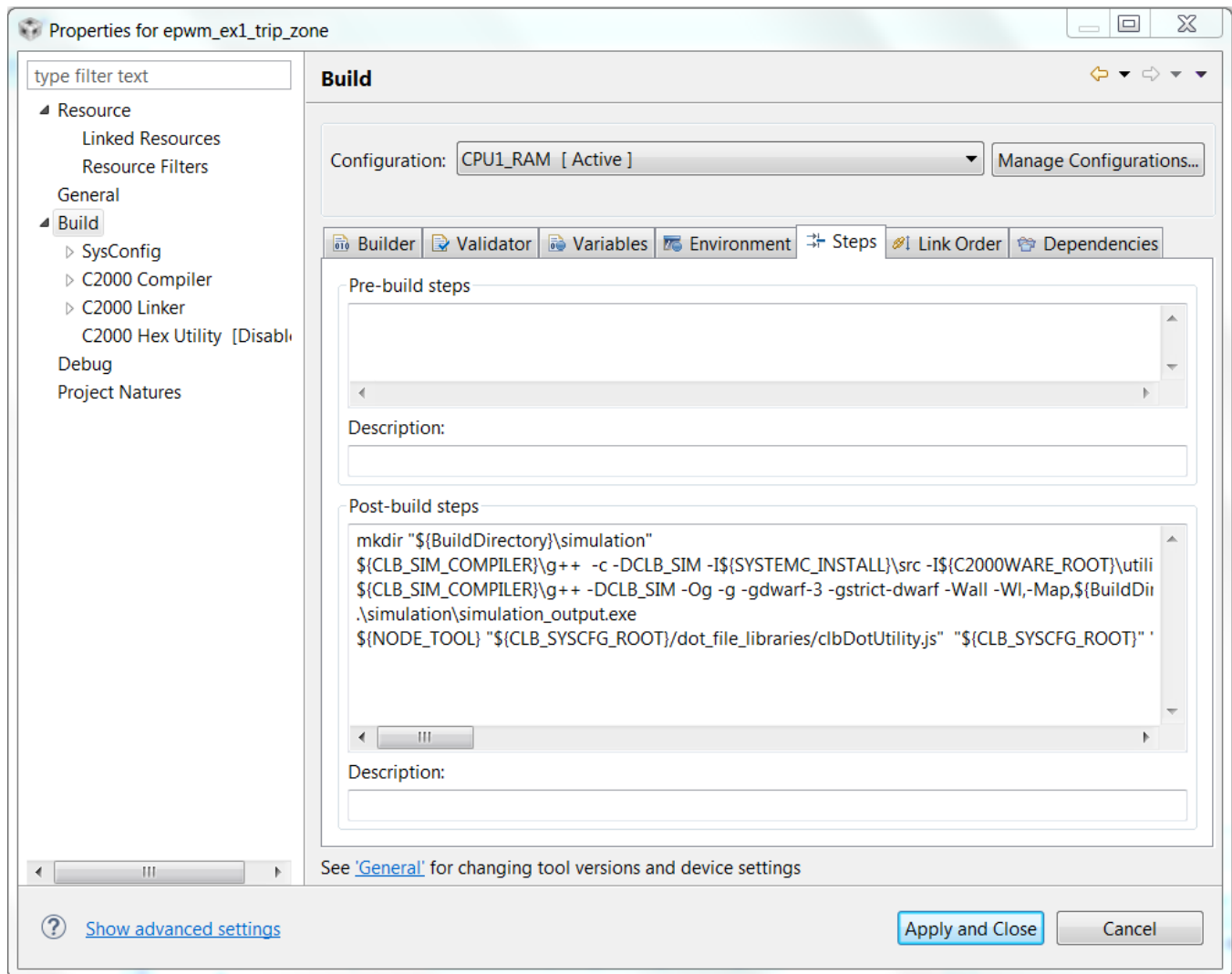


图 33. post-build 步骤

7. 接下来，依次打开“Build”→“SysConfig”→“Basic Options”并将以下内容添加到根系统配置元数据列表中
  - a. `${CLB_SYSCFG_ROOT}/.metadata/product.json`

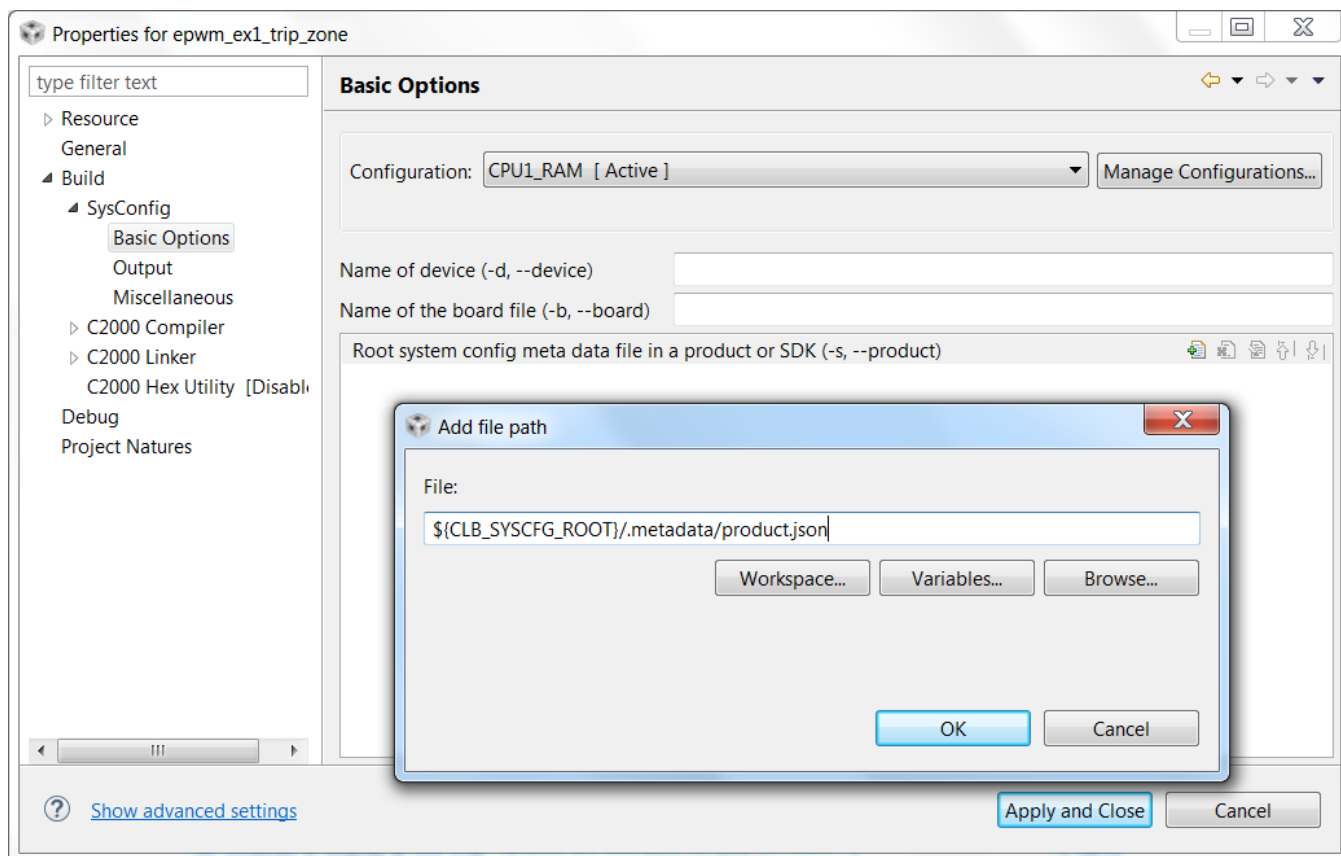


图 34. SysConfig SDK 路径

8. 最后依次单击“Apply”和“Close”。

9. 在构建该项目之后，CLB 工具生成的内容将出现在“Build Directory”中。图 35 显示了在 *epwm\_ex1\_trip\_zone* driverlib 示例中添加 CLB 支持后的相关示例。

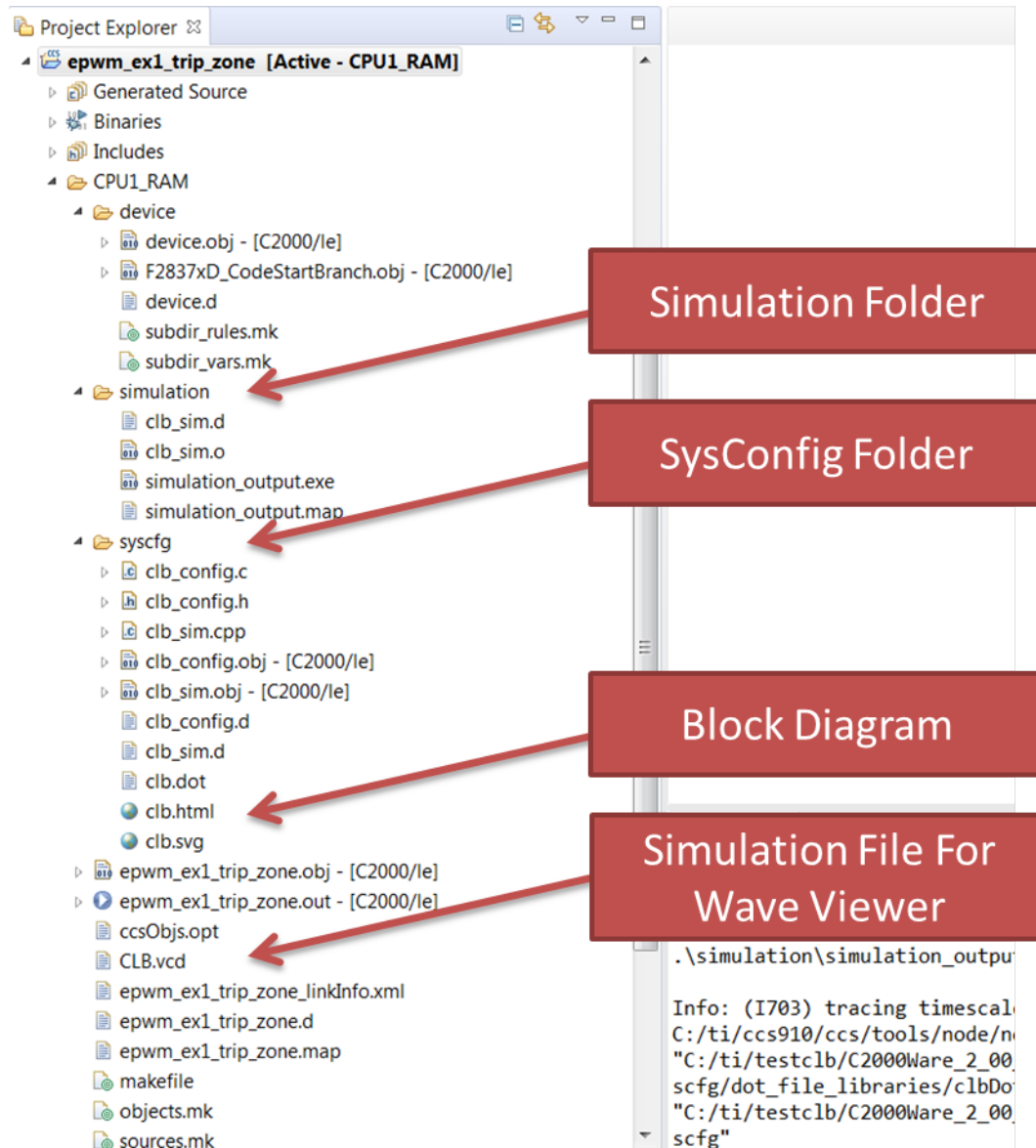


图 35. 具有 CLB 工具支持的 *epwm\_ex1\_trip\_zone*



## 7 常见问题解答 (FAQ)

问题：我的现有 CLB 项目与最新的 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”。

回答：SysConfig 中的最新更改现在需要修改和更新项目的 .syscfg 文件，以反映 TILE 资源在 CLB 包中的新位置。

---

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");
```

## 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

Copyright © 2022，德州仪器 (TI) 公司