Application Note

C2000™ Position Manager PTO API 参考指南

Ozino Odharo, Subrahmanya Bharath, Peter Luong and Lori Heustess

摘要

脉冲序列输出 (PTO) 和脉冲序列输入 (PTI) 是描述各种形式的信号脉冲流的通用名称。PTO 库 API 利用 C2000 可配置逻辑块 (CLB) 来协助处理各种 PTO 信号和 PTI 信号。这包括生成输出信号以及对输入信号（例如 QEP、CwCCW、脉冲和方向）进行解码。

内容

插图清单

English Document: SPRAC77

表格清单

商标

1 引言

C2000 实时 MCU 脉冲序列输出 (PTO) API 利用类型 1 或更高版本的可配置逻辑块 (CLB)，来生成指定的 PTO 或解码 PTI（脉冲序列输入）。

备注

一些 API 适用于脉冲序列输入 (PTI)，另外一些则适用于脉冲序列输出 (PTO)。为简单起见，示例、库和目录结构均使用后缀“pto”来标识属于此库的内容。

本文档介绍了用于下列每个模块的实现和相关软件：

PulseGen：输出一个简单脉冲和一个方向指示信号。

QepDiv：调节正交编码脉冲输入（QEP-A、QEP-B 和 QEP-Index）以输出减频的 PTO 信号。

Abs2Qep：将绝对位置的变化转换为等效的 QEP-A/B 和 QEP-I 信号。

QepOnClb 使用 CLB 实现基本 QEP 解码器。

提供两类软件：

应用程序工程示例：小型应用程序，用于配置 C2000 实时 MCU、合并适当的参考库并演示功能。节 6 介绍了如何访问源代码，将工程导入 CCS，然后构建和运行该示例。

参考 API 库：模块的软件实现。节 7 包括每个 API 的说明、如何访问源代码以及如何重新构建库。节 8 说明了如何在您的工程中使用 API。

备注

您将需要安装相应的开发工具来构建基于 CLB 的工程。如需更多信息，请参阅 *CLB 工具用户指南*。

2 PTO – PulseGen

PTO-PulseGen 函数可用于生成应用所需的脉冲和方向输出。图 2-1 所示为 PulseGen 输出，图 2-2 所示为实现图。

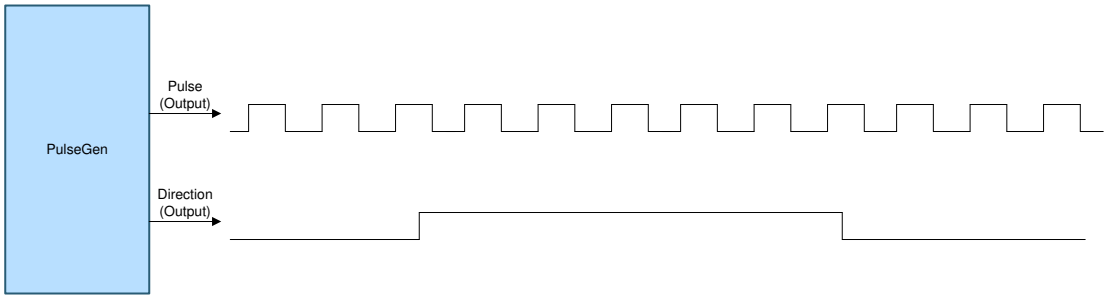


图 2-1. PulseGen 输出图

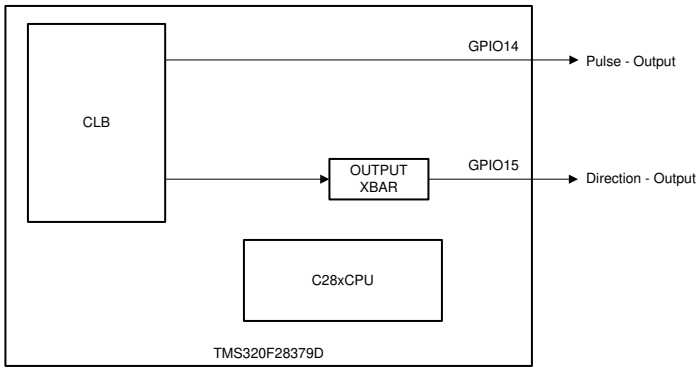


图 2-2. PulseGen 实现图

备注

CLB 和 MCU 边界之间的互连可能因器件或示例而异。具体的互连路由信息，请参阅节 6。

2.1 PulseGen 实现概述

本节概述了如何实现 PTO-PulseGen 接口。此接口由以下组件实现：

- **C28x CPU**
  - 初始化 PulseGen 接口，配置 CLB、XBAR 和 GPIO。
  - 向 CLB 提供脉冲数和每个脉冲的持续时间。

- **可配置逻辑块 (CLB) 类型 1 或更高版本**
  - 生成由软件接口函数定义的脉冲和方向。
- **器件互连 (XBAR)**
  - 根据需要进行配置以实现往返于 CLB 的输出信号路由。

2.2 PulseGen 限制

PTO-PulseGen 操作具有以下使用限制：

- PTO 周期的最小周期数必须为 1000 个周期（在 200MHz 系统时钟下，这一数值对应于 10µs [例如，100KHz]）。
- 为避免与 PTO 更新发生冲突，中断时间的周期数必须介于 PTO 周期的 40% 到 60% 之间。
- PTO-PulseGen 输出的最大频率在 200MHz CPU CLK 时为 5MHz。请参阅 C2000Ware MotorControl SDK 中提供的示例。

2.3 PulseGen CLB 配置

若要实现节 2.1 中描述的所需功能，需要在 CLB 逻辑块中使用以下资源。

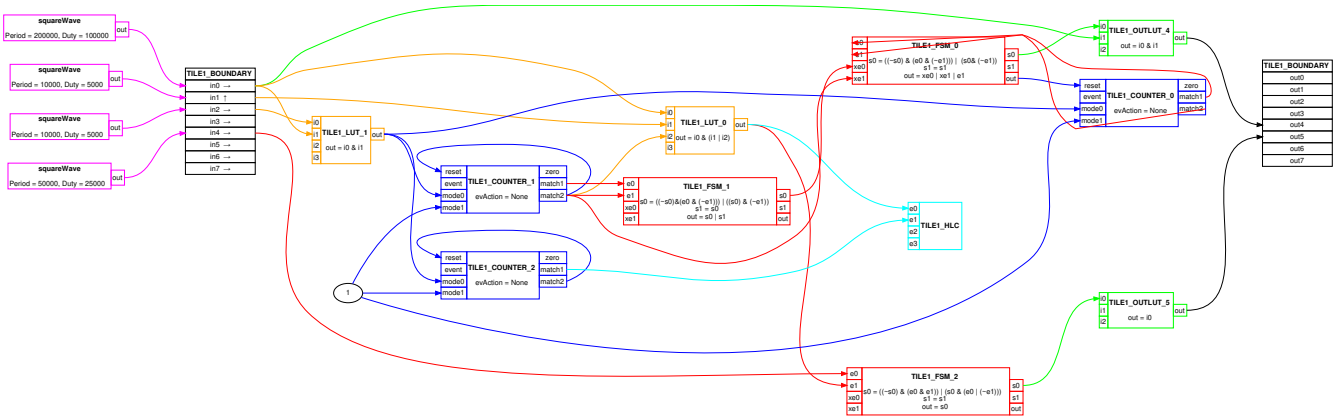


图 2-3. PulseGen CLB 逻辑块图

备注

节 7 介绍了如何在 Code Composer Studio™ 中构建库工程。通过构建工程，CCS 将重新生成 CLB 逻辑块图（clb.svg 或 clb.html）和对象 (.lib)。CLB 逻辑块图将位于 `RELEASE/syscfg` 目录下。

具体实现方式请参阅表 2-1 中的说明和图 2-3 中的图示。

表 2-1. PulseGen CLB 逻辑块 1

资源	功能	注意事项
输入		
In0	开/关控制（通过 GPREG）	启用 CLB
In1	上升沿检测	通过 EPWM1A
In2	开/关控制（通过 GPREG）	运行信号（PTO 启动/停止）
In3	未使用	未使用
In4	开/关控制（通过 GPREG）	设置 PTO 方向
In5	未使用	未使用
In6	未使用	未使用
In7	未使用	未使用
输出		
Out0	未使用	未使用
Out1	未使用	未使用
Out2	未使用	未使用

表 2-1. PulseGen CLB 逻辑块 1 (continued)

资源	功能	注意事项
Out3	未使用	未使用
Out4	发送使能	通过输出 XBar；PTO 脉冲输出
Out5	发送使能	通过输出 XBar；PTO 方向输出
Out6	未使用	未使用
Out7	未使用	未使用
逻辑资源		
LUT0	HLC 中 Event0 的输入	使用 in1 或 CNT1 匹配值对编码器输入进行边沿检测。在 HLC 中触发事件以将新值加载到 HLC 寄存器中
LUT1	CNT 1、2、3 的 Mode0 输入	用于确定 CNT1、CNT2 和 CNT3 所选模式的逻辑。启动所有三个计数器。
LUT2	未使用	未使用
FSM0	脉宽生成	该状态机将与 CNT0 一同生成若干高和低脉冲宽度。输出将设置 CNT0 的复位值。
FSM1	活跃周期和完整周期生成	根据 CNT1 的 match1 和 match2 输出来设置活跃周期和完整周期的值。输出活跃周期持续时间内的脉冲数，并且在完整周期和活跃周期差值之间不输出脉冲数
FSM2	PTO 输出方向生成	生成 PTO 输出方向。输出方向一直保持到由 FSM1 设置的完整周期结束。
CNT0	脉宽生成	计数器 Match1 和 Match2 值决定了高脉冲宽度和低脉冲宽度的触发器。将匹配值加载到 FSM0 输入 e0 和 e1。
CNT1	活跃周期和完整周期时钟生成	生成 FSM1 和 FSM2 所需的输入。Match1 决定了活跃期间的触发器。Match2 决定了完整期间的触发器。FSM1 使用匹配事件来生成活跃周期和完整周期。Match2 用作 FSM0 中额外的外部输入，以确定保持 PTO 输出方向的时间。
CNT2	完整周期计数器	Match1 事件用于触发 HLC 中的中断。当信号的完整周期达成后将重置计数器
高级控制器		
HLC	Event0 用于触发任务，Event1 用于触发中断	Event0 用于将 PTO 的新选项从 C28 内核加载到 CLB，Event1 用于根据 CNT2 的 match1 事件生成中断（对应于完整周期）。新的 PTO 选项在此事件后生效。

2.4 PulseGen 输入和输出信号

PTO-PulseGen 接口的芯片级输入：无。

PTO-PulseGen 接口的芯片级输出：脉冲输出和方向。在提供的示例中，这些输出将路由至 GPIO，如节 6 所述。

3 PTO – QepDiv

QepDiv PTO 函数可用于从 QEP 输入生成分离的脉冲流。图 3-1 所示为 QepDiv 输入和输出图。

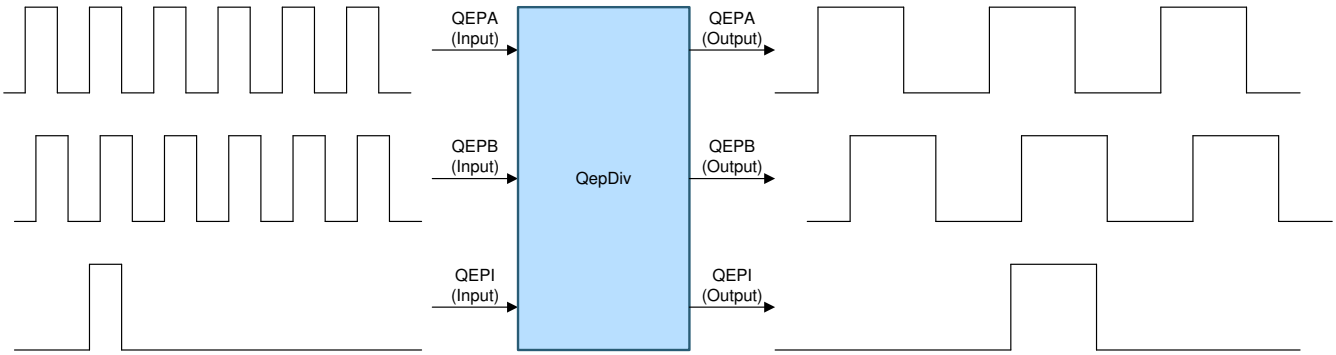


图 3-1. QepDiv 输入和输出图

图 3-2 所示为 CLB 互连图。

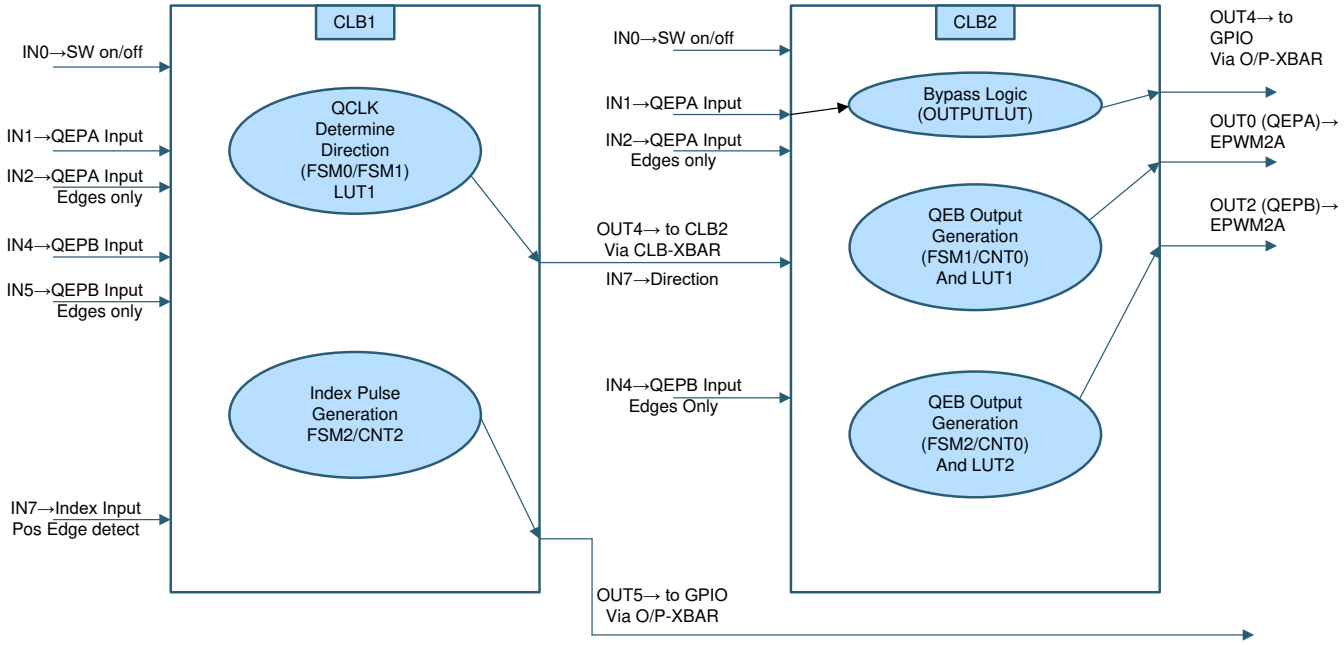


图 3-2. QepDiv 互连图

图 3-3 所示为 QepDiv 接口的实现图。

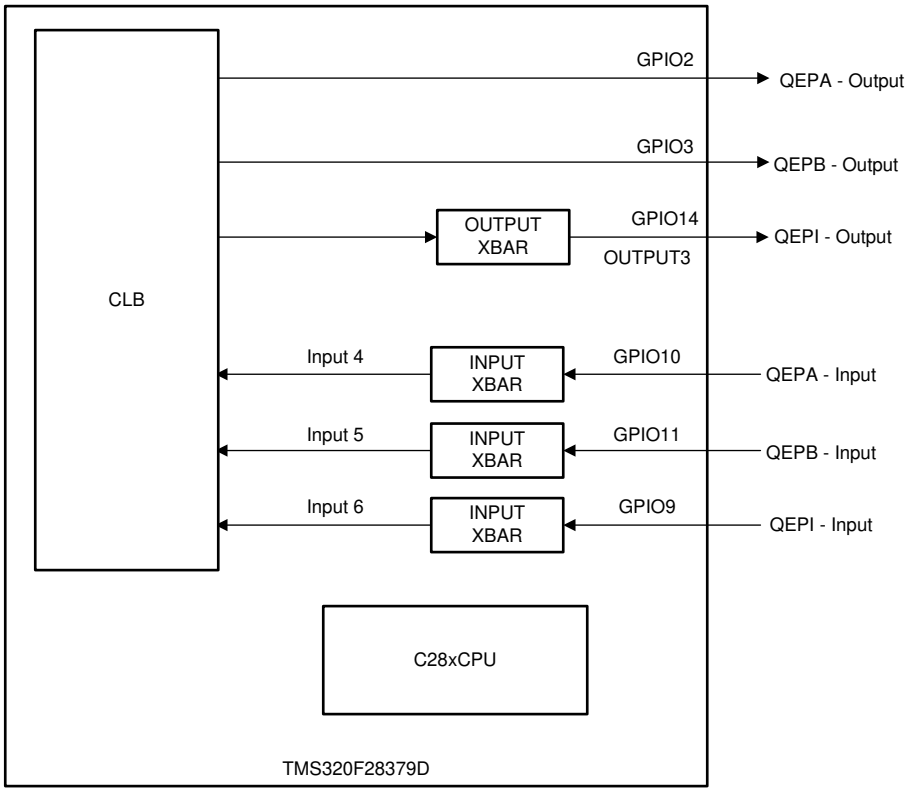


图 3-3. 实现图

备注

CLB 和 MCU 边界之间的互连可能因器件或示例而异。有关具体的互连路由信息，请参阅节 6。

3.1 QepDiv 实现概述

本节概述了如何实现 PTO QepDiv 接口。此接口主要由以下组件实现：

- **C28x CPU**
  - 此 CPU 可根据需要初始化 CLB、XBAR 和 GPIO 的功能和配置。
- **可配置逻辑块 (CLB) 类型 1 或更高版本**
  - 监测连接到 GPIO 的输入信号 QEP-A、QEP-B 和 QEP-I
  - 检测运动方向以及任何的方向变化
  - 检测输入信号的边沿
  - 实现分频功能并生成调节的输出：PTO-QEP-A、PTO-QEP-B 和 PTO-QEP-I。
- **器件互连 (XBAR)**
  - 输入和输出 XBAR 用于在适用的情况下将信号传入和传出 CLB。

3.2 QepDiv 限制

PTO-QepDiv 操作具有以下使用限制：

- QepDiv 接口可实现 /1、/2、/4、/8... 一直到 /1024 和 /2048 的分频因子。
- 输入信号（QEP-A 和 QEP-B）的最大频率限制为 5MHz。
- 检测到索引输入信号的上升沿时，在索引输出端将生成索引脉冲。
- 索引脉冲的宽度可由用户定义。请参阅 C2000Ware MotorControl SDK 中提供的 `pto_qepdiv_config` 函数和相应示例。
- 分频器值的计算如下：
  - 输出 QEP-A 或 QEP-B 的频率 = 输入 QEP-A 或 QEP-B 的频率 / (2 × 分频器值)

3.3 QepDiv 分频器设置和初始化

分频器的初始化通过以下函数完成：

- CLB2 中的 COUNTER_0 用于计数器 match2 值的 divider*4。
- CLB2 中的 COUNTER_0 用于计数器 match1 值的 divider*2。

使用 CLB1 中的 COUNTER_2 控制索引脉冲宽度以设置 match1 值。

```
uint16_t
pto_qepdiv_config(uint16_t divider, uint16_t indexWidth)
{
    CLB_writeInterface(CLB2_BASE, CLB_ADDR_COUNTER_0_MATCH2, divider * 4);
    CLB_writeInterface(CLB2_BASE, CLB_ADDR_COUNTER_0_MATCH1, divider * 2);
    CLB_writeInterface(CLB1_BASE, CLB_ADDR_COUNTER_2_MATCH1, indexWidth - 1);
    return(divider);
}
```

3.4 QepDiv CLB 配置

PTO API 实现的源文件位于 `[C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\source` 下。

若要实现节 3.1 中描述的所需功能，需要在 CLB 逻辑块中使用以下资源。

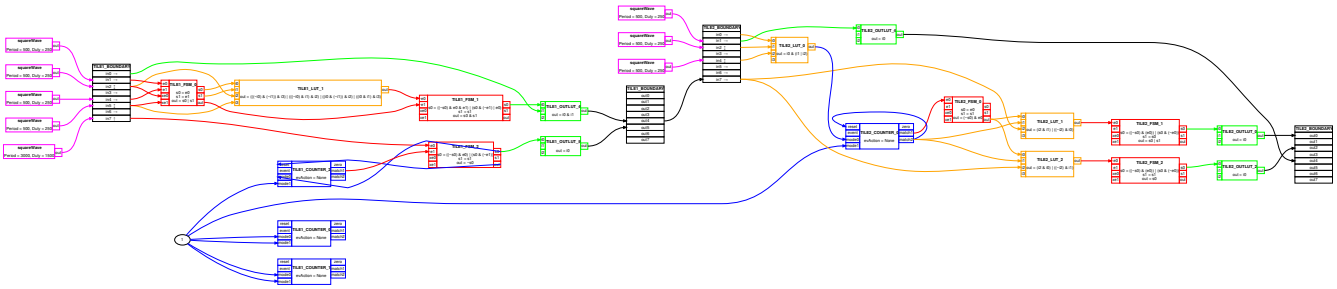


图 3-4. QepDiv CLB 逻辑块图

备注

您可以为每个相应的器件导入并构建 QepDiv API 参考工程（位于 `[C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\ccs` 中）。通过重新构建编译的对象，将重新生成 CLB 逻辑块图（clb.svg 或 clb.html）和对象 (.lib)。CLB 逻辑块图将位于 `RELEASE/syscfg` 目录下。

具体实现请参阅以下详细说明和图 3-4 中的图示。

表 3-1. QepDiv CLB 逻辑块 1

资源	功能	注意事项
		输入
In0	开/关控制（通过 GPREG）	启用 CLB
In1	开/关控制（通过 GPREG）	QEPA（通过 EPWM4A）
In2	边沿检测	QEPA（通过 EPWM4A）
In3	未使用	未使用
In4	开/关控制（通过 GPREG）	QEPB（通过 EPWM5A）
In5	边沿检测	QEPB（通过 EPWM5A）
In6	未使用	未使用
In7	边沿检测	QEPI（通过 EPWM4B）
		输出
Out0	未使用	未使用
Out1	未使用	未使用
Out2	未使用	未使用
Out3	未使用	未使用
Out4	发送使能	PTO 方向（通过输出 XBar）；CLB 2 的输入
Out5	发送使能	QEPI 输出（通过 OUTPUTXBAR3）
Out6	未使用	未使用
Out7	未使用	未使用
		逻辑资源
LUT0	未使用	未使用
LUT1	结合 FSM0 和 FSM1 以确定 QCLK 方向	为 FSM1 提供外部输入 0 的输入
LUT2	未使用	未使用
FSM0	QEPA 和 QEPB 之间的交替输入	该状态机检查 QEP 信号并在不同信号之间交替
FSM1	设置 QCLK 方向	使用 LUT1 和 FSM0 的输出来设置 QCLK，进而设置方向。输出将路由到 CLB2 作为输入方向
FSM2	索引脉冲生成	接受 QEPI 输入并使用 CNT2 Match2 值来设置 QEPI 输出周期和占空比。
CNT0	设置索引脉冲宽度值	加载通过 CLB_writeInterface 函数设置的 indexWidth-1 值
CNT1	设置分频器值	加载通过 CLB_writeInterface 函数设置的 divider*4 值
CNT2	设置分频器值	加载通过 CLB_writeInterface 函数设置的 divider*2 值
		高级控制器
HLC	未使用	未使用

表 3-2. QepDiv CLB 逻辑块 2

资源	功能	注意事项
	输入	
In0	开/关控制（通过 GPREG）	启用 CLB
In1	开/关控制（通过 GPREG）	QEPA（通过 EPWM4A）
In2	边沿检测	QEPA（通过 EPWM4A）
In3	未使用	未使用
In4	边沿检测	QEPB（通过 EPWM5A）
In5	未使用	未使用
In6	未使用	未使用
In7	开/关控制（通过 GPREG）	从 CLB1 out4 路由的 PTO 方向
	输出	
Out0	发送使能	QEPA 输出（通过 EPWM2A）
Out1	未使用	未使用
Out2	发送使能	QEPB 输出（通过 EPWM2B）
Out3	未使用	未使用
Out4	发送使能	旁路逻辑
Out5	未使用	未使用
Out6	未使用	未使用
Out7	未使用	未使用
	逻辑资源	
LUT0	QEPA/QEPB 信号输入	当逻辑块为开启状态时，将所选的 QEP 信号发送到 CNT0 作为 mode0 输入
LUT1	生成高低值	高低交替
LUT2	未使用	未使用
FSM0	QEP 脉宽生成	该状态机将与 CNT0 一同为 LUT1 和 LUT2 生成若干高和低脉冲宽度。
FSM1	QEPA 信号生成	使用 CNT0 和 LUT1 生成 QEPA 输出。
FSM2	QEPB 信号生成	使用 CNT0 和 LUT2 生成 QEPB 输出。
CNT0	用于输出 QEP 信号生成的计数器	计数器 Match1 值是 FSM0 的外部输入。Match2 值是计数器的复位值。match2 值传递给 LUT1 和 LUT2。
CNT1	未使用	未使用
CNT2	未使用	未使用
	高级控制器	
HLC	未使用	未使用

4 PTO – Abs2Qep

PTO-Abs2Qep 函数会将绝对位置的变化转换为正交编码器脉冲序列输出。图 4-1 所示为 PTO-Abs2Qep 接口的实现图。

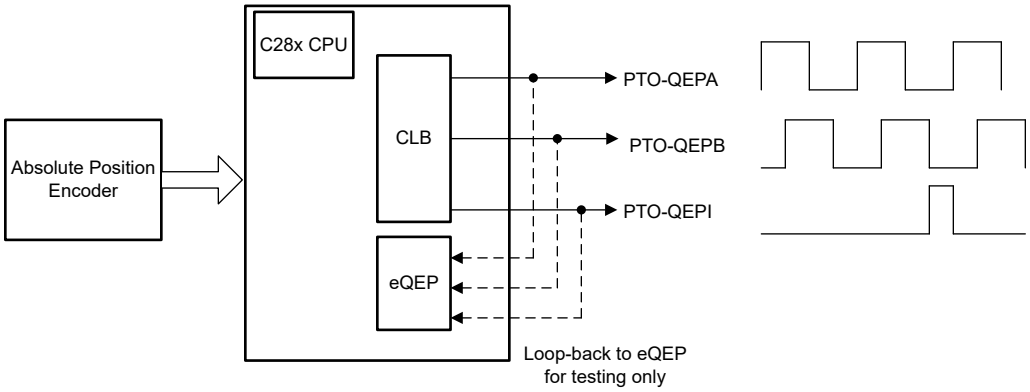


图 4-1. Abs2Qep 实现图

4.1 Abs2Qep 芯片资源

Abs2Qep 实现方案中使用了以下 C2000 资源：

- **C28x CPU**
  - 初始化 Abs2Qep 接口，配置 CLB、输入/输出 XBAR 和 GPIO。
  - 将绝对位置的变化转换为等效的 QEP-A/QEP-B 和 QEP-I 脉冲。
  - 配置 CLB 以生成脉冲序列输出。
- **可配置逻辑块 (CLB) 类型 1 或更高版本**
  - 生成由 C28x 定义的 PTO-QEP-A/B 和 QEP-I 脉冲。
  - 通过设置一个 CLB 中断标签来指示脉冲序列已完成。
- **器件互连 (XBAR)**
  - 输入和输出 XBAR 用于在适用的情况下将信号传入和传出 CLB。

4.2 Abs2Qep 工作原理

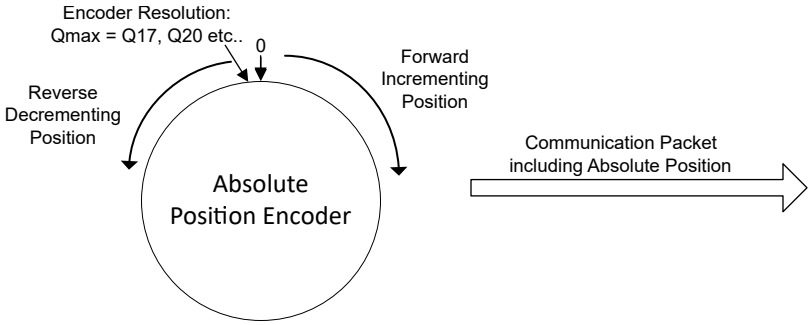


图 4-2. 绝对位置编码器

绝对编码器输出表示旋转轴的准确位置。如果 Qmax 是单次旋转的分辨率，则位置范围将是从 0 到 Qmax。Q17 = 2^{17} 或 Q20 = 2^{20} 范围内的分辨率很常见。当方向为正向（顺时针）时绝对位置增大，当方向为反向（逆时针）时绝对位置减小。

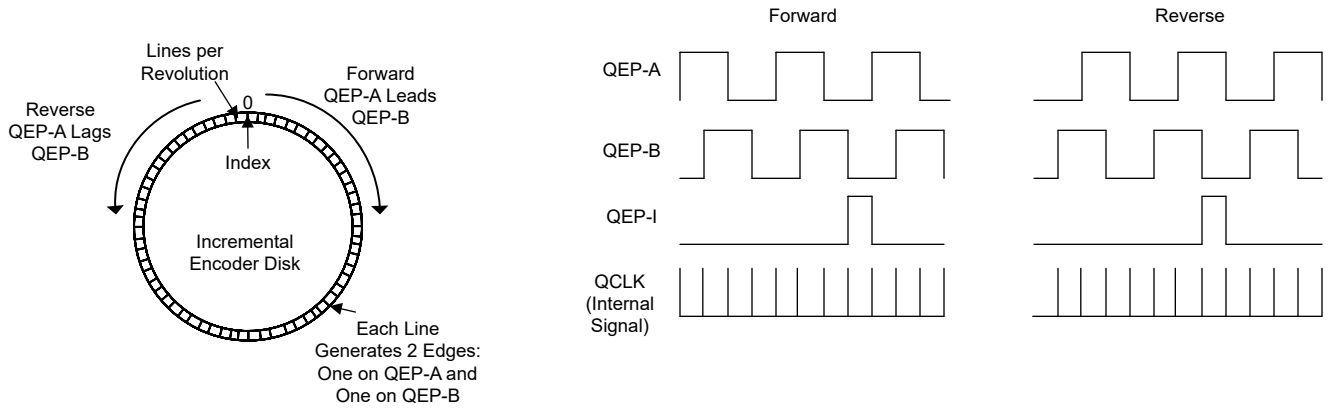


图 4-3. 增量位置编码器

增量编码器的输出是正交编码器脉冲 (QEP)。此脉冲序列包含以下输出：QEP-A、QEP-B 和 QEP-I，这些输出具有以下特性：

- QEP-A 和 QEP-B 之间的相位指示运动的方向。如果 QEP-A 领先 90°，则方向为正向（顺时针）。如果 QEP-A 滞后 90°，则方向为反向（逆时针）。
- QEP-A/B 频率与码盘速度成正比。
- 索引信号 QEP-I 表示过绝对零。

增量编码器的分辨率由码盘周围的线数决定。当每条线通过传感器时，QEP-A 上都会产生一个边沿（下降沿或上升沿），如图 4-3 所示。可通过第二个线环添加第二个通道（位于外环内部并偏离外环）。在这种情况下，该内部线环可以生成 QEP-B。例如，一个 1024 线编码器将有 1024 条 QEP-A 线和 1024 条 QEP-B 线，因此在一次完整旋转中总共有 2048 个 QEP 状态变化。

在 Abs2Qep 中，头文件中的一个可配置参数将定义每条线产生多少个 QEP 状态转换。QEP 状态转换由被称为 QCLK 的内部 CLB 信号进行控制，如图 4-3 所示。默认设置是每条线对应两个 QCLK 脉冲。

表 4-1 中的示例进一步阐明了这一点。

表 4-1. Abs2Qep 中线和 QCLK 之间的关系（正向）

线路	QCLK	QEP-A	QEP-B
1 号线外环	QCLK 1	上升沿	
1 号线内环	QCLK 2		上升沿
2 号线外环	QCLK 3	下降沿	
2 号线内环	QCLK 4		下降沿

4.2.1 Abs2Qep 转换公式

所有 Abs2Qep 转换计算均由 C28x 处理。根据结果，随后将配置 CLB 逻辑块以生成特定的 QEP 信号。节 4.3 详细介绍了 CLB 配置。

备注

转换公式中使用的参数可在 Abs2Qep 库头文件中进行配置。这些参数包括：绝对编码器分辨率、增量编码器每转线数、驱动器每分钟最大转数。

在 Abs2Qep 转换中会使用 ABS_TO_INCR 比率将绝对位置的变化映射为相应的 QEP 边沿数。边沿的任何一部分都会受到跟踪。如果边沿小数累积值达到 1，则会生成额外的边沿。

$$\text{ABS\_TO\_INCR} = \frac{\text{QCLK\_PER\_LINE x LINES\_PER\_REV}}{\text{ABS\_MAX\_POSITION}} = \frac{\text{QCLK\_PER\_REVOLUTION}}{\text{ABS\_MAX\_POSITION}} \quad (1)$$

其中

- LINES_PER_REV 为增量编码器分辨率。
- QCLK_PER_LINE 通常为 2。一个用于 QEP-A，另一个用于 QEP-B。
- ABS_MAX_POSITION = $2^{\text{ABS\_ENCODER\_RESOLUTION}}$。例如 2^{20}。

代表位置变化的 QCLK 或 QEP 边沿数量为：

$$\text{QCLK} = \text{ABS\_TO\_INCR x DELTA\_ABS\_POSITION} \quad (2)$$

其中：

- DELTA_ABS_POSITION = ABS_POSITION(n) - ABS_POSITION(n-1)，表示当前样片 (n) 和前一个样片 (n-1) 之间绝对位置的变化。
- QCLK 是表示位置变化所需的 QEP-A + QEP-B 边沿总数。

备注

方程式 2 中的这种简单转换假设**没有**过绝对零。有关过零检测，请参阅节 4.2.3

对于给定的位置变化，QCLK 的频率使得边沿能够在整个位置采样周期内均分。该频率以 CLB 时钟周期表示。

4.2.2 Abs2Qep 转换示例

给定以下参数：

- CLB 时钟 = 10 纳秒
- 位置采样周期 = 100 微秒或 10,000 个 CLB 时钟
- 绝对编码器分辨率 = ABS_MAX_POSITION = Q20 = 1048576
- 增量编码器分辨率 = 1024 条线。因此，QCLK_PER_REV = 2 x 1024 = 2048

ABS_TO_INCR 比率为：

$$\text{ABS\_TO\_INCR} = \frac{2 \text{ x 行}}{\text{Qmax}} = \frac{2048}{1048576} = 0.00195313 \quad (3)$$

表 4-2 显示了从绝对位置变化到生成的 QCLK 数量的转换示例。

请注意，在样片 2 和样片 3 中，边沿小数累积值大于 1。发生这种情况时会生成一个额外的 QCLK，并从边沿小数累积值减去一。

备注

显示的绝对位置样片仅用于演示用途。实际位置变化值可能比显示的值大得多，或者可能方向相反。

表 4-2. Abs2Qep 计算示例

样片	位置	增量位置 (1)	QCLK	小数边沿	生成的 QCLK 数	每个 QCLK 的 CLB 时钟数 (2)
0	0	0	0	0	0	0
1	24000	24000	46.875	0.875	46	217
2	53000	29000	56.6406	0.875 + 0.6406 = 1.515 → 0.515 (3)	56+1 (3)	175
3	62000	9000	17.5781	0.515 + 0.5781 = 1.09375 → 0.09375 (3)	17+1 (3)	555

(1) 位置(n) - 位置(n-1)。在本示例中，所有的变化都是正向的，并且没有过零。如果此值为负，则方向将相反。
(2) 每个 QCLK 脉冲之间的 CLB 时钟数。此值基于以 CLB 时钟表示的采样频率。本示例中将生成 10,000 个 CLB 时钟/QCLK
(3) 生成一个额外的 QCLK，并将小数部分调整 1。

4.2.3 Abs2Qep 过零检测

Abs2Qep 支持生成 QEP-I 以指示过绝对零。必须了解绝对编码器的最大每分钟转数 (RPM) 才能实现过零检测。RPM 与位置采样频率相结合可以决定可能的最大变化量。如果变化量大于最大值的绝对值，则已过绝对零。

考虑以下示例：

- 电机 MAX_RPM = 30,000 转/分钟 = 500 转/秒
- 位置采样频率 = 100 微秒
- 绝对编码器分辨率 = ABS_MAX_POSITION = Q20 = 2^{20}= 1048576

最大位置变化为：

$$500\frac{\text{revolutions}}{\text{second}} \text{ x } 100\frac{\text{microseconds}}{\text{sample}} = 0.05\frac{\text{revolutions}}{\text{sample}} \quad (4)$$

因此，幅度大于 0.05 x Qmax 的任何位置变化都会视为过零。

图 4-4 展示了正向过零。Position(n) 是一个相对较小的数字，而 Position(n-1) 是一个非常大的数字。因此，Position(n) - Position(n-1) 是一个负数，其幅度大于 ABS_MAX_POSITION。

在本示例中，Abs2Qep 使用两个测量值的总和来确定等效的 QEP 脉冲：

- (A) Position(n-1) 与 Qmax 之间的增量
- (B) 0 与 Position(n) 之间的增量

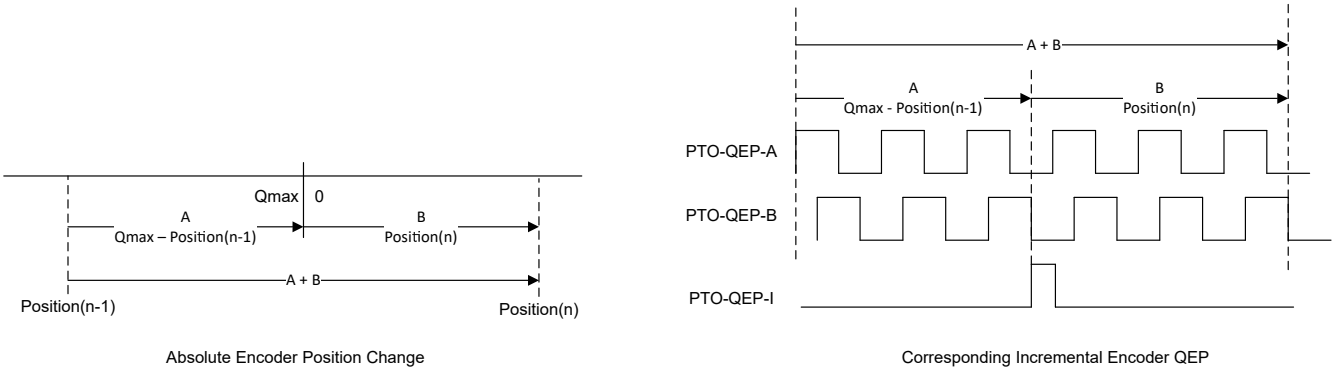


图 4-4. Abs2Qep 正向过零

图 4-5 展示了反向过零。Position(n) 是一个很大的值，而 Position(n-1) 相对较小。因此，Position(n) - Position(n-1) 将是正数，且幅度大于 ABS_MAX_POSITION。

在本示例中，Abs2Qep 使用以下测量值的总和：

- (A) Position(n-1) 与零之间的增量
- (B) Qmax 与 Position(n) 之间的增量

请注意，在生成的 PTO 中，QEP-B 领先 90°，表示反向。

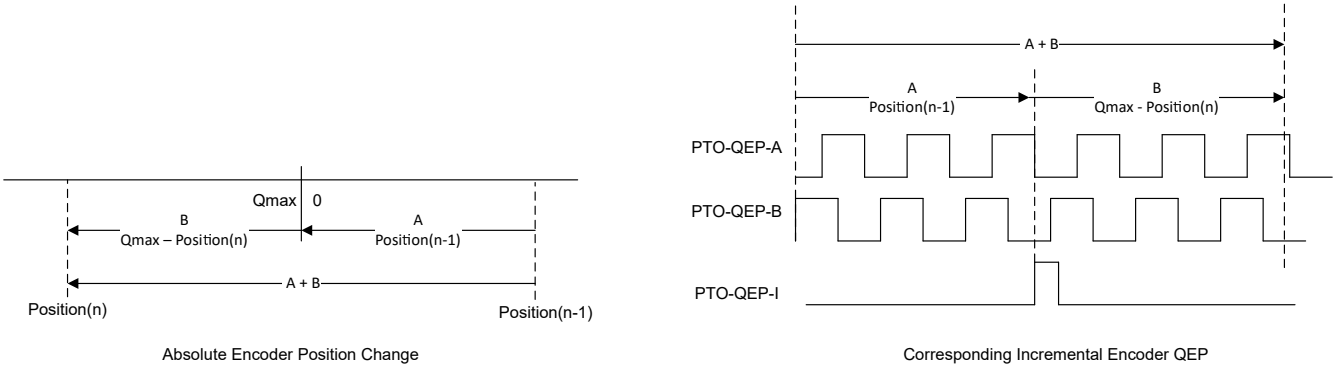


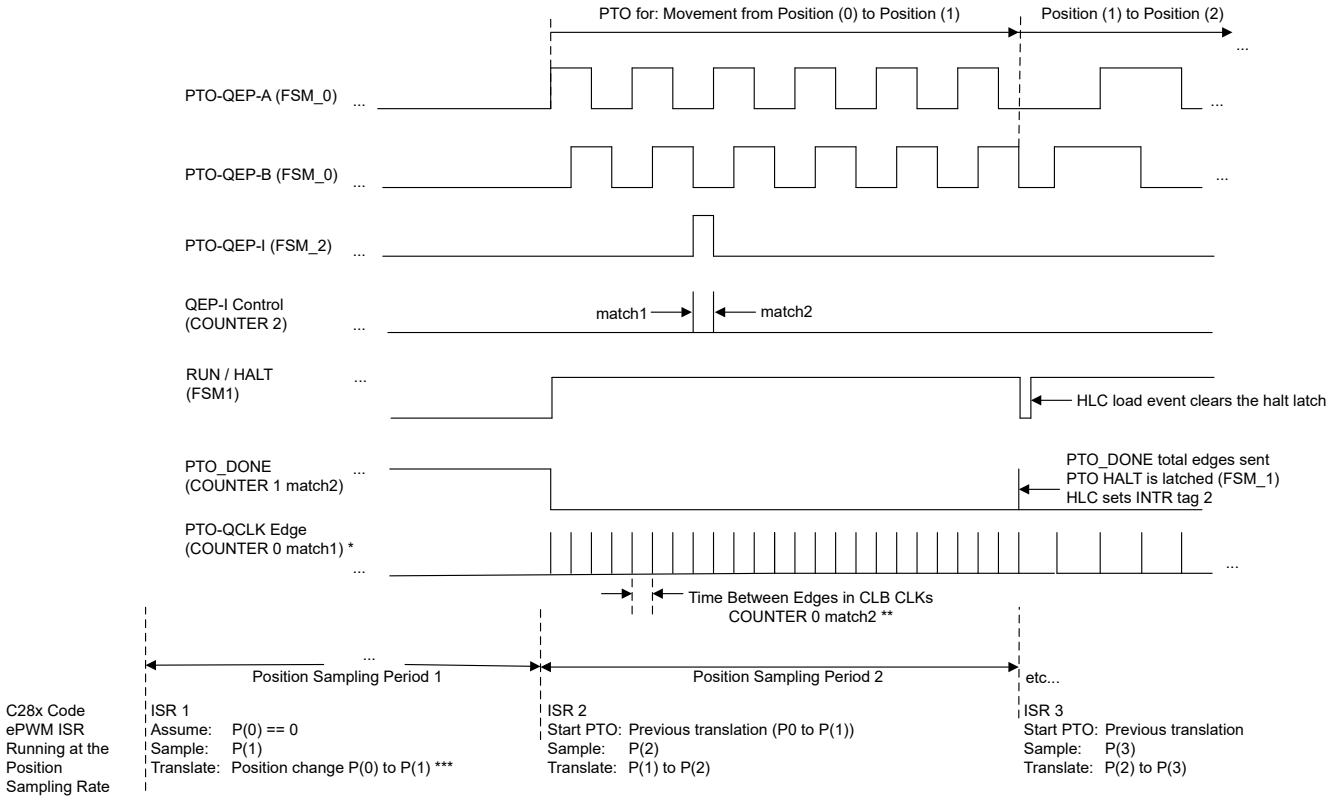
图 4-5. Abs2Qep 反向过零

4.3 Abs2Qep CLB 配置

如节 4.2 所述，将绝对位置转换为 QCLK 脉冲后，C28x 会将 PTO 参数加载到 HLC 的 FIFO 中。当需要启动 PTO 时，通过 GPREG 位提供的命令将指示 HLC 从 FIFO 中提取参数，将这些参数加载到计数器中，然后启动 PTO。一旦启动，CLB 会独立生成 PTO-QEP 波形。

一旦 PTO 完成，HLC 将设置一个 CLB 中断标签。C28x 可以使用这个标签在加载新配置之前检查 PTO 是否完成。

图 4-6 显示了 PTO-QEP 波形以及它们与 CLB 组件的关系。在本示例中，位置采样周期由 C28x 上的 ePWM ISR 控制。



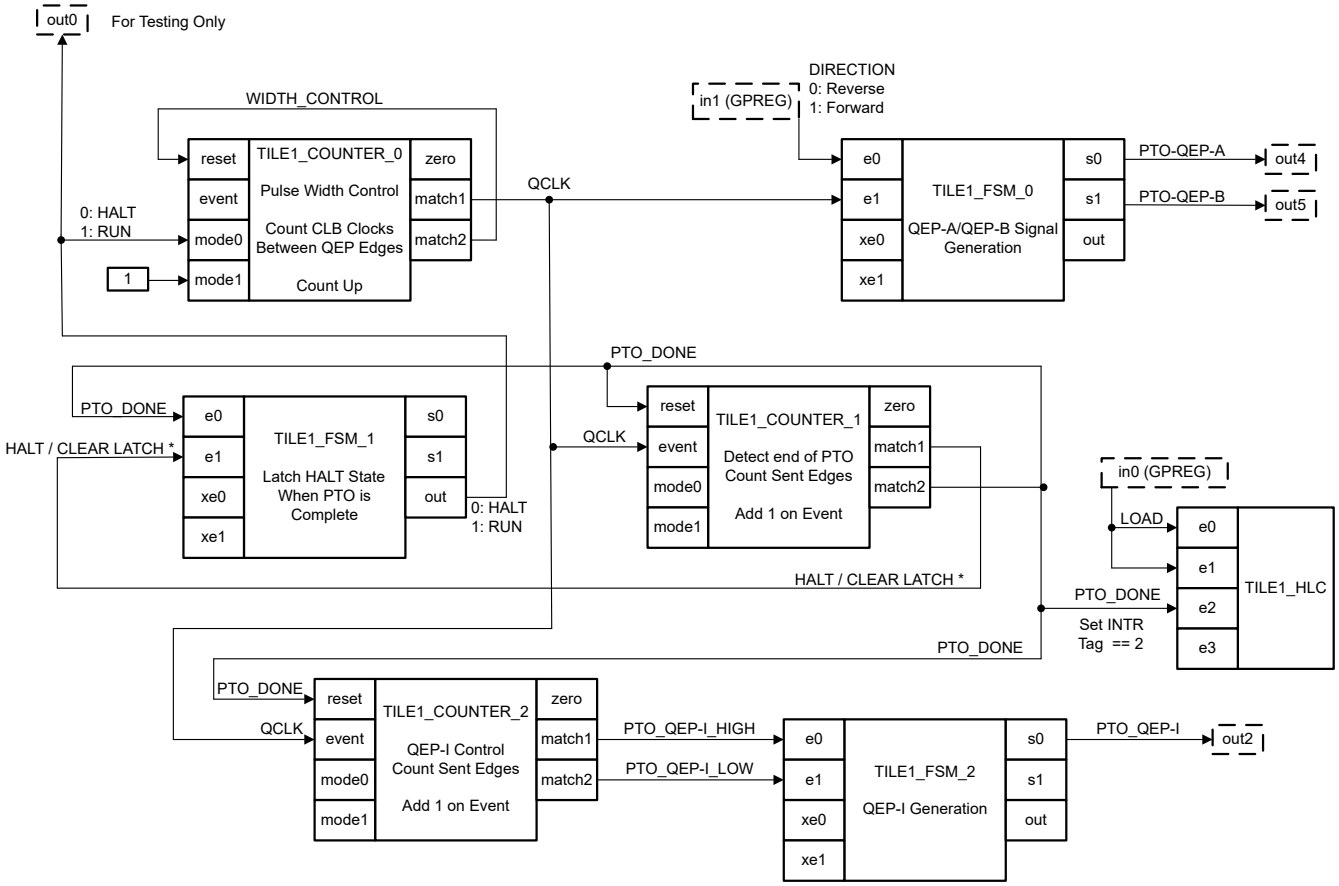
A. Counter 0 match 1 会初始化为固定值并且不会改变。此匹配会产生 QCLK 脉冲。这会将 QCLK 切换开关置于计数开始处，从而缩短前一次 PTO 停止和下一次 PTO 开始之间的时间。

B. QCLK 脉冲之间的时间由 COUNTER 0 match 2 控制。此匹配会重置 COUNTER 0。

C. P(n) 代表样片 n 的绝对位置。

图 4-6. Abs2Qep 系统波形示例

逻辑块的方框图如图 4-7 所示，表 4-3 详细描述了每个 CLB 组件的功能。



A. HALT/CLEAR LATCH 在 LOAD (加载) 事件期间由 HLC 直接控制。

B. PTO_DONE 由 COUNTER 1 递增控制，并在 LOAD 事件期间直接由 HLC 控制。

图 4-7. Abs2Qep CLB 逻辑块的方框图

表 4-3. Abs2Qep CLB 逻辑块 1

资源	功能	注意事项
		输入
In0	LOAD 控制上升沿： 加载新的 PTO 配置 (HLC)。	连接到 GPREG 位 0。 在加载新配置之前，检查并确认最后一个 PTO 已完成 (INTR 标签 == 2)
In1	DIRECTION 控制 1：顺时针 (正向) 0：逆时针 (反向)	连接到 GPREG 位 1。 仅在最后一个 PTO 已完成 (Intr 标签 2) 时更改
In2	未使用	未使用
In3	未使用	未使用
In4	未使用	未使用
In5	未使用	未使用
In6	未使用	未使用
In7	未使用	未使用
		输出
Out0	未使用	未使用
Out1	未使用	未使用
Out2	PTO_QEP-I 索引从 0 到 1 的转换表示已过绝对零位置。	索引输出信号。

表 4-3. Abs2Qep CLB 逻辑块 1 (continued)

资源	功能	注意事项
Out3	未使用	未使用
Out4	PTO_QEP-A	PTO 正交输出 A
Out5	PTO_QEP-B	PTO 正交输出 B
Out6	未使用	未使用
Out7	未使用	未使用
逻辑资源		
LUT0	未使用	未使用
LUT1	未使用	未使用
LUT2	未使用	未使用
FSM0	生成 PTO_QEP-A 和 PTO-QEP-B	在每个 QCLK 输入端生成 1 个边沿。QEP-A/B 的超前/滞后基于当前状态和 DIRECTION 输入信号。
FSM1	生成 HALT/RUN 信号	如果以下任一条件成立，则停止 PTO 输出： • PTO 完成。该停止状态会被锁存，PTO 将保持停止，直到通过 HALT/RUN 控制输入来清除锁存器为止 • HALT/RUN 控制输入处于高电平。
FSM2	生成 PTO_QEP-I 信号	根据 QEP-I 控制信号强制 PTO_QEP-I 为高电平和低电平。如果需要，允许用户将 QEP-I 配置为在多个 QCLK 内保持高电平。
CNT0	生成 QCLK（PTO 宽度控制）信号	每个 CLB 时钟的计数加 1。 • match1：固定值。在计数开始位置附近生成 QCLK 信号。这种放置方法可以缩短上一次 PTO 停止和下一次 PTO 开始之间的时间。 • match2：QEP 边沿之间的 CLB 时钟数。每个 match2 事件都会重置计数器。
CNT1	PTO 边沿计数控制	每个 QCLK 事件递增 1，由此可计算 PTO 期间发送的总 QEP-A + QEP-B 边沿数。 • match1：由 HLC 操作以清除停止锁存条件并启动 PTO。 • match2：待发送的边沿数。达到该数量后，便会置位 PTO_DONE 信号。这会锁存暂停状态，还会重置边沿计数并重置 QEP-I 控制。
CNT2	PTO_QEP-I 控制	每个 QCLK 事件递增 1，由此可计算 PTO 期间发送的总 QEP-A + QEP-B 边沿数。 • match1：PTO_QEP-I 将变为高电平的边沿 • match2：PTO_QEP-I 将变为低电平的边沿 **备注** 如果 PTO-QEP-I 在整个 PTO 内应保持低电平，则将 match1 和 match2 配置为一个大数字以避免匹配。（如 0xFFFFFFFF）。
高级控制器		
HLC	事件 1：加载新的 PTO 配置。	响应来自 C28x 的 LOAD 输入端的上升沿。这将配置并启动一个新的 PTO。如需了解所有步骤，请参阅节 4.3.3 中的程序说明。
	事件 2：信号 PTO 已完成。	通过设置中断标签 2 来响应 PTO 信号的完成。此时，可安全加载新的 PTO 计数器配置。

4.3.1 Abs2Qep QEP-A/B 脉冲序列生成

QEP-A 和 QEP-B 信号由有限状态机 (FSM) 生成。状态图如图 4-8 所示，并具有以下特征：

- 当 QCLK = 1 时发生状态转换。
- 当 QCLK = 0 时状态保持不变。
- 一次只改变一个 QEP 信号。
- 首先转换哪个信号取决于来自 C28x 的方向输入。

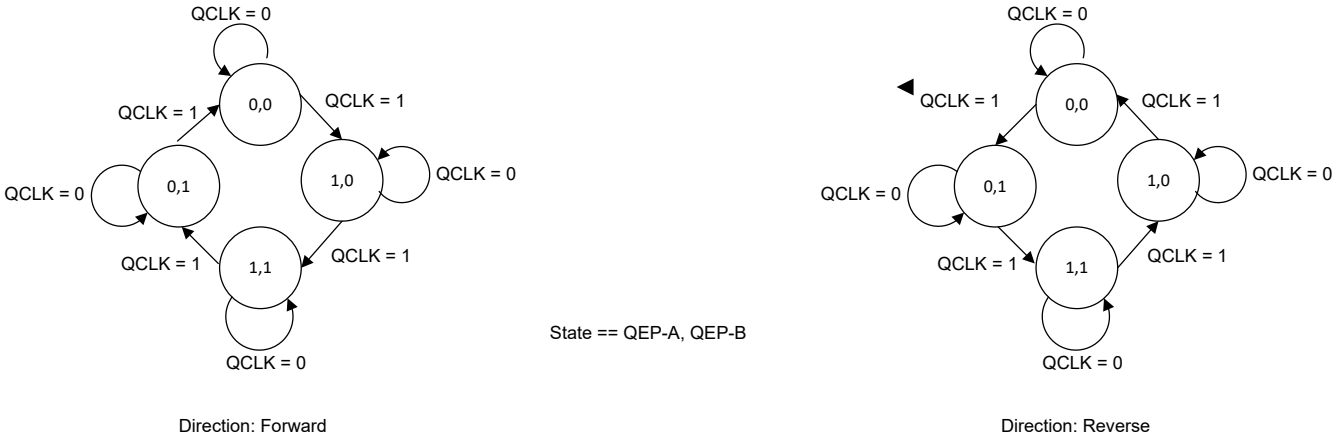


图 4-8. Abs2Qep PTO 状态图

表 4-4 和表 4-5 是对应的卡诺图。通过检查图中的每个“1”或使用卡诺图解算器来确定所得方程。x 用于指示无效的状态。请注意，无需进一步简化方程；可按所示方式将这些方程输入到 CLB 工具中。使用 OR 运算符可从各个组成部分建立完整的方程，如仿真结果所示（图 4-9）。

表 4-4. QEP-A (s0) 信号生成卡诺图

		方向 (e0) = 1（正向） 下一个状态 QCLK、QEP-B（e1、s1）						方向 (e0) = 0（反向） 下一个状态 QCLK、QEP-B（e1、s1）			
		00	01	11	10			00	01	11	10
当前状态 s0、s1（QEP-A、B）	00	0	0	x (2)	1 (2)	当前状态 s0、s1（QEP-A、B）	00	0	0	0	x
	01	0	0	x	0		01	0	0	1 (3)	x (3)
	11	1(1)	1(1)	0	x		11	1(1)	1(1)	x (3)	1 (3)
	10	1(1)	1(1)	1(2)	x (2)		10	1(1)	1(1)	x	0

(1) s0_1 = (e0 & s0 & !e1) | (!e0 & s0 & !e1) = s0 & !e1
(2) s0_2 = e0 & !s1 & e1
(3) s0_3 = !e0 & s1 & e1

表 4-5. QEP-B (s1) 信号生成卡诺图

		方向 (e0) = 1（正向） 下一个状态 QCLK、QEP-A（e1、s0）						方向 (e0) = 0（反向） 下一个状态 QCLK、QEP-A（e1、s0）			
		00	01	11	10			00	01	11	10
当前状态 s0、s1（QEP-A、B）	00	0	0	0	x	当前状态 s0、s1（QEP-A、B）	00	0	0	x (3)	1 (3)
	01	1(1)	1(1)	x	0		01	1(1)	1(1)	1(3)	x (3)
	11	1(1)	1(1)	x (2)	1 (2)		11	1(1)	1(1)	0	x
	10	0	0	1 (2)	x (2)		10	0	0	x	0

(1) s1_1 = (e0 & s1 & !e1) | (!e0 & s1 & e1) = s1 & !e1
(2) s1_2 = e0 & s0 & e1
(3) s1_3 = !e0 & !s0 & e1

图 4-9 所示为 SystemC 仿真结果。

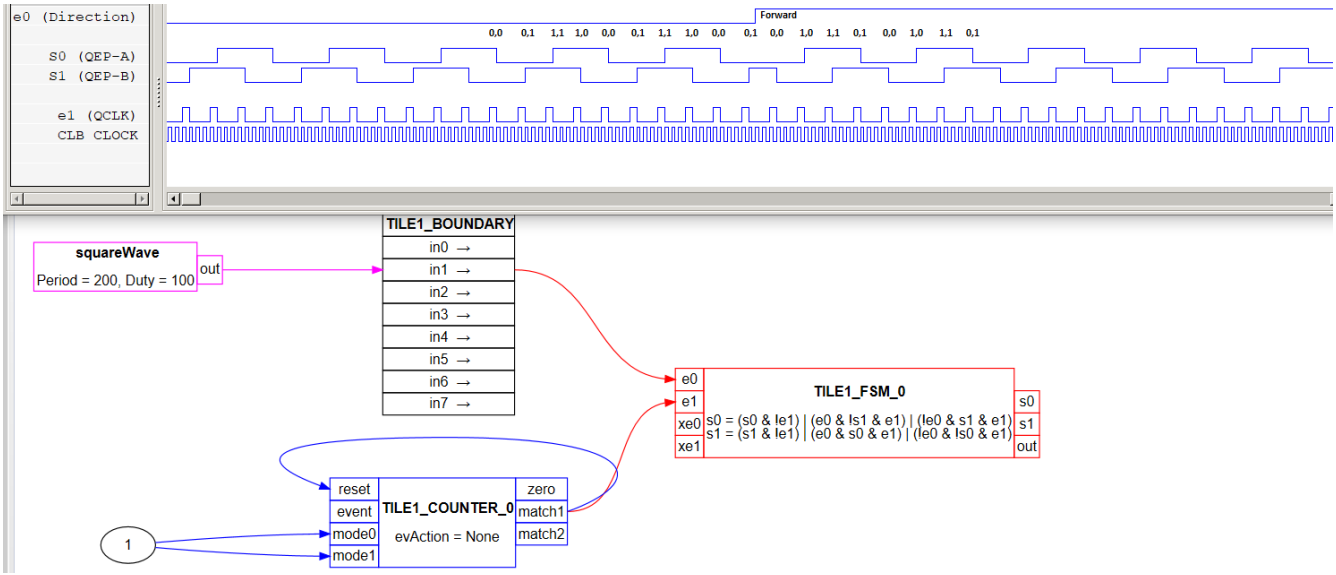


图 4-9. 仿真 QEP-A 和 QEP-B 生成

4.3.2 Abs2Qep 停止锁存器

HALT_LATCH 和 RUN/HALT 输出是使用有限状态机实现的。输出信号连接到产生 QCLK 的计数器 mode0 输入端。RUN/HALT 输出仅取决于锁存器的当前状态以及来自 CPU 的 HALT 信号。如果 HALT 信号为低电平且未设置锁存器，则将生成 QCLK (COUNTER mode0 = out = 1)。在所有其他情况下，不会生成 QCLK (COUNTER mode0 = out = 0)。这将表示为 out = !(s0 | e1)。

表 4-6. RUN/HALT 输出

s0 (LATCH)	e1 (HALT/CLEAR LATCH)	out = !(s0|e1)	QCLK 生成
0	0	1	运行
0	1	0	停止
1	0	0	停止
1	1	0	停止

HALT_LATCH 是在 PTO_DONE 的上升沿设置的。该设置状态将保持，直到 CPU 发出的 HALT/CLEAR_LATCH 信号的上升沿将状态清除为止。

表 4-7. HALT_LATCH 卡诺图

		PTO_DONE、CLEAR_LATCH (e0、e1)			
		00	**01**	**11**	**10**
s0 (LATCH)	**0**	0	0	0	1 (2)
	1	1 (1)	0	0	1 (1)、(2)

(1) s0_1 = s0 & !e1
(2) s0_2 = e0 & !e1

图 4-10 中显示了停止锁存器的 SystemC 仿真。

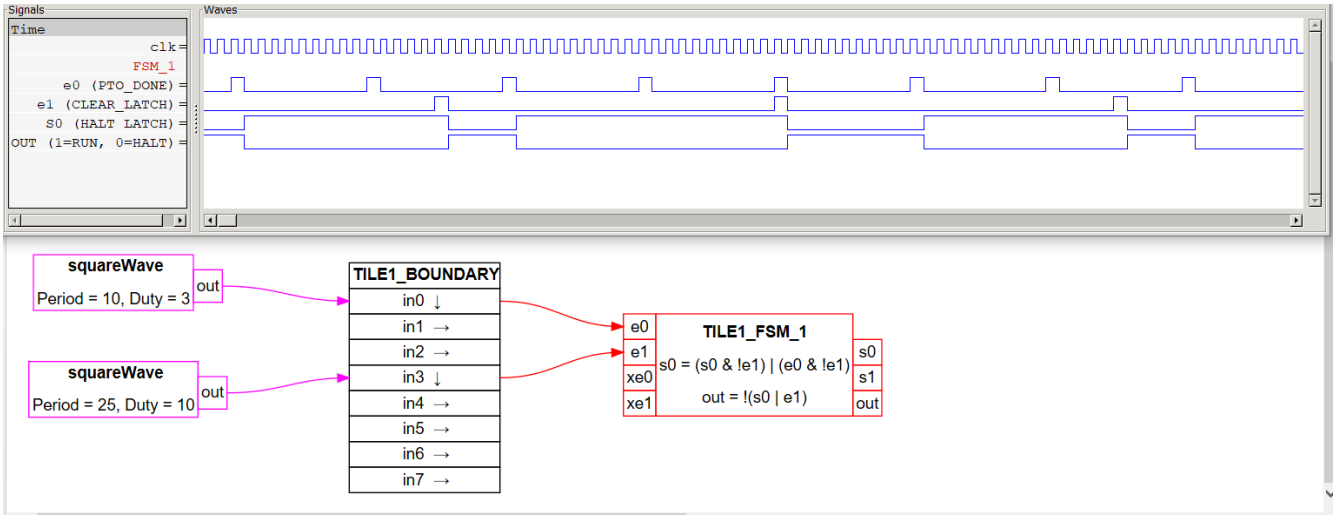


图 4-10. 停止锁存器的仿真

4.3.3 Abs2Qep 高级控制器 (HLC)

高级控制器在编程后可以：

- 启动 PTO 信号的生成
- 修改 HALT/CLEAR_LATCH 和 PTO_DONE 信号
- 加载生成 PTO 所需的计数器匹配值
- 标记 PTO 状态的结束

表 4-8. Abs2Qep HLC 寄存器用途

R0 和 R1	用于从 FIFO 拉取数据。
R2	在 CLB 配置期间初始化为零。用于将零加载到匹配基准，以便将给定信号操纵到高电平状态。
R3	在 CLB 配置期间初始化为 0xFFFFFFFF。用于将一个较大值加载到匹配基准，以便将给定信号操纵到低电平状态。

表 4-9. Abs2Qep HLC 程序

加载：事件 0、事件 1		
指令编号	**操作码**	**说明**
Program0：0	`MOV_T1 R2、C1`	置位 HALT/CLEAR_LATCH。COUNTER 1 已被 PTO_DONE 重置（计数 == 0）。加载值为零的 match1 基准将强制 HALT/CLEAR_LATCH 上出现上升沿。
Program0：1	`MOV_T2 R3、C1`	强制 PTO_DONE 信号处于低电平状态。
Program0：2、3	`PULL R0` `MOV_T2 R0、C1`	加载要生成的 QCLK 的数量。 注意：对于零个 QCLK 的情况：由于 COUNTER_1 计数 == 0，QCLK 值为零将强制 PTO_DONE 回到高电平状态。
Program0：4、5	`PULL R1` `MOV_T2 R0、C0`	加载两个 QCLK 之间的 CLB 时钟的数量。当计数器达到此值时，它将重置为零。

Program0：6、7
Program1：0，1
	`PULL R0`
	`MOV_T1 R0、C2`
	`PULL R0`
	`MOV_T2 R0、C2`
		配置哪个 QCLK 边沿将强制 PTO-QEP-I 处于高电平和低电平状态。如果 PTO-QEP-I 应保持在低电平状态，则将通过 FIFO 传递较大的值。

Program1：2	`MOV R1、C0`	将 COUNTER_0 设置为零。这可以防止计数器在加载零脉冲配置时递增 1。

表 4-9. Abs2Qep HLC 程序 (continued)

加载：事件 0、事件 1		
指令编号	**操作码**	**说明**
Program1：3	INTR 1	作为标签用于指示事件 0 加上事件 1 完成。这将放置在最后一条指令旁边，以防止其与事件 2 中的 INTR 指令背靠背。
Program1：4	MOV_T1 R3、C1	强制 HALT/CLEAR_LATCH 信号处于低电平状态。如果 PTO_DONE 信号处于低电平状态，这将启动 PTO 信号的生成。如果 PTO_DONE 处于高电平状态，则将设置 HALT_LATCH。
PTO_DONE：事件 2		
指令编号	**操作码**	**说明**
Program2：0	INTR 2	作为标签用于指示事件 2 已完成或 PTO 已完成。

4.4 Abs2Qep 输入和输出信号

Abs2Qep 接口的芯片级输入：在示例中，仿真的绝对位置由测试函数生成，不需要外部输入。如果使用绝对编码器，则应按照绝对编码器接口文档中的说明对其进行连接。

Abs2Qep 接口的芯片级输出：输出信号为 PTO-QEP-A、PTO-QEP-B 和 PTO-QEP-I 信号。在提供的示例中，这些输出映射到 GPIO，如节 6 所述

5 PTO – QepOnClb QEP 解码器

QepOnClb 配置 CLB 块，以实现简单的 QEP 解码器模块。图 5-1 显示了 QepOnClb 的实现图。

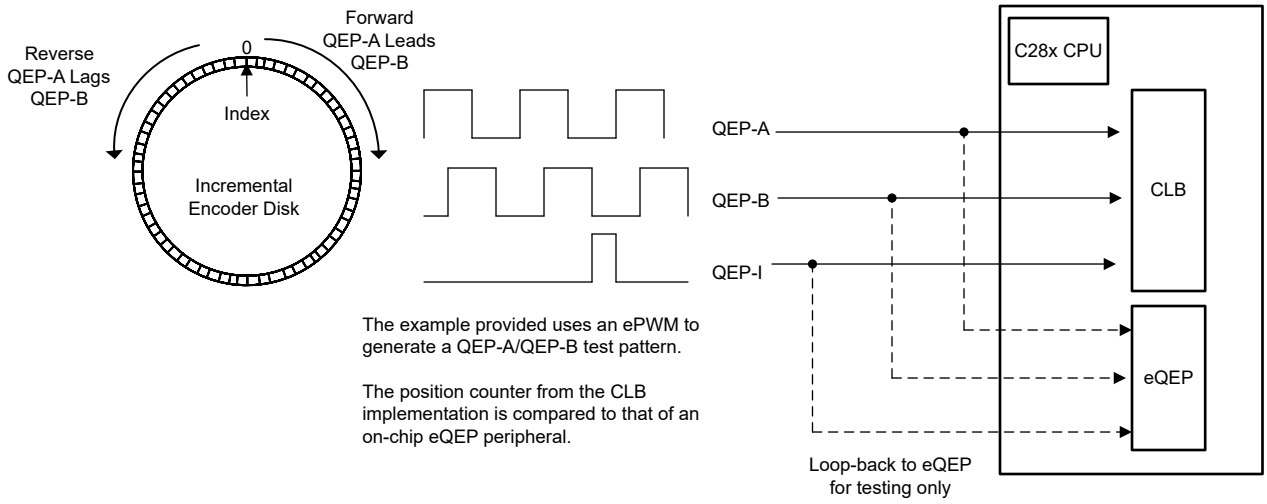


图 5-1. CLB 上的 QEP 实现图

5.1 QepOnClb 和 eQEP 的比较

此解码器实现方式与正交计数模式下运行的 C2000 eQEP 外设类似。表 5-1 简要比较了支持的特性。在某些情况下可能进行了修改。请参考器件特定的技术参考手册，了解 eQEP 外设的详细说明。

表 5-1. QepOnClb 和 eQEP（0 类）比较概述

	eQEP 特性	QepOnClb 支持
正交时钟模式（QEP-A/B 信号）	检测方向、计数和无效状态转换。	已实现。 比较 QEP-A/B 信号的当前状态与之前状态。相位关系决定了运动方向。如果两个信号同时改变，将检测到无效状态转换。
	无效状态转换会设置标志，还有可能生成中断。	已实现。 CLB 将针对无效状态转换发出中断。如果需要标志，可使用 CLB 中断标签，中断会被禁用。
	配置 4 个计数器：QEPA/B 的上升和下降沿	实现的结果： QEPA/B 的上升和下降沿都会生成计数。 修改 QCLK 状态机可改变此行为。
	反向计数（反向 QEP-A/B 输入）	未实现。 实现方式为修改 CLB 配置，交换 QCLK 状态机输入。
方向计数模式（XCLK 和 DIR 信号）	QEP-A 成为 XCLK，QEP-B 成为 DIR。	未实现。 实现方式为修改 CLB 配置并 (1) 从方向解码 LUT 断开 DIR (QEP-B) 连接，(2) 将 DIR 直接连接到计数器的方向控制，模式 1。
QEP-I（索引或零信号）	锁存位置计数器，稍后可使用 driverlib 函数进行读取。可配置为上升沿、下降沿或基于事件标记/软件索引标记。	实现方式为： 在 QEP-I 上升沿锁存位置计数器。可使用提供的库函数从 HLC FIFO 读取位置计数器值。在一些器件上可配置 HLC，以响应下降沿而不是上升沿。
	初始化位置计数器。	未实现。 实现方式是修改 HLC 程序，在初始化计数器时使用从 FIFO 提取的值。
	复位位置计数器。	未实现。 实现方式是将 QEP-I 信号路由到 QEP 复位生成 LUT 和 LUT 方程。
QEP-S（选通信号）	锁存或复位位置计数器。	未实现。 可通过以下 QEP-I 示例实现。
位置计数器工作模式	在发生索引、位置上限、第一索引或单位超时事件时复位。	实现方式为： 在达到位置上限值时复位。此值通过提供的库函数 pto_qeponclb_configMaxCounterPos() 进行配置。请参阅节 7.6。
位置比较器件		未实现。
边沿捕捉器件		未实现。
看门狗		未实现。
单位时基计时器 (QUTMR)		未实现。

5.2 QepOnClb 芯片资源

QepOnClb 实现方案中使用了以下 C2000 资源：

- **C28x CPU**
  - 初始化 QepOnClb 接口，配置 CLB、输入/输出 XBAR 和 GPIO。
  - 配置 CLB 以实现基本 QEP 解码器模块的正交计数模式。
- **可配置逻辑块 (CLB) 类型 1 或更高版本**
  - 1 个 CLB 逻辑块。请参阅节 5.4 了解使用的具体 CLB 块。
  - 32 位 QEP 位置计数器，具有可编程位置上限。
  - QEP 方向解码，来自 QEP-A/B

  - QEP 状态转换错误检测
  - 锁存 QEP-I 上的位置计数器
- **器件互连 (XBAR)**
  - 输入和输出 XBAR 用于在适用的情况下将信号传入 CLB。

5.3 QepOnClb 工作原理

QEP 解码器解读来自增量编码器的脉冲序列输出。基本 QEP 脉冲序列包含信号 QEP-A、QEP-B 和 QEP-I，如图 5-3 中所示。这些信号具有以下特性：

- QEP-A/B 相指示运动方向。如果 QEP-A 的上升沿超前 90°，则方向为正向（顺时针）。如果 QEP-A 的上升沿滞后，则方向为反向（逆时针）。如图 5-2 所示。
- QEP-A/B 频率与码盘速度成正比。
- 索引信号 QEP-I 表示过绝对零。

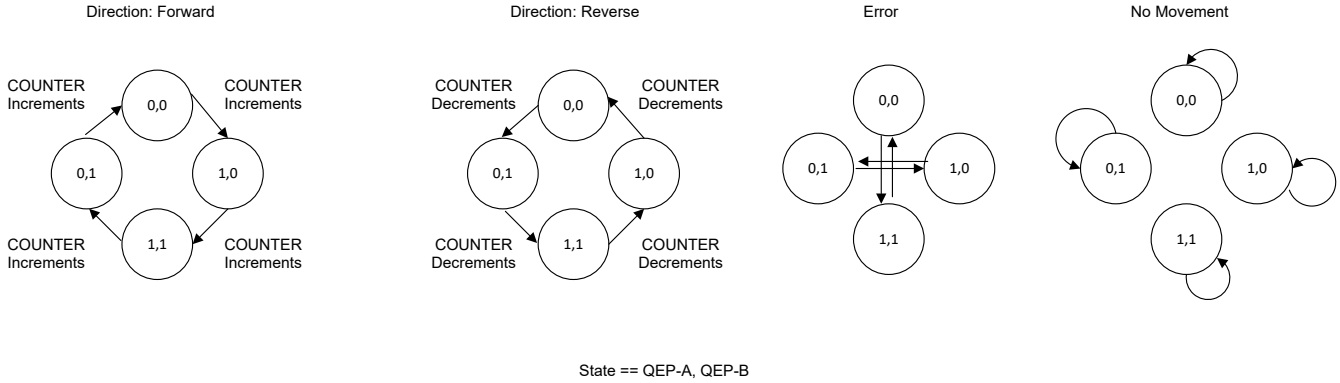


图 5-2. QEP-A、QEP-B 状态图

设计方法：

1. 选择满足映射解码器要求的 CLB 组件。表 5-2 提供了此类映射的示例。
2. 绘制波形，有助于直观展示 CLB 块之间的理想交互。图 5-3 包含示例 QEP 波形、CLB 生成的信号以及用于实现此特性的相应 CLB 块。
3. 定义 LUT 和 FSM 模块的公式。节 5.4 提供了各模块的详细说明。

表 5-2. 解码器特性与 CLB 块之间的映射示例

解码器功能	CLB 块映射
32 位位置计数器	直接映射 CLB 32 位计数器模块。连接 match1 和 match2 可进行复位，对于事件，可实现介于 0 和位置上限 (MAXPOS) 间的计数。
过去状态存储器	有效状态转换、方向和错误的检测都依赖于 QEP-A/B 过去的状态，这可映射能够存储过去状态的 FSM。
过去和当前状态的比较	如果可以从 FSM 中得到过去状态，LUT 即可比较当前和过去状态。如果没有 LUT，FSM 也可提供这一功能。需要对方向检测和错误检测进行比较。
中断和计数器捕捉	捕捉计数器的值并中断 CPU 映射 HLC 的功能。
CPU 对解码器的输入，例如复位和使能	控制位从 CPU 经过 GPREG 路由到 LUT，并与其他系统信号相结合（OR 或 AND）。

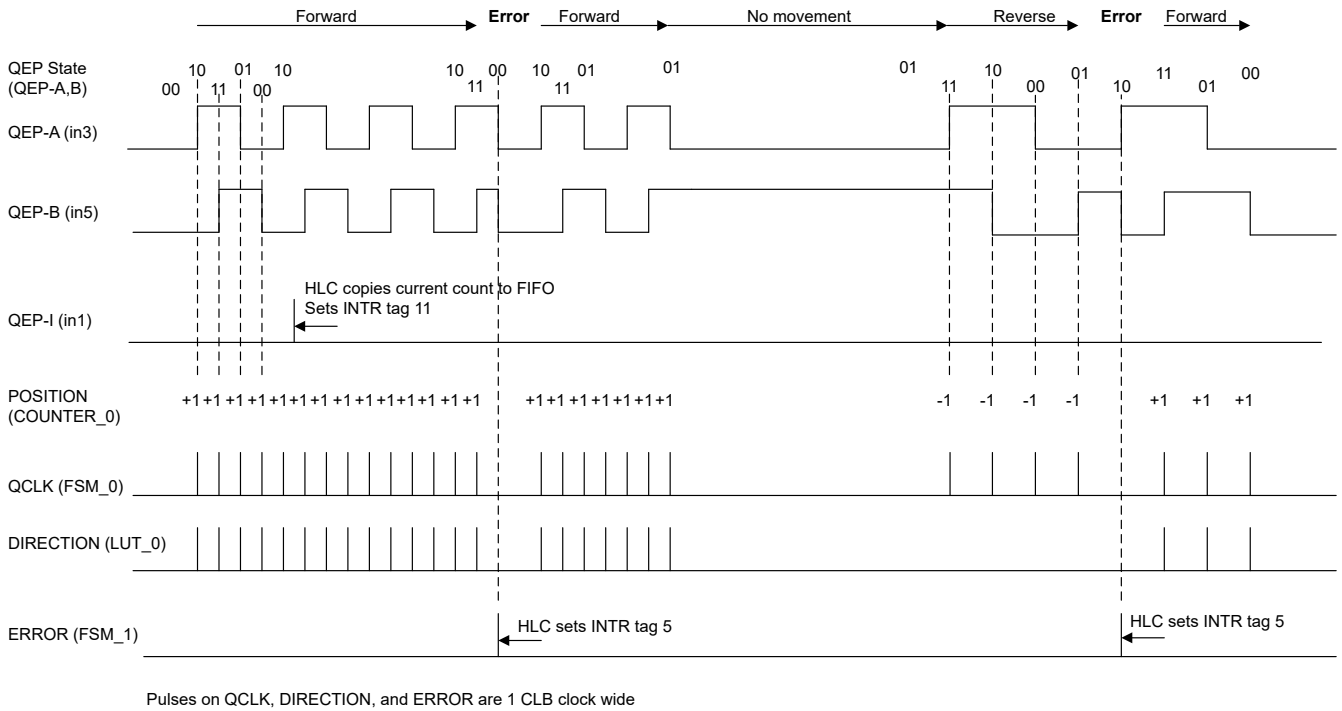


图 5-3. QepOnClb 波形示例

5.4 QepOnClb CLB 资源

图 5-4 中展示了解码器 CLB 配置，并在表 5-3 中进一步说明。

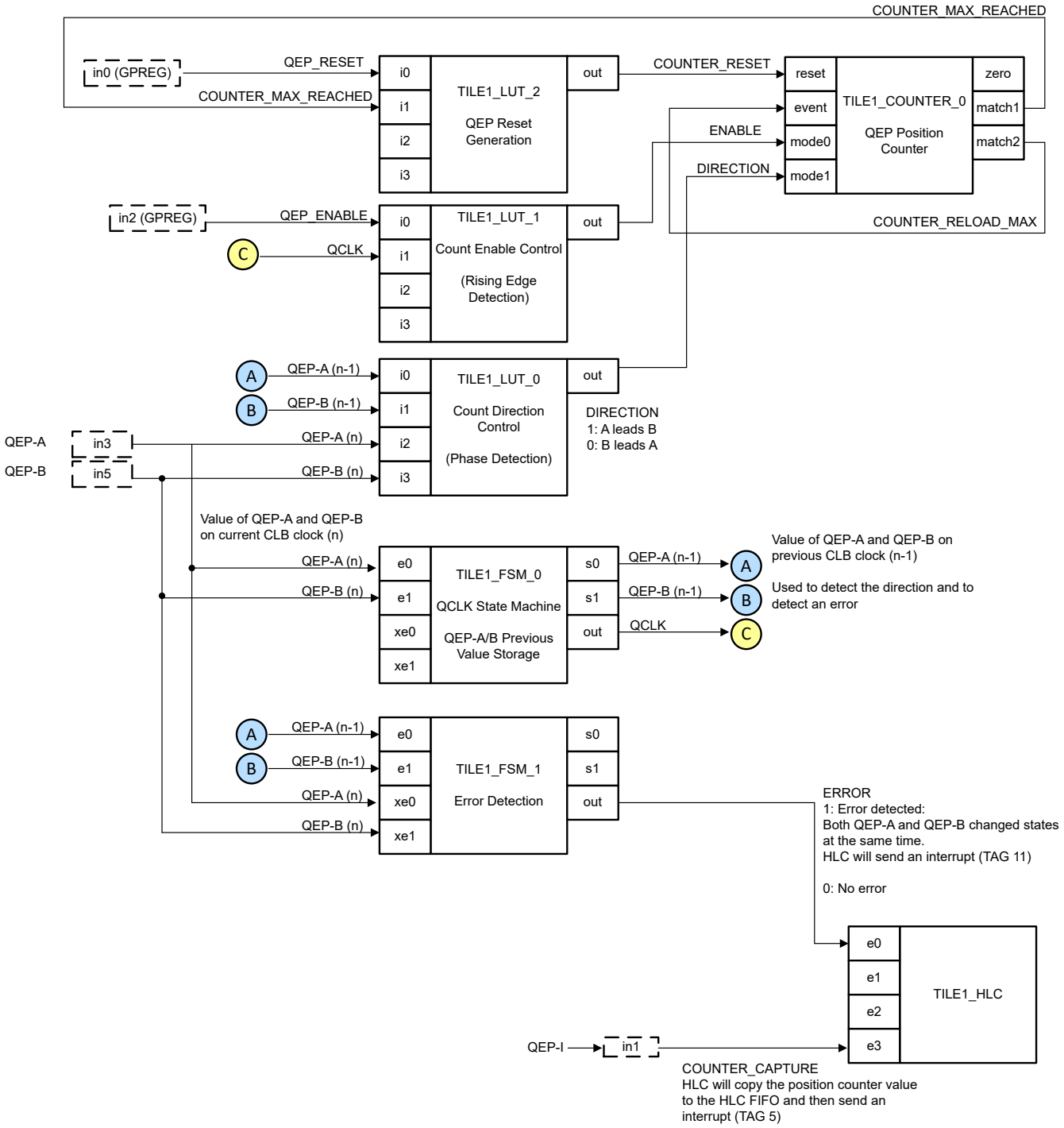


图 5-4. QepOnClb 逻辑块的方框图

表 5-3. QepOnClb 逻辑块 1

资源	功能	注意事项
		输入
In0	QEP_RESET	连接到 GPREG 位 0，对位置计数器复位进行软件控制。 • 1：复位位置计数器。计数器将保持复位状态，直到将 0 写入此位。 • 0：将计数器从复位状态释放。如果 QEP_ENABLE 为 1，位置计数器将根据需要递增/递减。
In1	QEP-I	根据设计，此信号的上升沿将提示 HLC 将当前的位置计数器存储在 FIFO 中。这与 eQEP 的上升沿锁存模式类似。
In2	QEP_ENABLE	连接到 GPREG 位 2。从软件提供位置计数器的启用/禁用开关。 • 1：QEP 已启用。如果复位信号 QEP_RESET 也为 0，位置计数器将根据需要递增/递减。 • 0：QEP 禁用。位置计数器停止递增/递减。
In3	QEP-A	QEP-A 和 QEP-B 的状态转换用于检测运动、运动方向或错误。
In4	未使用	未使用
In5	QEP-B	QEP-A 和 QEP-B 的状态转换用于检测运动、运动方向或错误。
In6	未使用	未使用
In7	未使用	未使用
		输出
Out0	未使用	未使用
Out1	未使用	未使用
Out2	未使用	未使用
Out3	未使用	未使用
Out4	未使用	未使用
Out5	未使用	未使用
Out6	未使用	未使用
Out7	未使用	未使用
		逻辑资源
LUT0	计数方向控制	确定运动方向。比较 QEP-A、QEP-B 的当前状态与过去状态，以解码相位。输出可适当地设置位置计数器的模式。 • QEP-A 超前（正向）：DIRECTION 为 1，位置计数器模式为递增。 • QEP-A 滞后（反向）：DIRECTION 为 0，位置计数器模式为递减。
LUT1	计数启用控制	使位置计数器递增或递减一。以下两种条件均满足时可实现： 1. QEP_ENABLE 为 1 且 2. QCLK 为 1
LUT2	QEP 复位生成	满足以下任一条件时位置计数器将复位： 1. QEP_RESET 为 1 或 2. 位置计数器 match2 有效
FSM0	QCLK 状态机	此 FSM 有两种功能： • 监控 QEP-A/B 信号，检测有效的状态改变。这种改变会将 QCLK 拉高，使位置计数器递增或递减。请参见 LUT1。 • 存储之前的 QEP-A/B 电平。之前的状态用于检测运动方向或错误。请参见 LUT0 和 FSM1。
FSM1	错误检测	比较之前的 QEP-A/B 状态与当前状态。如果两个信号同时改变，则内部 ERROR 信号将强制拉高。根据设计，ERROR 的上升沿将触发 HLC 发送标签为 11 的中断。

表 5-3. QepOnClb 逻辑块 1 (continued)

资源	功能	注意事项
FSM2	未使用	未使用
CNT0	位置计数器	如果启用 (QCLK 为 1) ，则在每个 CLB 时钟周期递增或递减一。位置计数器的位置上限 (MAXPOS) 根据以下条件指定： • 负载：位置上限减 1 (MAXPOS - 1)。match2 触发一个事件后，此值将载入计数器。 • match1：达到位置上限 (MAXPOS)后复位计数器。 • match2：0xFFFFFFFF 触发一个计数器事件，将负载值载入计数器。
CNT1	未使用	未使用
CNT2	未使用	未使用
高级控制器		
HLC	事件 0：检测到错误。	向 CPU 发送错误中断，标签为 11。
	事件 3：计数器捕捉	将当前位置计数器复制到 FIFO，响应 QEP-I 上升沿。然后 HLC 会中断 CPU 并设置标签 5。

5.4.1 QepOnClb QCLK 状态机

QCLK 状态机具有两个功能：(1) 保留一份 QEP-A 和 QEP-B 之前电平的副本，(2) 检测有效的 QEP 状态改变，使位置计数器递增或递减。

若要创建一份 QEP-A 和 QEP-B 之前信号的副本，使用以下状态公式：

- s0 next = QEP-A (n) = e0
- s1 next = QEP-B (n) = e1

若要确定 QEP 状态改变是否有效，需要将之前的 QEP-A/B 值与当前值进行比较。检测到有效的 QEP 状态转换时，FSM 会将 QCLK 拉高。此信号可启用位置计数器，根据 DIRECTION 信号使其递增或递减。

表 5-4 中介绍了四种可能的情况：

1. 无效的状态改变，QCLK = 0，位置计数器不变
2. 无运动，QCLK = 0，位置计数器不变
3. 正向运动，QCLK = 1，位置计数器递增或递减
4. 反向运动，QCLK = 1，位置计数器递增或递减

生成的公式由 OR 运算符连接，输入到 CLB 工具中以获得 FSM 的输出。

表 5-4. QCLK 状态机卡诺图

		当前状态 e0，e1 QEP-A(n)，B(n)			
		00	**01**	**11**	**10**
之前的状态 s0，s1 QEP-A(n-1)，B(n-1)	**00**	0 无运动	1 (2) 反向	0 无效	1 (4) 正向
	01	1 (1) 正向	0 无运动	1 (3) 反向	0 无效
	11	0 无效	1 (2) 正向	0 无运动	1 (4) 反向
	10	1 (1) 反向	0 无效	1 (3) 正向	0 无运动

(1) (!s0 & s1 & !e0 & !e1) + (s0 & !s1 & !e0 & !e1)
(2) (!s0 & !s1 & !e0 & e1) + (s0 & s1 & !e0 & e1)
(3) (!s0 & s1 & e0 & e1) + (s0 & !s1 & e0 & e1)
(4) (!s0 & !s1 & e0 & !e1) + (s0 & s1 & e0 & !e1)

5.4.2 QepOnClb 方向解码

LUT 比较当前 QEP-A/B 信号与之前的 QEP-A/B 信号，以确定方向。之前的 QEP-A/B 值由节 5.4.1 中介绍的 QCLK 状态机 (FSM) 提供。

表 5-5 中展示了 4 种可能的情况：

1. 无效的状态变化：DIRECTION = 无关，看作 0
2. 无运动：DIRECTION = 无关，看作 0
3. QEP-A 上升沿超前：正向运动，DIRECTION = 1
4. QEP-B 上升沿超前：反向运动，DIRECTION = 0

对于第 1 种和第 2 种情况，不必关注 DIRECTION 信号。在这种情况下，QCLK 状态机使 QCLK 保持低电平。低 QCLK 会禁用位置计数器，它不会发生递增或递减。对于无关类情况，使用 DIRECTION = 0。

生成的公式由 OR 运算符连接，可在 CLB 工具中查看。

表 5-5. 方向检测卡诺图

		当前状态 i2，i3 QEP-A(n)，B(n)			
		00	**01**	**11**	**10**
之前的状态 i0，i1 QEP-A(n-1)，B(n-1)	**00**	0 无运动	0 反向	0 无效	1 (4) 正向
	01	1 (1) 正向	0 无运动	0 反向	0 无效
	11	0 无效	1 (2) 正向	0 无运动	0 反向
	10	0 反向	0 无效	1 (3) 正向	0 无运动

(1) (!i0 & i1 & !i2 & !i3)
(2) (i0 & i1 & !i2 & i3)
(3) (i0 & !i1 & i2 & i3)
(4) (!i0 & !i1 & i2 & !i3)

5.4.3 QepOnClb 错误检测

FSM 会比较当前 QEP-A/B 信号与之前的 QEP-A/B 信号，以检测错误。之前的 QEP-A/B 值由节 5.4.1 中介绍的 QCLK 状态机 (FSM) 提供。

备注

此逻辑也可使用 LUT，因为 QEP-A 和 QEP-B 值之前的状态是由一个单独的 FSM 提供的。但逻辑块中的所有 LUT 均已使用，因此要利用未使用的 FSM 的输出来生成 ERROR 信号。

表 5-6 中介绍了三种可能的情况：

1. 正向或反向的有效运动，ERROR = 0
2. 无运动，ERROR = 0
3. QEP-A/B 同时改变值，ERROR = 1

表 5-6. 错误检测卡诺图

		当前状态 xe0，xe1 QEP-A(n)，B(n)			
		00	**01**	**11**	**10**
之前的状态 e0，e1 A(n-1)，B(n-1)	**00**	0 无运动	0 反向	1(3) 无效	0 正向
	01	0 正向	0 无运动	0 反向	1(4) 无效
	11	1 (1) 无效	0 正向	0 无运动	0 反向
	10	0 反向	1 (2) 无效	0 正向	0 无运动

(1) (!xe0 & !xe1 & e0 & e1)
(2) (!xe0 & xe1 & e0 & !e1)
(3) (xe0 & xe1 & !e0 & !e1)
(4) (xe0 & !xe1 & !e0 & e1)

5.4.4 QepOnClb 仿真波形

本节提供了 QEP 实现的仿真。更多有关 CLB 仿真的信息，请参阅 CLB 工具用户指南 SPRUIR8。

备注

1. QEP-A/B 输入激励由 CLB 逻辑块 4 生成，用到了一个计数器和两个 FSM，如图 5-5 中所示。每个 FSM 的 e0 输入发生改变时，s0 输出会进行切换。FSM 模块的输出连接到逻辑块 1 仿真输入 in3 和 in5。
2. 仅显示位置计数器的最后 5 位，以方便读取。
3. 此示例指定的位置上限 MAXPOS 为 0xD。

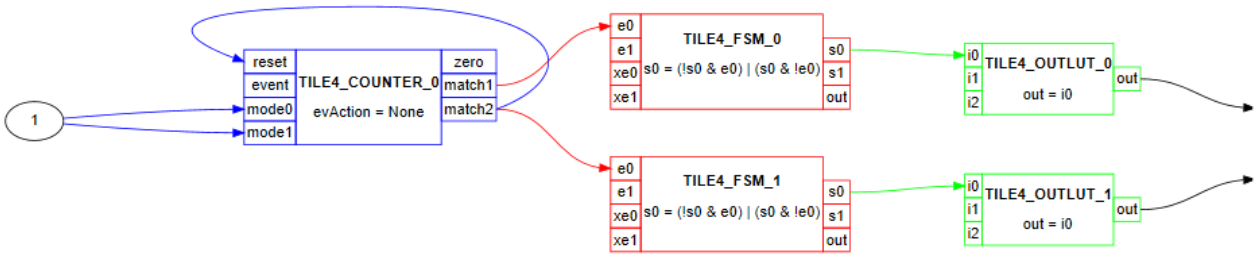


图 5-5. QepOnClb 仿真激励

图 5-6 和图 5-7 展示了正向运动仿真。match1 (MAXPOS) 输出接回计数器的复位（显示为橙色）。计数器达到 MAXPOS 时，下一个 CLB 时钟将复位为 0。

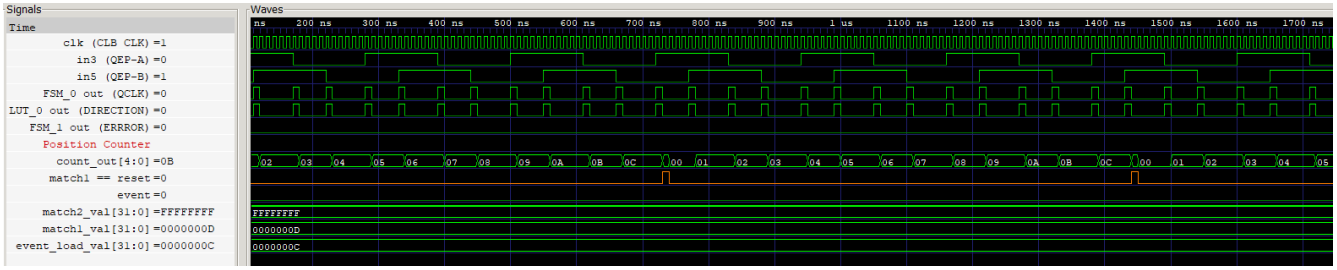


图 5-6. QepOnClb 正向仿真波形

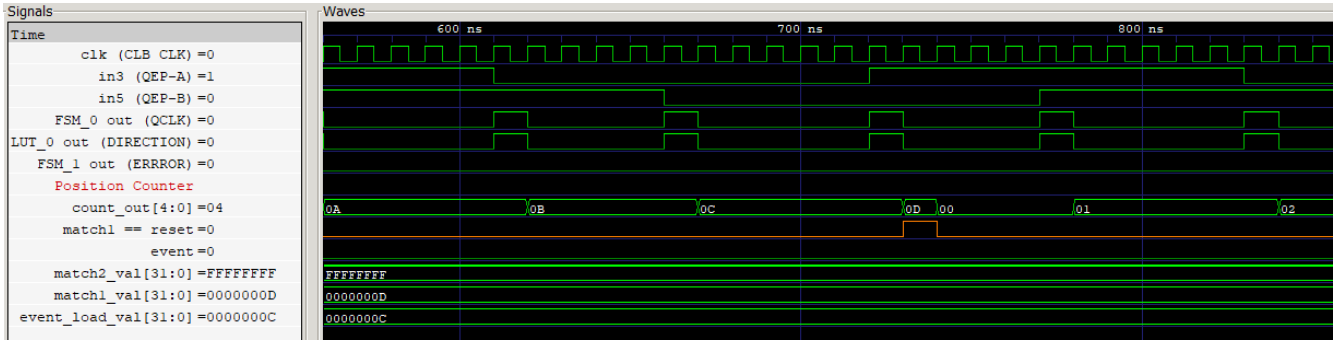


图 5-7. QepOnClb 正向 MAXPOS 仿真

图 5-8 和图 5-9 展示了反向运动仿真。match2 输出接回计数器的事件输入。如果计数器转换小于零 (0xFFFFFFFF)，负载值 (MAXPOS -1) 将载入计数器。

备注

仅显示位置计数器的低 5 位，以方便读取。0x1F 对应于 0xFFFFFFFF。

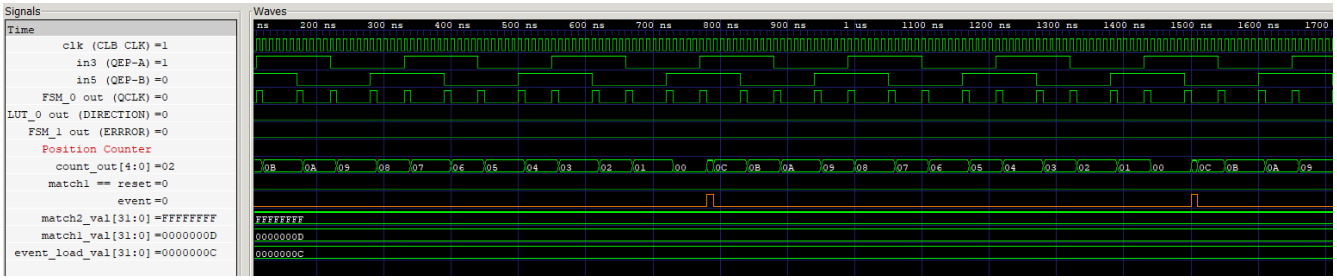


图 5-8. QepOnClb 反向仿真波形

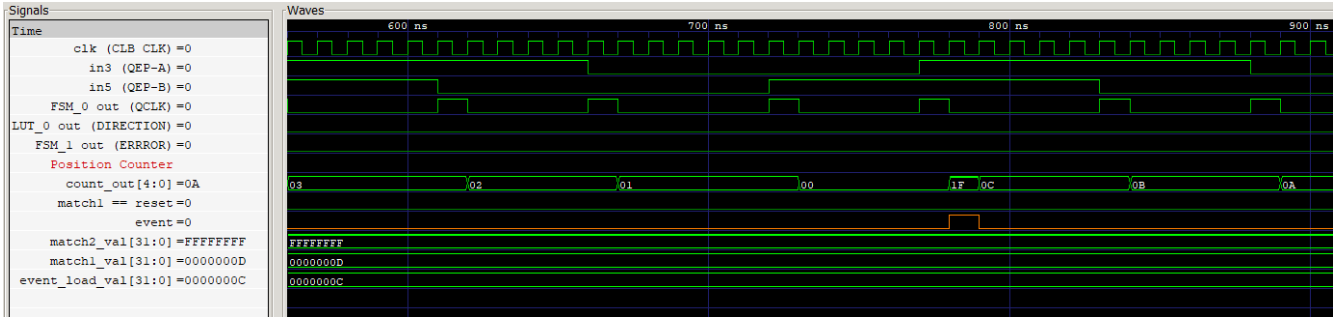


图 5-9. QepOnClb 反向 MAXPOS 仿真

图 5-10 展示了 QEP-A/B 同时转换时，错误检测信号为高电平。

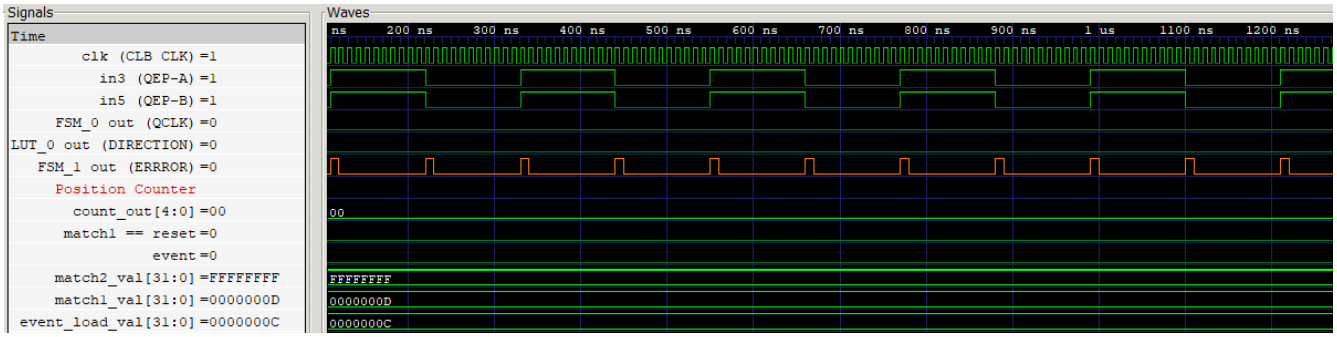


图 5-10. QepOnClb 错误检测仿真波形

6 示例工程

示例工程可以在表 6-1 所示的目录中找到。这些示例工程使用节 7 中所述的 API 库工程。

备注

一些 API 适用于脉冲序列输入 (PTI)，另外一些则适用于脉冲序列输出 (PTO)。为简单起见，示例、库和目录结构均使用后缀“pto”来标识属于此库的内容。

表 6-1. 示例解决方案的位置

C:\ti\c2000\C2000Ware_MotorControl_SDK_[version]\	SDK 的默认安装位置。([SDK])
[SDK]\solutions\boostxl_posmgr\	特定于器件的解决方案基础安装目录 ([pto_base])
[pto_base]\shared\source	与器件无关并在不同器件示例中使用的源代码。
[pto_base]\[device]\source	示例工程的源代码。包括 .c 和 .syscfg 文件。
[pto_base]\[device]\include	特定于示例的头文件。
[pto_base]\[device]\ccs\[pto_example]	Code Composer Studio (CCS) projectspec 文件。用于将工程导入您的 CCS 工作区。
[pto_base]\[device]\cmd	示例工程链接器命令文件 (.cmd)。

6.1 硬件要求

表 6-2 介绍了用于运行和测试 PTO 示例的硬件。

表 6-2. 硬件

器件	硬件
TMS320F28388D	F28388D controlCARD 评估模块 (TMDSCNCD28388D) 和扩展坞 (TMDSHSECDOCK)
TMS320F28379D	F28379D LaunchPad 开发套件 (LAUNCHXL-F28379D)
TMS320F280025	F280025C LaunchPad 开发套件 (LAUNCHXL-F280025C)
TMS320F280039C	F280039C LaunchPad 开发套件 (LAUNCHXL-F280039C (1))
TMS320F280049	F280049C LaunchPad 开发套件 (LAUNCHXL-F280049C)

(1) LAUNCHXL-F280039C 器件是即将推出的 LaunchPad，将于 2022 年第二季度发布。

6.2 安装 Code Composer Studio 和 C2000WARE-MOTORCONTROL-SDK™

安装所需的软件来构建和运行 PTO 示例：

1. 安装 Code Composer Studio v11.0.0 或更高版本（如果尚未在 PC 上安装）
2. 安装 C2000WARE-MOTORCONTROL-SDK v4.00.00.00 或更高版本（如果尚未在 PC 上安装）

备注

只需要上述软件即可构建这些示例。若要重新构建基于 CLB 的库，还需要 CLB 工具。此工具包含在 Code Composer Studio (sysconfig) 和 SDK 的 C2000Ware 子组件（支持实用程序）中。若要运行基于 CLB 的仿真，需要安装其他工具（请参阅 *CLB 工具用户指南* 中关于这些工具的说明）。

6.3 导入并运行示例工程

1. 在 CCS 或更高版本中，依次点击“*Project -> Import CCS Projects…*”。
2. 导航至特定于器件的 PTO 解决方案 CCS 目录。该路径如表 6-1 所示。
3. 选择所需的 projectspec，然后点击“*Finish*”。
4. 构建工程：
   a. 在工程管理器窗口中右键点击工程名称
   b. 选择“*Rebuild Project*”
   c. 观察“Console”窗口中是否有任何构建错误或构建成功完成的信息。
5. 在构建完成后且没有错误的情况下，通过选择“*Run ->Debug*”来执行该工程。
6. 通过按“Run”按钮来运行代码。
7. 按照以下涉及具体示例的几节中的说明来监视信号和变量。

6.4 PulseGen 示例

验证并监测输出波形信号。使用的 GPO 映射如表 6-3 所示：

表 6-3. PulseGen 输出信号到 GPIO 映射

器件	方向（通过 OUTPUTXBAR 路由）	脉冲输出（通过 OUTPUTXBAR 路由）
TMS320F280025C	GPIO15	GPIO45
TMS320F280039C	GPIO31	GPIO26
TMS320F280049C	GPIO15	GPIO26
TMS320F28379D	GPIO15	GPIO14
TMS320F28388D	GPIO15	GPIO26

6.5 QepDiv 示例

如映射表所示，测试输入和 PTO 输出在内部进行路由。在表 6-4 中列出的函数中可找到用于路由信号的代码。

表 6-4. QepDiv 示例输入/输出信号路由

函数	位置	注意事项
输入信号路由：GPIO 至 CLB		
pto_qepdiv_setup_GPIO()	示例应用	将输入 GPIO 连接到 INPUTXBAR。
pto_qepdiv_initCLBXBAR()	库	将 INPUTXBAR 路由到全局 CLB AUXSIGx 信号。
pto_qepdiv_setupPeriph()	库	将逻辑块输入连接到 CLB 全局多路复用器、CLB 本地多路复用器或逻辑块的 GPREG。
输出路由：CLB 至 GPIO		
函数	**位置**	**注意事项**
pto_qepdiv_initCLBXBAR	库	将逻辑块的 out4 或 out5 连接到 OUTPUTXBAR
pto_qepdiv_startOperation()	库	通过 setOutputMask() 使 CLB 输出能够覆盖外设信号
pto_qepdiv_setup_GPIO()	示例应用	将 GPIO 输出连接到外设或 OUTPUTXBAR

表 6-5. F28002x、F28003x、F28004x、F2837x 和 F2838x QepDiv 输出 GPIO 映射

QepDiv 输入			QepDiv 输出		
输入信号	**连接目标（供演示）** (1)	**路由到 CLB**	**输出信号**	**从 CLB 路由**	**GPIO 引脚**
QEP-A：GPIO10	EPWM4A/GPIO6 或外部信号	INPUTXBAR4 → AUXSIG0 → Tile1 in1、in2 和 Tile2 in1、in2	PTO_QEP-A	覆盖 PWM2A	GPIO2
QEP-B：GPIO11	EPWM5A/GPIO8 或外部信号	INPUTXBAR5 → AUXSIG1 → Tile1 in4、in5 和 Tile2 in4	PTO_QEP-B	覆盖 PWM2B	GPIO3
QEP-I：GPIO9	EPWM4B/GPIO7 或外部信号	INPUTXBAR6 → AUXSIG2 → Tile1 in7	PTO_QEP-I	Tile1 out5 → OUTPUTXBAR3	GPIO5

(1) 本示例使用备用 EPWM 来提供 QEP 输入。这些仅用于测试目的，并不对应于实时使用情况。您可以选择将这些 EPWM 输出连接到 QepDiv 输入信号，也可以选择连接其他外部信号。

表 6-6 列出了使用 EPWM 作为 QepDiv 输入时所需的连接。

表 6-6. QepDiv 测试输入连接

板	EPWM4A 至 QEP-A	EPWM5A 至 QEP-B	EPWM4B 至 QEP-I
LAUNCHXL-F280025C	78 (IO.6) 至 14 (IO.10)	76 (IO.16) 至 15 (IO.11)	77 (IO.7) 至 7 (IO.9)
LAUNCHXL-F280039C	78 (IO.6) 至 36 (IO.10)	76 (IO.16) 至 35 (IO.11)	77 (IO.7) 至 7 (IO.9)
LAUNCHXL-F280049C	78 至 40	38 至 39	77 至 37
LAUNCHXL-F28379D	80 至 76	78 至 75	79 至 77
TMDSCNCD28388D	54 至 61	57 至 63	56 至 59

6.6 Abs2Qep 示例

本示例使用 PWM 计时器来模拟位置采样率。绝对位置值由测试函数提供。通过/失败标准假设 PTO 的输出端从外部连接到 eQEP 外设。

6.6.1 观察变量

以下观察变量将提供通过/失败信息：

- passCount
- failCount
- deltaMax
- EQep1Regs.QPOSCNT

绝对位置的角度将与对应于 eQEP 位置计数器 (QPOSCNT) 的角度进行比较。如果差值小于指定的阈值，则 passCount 会增加 。如果不是，则 failCount 会增加。最大差值会记录在 deltaMax 中。

6.6.2 测试信号

提供了两个测试信号以协助查看波形：

- **测试信号 1：**在每个 PWM ISR 开始时切换。如果 PTO 方向是正向，则会保持高电平。如果 PTO 方向是反向，则会保持低电平。
- **测试信号 2：** Abs2Qep 内部的 HALT/RUN 信号。该信号可用于准确显示 PTO 在何处停止，以及它何时相对于测试信号 1 的 ISR 切换进行重新启动。

6.6.3 引脚用途和测试连接

PTO 的输出通过 OUTPUTXBAR 并以覆盖 PWM1-B 的方式在内部路由到 GPIO。在表 6-7 列出的函数中可以找到用于路由信号的代码。

表 6-7. Abs2Qep 输出信号路由

函数	位置	注意事项
pto_abs2qep_initCLBXBAR	库	将逻辑块 out4/5 连接到 OUTPUTXBAR
pto_abs2qep_setupPeriph	库	使输出能够覆盖外设。
pto_setupGPIO	示例应用	将 OUTPUTXBAR 连接到 GPIO 输出

表 6-8. F2838xD Abs2Qep 输出 GPIO 映射

Abs2Qep 输出			测试连接
Abs2Qep 信号	**从 CLB 路由到 GPIO**	**GPIO/TMDSHSECDOCK 引脚**	**将 Abs2Qep 输出连接到：**
PTO-QEP-A	Tile1 out4 到 OUTPUTXBAR7	GPIO16/引脚 67	GPIO20_EQEP1A/引脚 68
PTO-QEP-B	Tile1 out5 到 OUTPUTXBAR2	GPIO3/引脚 55	GPIO21_EQEP1B/引脚 70
PTO-QEP-I	Tile1 out2，覆盖 PWM1-B	GPIO1/引脚 51	GPIO99_EQEP1I/引脚 96
测试信号 1：系统 PWM ISR/方向	无	GPIO32/引脚 85	使用示波器进行监视
测试信号 2：内部 HALT/RUN	Tile1 out0，覆盖 PWM1-A	GPIO0/引脚 49	使用示波器进行监视

表 6-9. F2837xD Abs2Qep 输出 GPIO 映射

Abs2Qep 输出			测试连接
Abs2Qep 信号	**从 CLB 路由到 GPIO**	**GPIO/LAUNCHXL-F28379D 引脚**	**将 Abs2Qep 输出连接到：**
PTO-QEP-A	Tile1 OUT4 到 OUTPUTXBAR7	GPIO16/J4-33	GPIO20_EQEP1A/J14-1
PTO-QEP-B	Tile1 OUT5 到 OUTPUTXBAR2	GPIO3/J4-37	GPIO21_EQEP1B/J14-2
PTO-QEP-I	Tile1 OUT2，覆盖 PWM1-B	GPIO1/J4-39	GPIO99_EQEP1I/J14-3
测试信号 1：系统 PWM ISR/方向	无	GPIO32/J1-2	使用示波器进行监视
测试信号 2：内部 HALT/RUN	Tile1 OUT0，覆盖 PWM1-A	GPIO0/J4-40	使用示波器进行监视

表 6-10. F280049C Abs2Qep 输出 GPIO 映射

Abs2Qep 输出			测试连接
Abs2Qep 信号	从 CLB 路由到 GPIO	GPIO/LAUNCHXL-F280049C 引脚(1)	将 Abs2Qep 输出连接到：
PTO-QEP-A	Tile1 OUT4 到 OUTPUTXBAR1	GPIO24/J8-55	GPIO10_EQEP1A/J4-40
PTO-QEP-B	Tile1 OUT5 到 OUTPUTXBAR2	GPIO3/J6-75	GPIO11_EQEP1A/J4-39
PTO-QEP-I	Tile1 OUT2，覆盖 PWM1-B	GPIO1/J6-79	GPIO9_EQEPI/J4-37
测试信号 1：PWM ISR/方向	-	GPIO13/J1-3	使用示波器进行监视。
测试信号 2：HALT/RUN	Tile1 OUT0，覆盖 PWM1-A	GPIO0/J6-80	使用示波器进行监视。

(1) J8 和 J6 在 RevA F28004x LaunchPad 的丝印层上互换。此表中的 J8 和 J6 引脚编号对应 Rev A 上的丝印层。若要确认这是否适用于您的电路板，请参阅 *C2000™ Piccolo™ F28004x 系列 LaunchPad™ 开发套件* 已修订章节中的“已知问题”。

表 6-11. F280025C Abs2Qep 输出 GPIO 映射

Abs2Qep 输出			测试连接
Abs2Qep 信号	从 CLB 路由到 GPIO	GPIO/LAUNCHXL-F280025C 引脚	将 Abs2Qep 输出连接到：
PTO-QEP-A	Tile1 OUT4 到 OUTPUTXBAR1	GPIO24/J5-44/45	GPIO10_EQEP1A/J2-14
PTO-QEP-B	Tile1 OUT5 到 OUTPUTXBAR2	GPIO3/J4-37	GPIO11_EQEP1A/J2-15
PTO-QEP-I	Tile1 OUT2，覆盖 PWM1-B	GPIO1/J4-39	GPIO9_EQEPI/J1-7
测试信号 1：PWM ISR/方向	-	GPIO13/J8-79	使用示波器进行监视。
测试信号 2：HALT/RUN	Tile1 OUT0，覆盖 PWM1-A	GPIO0/J4-40	使用示波器进行监视。

表 6-12. F280039C Abs2Qep 输出 GPIO 映射

Abs2Qep 输出			测试连接
Abs2Qep 信号	从 CLB 路由到 GPIO	GPIO/LAUNCHXL-F280039C 引脚	将 Abs2Qep 输出连接到：
PTO-QEP-A	Tile1 OUT4 到 OUTPUTXBAR1	GPIO24/J1-8	GPIO10_EQEP1A/J4-36
PTO-QEP-B	Tile1 OUT5 到 OUTPUTXBAR2	GPIO3/J4-37	GPIO11_EQEP1A/J4-35
PTO-QEP-I	Tile1 OUT2，覆盖 PWM1-B	GPIO1/J4-39	GPIO9_EQEPI/J1-7
测试信号 1：PWM ISR/方向	-	GPIO13/J8-79	使用示波器进行监视。
测试信号 2：HALT/RUN	Tile1 OUT0，覆盖 PWM1-A	GPIO0/J4-40	使用示波器进行监视。

6.7 QepOnClb 示例

此示例在 CLB 外设上实现了一个简单的 QEP 解码器。将此基于 CLB 的 QEP 解码器的输出与基于 eQEP 的 QEP 解码器输出进行比较。器件生成 EPWM 信号，用作测试输入来模拟 QEP-A 和 QEP-B 信号。

6.7.1 观察变量

以下观察变量提供 QepOnClb 信息，用于比较 eQEP 和 CLBQEP 的性能：

- eqepPosition
- clbqepPosition
- deltaPosition
- maxDeltaPosition
- interruptCount
- sendIndex

在示例执行期间，eqepPosition 和 clbqepPosition 分别跟踪 eQEP 和 CLBQEP 的位置。如果 eqepPosition 和 clbqepPosition 不同，差别将显示在 deltaPosition 变量中。在整个程序执行过程中发现的最大 deltaPosition 存储在 maxDeltaPosition 中。

6.7.2 接头引脚连接

在本例中，EPWM 由器件生成，旨在模拟测试 QEP 信号。这些 EPWM 信号需要从外部路由到 CLB INPUTXBAR 和板载 eQEP 外设。以下两个表介绍了需要根据所用器件进行的必要引脚连接。

备注

EPWM 信号作为测试输入，展示了 QepOnClb 示例的功能。如有需要，用户也可以将外部 QEP-A 和 QEP-B 信号路由到 INPUTXBAR 和 eQEP 外设。

基于 CLB 的 QEP 解码器模块经配置，可接受对应 QEP-A、QEP-B 和 QEP-I 的三个输入。QEP-A 信号应路由到 INPUTXBAR2，QEP-B 信号应路由到 INPUTXBAR1，QEP-I 信号应路由到 INPUTXBAR3。

表 6-13 列出了将 EPWM 信号路由到 CLB X-BAR 所需的连接。

表 6-13. QepOnClb EPWM 到 CLB INPUTXBAR 的连接

板	EPWMA 到 INPUTXBAR2 (QEP-A)	EPWMB 到 INPUTXBAR1 (QEP-B)	GPIO 到 INPUTXBAR3 (QEP-I)
LAUNCHXL-F280025C	GPIO0 (J4-40) 到 GPIO8 (J2-12)	GPIO1 (J4-39) 到 GPIO9 (J1-7)	GPIO2 (J4-38) 到 GPIO27 (J2-11)
LAUNCHXL-F280039C	GPIO0 (J4-40) 到 GPIO8 (J2-15)	GPIO1 (J4-39) 到 GPIO9 (J1-7)	GPIO2 (J4-38) 到 GPIO27 (J6-59)
LAUNCHXL-F280049C	GPIO10 (J4-40) 到 GPIO39 (J2-13)	GPIO11 (J4-39) 到 GPIO40 (J1-4)	GPIO8 (J4-38) 到 GPIO27 (J6-59)
LAUNCHXL-F28379D	GPIO0 (J4-40) 到 GPIO18 (J1-4)	GPIO1 (J4-39) 到 GPIO40 (J5-50)	GPIO2 (J4-38) 到 GPIO27 (J6-52)
TMDSCNCD28388D	GPIO0（引脚 49）到 GPIO18（引脚 71）	GPIO1（引脚 51）到 GPIO40（引脚 89）	GPIO2（引脚 53）到 GPIO27（引脚 81）

将 EPWM 信号路由到 eQEP 外设进行环回测试，这是为了与基于 CLB 的 QEP 模块进行比较。

表 6-13 列出了将 EPWM 信号路由到 eQEP 外设所需的连接。

表 6-14. QepOnClb EPWM 到 eQEP 的连接

板	EPWMA 到 eQEP-A	EPWMB 到 eQEP-B	GPIO 到 eQEP-I
LAUNCHXL-F280025C	GPIO0 (J4-40) 到 GPIO25 (J4-31)	GPIO1 (J4-39) 到 GPIO29 (J1-4/5)	GPIO2 (J4-38) 到 GPIO23 (J2-13)
LAUNCHXL-F280039C	GPIO0 (J4-40) 到 GPIO25 (J6-51)	GPIO1 (J4-39) 到 GPIO29 (J1-4/5)	GPIO2 (J4-38) 到 GPIO23 (J2-11)
LAUNCHXL-F280049C	GPIO10 (J4-40) 到 GPIO35 (J1-10)	GPIO11 (J4-39) 到 GPIO37 (J1-9)	GPIO8 (J4-38) 到 GPIO59 (J2-11)
LAUNCHXL-F28379D	GPIO0 (J4-40) 到 GPIO20 (QEP1A)	GPIO1 (J4-39) 到 GPIO21 (QEP1B)	GPIO2 (J4-38) 到 GPIO23 (QEP1I)
TMDSCNCD28388D	GPIO0（引脚 49）到 GPIO20（引脚 68）	GPIO1（引脚 51）到 GPIO21（引脚 70）	GPIO2（引脚 53）到 GPIO23（引脚 74）

7 库源代码和工程

本节介绍了如何导入和重新构建库，并提供每个 API 函数的说明。每个 Code Composer Studio 库工程都包含可配置逻辑块 (CLB) 的配置信息。若要了解如何修改 CLB 的配置，请参阅 *CLB 工具用户指南*。

备注

一些 API 适用于脉冲序列输入 (PTI)，另外一些则适用于脉冲序列输出 (PTO)。为简单起见，示例、库和目录结构均使用后缀“pto”来标识属于此库的内容。

7.1 查找库源代码

PTO API 和源代码可以在表 7-1 所示的位置找到。

表 7-1. PTO 库的位置

C:\ti\c2000\C2000Ware_MotorControl_SDK_[version]	SDK 的默认安装位置。([SDK])
[SDK]\libraries\position_sensing\pto	库基础安装目录 ([lib_base])
[lib_base]\ccs\[device]	参考库的 Code Composer projectspec 文件。使用这些工程可为每个器件重新构建库。
[lib_base]\lib	生成的库目标文件 (.lib)
[lib_base]\source	库源代码 (.c) 和 CLB 配置 (.syscfg) 文件。
[lib_base]\include	库头文件。通过 #include 在调用库的应用程序工程中包含这个文件。
[lib_build_dir]\RELEASE\syscfg	CLB 逻辑块图的位置。通过重新构建编译的对象，CCS 将重新生成 CLB 逻辑块图 (clb.svg 或 clb.html) 和对象 (.lib)。您可以通过 CCS Project Explorer 访问 .svg 或 html 文件。

7.2 导入和构建库工程

若要重新构建 API 库对象，请执行以下过程：

1. 安装节 6.2 中描述的必要软件工具 (如果尚未安装)。
2. 在 CCS 或更高版本中，依次点击“*Project -> Import CCS Projects…*”。
3. 导航至器件对应的 Code Composer Studio (CCS) projectspec 目录，请参阅表 7-1。
4. 选择所需的库工程，然后点击“*Finish*”。
5. 在 CCS Project Explorer 窗口中，展开所选的工程，然后打开对应的 SysConfig 文件 (例如，“pto_pulsegen.syscfg”)。
6. 检查逻辑块的配置，并观察 LUT 和 LUT 中的逻辑表达式以及输出 LUT。
7. 在 CCS 菜单中依次选择“*Project -> Build Project*”。
8. 此时会在 [lib_base]\lib 文件夹中生成一个输出对象 (.lib)。此文件将包含在 PTO 示例工程中。此对象是 PTO 源文件的已编译对象。
9. [可选] – 如需了解如何运行基于 CLB 的工程的仿真，请参阅 *CLB 工具用户指南* 中的*运行仿真* 一节。

7.3 PTO - PulseGen API

本节详细介绍了 PulseGen 库函数。有关查找源代码和重新构建库的信息，请参阅节 7。

PulseGen 包含文件：*pto_pulsegen.h*

表 7-2. PTO-PulseGen API 函数

名称	说明	类型
pto_pulsegen_reset	用于重置由早期配置所设置的 pulsegen 参数并开始全新的设置。如果脉冲生成需要重置并在稍后阶段再次启动，则需要调用该函数。	初始化时间
pto_pulsegen _setupPeriph	在系统初始化期间使用该函数设置 CLB 和其他互连 XBAR。每次系统复位后都需要调用该函数。在调用设置外设函数之前，不会执行任何事务。	初始化时间
pto_pulsegen _startOperation	该函数将启动接口上的脉冲生成。在 pto_pulsegen_setupPeriph 之后进行调用。执行由先前函数设置的事务。请注意，设置和启动操作是单独的函数调用。您可以根据需要设置外设，并根据需要在其他时间使用该函数调用来启动实际的脉冲生成。	运行时
pto_pulsegen _runPulseGen	这是定期调用的运行时函数，用于根据应用程序的需要动态配置和更改脉冲生成要求。需要使用相应的参数（例如脉冲数、周期、持续时间等）定期调用该函数。稍后的章节将提供相关详细信息。	运行时

7.3.1 pto_pulsegen_runPulseGen

说明

这是定期调用的运行时函数，用于根据应用程序的要求动态配置和更改脉冲生成要求。必须使用相应的参数（例如脉冲数、周期、持续时间等）定期调用该函数。

定义

```
uint16_t pto_pulsegen_runPulseGen(
        uint32_t pulseLo,
        uint32_t pulseHi,
        uint32_t ptoActivePeriod,
        uint32_t ptoFullPeriod,
        uint32_t ptoInterruptTime,
        uint16_t ptoDirection,
        uint16_t run
        );
```

参数

输入：

- pulseLo – 低脉宽
- pulseHi – 高脉宽
- ptoActivePeriod – 发出脉冲的周期；小于 ptoFullPeriod
- ptoFullPeriod – 完整的 PTO 期间
- ptoInterruptTime – 对 CPU 产生中断的时间
- ptoDirection – 方向输出；新的周期开始时锁定在方向输出上
- run – 值指示 1（运行）和 0（停止）。在新周期开始时进行采样以确定继续还是停止脉冲生成

返回：

- Val – 如果函数执行成功，其将返回 ptoFullPeriod 作为返回值

使用

在 pto_pulsegen.c 中提供了一个名为 pto_setOptions 的示例配置函数作为示例，旨在协助 pto_pulsegen_runPulseGen 函数以及执行中间计算。请参阅以下代码计算示例，其中说明了如何生成该函数的各种参数。更多详细信息，请参阅 pto_pulsegen.c 和 pto_pulsegen.h 文件。

```
uint32_t pto_setOptions(
        uint32_t numPulses, //number of pulses needed to be generated in next period
        uint32_t Period,        // PTO period in clock cycles
        uint32_t ptoInterruptTime, // Interrupt generation time
        uint16_t ptoDirection,    // Direction output
        uint16_t run)        //run-stop condition.
{
    uint32_t pulseFreq, reminder;
    uint32_t pulseLo;
    uint32_t pulseHi;
    uint32_t ptoActivePeriod;
    uint32_t ptoFullPeriod;
    pulseFreq = Period / numPulses;
    reminder = Period - (pulseFreq * numPulses);
    pulseLo = (pulseFreq/2 );
    pulseHi = pulseFreq;
    ptoActivePeriod = (pulseFreq * numPulses);
    ptoFullPeriod = Period;
    pto_pulsegen_runPulseGen(
            pulseLo,
            pulseHi,
            ptoActivePeriod,
            ptoFullPeriod,
            ptoInterruptTime,
            ptoDirection,
            run);
    return(reminder);
}
```

7.3.2 pto_startOperation

说明

该函数可启动脉冲生成。必须在 pto_pulsegen_setupPeriph 之后调用该函数。因此，pto_pulsegen_startOperation 将启动先前设置的脉冲生成。

备注

通过分开的函数调用设置和启动操作。用户可以设置传输，并根据需要在其他时间使用该函数调用来启动脉冲生成。

定义

```
void pto_pulsegen_startOperation(void);
```

参数

输入：无

返回：无

使用

示例代码：

```
pto_initPulsegen ();
SysCtl_delay (800L);
pto_pulsegen_startOperation ();
retval1 = pto_pulsegen_runPulseGen (7, 15, 960, 990, 500, 1, 1);
```

7.3.3 pto_pulsegen_setupPeriph

说明

该函数在系统初始化期间执行 CLB 和其他互连 XBAR 的设置。每次系统复位后都必须调用该函数。在调用设置外设函数之前，不会执行任何事务。

定义

```
void pto_pulsegen_setupPeriph (void);
```

参数

输入：无

返回：无

使用

示例代码：

```
pto_pulsegen_setupPeriph();
```

7.3.4 pto_pulsegen_reset

说明

该函数重置由早期配置（PulseGen 函数调用）所设置的 pulsegen 参数并开始新的设置。如果脉冲生成必须重置并在稍后阶段再次启动，则必须调用该函数。

定义

```
void pto_pulsegen_reset (void);
```

参数

输入：无

返回：无

使用

示例代码：

```
pto_pulsegen_reset();
```

7.4 PTO - QepDiv API

本节详细介绍了 QepDiv 库函数。有关查找源代码和重新构建库的信息，请参阅节 7。

QepDiv 包含文件：*pto_qepdiv.h*

表 7-3. PTO-QepDiv API 函数

名称	说明	类型
pto_qepdiv_reset	用于重置由早期配置所设置的 qepdiv 参数并开始全新的设置。如果脉冲生成需要重置并在稍后阶段再次启动，则需要调用该函数。	初始化时间
pto_qepdiv _setupPeriph	在系统初始化期间使用该函数设置 CLB 和其他互连 XBAR。每次系统复位后都需要调用该函数。在调用设置外设函数之前，不会执行任何事务。	初始化时间
pto_qepdiv _startOperation	该函数将启动接口上的脉冲生成。在 pto_qepdiv_setupPeriph 之后进行调用。执行由先前函数设置的事务。请注意，设置和启动操作是单独的函数调用。用户可以根据需要设置外设，并根据需要在其他时间使用该函数调用来启动实际的脉冲生成。	运行时
pto_qepdiv_config	该函数用于配置分频器，分频器值无法动态改变。在重新配置功能之前，用户需要使用 pto_qepdiv_reset 来重置模块。	运行时

7.4.1 pto_qepdiv_config

说明

该函数可配置分频器。分频器值不能动态更改。在重新配置功能之前，用户必须使用 pto_qepdiv_reset 来重置模块。

定义

```
pto_qepdiv_config(uint16_t divider, uint16_t indexWidth);
```

参数

输入：

- 分频器 – 分频器的值
- 索引宽度 – 索引脉冲输出保持开启的周期数

返回：

- Val – 如果函数执行成功，则将返回 ptoFullPeriod 作为返回值

使用

示例代码：

```
retval1 = pto_qepdiv_config(4, 10);
    pto_qepdiv_startOperation(1);
```

7.4.2 pto_startOperation

说明

该函数可启动脉冲生成。只能在 pto_qepdiv_setupPeriph 之后调用该函数。因此，pto_qepdiv_startOperation 函数将启动先前设置的脉冲生成。

备注

通过分开的函数调用设置和启动操作。用户可以设置传输，并根据需要在其他时间使用该函数调用来启动脉冲生成。

定义

```
void pto_qepdiv_startOperation(uint16_t run);
```

参数

输入（为启动或停止函数而需要传递的参数）：

- 启动 = 1
- 停止 = 0

返回：无

使用

示例代码：

```
retval1 = pto_qepdiv_config(4, 10);
    pto_qepdiv_startOperation(1);
```

7.4.3 pto_qepdiv_setupPeriph

说明

在系统初始化期间使用 pto_qepdiv_setupPeriph 函数设置 CLB 和其他互连 XBAR。每次系统复位后都必须调用该函数。在调用设置外设函数之前，不会执行任何事务。

定义

```
void pto_qepdiv_setupPeriph (void);
```

参数

输入：无

返回：无

使用

示例代码：

```
pto_qepdiv_setupPeriph();
```

7.4.4 pto_qepdiv_reset

说明

用于重置由早期配置所设置的 qepdiv 参数并开始全新的设置。如果脉冲生成必须重置并在稍后阶段再次启动，则必须调用该函数。

定义

```
void pto_qepdiv_reset (void);
```

参数

输入：无

返回：无

使用

示例代码：

```
pto_qepdiv_reset();
```

7.5 PTO - Abs2Qep API

本节详细介绍了 Abs2Qep 库函数。有关查找源代码和重新构建库的信息，请参阅节 7。

Abs2Qep 包含文件：*pto_abs2qep.h*

表 7-4. PTO-Abs2Qep API 函数

名称	说明	类型
pto_abs2eqep _setupPeriph	在系统初始化期间使用该函数设置 CLB 和其他互连 XBAR。每次系统复位后都要调用该函数。在调用设置外设函数之前，不会执行任何事务。	初始化时间
pto_abs2qep _translatePosition	将绝对位置的变化转换为等效的增量位置变化。该函数将配置 CLB 以生成：QCLK 的数量、QCLK 频率和 QEP-I 脉冲。用于转换的参数可以在库头文件中进行配置。该函数将配置加载到 HLC FIFO 中。	运行时
pto_abs2qep _runPulseGen	该函数将启动接口上的脉冲生成。在 pto_abs2qep_translatePosition 设置 CLB HLC FIFO 后进行调用。请注意，通过分开的函数调用设置和启动操作。	运行时

7.5.1 Abs2Qep API 配置

库的头文件 `pto_abs2qep.h` 中包含一些参数，可修改这些参数以根据不同的编码器和位置采样率对库进行配置。这些参数包括：

- 位置采样周期
- 电机每分钟最大转数 (RPM)
- 绝对编码器分辨率
- 增量编码器每转线数

7.5.2 pto_abs2qep_runPulseGen

说明

在运行时调用的函数，用于启动新的 PTO。该函数在启动新 PTO 之前检查前一个 PTO 是否已完成。

备注

通过分开的函数调用设置和启动操作。必须在该函数之前调用设置 (pto_abs2qep_translatePosition)。您可以设置传输，并根据需要在其他时间使用该函数调用来启动脉冲生成。

定义

```
    void
    pto_abs2qep_runPulseGen(
        uint16_t ptoDirection
    );
```

参数

输入：

- ptoDirection：PTO 的方向。此参数决定哪个信号将引导 QEP-A 或 QEP-B。

返回：无

使用

```
// Call to sample a new absolute position
    ....
// Translate change from previous position to PTO configuration
    ptoDirection = pto_abs2qep_translatePosition(absolutePosition);
    ....
// Start the last configuration
    pto_abs2qep_runPulseGen(ptoDirection);
```

7.5.3 pto_abs2qep_setupPeriph

说明

该函数在系统初始化期间执行 CLB 和 XBAR 互连的设置。每次系统复位后都必须调用该函数。在调用设置外设函数之前，不会执行任何事务。

定义

```
void pto_abs2qep_setupPeriph(void);
```

参数

输入：无

返回：无

使用

```
pto_abs2qep_setupPeriph();
...
//
// GPIO and other system peripheral configuration
//
```

7.5.4 pto_abs2qep_translatePosition

说明

该函数将绝对位置的变化转换为要加载到 CLB FIFO 中的等效 PTO 配置。相关信息包括：

- 生成 QEP-A 和 QEP-B 脉冲所需的 QCLK 数量
- 如果过零，则包括应在哪个 QCLK 边沿将 QEP-I 驱动为高电平和低电平的信息
- 每个 QCLK 之间的 CLB 时钟数
- 位置变化的方向。

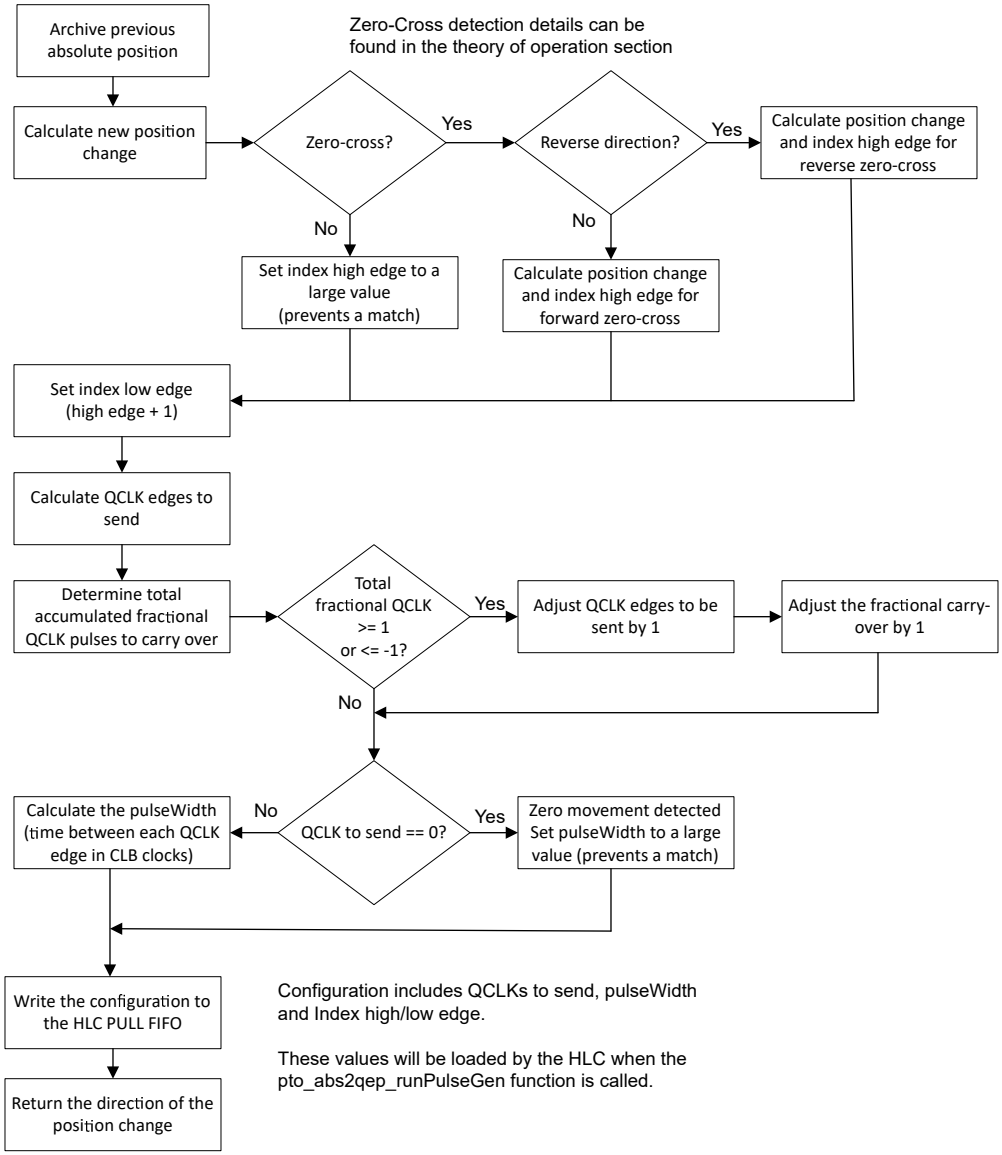


图 7-1. Abs2Qep 转换函数

定义

```
uint16_t
    pto_abs2qep_translatePosition(
        uint32_t positionNew
    );
```

参数

输入：

- positionNew - 由系统采样的新绝对位置。转换函数会将此值与前一个样片进行比较以确定位置变化。

返回：ptoDirection - 指示 PTO 的方向。

- PTO_ABS2QEP_CLOCKWISE_PTO
- PTO_ABS2QEP_COUNTERCLOCKWISE_PTO

备注

该函数将 PTO 配置直接加载到 HLC PULL FIFO 中。

使用

```
// Call to sample a new absolute position
    ....
// Translate change from previous position to PTO configuration
    ptoDirection = pto_abs2qep_translatePosition(absolutePosition);
    ....
// Start the last configuration
    pto_abs2qep_runPulseGen(ptoDirection);
```

7.6 PTO - QepOnClb API

本节详细介绍了 QepOnClb 库函数。有关查找源代码和重新构建库的信息，请参阅节 7。

QepOnClb 头文件：*pto_qeponclb.h*

表 7-5. PTO-QepOnClb API 函数

名称	说明	类型
pto_qeponclb_setupPeriph	在系统初始化期间使用该函数设置 CLB 和其他互连 XBAR。每次系统复位后都需要调用该函数。在调用设置外设函数之前，无法完成解码。	初始化时间
pto_qeponclb_initCLBQEP	辅助函数，可初始化 CLB 外设以模拟 QEP 解码器。此函数在 pto_qeponclb_setupPeriph 函数中使用，属于 CLBQEP 系统初始化的一部分。	初始化时间
pto_qeponclb_configMaxCounterPos	此函数配置 CLBQEP 的计数器值上限，并加载为 CLB 逻辑块中的一个参数。	初始化时间
pto_qeponclb_enableCLBQEP	用于启用 CLBQEP 计数器，以开始解码。CLBQEP 计数器需要在启用前进行初始化。	运行时
pto_qeponclb_resetCLBQEP	用于复位之前配置所设置的 CLBQEP 参数。如果计数器需要复位并在稍后的实例中再次启动，则需要调用该函数。	运行时
pto_qeponclb_getCounterVal	此函数从 CLB 中捕捉计数器 0 的当前值。	运行时
pto_qeponclb_getCLBQEPPos	如果要从 CLB 捕捉基于 CLB 的 QEP 外设的当前位置，需要调用此函数。	运行时
pto_qeponclb_clearFIFOptr	如果需要清除 CLB 的 FIFO 指针，需要调用此函数。读取由 HLC 推送的 FIFO 值后，需要调用此函数，使下一次的 PUSH 发生在正确的位置。	运行时

7.6.1 pto_qeponclb_setupPeriph

说明

此函数在系统初始化期间设置 CLB 和其他互连 XBAR，以允许使用 CLBQEP 外设。每次系统复位后都需要调用该函数。在调用设置外设函数之前，无法完成 QEP 解码。

定义

```
void pto_qeponclb_setupPeriph(uint32_t maxPosition);
```

参数

输入：

- maxPosition - CLBQEP 的计数器值上限

返回：无

使用

示例代码：

```
uint32_t max_position;
max_position = 500;
pto_qeponclb_setupPeriph(max_position);
```

7.6.2 pto_qeponclb_initCLBQEP

说明

本函数是一种辅助函数，可初始化 CLB 外设以模拟 QEP 解码器。此函数在 pto_qeponclb_setupPeriph 函数中使用，属于 CLBQEP 系统初始化的一部分。

定义

```
void pto_qeponclb_initCLBQEP(uint32_t maxPosition);
```

参数

输入：

- maxPosition - CLBQEP 的计数器值上限

返回：无

使用

示例代码：

```
uint32_t max_position;
max_position = 500;
pto_qeponclb_initCLBQEP(max_position);
```

7.6.3 pto_qeponclb_configMaxCounterPos

说明

此函数会配置 CLBQEP 的计数器值上限，并加载此值，作为 CLB 逻辑块中的一个参数。

定义

```
void pto_qeponclb_configMaxCounterPos(uint32_t clbBase, uint32_t maxPosition);
```

参数

输入：

- clbBase – CLB 逻辑块的基址
- maxPosition - CLBQEP 的计数器值上限

返回：无

使用

示例代码：

```
uint32_t max_position;
max_position = 500;
pto_qeponclb_configMaxCounterPos(max_position);
```

7.6.4 pto_qeponclb_enableCLBQEP

说明

此函数用于启用 CLBQEP 计数器，以开始解码。CLBQEP 计数器需要在启用前进行初始化。

Definition

```
void pto_qeponclb_enableCLBQEP(uint32_t clbBase, uint32_t enableCapture);
```

参数

输入：

- clbBase – CLB 逻辑块的基址
- enableCapture - 对应启用 CLBQEP 的 GPREG 位

返回：无

使用

示例代码：

```
pto_qeponclb_enableCLBQEP(0x00003000, (1 << 2));
```

7.6.5 pto_qeponclb_resetCLBQEP

说明

此函数用于复位由之前的配置设置的 CLBQEP 参数。如果计数器需要复位并在稍后的实例中再次启动，则需要调用该函数。

定义

```
void pto_qeponclb_resetCLBQEP(uint32_t clbBase, uint32_t resetCounter);
```

参数

输入：

- clbBase – CLB 逻辑块的基址
- resetCounter - 对应复位 CLBQEP 的 GPREG 位

返回：无

使用

示例代码：

```
pto_qeponclb_resetCLBQEP(0x00003000, (1 << 0));
```

7.6.6 pto_qeponclb_getCounterVal

说明

此函数从 CLB 中捕捉计数器 0 的当前值。

定义

```
uint32_t pto_qeponclb_getCounterVal(uint32_t clbBase);
```

参数

输入：

- clbBase – CLB 逻辑块的基址

返回：

- Val - 该函数将返回 counterVal 作为返回值

使用

示例代码：

```
#Define CLB1_BASE 0x00003000U
uint32_t retVal1;
retVal1 = pto_qeponclb_getCounterVal(CLB1_BASE);
```

7.6.7 pto_qeponclb_getCLBQEPPos

说明

如果要从 CLB 捕捉基于 CLB 的 QEP 外设当前的位置，需要调用此函数。

定义

```
uint32_t pto_qeponclb_getCLBQEPPos(uint32_t clbBase);
```

参数

输入：

- clbBase – CLB 逻辑块的基址

返回：

- Val - 该函数将返回 clbqepPos 作为返回值

使用

示例代码：

```
#Define CLB1_BASE 0x00003000U
uint32_t retVal1;
retVal1 = pto_qeponclb_getCLBQEPPos(CLB1_BASE);
```

7.6.8 pto_qeponclb_clearFIFOptr

说明

如果需要清除 CLB 的 FIFO 指针，需要调用此函数。读取由 HLC 推送的 FIFO 值后，需要调用此函数，使下一次的 PUSH 发生在正确的位置。

定义

```
void pto_qeponclb_clearFIFOptr(uint32_t clbBase);
```

参数

输入：

- clbBase – CLB 逻辑块的基址

返回：无

使用

示例代码：

```
pto_qeponclb_clearFIFOptr(0x00003000);
```

8 在工程中使用参考 API

以下几节将介绍在工程中使用一个或多个库所需的步骤。其中包括：

- 添加库头文件
- 更新 Code Composer Studio 选项以在库中进行链接
- 修改往返于 CLB 的内部路由
- 为了调用 API 功能而需要执行的初始化步骤

备注

一些 API 适用于脉冲序列输入 (PTI)，另外一些则适用于脉冲序列输出 (PTO)。为简单起见，示例、库和目录结构均使用后缀“pto”来标识属于此库的内容。

8.1 将 PTO 支持添加到工程中

按照以下说明将 PTO API 添加到工程中。

备注

具体位置可能有所不同，这取决于安装 C2000Ware_MotorControl_SDK 的位置以及工程正在使用的其他库。

1. 在应用程序中包含 PTO 头文件，即 `{ProjectName}.h`。

```
#include "pto_pulsegen.h"
#include "pto_qepdiv.h"
#include "pto_abs2qep.h"
#include "pto_qeponclb.h"
```

2. 在 Code Composer Studio (CCS) 中，右键点击工程，然后导航至 *Project Properties → Build → C2000 Compiler → Include Options*
   a. 将头文件目录添加到“*#include search path*”下（请参阅图 8-1）

      PTO 头文件的路径为 `${SDK_ROOT}\libraries\position_sensing\pto\include`。

备注

${SDK_ROOT} 是 CCS 用来指示 SDK 安装位置的变量。可在 *Project Properties → Resource → Linked Resources* 下查看该变量的定义。

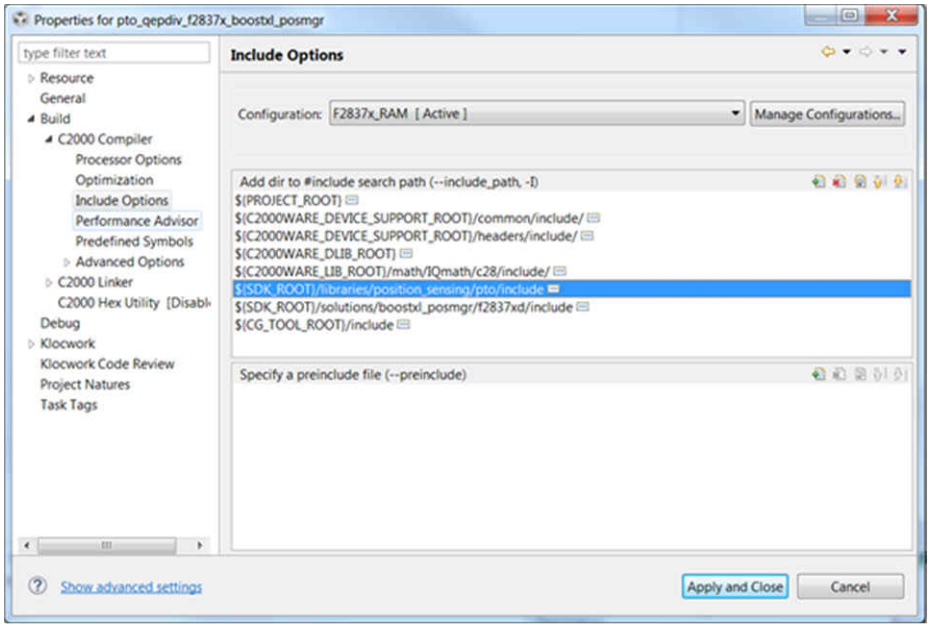


图 8-1. 使用 PTO 参考 API 的工程的编译器包含选项

3. 将已编译的库文件添加到工程中：
   a. 在“Project Explorer”窗口中右键点击工程名称
   b. 导航至“*Project Properties → Build → C2000 Linker → File Search Path*”
   c. 将库目录添加到“*library search path*”下
   d. 将库的名称添加到“*Include library file*”下
   e. 点击“*Apply and Close*”

   PTO 编译的库目标文件位于：`[C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\lib`。

图 8-2 和图 8-3 中显示的示例展示了为包含 PTO API 编译的目标文件，需要对链接器选项进行哪些更改。

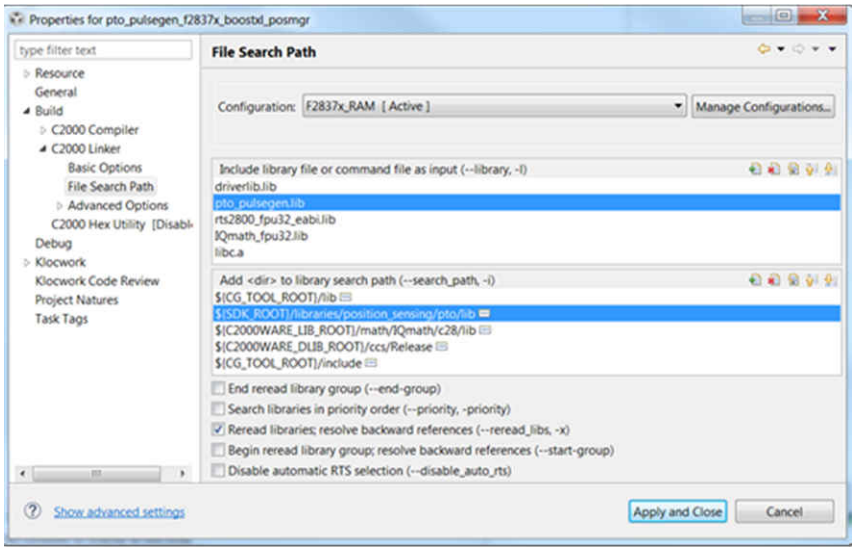


图 8-2. C2000™ 链接器选项 – PulseGen

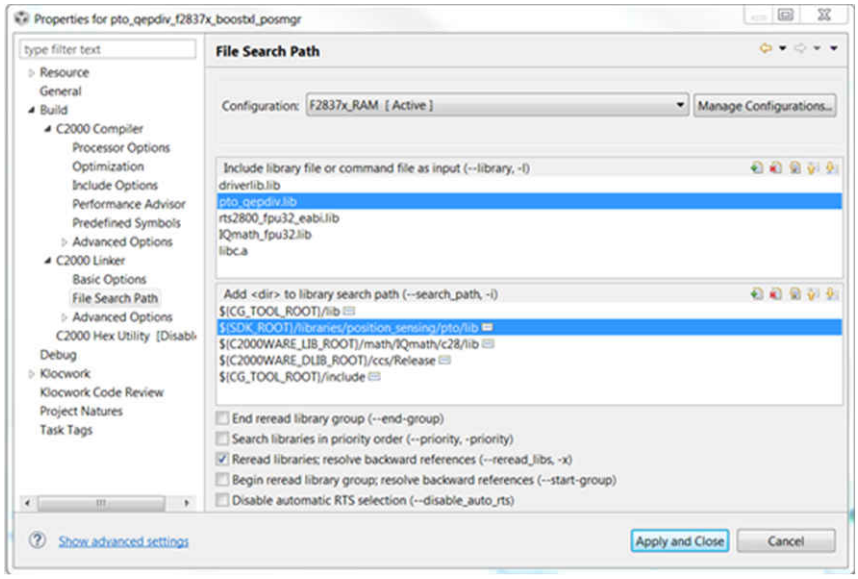


图 8-3. C2000 链接器选项 – QepDiv

备注

具体位置可能有所不同，这取决于安装 C2000Ware_MotorControl_SDK 的位置以及工程正在使用的其他库。

8.2 往返于 CLB 的路由

下一步是了解往返于 CLB 的路由，以及它将如何集成到您的工程需求中。

提供的示例在 CLB 和特定 GPIO 引脚之间路由信号。输入/输出路由在节 6 中有相应说明。但是，您的应用可能需要将信号路由到不同的引脚或不同的 XBAR 或不同的外设。具体修改因情况而异。

在某些情况下，只需对应用程序代码进行简单的更改即可。

路由是通过 INPUTXBARx 到达 CLB AUXSIGx 全局多路复用器。您想将输入更改为不同的 GPIO。这可以通过更改连接到 INPUTXBARx 的 GPIO 来实现。

在其他情况下，可能需要更改 CLB 库或将设计移至另一个逻辑块。例如：

如果 Tile 输出无法使用 OUTPUTXBAR，则它可能会覆盖外设输出。每个逻辑块可用的外设各不相同。例如，ePWM1 可被 Tile1 覆盖，ePWM2 被 Tile2 覆盖，依此类推。此处的更改可能需要将输出路由到不同的 OUTPUT 或不同的逻辑块。

8.3 初始化步骤

下一节将介绍为了使用 PTO 库而需要执行的具体初始化步骤。相关示例应仅用作参考。

8.3.1 PTO-PulseGen API 初始化

PTO PulseGen API 函数的初始化和正常运行需要执行以下步骤。

1. 通过调用 `pto_pulsegen_setupPeriph()` 函数来初始化和设置外设配置。
2. 设置配置所需的 GPIO。更多信息，请参阅 `pto_setupGPIO`。
3. 若要设置脉冲生成配置，请参阅 `pto_setOptions` 函数。
4. `ptoISR` 用作主中断服务例程 (ISR)。若要了解如何更新 PTO 配置，请参阅此 ISR。

8.3.2 PTO-QepDiv API 初始化

PTO QepDiv API 函数的初始化和正常运行需要执行以下步骤。

1. 通过调用 `pto_qepdiv_setupPeriph()` 函数来初始化和设置外设配置。
2. 设置配置所需的 GPIO。
3. 若要设置该配置，请参阅 `pto_qepdiv_config()` 函数。
4. 调用 `pto_qepdiv_startOperation()` 以启动 QepDiv 配置。

8.3.3 PTO-Abs2Qep API 初始化

若要初始化和配置 PTO Abs2Qep API 函数，需要执行以下步骤。

1. 查看库头文件 `pto_abs2qep.h`，并更新为了与系统匹配而需要的任何配置信息。如果进行了修改，则应按照节 7 所述重新构建库。配置包括：
   a. 驱动分辨率
   b. 最大驱动 RPM
   c. 增量编码器每转线数
   d. 位置采样率
2. 如节 8.2 所述，设置 GPIO 和往返于 CLB 的路由。
3. 设置 CLB，请参阅 `pto_abs2qep_setupPeriph()`。请注意，在该函数的末尾会进行一次调用以通过位置 0 设置 CLB。

```
pto_abs2qep_translatePosition(0);
pto_abs2qep_runPulseGen(PTO_ABS2QEP_CLOCKWISE_PTO);
```

4. 配置一个计时器以在采样绝对位置时启动 ISR。该示例使用了 ePWM3。
5. 在采样 ISR 中执行以下操作（请参阅 `pto_EPWM3ISR`）：
   a. 开始先前的 PTO 转换：`pto_abs2qep_runPulseGen(ptoDirection)`
   b. 采样一个新的绝对位置 `AbsolutePositionNext = <application dependent function>()`
   c. 转换下一个 PTO。`ptoDirection = pto_abs2qep_translatePosition(absolutePositionNext)` 下次运行 ISR 时将运行此转换。

8.3.4 PTO-QepOnClb API 初始化

若要初始化和配置 PTO QepOnDiv API 函数，需要执行以下步骤。

1. 通过调用 `pto_qeponclb_setupPeriph()` 函数来初始化和设置外设配置。
2. 将值作为参数传递到 `pto_qeponclb_setupPeriph()` 和 `pto_qeponclb_configMaxCounterPos()` 函数中，可配置计数器位置上限。
3. 设置配置所需的 GPIO。
4. `epwmISR` 用作主中断服务例程 (ISR)。若要了解如何更新 PTO 配置，请参阅此 ISR。
5. 调用 `pto_qeponclb_enableCLBQEP()` 以启用 CLBQEP 进行解码。

9 参考文献

- 德州仪器 (TI)： *《CLB 工具用户指南》*
- 德州仪器 (TI)：*C2000™ Piccolo™ F28004x 系列 LaunchPad™ 开发套件*

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 C (May 2021) to Revision D (January 2022)	Page
• 添加了 QEP 实现	2
• 添加新的节 5。	21
• 添加了新的 节 5.1。	22

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