使用 C2000™ 可配置逻辑块 (CLB) 进行设计

Nima Eskandari

摘要

C2000 可配置逻辑块 (CLB) 是一组可配置的逻辑模块，它们通过软件互联以实施定制的数字逻辑功能。在本应用报告中，我们将设计并测试一个简单的定制数字逻辑系统。我们会介绍和设置 CLB 模块内的每个可配置块，以实施所需的定制系统。

内容

附图目录

SPRACL3 — http://www-s.ti.com/sc/techlit/SPRACL3

附表目录

商标

C2000, s1, Code Composer Studio are trademarks of Texas Instruments.

1 简介

为了了解 CLB，本文档将分别介绍每个可配置块，并说明它们如何配合使用。CLB 子系统包含多个相同的逻辑块。F2837xD CLB 子系统中有四个相同的逻辑块，其他器件包含的逻辑块数量可能更多，也可能更少。每个逻辑块都包含：

- 4 输入查找表 (LUT4) 子模块
- 计数器子模块
- 有限状态机 (FSM) 子模块
- 输出 3 输入查找表（输出 LUT）子模块
- 高级控制器 (HLC) 子模块

设计一个简单的 4 态状态机。使用 CLB 逻辑块中的每个子模块，并显示它们的功能和用例。

2 补充在线信息

有关特定 C2000 器件上的 CLB 模块的更多信息，请参阅特定器件数据表和相应的技术参考手册 (TRM)。

本应用报告是使用 TMS320F2837xD 系列器件撰写的。下面列出了用于本应用报告的数据表和 TRM：

- *TMS320F2837xD 双核 Delfino™ 微控制器数据表*
- *TMS320F2837xD 双核 Delfino™ 微控制器技术参考手册*

TI E2E™ 社区还提供了其他支持。

3 设计概述

本文档中使用的设计是一个简单的四态状态机。这四种状态为：

- 已打开
- 正在关闭
- 已关闭
- 正在打开

此设计类似于简单的车库门打开方式。我们使用两个外部输入来与状态机交互。其中包括：

- BOUNDARY IN0：按钮输入，根据当前状态打开或关闭门
- BOUNDARY IN1：传感器输入，指示门是否已完成关闭或打开动作

图 1 显示了 FSM 的状态以及如何利用外部输入在这些状态之间转换。

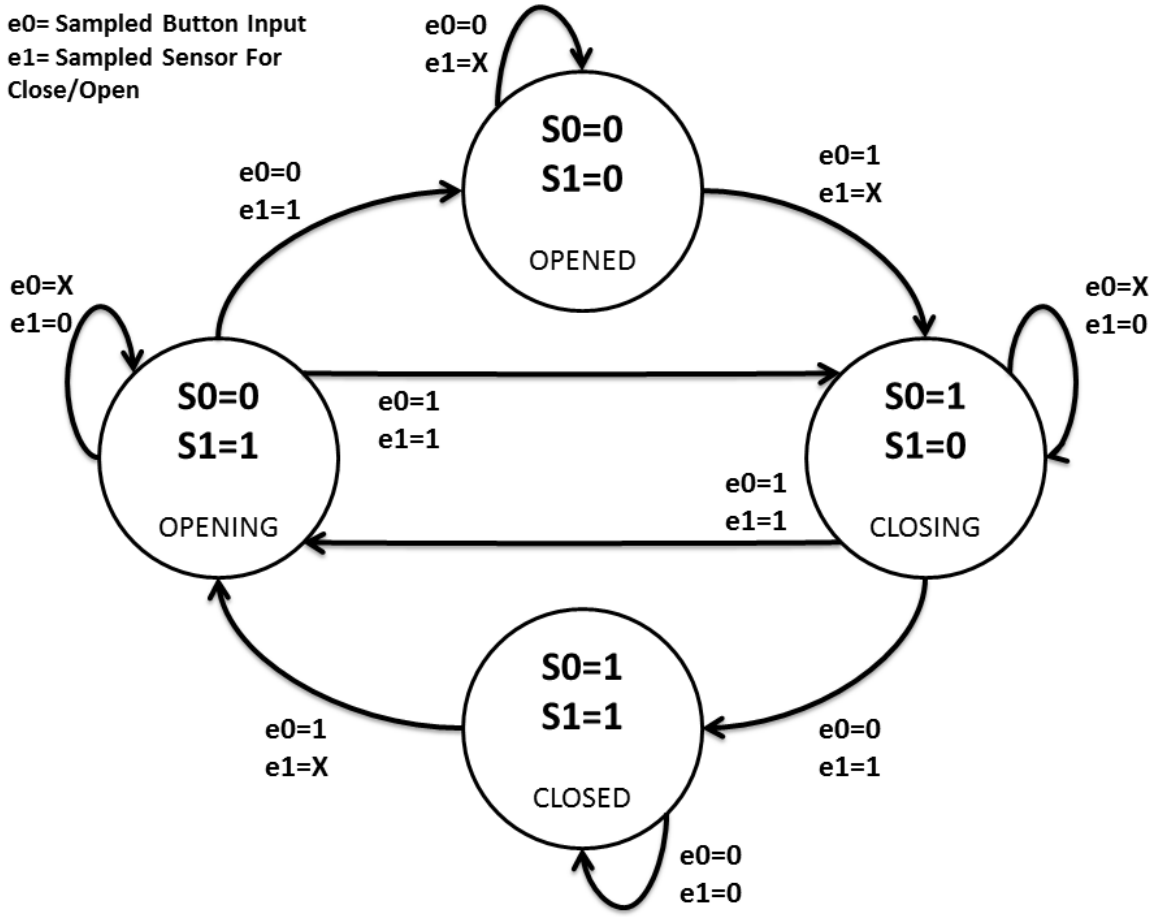

图 1. 状态机图

当 BOUNDARY IN0（按钮输入）为高电平时，系统将从已打开或已关闭状态转换为正在打开或正在关闭状态。当 BOUNDARY IN1（传感器输入）为高电平时，系统将从正在打开或正在关闭状态转换为已打开或已关闭状态。当 IN0 和 IN1 同时为高电平时，系统将从正在关闭转换为正在打开状态，或者从正在打开转换为正在关闭状态。所有其他输入组合都不会导致在状态机的状态之间转换。

4 对输入采样

设计这个特定系统的第一步是对外部输入进行采样，以便对最终的设计进行测试。我们将使用计数器每 50000 个时钟周期对 BOUNDARY IN0 和 IN1 进行一次采样，而不是将来自 GPIO 的源输入用作 BOUNDARY IN0 和 IN1。使用示波器显示正在关闭与正在打开状态之间的转换，以便于查看轮询结果。10 节 显示了这种情况。

将计数器模块 (COUNTER0) 与 LUT0 和 LUT1 配合使用，以生成采样输入信号，供 FSM 子模块随后使用。采样逻辑的要求如下所示：

- 使用 COUNTER0 每 50000 个时钟周期生成一个信号 (match1_val = 50000)
- 始终启用计数器 (mode0 = 1) 并进行计数 (mode1 = 1)
- 在发生 MATCH1 事件时重置计数器
- 使用 LUT0 和 LUT1
- 将两个输入信号（BOUNDARY IN0 和 IN1）与 COUNTER0 MATCH1 事件信号配合使用，以生成采样输入信号，供 FSM 随后使用

图 2 和图 3 显示了这些要求以及如何在示例项目的 SysConfig 设置中使用这些要求。

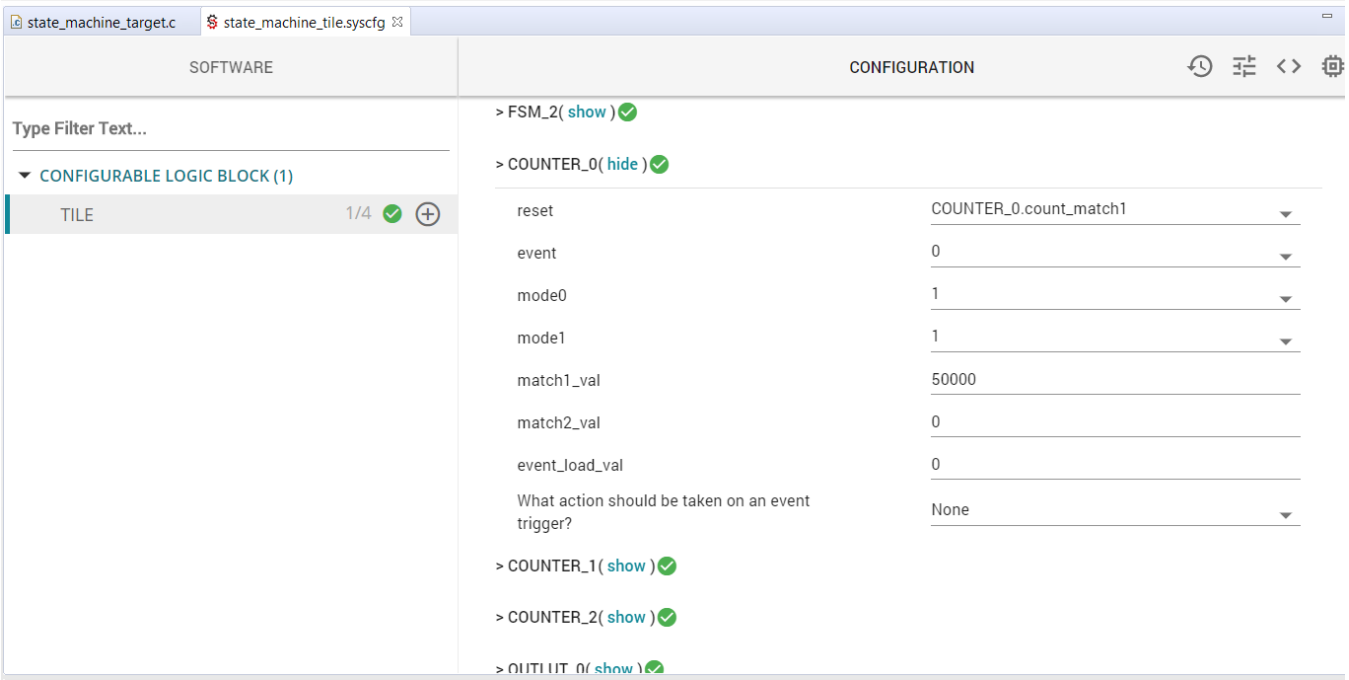

图 2. COUNTER0 配置

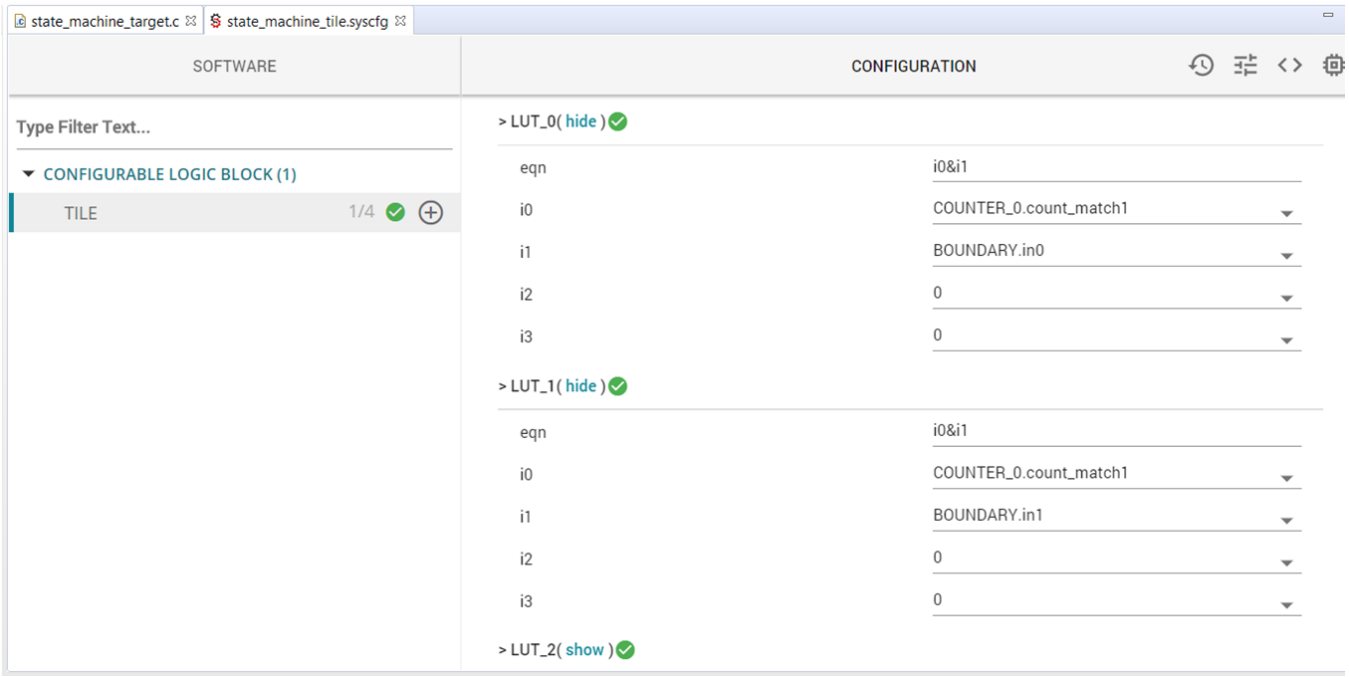

图 3. LUT0 和 LUT1 配置

5 在 FSM 子模块中部署状态机

图 1 中的状态机是利用上一节所生成的采样输入部署的。首先创建一个完整的真值表，其中包含输入以及状态机的当前状态和次态。FSM 子模块也会生成一个输出信号。在此设计中，当系统的状态转换为正在打开或正在关闭时，会将输出设置为高电平。稍后会在设计中使用此输出（用于门控 PWM 信号）。表 1 显示了此设计的完整真值表。

表 1. FSM 真值表

s0	s1	e0	e1	s0 次态	s1 次态	输出
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	1
0	0	1	1	1	1	1
0	1	0	0	0	1	0
0	1	0	1	0	0	0
0	1	1	0	0	1	0
0	1	1	1	1	0	1
1	0	0	0	1	0	0
1	0	0	1	1	1	0
1	0	1	0	1	0	0
1	0	1	1	0	1	1
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	0	1	1
1	1	1	1	0	1	1

使用 FSM 真值表，为 FSM 的 s0 态、s1 态和输出创建卡诺图。卡诺图有助于查找用于 s0、s1 和输出的 FSM 公式。

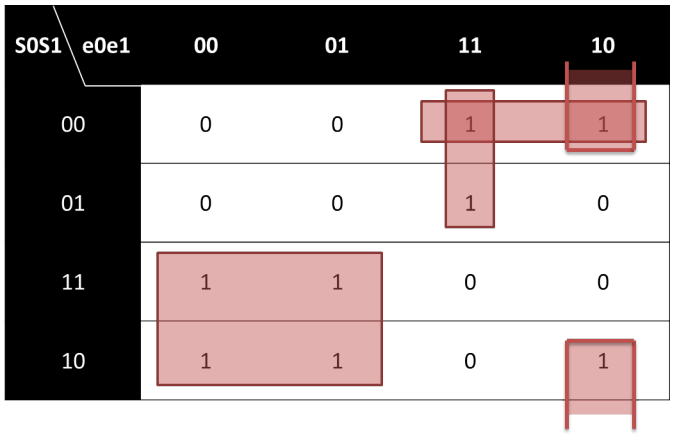

图 **4. FSM1 S0** 卡诺图

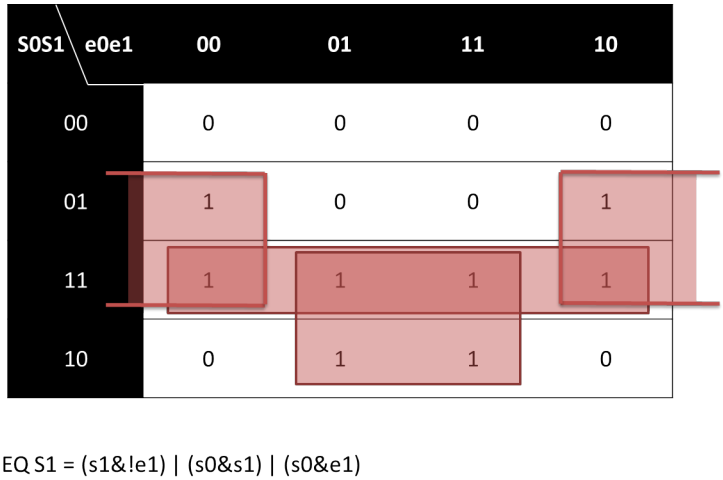

图 **5. FSM1 S1** 卡诺图

S0S1 \ e0e1	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	0	0	1	1
10	0	0	1	0

EQ OUTPUT = (e0&e1) | (!s0&!s1&e0) | (s0&s1&e0)

图 **6. FSM1** 输出卡诺图

F28379D LaunchPad（用于测试此设计）会在 LaunchPad 上的 LED 上显示 FSM1 的 s0 和 s1™。OUTLUT4 和 OUTLUT5 将这些信号从 FSM1 导出到输出 X-BAR 中， 并通过选择好的X-BAR 以驱动 GPIO。当 GPIO34 和 GPIO31 的输出被拉低时，LaunchPad 上的 LED 将打开。FSM1 的 s0 和 s1 会因此反相，随后会通过 OUTLUT4 和 OUTLUT5 从 CLB 中输出。当 s0 和 s1 为高电平时，此反相会将 LED 打开。图 7 显示了 OUTLUT4 和 OUTLUT5 的 SysConfig 配置。

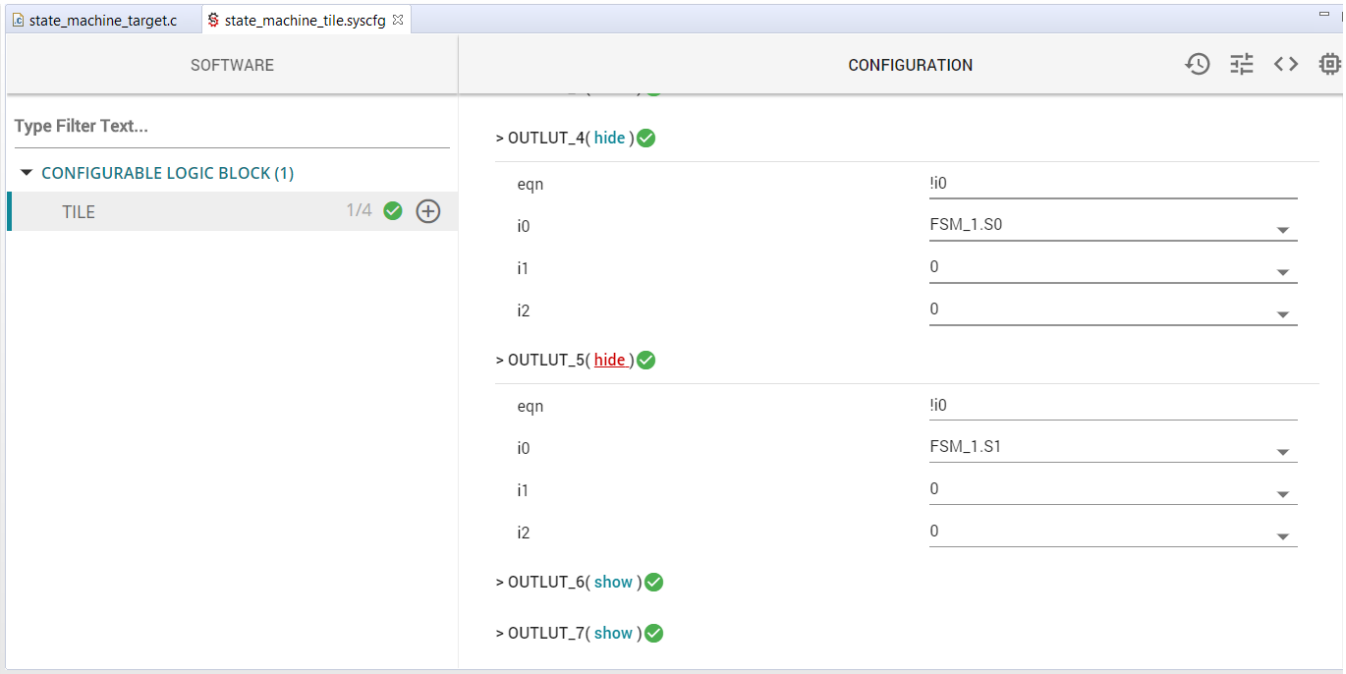

图 **7. OUTLUT4** 和 **OUTLUT5** 配置

6 生成 PWM 信号

CLB 可以使用自己的可配置块来生成 PWM 波形。假设需要在系统的状态为正在打开或正在关闭时生成 PWM 信号。对于此系统，来自 CLB 逻辑块的两个 PWM 信号分别是：

- 正在打开状态：第一个 PWM 信号有效，并具有可修改的周期和占空比，而第二个 PWM 信号保持低电平。
- 正在关闭状态：第一个 PWM 信号保持低电平，而第二个 LOW 信号有效并具有可修改的周期和占空比。

FSM0、FSM2 和 COUNTER1 生成 PWM 信号。计数器子模块用作 PWM 的主要部分，FSM0 和 FSM2 则构成逻辑的剩余部分。需要这些角色才能生成 PWM 波形。

将 COUNTER1 设置为在倒计数模式下运行，并在计数器达到 ZERO 或指定的匹配值时生成信号。以下各项总结了对 COUNTER1 的要求：

- 从 load_val (mode1 = 0) 倒计数。
- 当发生 counter=ZERO 事件时，将 load_val 加载到计数器中。
- 使用 COUNTER1 的 load_val 寄存器设置 PWM 信号的周期。
- 在后续步骤中使用在 counter=match1_val 时生成的事件。
- 使用 COUNTER1 的 match1_val 寄存器设置 PWM 的占空比。
- 计数器的启用信号是 mode0。此信号应当只在系统处于正在打开或正在关闭状态时有效（此信号将在后续步骤中创建，做为计数器的输入信号）。
- 当检测到状态转换为正在打开或正在关闭状态时，重置计时器（已经可以从 FSM1 输出获得此信号）。

这样就设置了基于时间的 PWM 计数器。接下来，创建只在系统处于正在打开或正在关闭状态时有效的信号。要创建此信号，需要将 FSM2 用作 3 输出 LUT 而非有限状态机。设置 FSM2 以满足以下要求：

- S0：只在正在打开状态激活时有效
- S1：只在正在关闭状态激活时有效
- 输出：在正在关闭或正在打开状态激活时有效

将来自 FSM2 的输出用作 COUNTER1 mode0 的输入。COUNTER1 的 Mode0 只在系统处于正在关闭或正在打开状态时有效。

图 8 和图 9 显示了 FSM2 和 COUNTER1 的 SysConfig 配置。

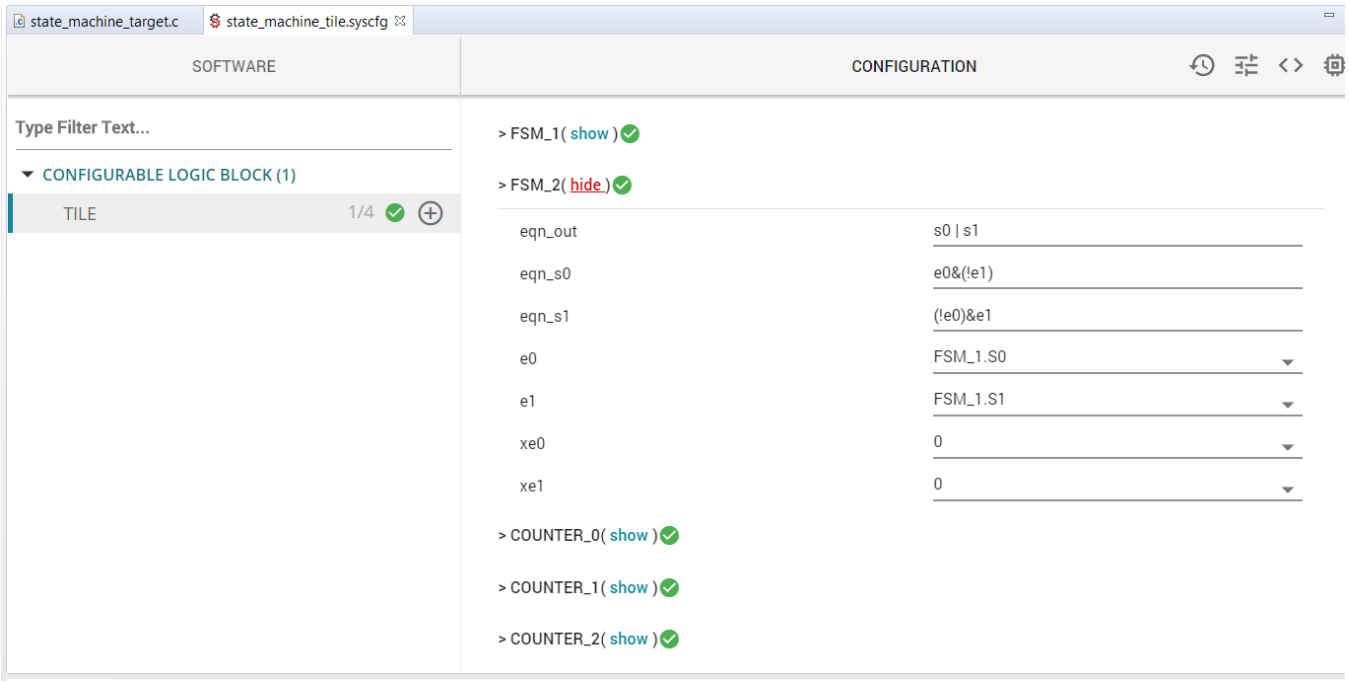

图 8. FSM2 配置

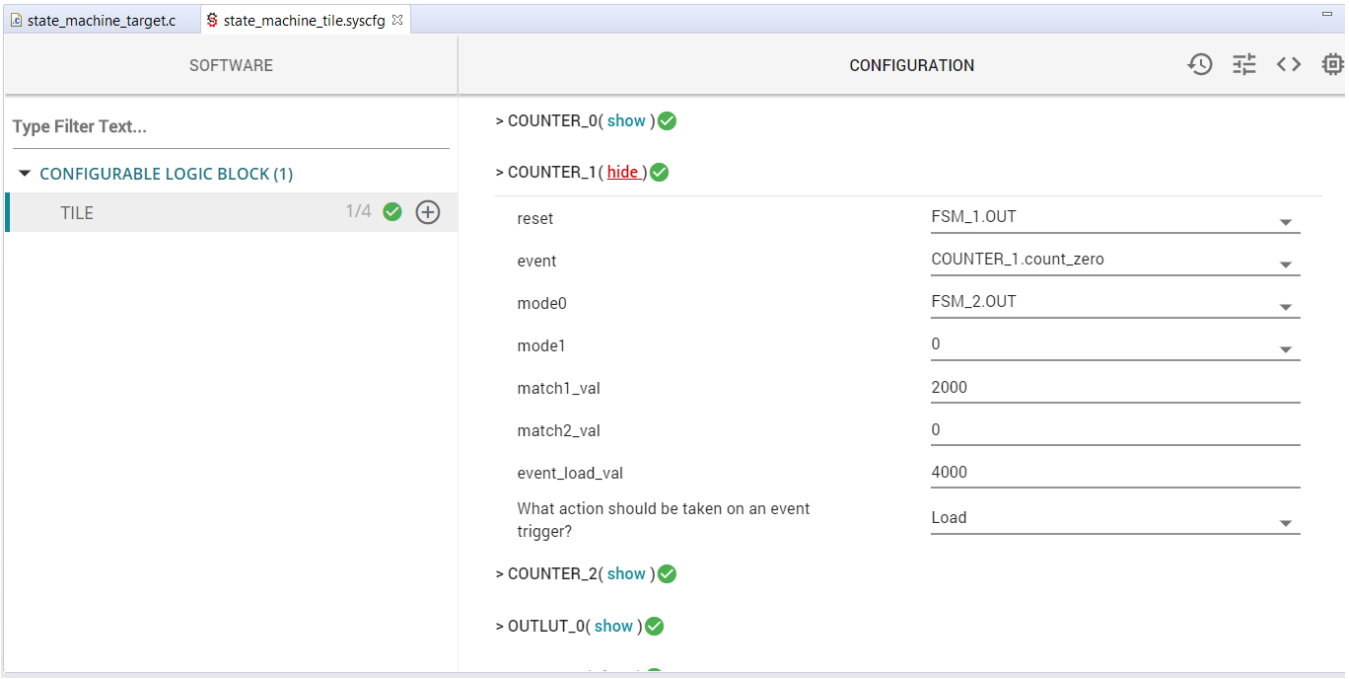

图 9. COUNTER1 配置

创建 PWM 信号的最后一步是使用 FSM 模块，根据 COUNTER1 和 FSM2 生成的事件和信号生成 PWM 波形。使用 FSM0 s0 生成 PWM 信号。当 COUNTER1 发生 counter=ZERO 事件时，s0 必须清除为低电平。当 COUNTER1 发生 counter=match1_val 事件时，s0 必须设置为高电平。

图 10 显示了 FSM0 的 SysConfig 配置。

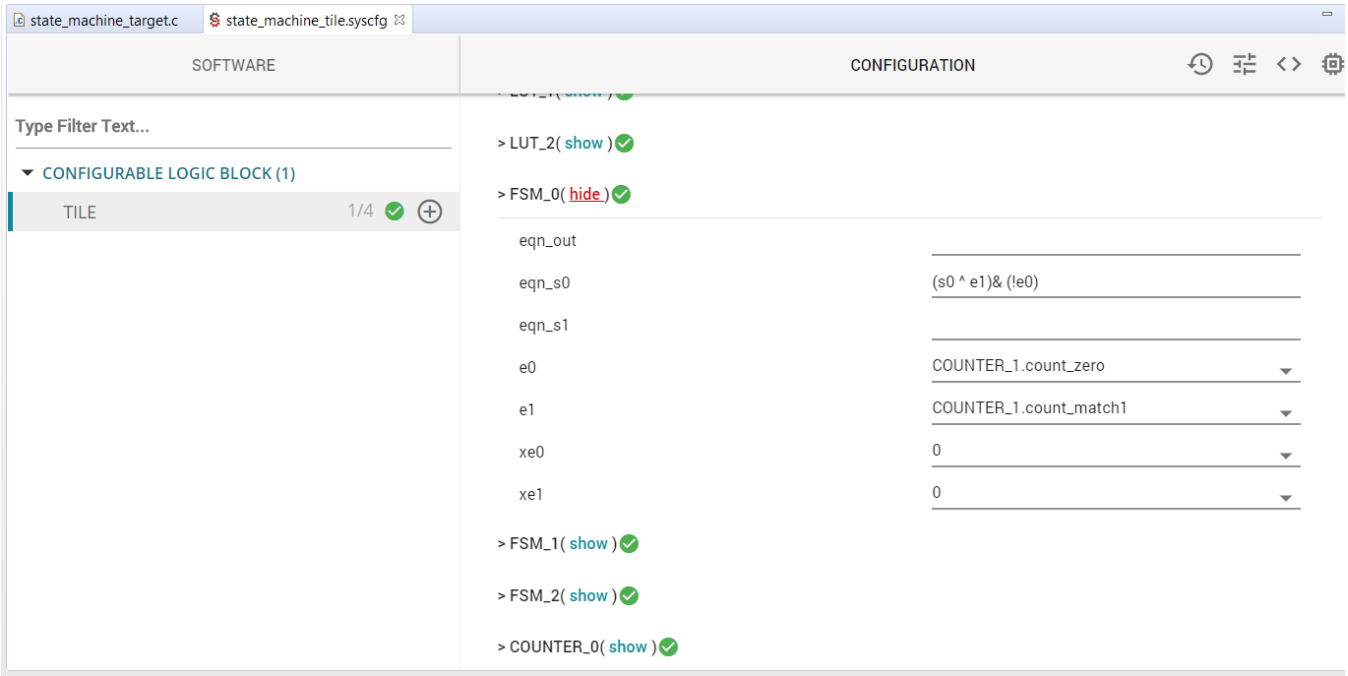

图 10. FSM0 配置

最后，将 PWM 信号拆分为两个信号：一个用于正在关闭状态，一个用于正在打开状态。OUTLUT0 和 OUTLUT2 输出这两个信号。当系统处于正在打开状态时，OUTLUT0 将输出 PWM 信号，OUTLUT2 将保持低电平。当系统处于正在关闭状态时，OUTLUT2 将输出 PWM 信号，OUTLUT0 将保持低电平。图 11 显示了 OUTLUT0 和 OUTLUT2 的 SysConfig 配置。

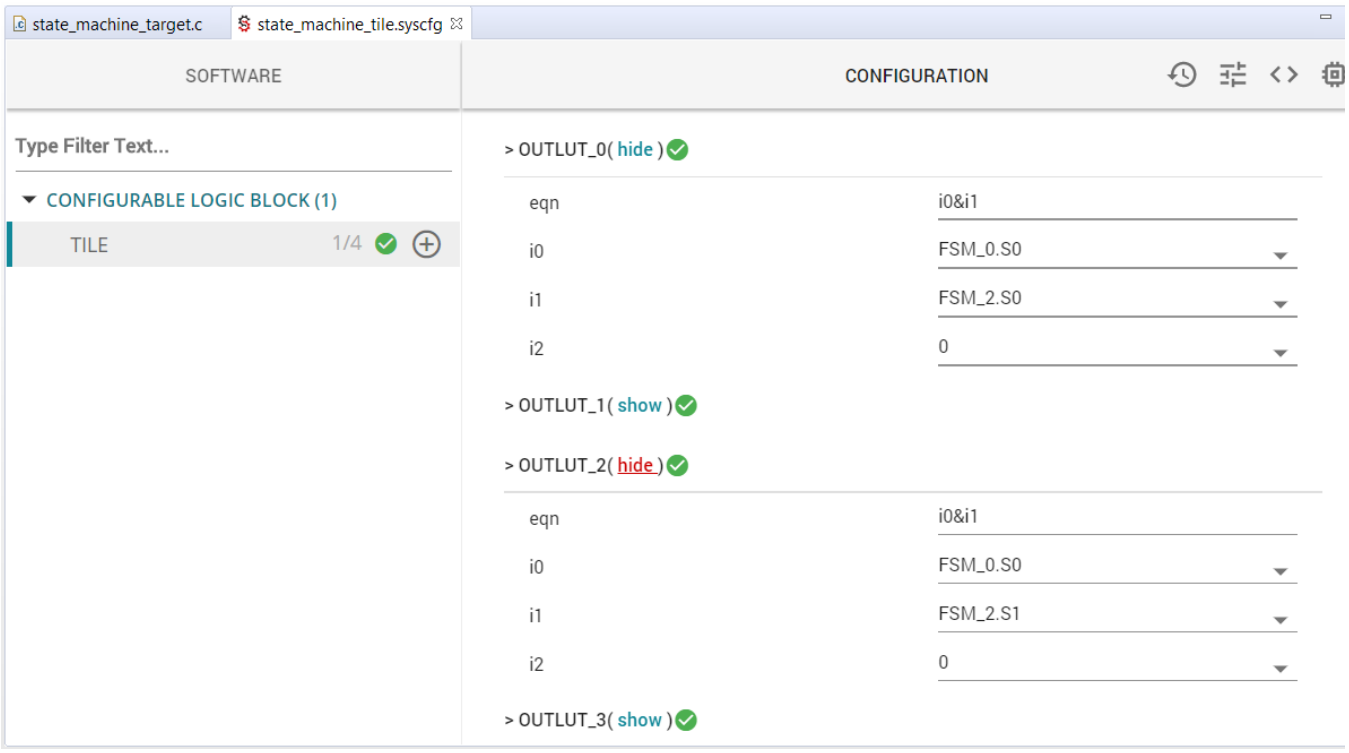

图 11. OUTLUT0 和 OUTLUT2 配置

在下一步，可以在运行期间随时使用 C28x 内核和 HLC 子模块来更新 PWM 周期和占空比。

7 修改 PWM 周期和占空比

需要更新 COUNTER1 的 load_val 和 match1_val，以修改 CLB 逻辑块生成的 PWM 的周期和占空比。C28x 内核能够访问 COUNTER1 中的这两个寄存器并根据需要对它们进行更新。不过，最好在 COUNTER1 发生 counter=ZERO 事件时更新 load_val 和 match1_val，以免发生任何不可预测的行为。在此设计中，C28x 内核提出请求之后，计数器值会在下一个 counter=ZERO 事件时被更新。为了加载计数器值，将使用 HLC 子模块和推挽接口。推挽接口是 C28x 内核和 HLC 子模块用来传输数据的 4 级深度 FIFO。C28x 内核将新的周期和占空比值写入到推挽 FIFO 中。HLC 读取这些值并在发生下一个 counter=ZERO 事件时更新 COUNTER1 寄存器。C28x 还必须向 HLC 发出信号，以表明推挽 FIFO 中提供了新值。使用 GPREG 寄存器（C28x 可以访问的寄存器）的 BIT2 (0b00000100) 创建此信号。将 GPREG 寄存器位连接到 CLB 逻辑块的 BOUNDARY 输入。HLC 在输入事件被激活时执行自己的任务。

在此设计中，使用 EVENT0 向 HLC 发出信号，以使其读取推挽 FIFO 并更新 PWM 的值。如果在 HLC 的 EVENT0 上检测到上升沿，则执行指定的指令。由于 C28x 内核必须向 HLC 发出信号以使其在发生下一个 COUNTER1 counter=ZERO 事件时开始更新 PWM 值，因此属于 HLC 的 EVENT0 输入必须是对 COUNTER1 counter=ZERO 和 BOUNDARY IN2 执行与运算之后得到的信号。使用 LUT2 对这两个信号执行逻辑与运算。随后将 LUT2 的输出连接到 HLC 的 EVENT0 输入。

图 12 显示了 LUT2 的 SysConfig 配置。

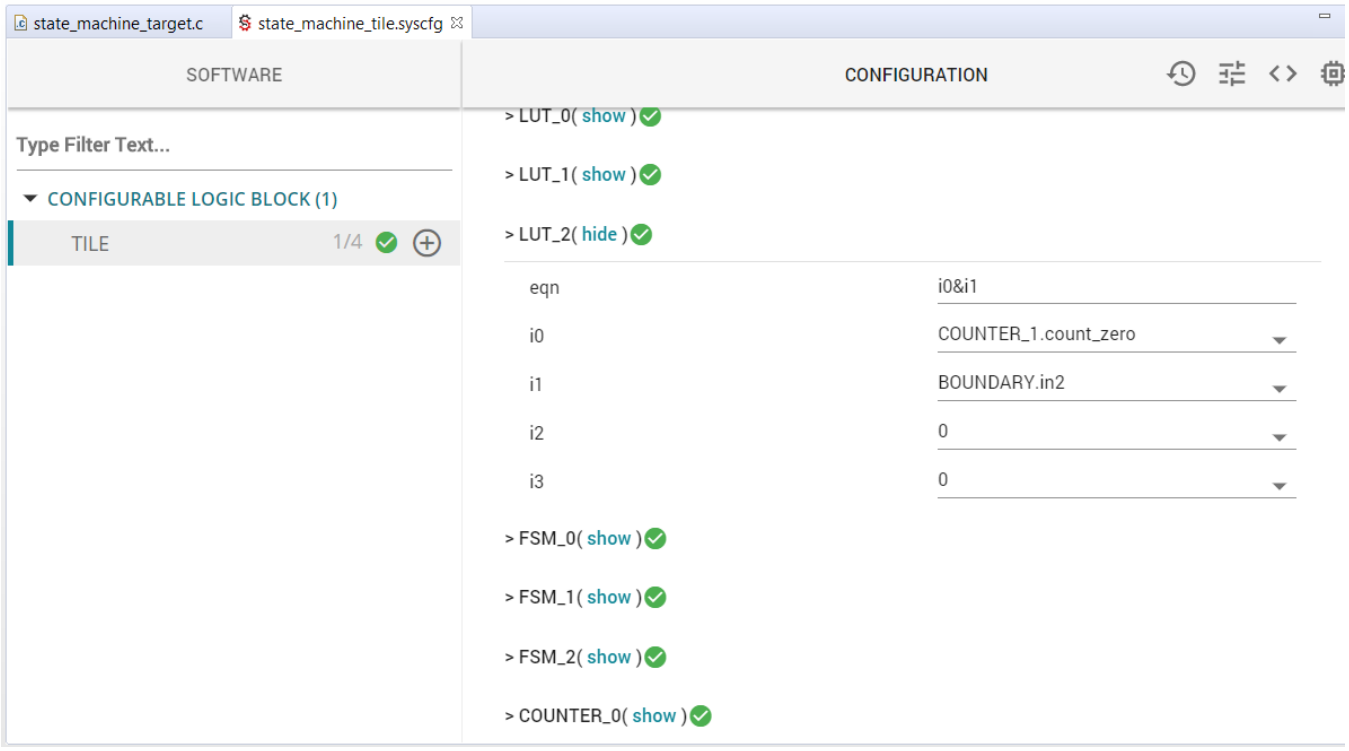

图 12. LUT2 配置

接下来，定义 HLC 的指令。C28x 内核先后使用新的周期值和新的占空比更新 FIFO。HLC 必须按同样的顺序从 FIFO 中拉取这些值。首先，HLC 必须将周期值拉到 COUNTER1 的有效计数器寄存器中（此值也会写入到将在下一步中讨论的 load_val 中）。必须将从 FIFO 拉取的下一个值写入到 COUNTER1 的 match1_val 中。不过，“拉取”指令无法将值拉到计数器子模块的 match1_val 中。唯一能够向计数器的匹配值执行写入的指令是 MOV_T1 和 MOV_T2。这两个指令分别访问 match1_val 和 match2_val。使用“拉取”指令将 FIFO 中的值拉到 HLC 的 R1 寄存器中。随后，R1 值将使用 MOV_T1 指令更新 COUNTER1 的 match1_val。

INTR 指令是 HLC 执行的最终指令。INTR 指令在 C28x 内核上生成中断，此中断向 C28x 内核表明已执行更新。FIFO 现在为空，可以再次使用。C28x 内核也会清除 GPREG BIT2，以使下一个 counter=ZERO 事件不会生成另一个 HLC 事件（导致 FIFO 为空时被读取并导致下溢）。INTR 指令需要一个被称为“中断标签”的参数。读取 C28x 内核读取此标签，以了解为何造成中断。HLC 可以在逻辑块中发生不同的事件时生成多个中断；但在此设计中，只会使用一个中断。与 INTR 指令配合使用的中断标签是“1”，C28x 内核会在处理中断服务例程时确保中断标签为“1”。

图 13 显示了 HLC 子模块的 SysConfig 配置。

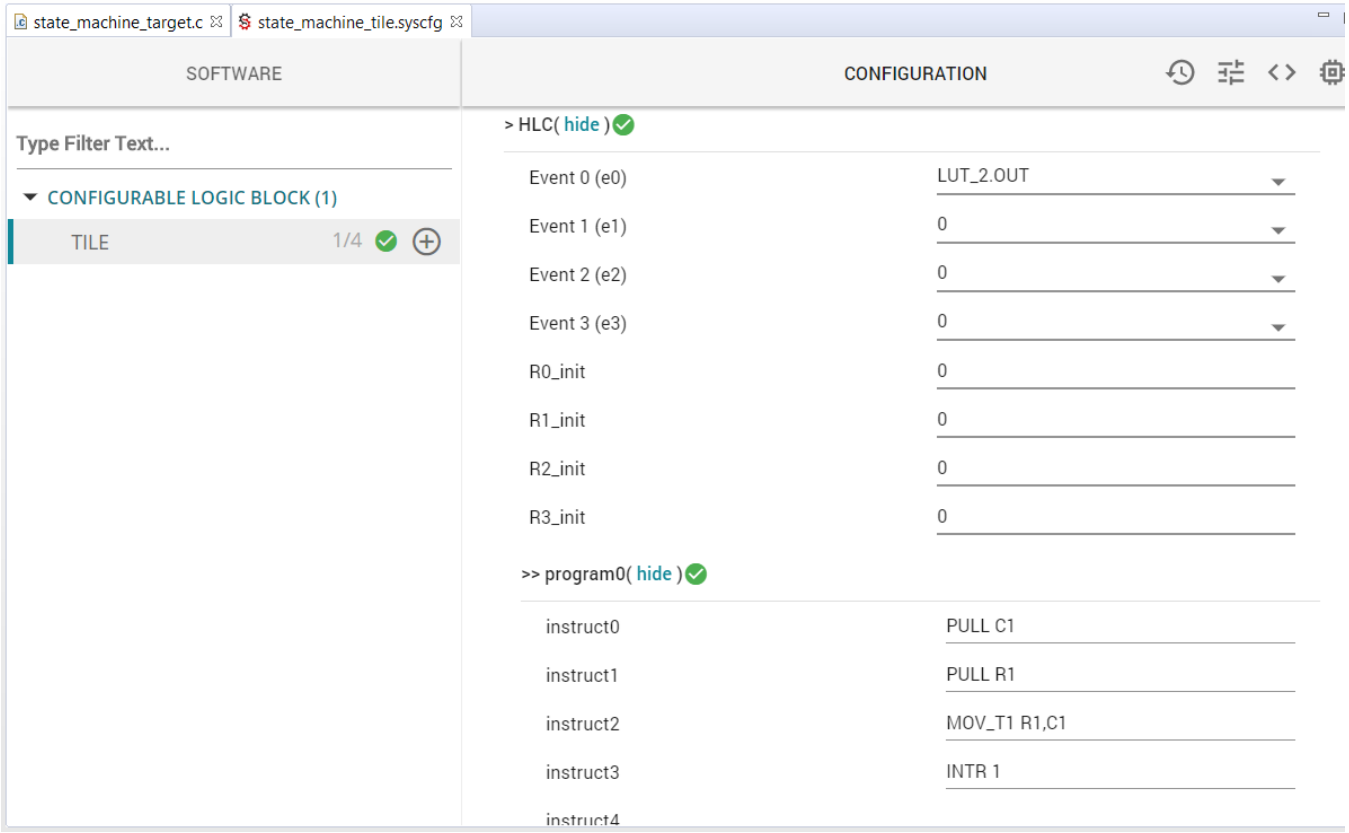

图 13. HLC 配置

在此设计的这一部分中尚未执行的最终一项操作是更新 COUNTER1 的 load_val。HLC 无法访问此寄存器，因此 C28x 必须更新此值才能设置 GPREG BIT2。如果不更新 load_val，将不会更新周期。如下代码片段展示了 C28x 内核如何更新 PWM 周期和占空比以及在收到中断之后清除 GPREG。

```
void updateClbPwm(uint32_t period, uint32_t duty)
{
if (canUpdate)
   {
       canUpdate = 0;
       CLB_writeInterface(CLB1_BASE, CLB_ADDR_COUNTER_1_LOAD, period);
       HWREG(CLB1_BASE + CLB_LOGICCTL + CLB_O_BUF_PTR) = 0U;
       HWREG(CLB1_BASE + CLB_DATAEXCH + CLB_O_PULL(0)) = period;
       HWREG(CLB1_BASE + CLB_DATAEXCH + CLB_O_PULL(1)) = duty;
       CLB_setGPREG(CLB1_BASE, 1 << TRIGGER_PWM_UPDATE_SHIFT);
   }
}
__interrupt void clb1ISR(void)
{
    uint16_t tag = CLB_getInterruptTag(CLB1_BASE);
    if (tag == UPDATE_PWM_COMPLETED_TAG)
    {
        canUpdate = 1;
        CLB_setGPREG(CLB1_BASE, 0);
    }
    CLB_clearInterruptTag(CLB1_BASE);
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP5);
}
```

9 节 说明了如何使用 XBAR 模块为 CLB 逻辑块导入和导出信号。

8 已完成的设计

图 14 显示了此设计的整个方框图。您可以利用各个子模块之间的关系了解 CLB 逻辑块要实现的整体目标。

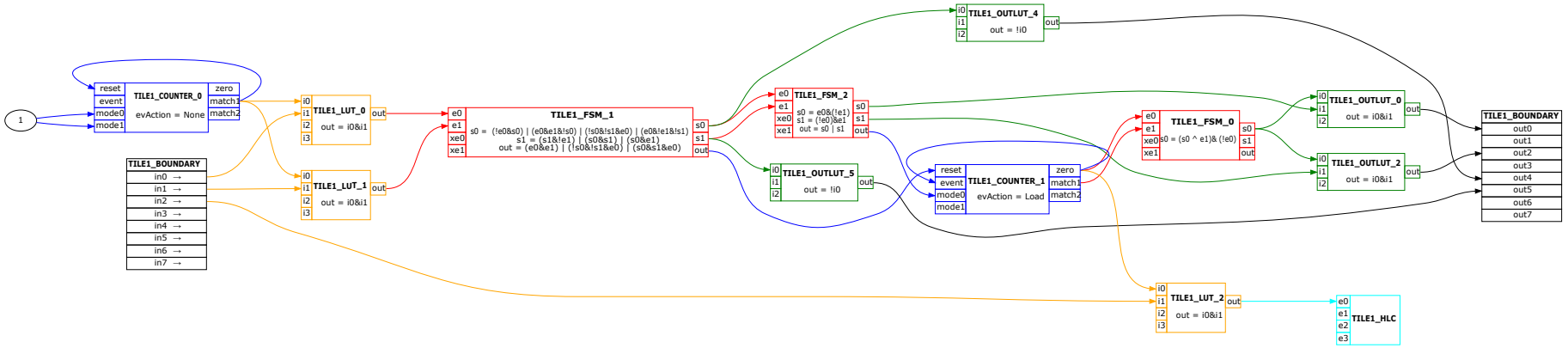

图 **14.** 已完成的设计方框图

选择要使用的子模块时要小心谨慎。例如，上述设计无法使用 FSM0 来取代 FSM1，原因在于硬件限制不允许选择 LUT1 输出作为 FSM0 e1 的输入。这是一个无效配置的示例。《TMS320F2837xD 双核 Delfino™ 微控制器技术参考手册》显示了这些无效设置的完整列表。硬件中的几组输入已经断开，以免形成逻辑环路。图 15 显示了 CLB SysConfig 工具为了向用户通报此设计错误而生成的警告。

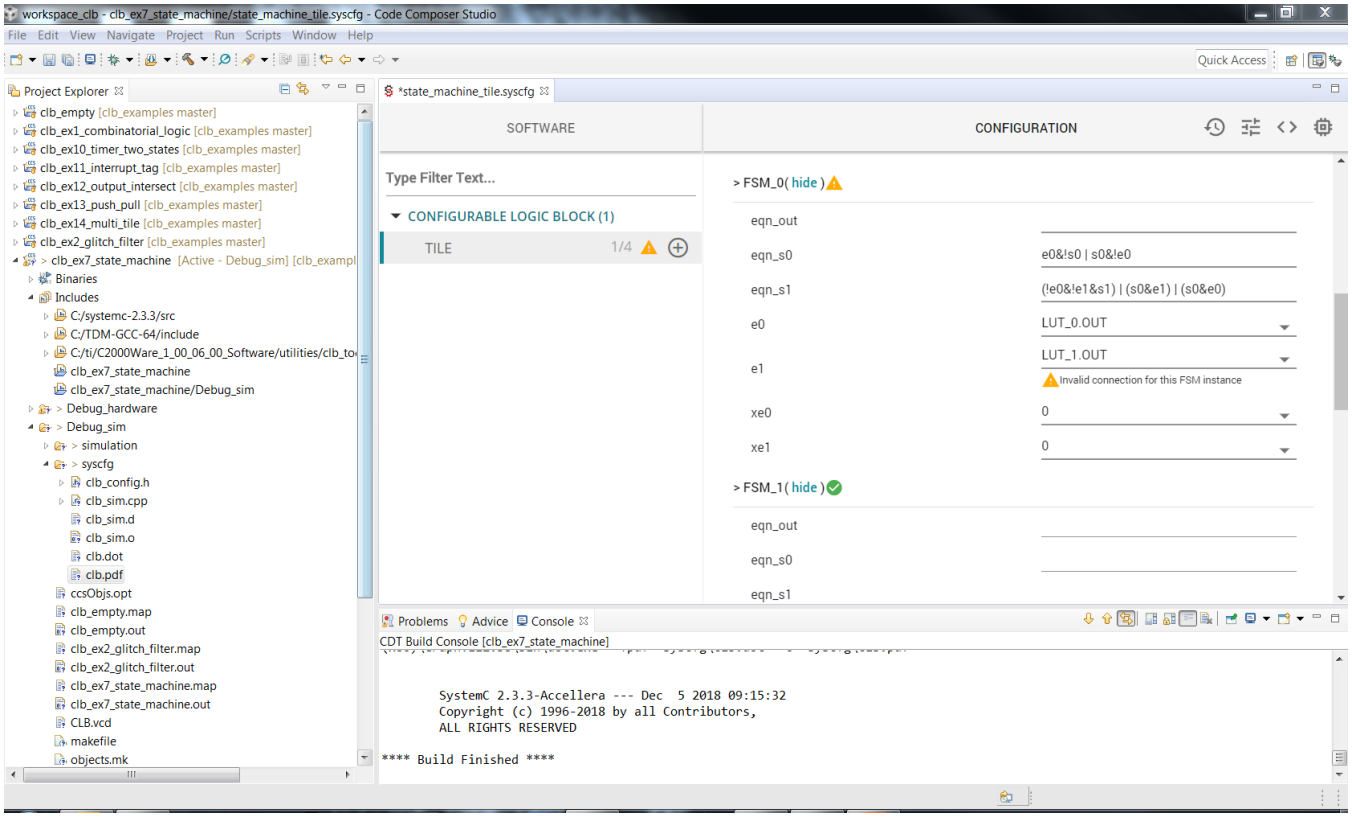

图 **15.** 此示例的无效连接

9 输入 X-BAR、输出 X-BAR 和 CLB X-BAR

可以使用 X-BAR 模块将外部信号导入到 CLB 中或者将信号从 CLB 中导出。在本节中，为本应用报告中设计的 CLB 逻辑块配置了 X-BAR 模块（输入 X-BAR、输出 X-BAR 和 CLB X-BAR）。

9.1 使用 X-BAR 将信号导入到 CLB 逻辑块中

要将信号从 GPIO 导入到 CLB（BOUNDARY INz，其中的 z 是 0 到 7 的任意数字），必须配置输入 X-BAR 和 CLB X-BAR。将信号从 GPIO 导入到 CLB 时，需要执行的步骤包括：

1. 照常配置 GPIO：
   a. 设置方向：输入/输出。
   b. 启用/禁用上拉。
   c. 设置其他 GPIO 配置。
2. 使用输入 X-BAR（例如 INPUTx，其中的 x 是 1 到输入最大数量之间的任意数字），并选择所需的 GPIO。
3. 在 CLB 模块内，为 BOUNDARY INz 选择全局输入而不是局部输入。
4. 使用 CLB X-BAR 选择输入 X-BAR 的 INPUTx 作为 AUXSIGy（其中的 y 是 0 到 7 的任意数字）。
5. 选择 AUXSIGy 作为 BOUNDARY INz 的全局输入。
6. 禁用 GPREG 输入，启用外部输入（全局输入 = AUXSIGy 输入 = INPUTx 输入 = GPIO 输入）。

图 16 显示了如何从全局输入、局部输入或 GPREG 位中选择 CLB BOUNDARY 输入。

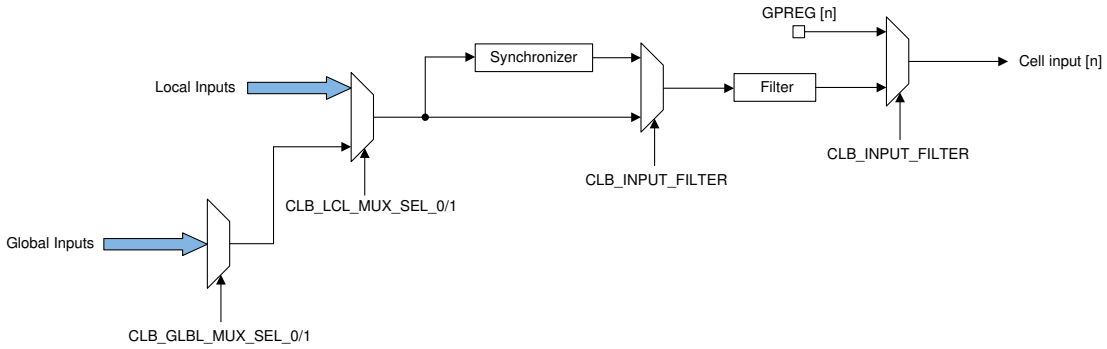

图 16. CLB BOUNDARY 输入多路复用

以下代码片段展示了如何使用器件驱动程序库来完成上述步骤。

```
//
// Configure GPIO4 for Button
//
GPIO_setPinConfig(GPIO_4_GPIO4);
GPIO_setDirectionMode(4, GPIO_DIR_MODE_IN);
GPIO_setPadConfig(4, GPIO_PIN_TYPE_PULLUP);
//
// Configure Input-XBAR INPUT1 to GPIO4
//
XBAR_setInputPin(XBAR_INPUT1, 4);
//
// Configure CLB-XBAR AUXSIG0 as INPUT1
//
XBAR_setCLBMuxConfig(XBAR_AUXSIG0, XBAR_CLB_MUX01_INPUTXBAR1);
XBAR_enableCLBMux(XBAR_AUXSIG0, XBAR_MUX01);

CLB_configLocalInputMux(CLB1_BASE, CLB_IN0, CLB_LOCAL_IN_MUX_GLOBAL_IN);
CLB_configGlobalInputMux(CLB1_BASE, CLB_IN0, CLB_GLOBAL_IN_MUX_CLB_AUXSIG0);
CLB_configGPInputMux(CLB1_BASE, CLB_IN0, CLB_GP_IN_MUX_EXTERNAL);
```

9.2 使用 X-BAR 从 CLB 逻辑块中导出信号

要将信号从 CLB (BOUNDARY OUTx) 导出到 GPIO，对于 OUT4 和 OUT5，请使用输出 X-BAR，OUT0-3 和 OUT6-7 会在 GPIO 处拦截特定外设输出。

如果使用 OUT4 和 OUT5，请使用输出 X-BAR 并执行以下步骤：

1. 照常配置 GPIO。
   a. 设置方向：输入/输出。
   b. 启用/禁用上拉。
   c. 设置其他 GPIO 配置。

2. 将 PINMUX 配置为使用输出 X-BAR。

以下代码片段展示了如何使用器件驱动程序库来完成上述步骤。

```
//
// Configure GPIO31 for OUTPUTXBAR8
//
GPIO_setPinConfig(GPIO_31_OUTPUTXBAR8);
GPIO_setDirectionMode(31, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(31, GPIO_PIN_TYPE_STD);
//
// Configure OUTPUT-XBAR OUTPUT8 as CLB1_OUT4
//
XBAR_setOutputMuxConfig(XBAR_OUTPUT8, XBAR_OUT_MUX01_CLB1_OUT4);
XBAR_enableOutputMux(XBAR_OUTPUT8, XBAR_MUX01);
//
// Configure GPIO34 for OUTPUTXBAR1
//
GPIO_setPinConfig(GPIO_34_OUTPUTXBAR1);
GPIO_setDirectionMode(34, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(34, GPIO_PIN_TYPE_STD);
//
// Configure OUTPUT-XBAR OUTPUT1 as CLB1_OUT5
//
XBAR_setOutputMuxConfig(XBAR_OUTPUT1, XBAR_OUT_MUX03_CLB1_OUT5);
XBAR_enableOutputMux(XBAR_OUTPUT1, XBAR_MUX03);
```

如果进行外设输出拦截（逻辑块 OUT0-3 和 OUT6-7）：

1. 为此特定外设照常配置 PINMUX。
   a. 示例：为 EPWM1A 配置 GPIO0。
2. 允许使用 OUT_EN 寄存器进行 CLB OUT0-3 和 OUT6-7 输出。

GPIO 的输出是 CLB OUT 而非外设输出。TRM 中的外设信号多路复用器表提到了哪个 CLB OUT 对应于哪个外设输出。以下代码片段展示了如何使用器件驱动程序库来完成上述步骤。

```
//
// Configure GPIO0 for EPWM1A which will be overriden by CLB0_OUT0
//
GPIO_setPinConfig(GPIO_0_EPWM1A);
GPIO_setDirectionMode(0, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(0, GPIO_PIN_TYPE_STD);
//
// Configure GPIO1 for EPWM1B which will be overriden by CLB0_OUT1
//
GPIO_setPinConfig(GPIO_1_EPWM1B);
GPIO_setDirectionMode(1, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(1, GPIO_PIN_TYPE_STD);
```

最后，必须启用 CLB 逻辑块输出。如下代码片段将使用器件驱动程序库完成此任务。

```
CLB_setOutputEnableMask(CLB1_BASE, 1 << 0 | 1 << 2);
```

10 运行示例项目

本节将说明如何测试我们为此系统设计的 Code Composer Studio™示例项目。我们在 TMS320F28379D LaunchPad 上测试了此示例。

10.1 设置和连接

图 17 显示了用于 BOUNDARY IN0 和 IN1 的引脚、用于正在关闭和正在打开状态的 PWM 输出以及用于显示 FSM1 子模块当前状态的 LED。

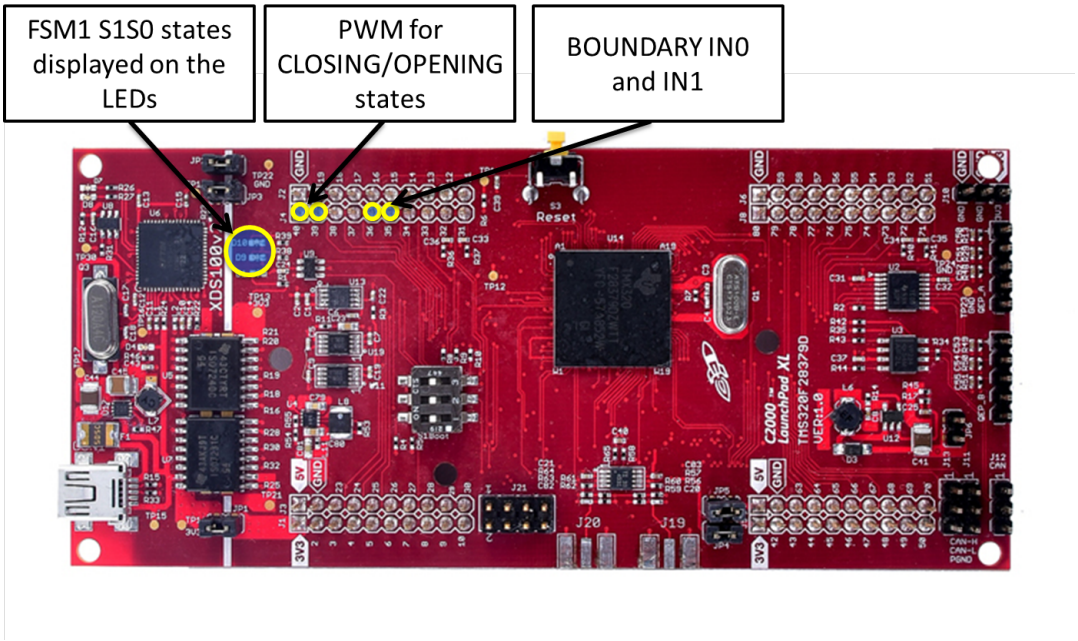

图 17. TSM320F28379D LaunchPad 连接

10.2 测试状态和转换

在运行此代码之前，使用一条跳线将 BOUNDARY IN0 和 IN1 拉到 GND。在这些 GPIO 上启用内部上拉。断开与 GND 之间的连接会将引脚拉到高电平。将示波器探头同时连接到 GPIO0 和 GPIO1，以便在系统处于正在关闭或正在打开状态时查看 PWM 信号。随后的几节将介绍如何在这些步骤之间转换，以查看 LaunchPad 上的这些更改产生的结果。

10.2.1 步骤 1

执行以下步骤，以查看从已打开状态向正在关闭状态的转换：

1. 运行内核。两个 LED 都已关闭，示波器上未显示任何 PWM 信号。
2. 断开 BOUNDARY IN0 与 GND 的连接。
   a. 状态机将从已打开状态转换为正在关闭状态。相应的 GPIO 上显示的 PWM 信号表示正在关闭。状态 LED 处于关-开状态，以显示 S1 = 0、S0 = 1。
3. 再次将 BOUNDARY IN0 连接到 GND。此更改不会产生任何变化。

10.2.2 步骤 2

执行以下步骤，以查看从正在关闭状态向已关闭状态的转换：

1. 断开 BOUNDARY IN1 与 GND 的连接。
   a. 状态机将从正在关闭状态转换为已关闭状态。PWM 信号将停止。状态 LED 处于开-开状态，以显示

S1 = 1、S0 = 1。

2. 再次将 BOUNDARY IN1 连接到 GND。此更改不会产生任何变化。

10.2.3 步骤 3

执行以下步骤，以查看从已关闭状态向正在打开状态的转换：

1. 断开 BOUNDARY IN0 与 GND 的连接。
   a. 状态机将从已关闭状态转换为正在打开状态。相应的 GPIO 上显示的 PWM 信号表示正在打开。状态 LED 处于开-关状态，以显示 S1 = 1、S0 = 0。
2. 再次将 BOUNDARY IN0 连接到 GND。此更改不会产生任何变化。

10.2.4 步骤 4

执行以下步骤，以查看从正在打开状态向已打开状态的转换：

1. 断开 BOUNDARY IN1 与 GND 的连接。
   a. 状态机将从正在打开状态转换为已打开状态。PWM 信号将停止。状态 LED 处于关-关状态，以显示 S1 = 0、S0 = 0。
2. 再次将 BOUNDARY IN1 连接到 GND。此更改不会产生任何变化。

10.2.5 步骤 5

在这最后一个测试中，输入 IN0 和 IN1 已从 GND 断开，导致系统的状态在正在关闭和正在打开之间连续跳跃。此跳跃会按采样计数器的频率切换状态和 PWM 信号。

10.3 测试 PWM 周期和占空比

最后要测试的是在运行时更改 PWM 周期和占空比的能力。执行以下步骤，以测试此功能：

1. 运行此代码并打开表达式窗口。
2. 添加 clbPwmUpdateNow、clbPwmDuty 和 clbPwmPeriod。
3. 启用自动刷新。

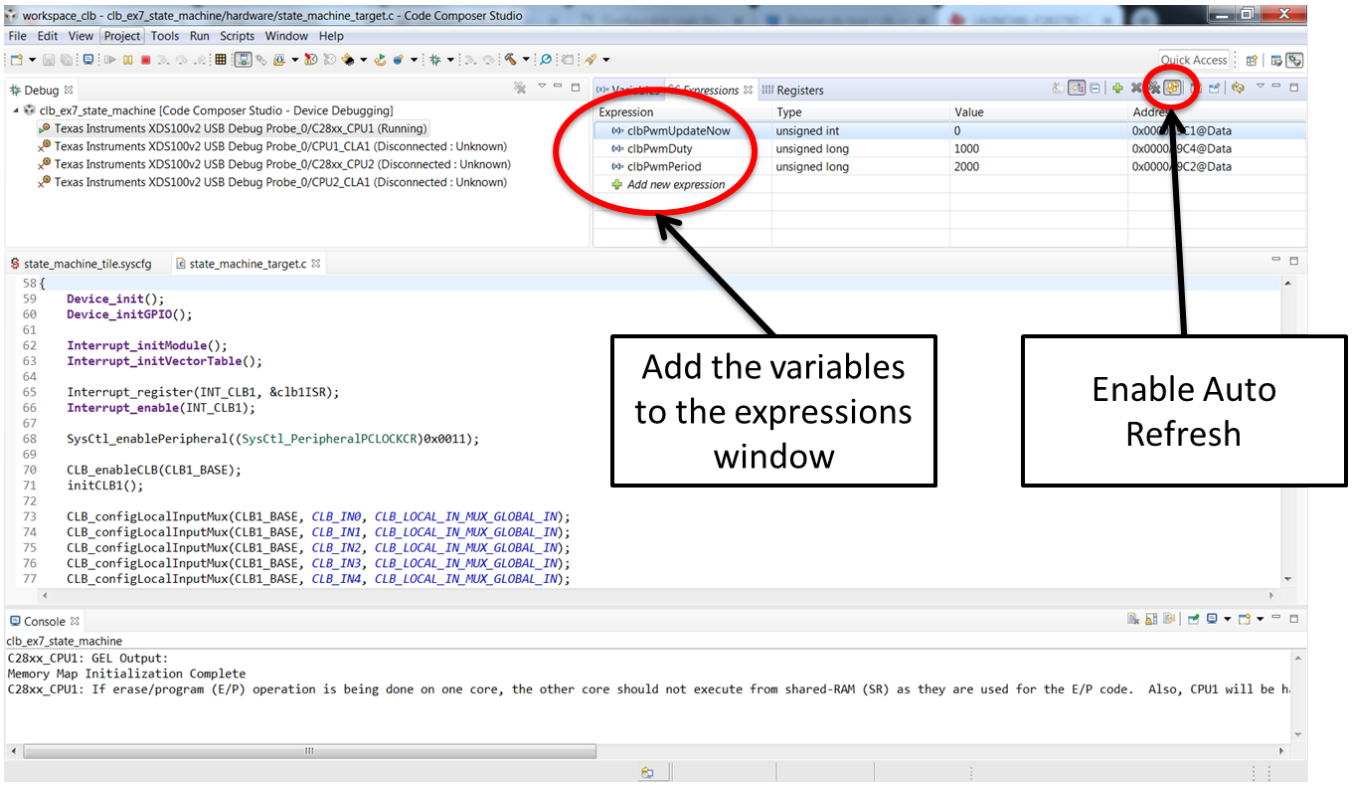

图 **18.** 使用表达式窗口更改 **PWM** 周期和占空比

4. 单击 clbPwmDuty 和 clbPwmPeriod 的值，并将它们更新为新值。
5. 单击 clbPwmUpdateNow 的值，并写入值“1”。
6. PWM 会更新，并显示在示波器上。

图 19 和图 20 显示了一个将 PWM 占空比设置为 1000、将周期设置为 2000 的示例。

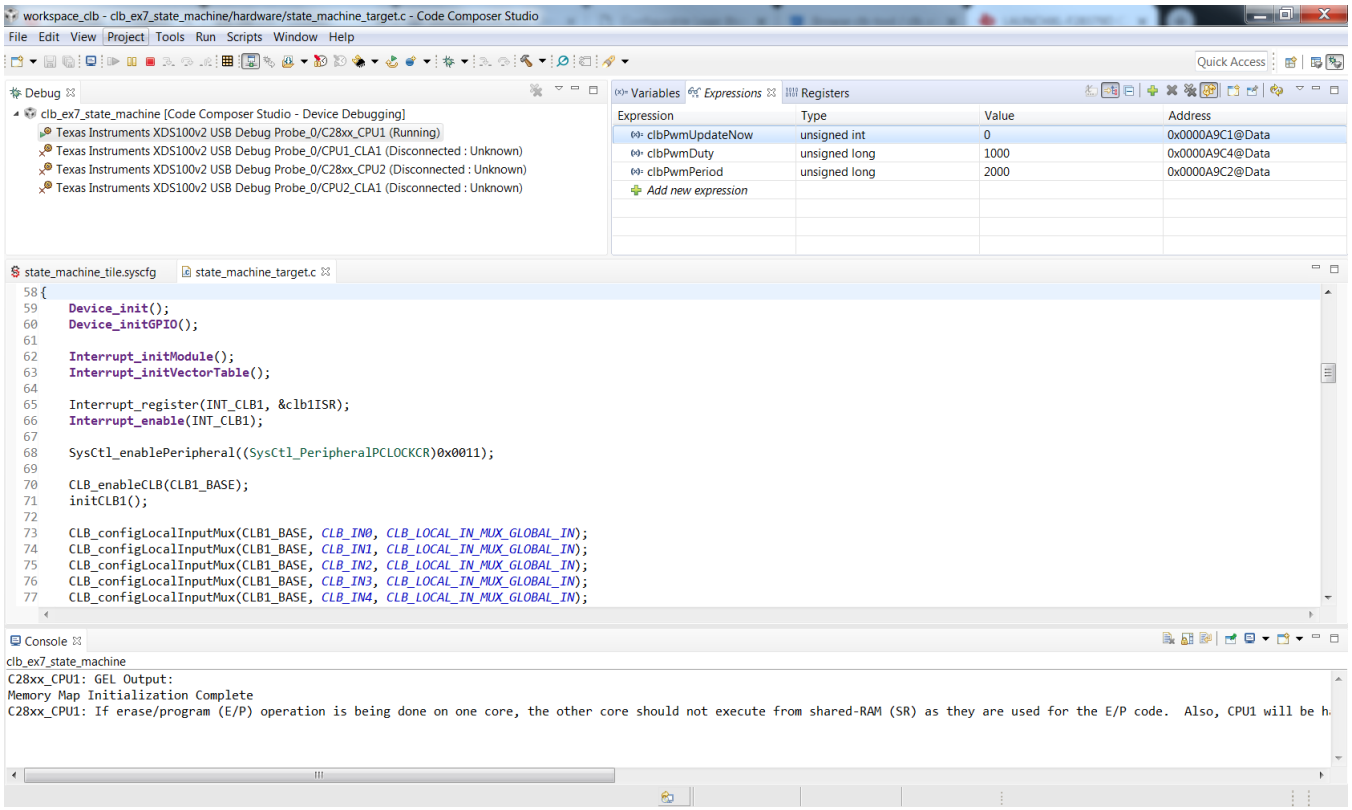

图 **19. PWM** 占空比为 **1000**，周期为 **2000**

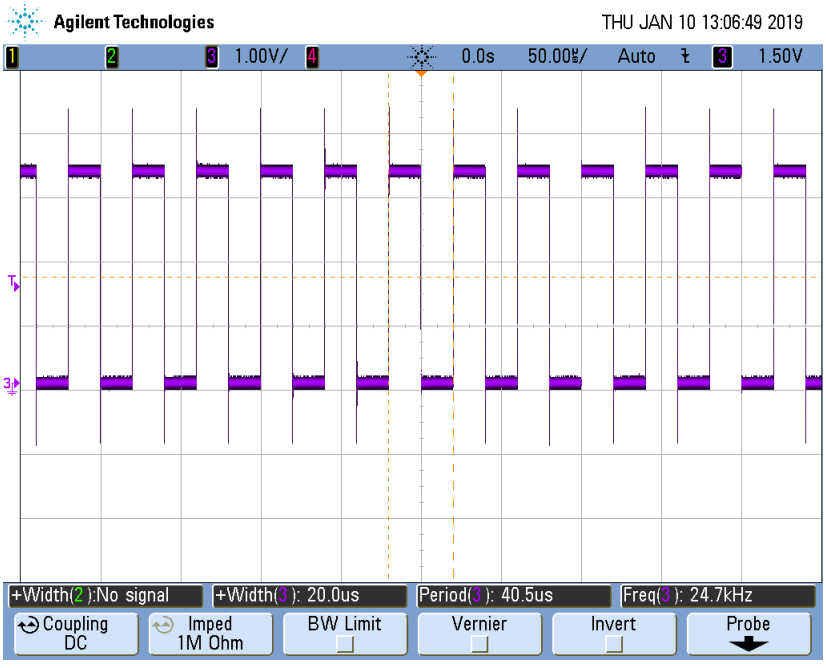

图 **20.** 示波器 **- PWM** 占空比为 **1000**，周期为 **2000**

图 21 和图 22 显示了一个将 PWM 占空比设置为 3000、将周期设置为 4000 的示例。

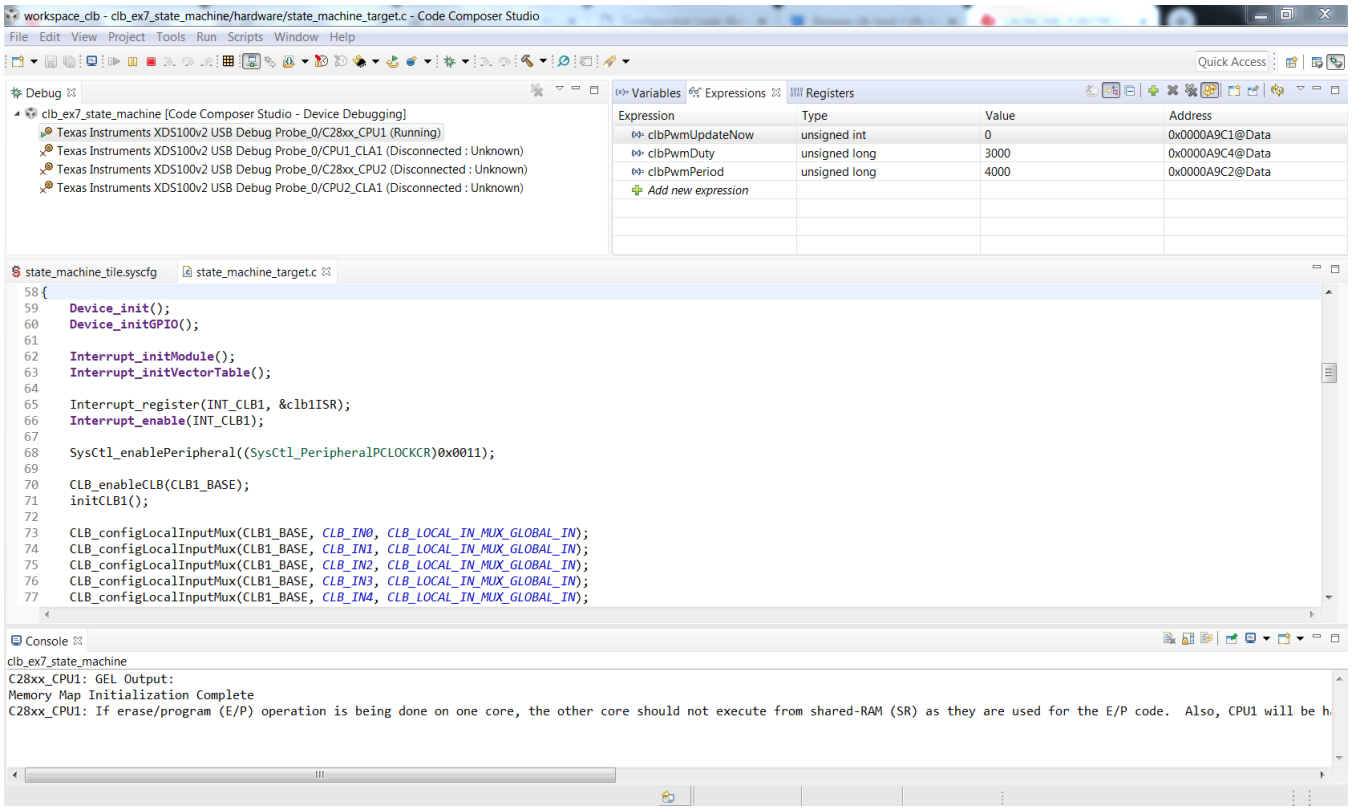

图 **21. PWM** 占空比为 **3000**，周期为 **4000**

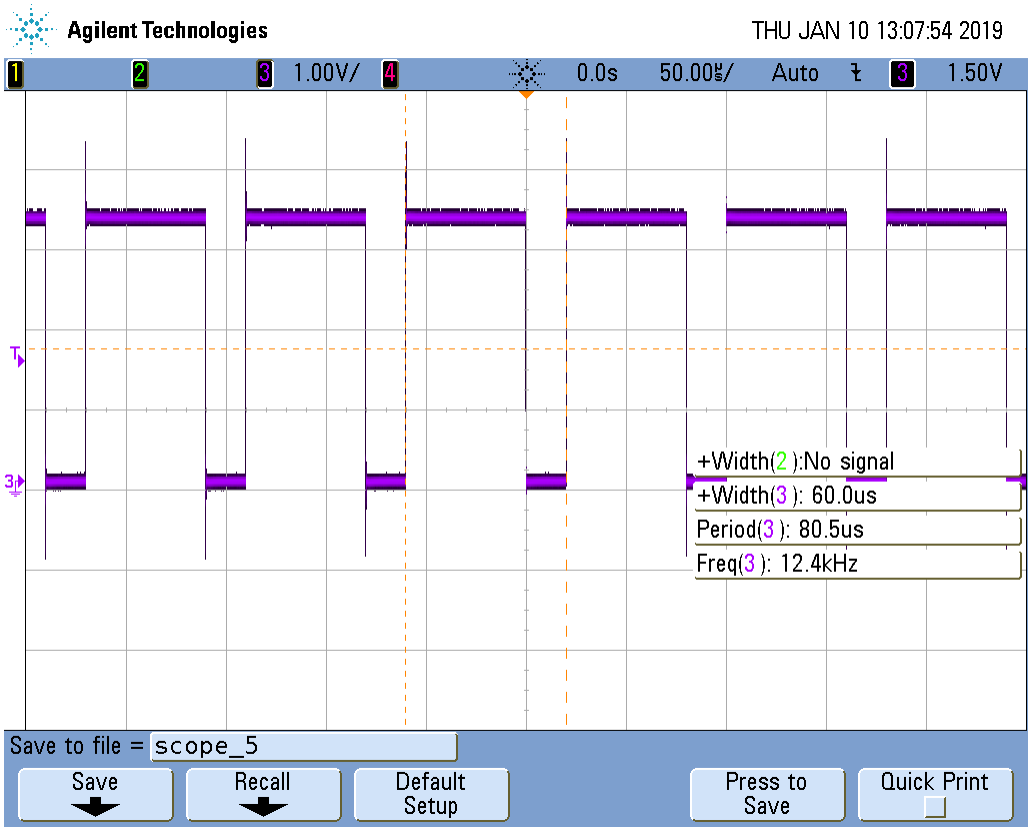

图 **22.** 示波器 **- PWM** 占空比为 **3000**，周期为 **4000**

11 总结

CLB 模块是一个功能强大、高度灵活的外设。它可以创建极其强大的设计，这些设计既可以独立运行，也可以用作对 C28x 内核的补充。还展示了 CLB Sysconfig 工具的使用方法以及 XBAR 模块为 CLB 模块导入和导出信号时所需的配置。

12 参考文献

- *TMS320F2837xD 双核 Delfino™ 微控制器技术参考手册*
- *TMS320F28004x Piccolo 微控制器技术参考手册*

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