User's Guide
SPRAC77A–March 2017–Revised November 2019

C2000™ Position Manager PTO API Reference Guide

Ozino Odharo, Subrahmanya Bharathi

ABSTRACT

Pulse-train output (PTO) is a generic name for describing various forms of pulse outputs. The APIs assist in generating various outputs of QEP, CwCCW, pulse and direction, and more.

Contents

1 Introduction 2
2 PTO Software 11
3 Module Summary 12
4 Using the PTO Reference APIs in projects 17
5 Hardware, Software, Testing Requirements 20

List of Figures

1 QepDiv Input and Output Diagram 2
2 QepDiv Interconnect Diagram 2
3 Implementation Diagram 3
4 CLB Tile Diagram for PTO QepDiv 5
5 PulseGen Output Diagram 7
6 Implementation Diagram 8
7 CLB Tile Diagram for PTO PulseGen 9
8 Compiler Include Options for Projects Using PTO Reference APIs 17
9 C2000 Linker Options – PulseGen 18
10 C2000 Linker Options – QepDiv 18

List of Tables

1 QepDiv CLB Tile 1 6
2 QepDiv CLB Tile 2 7
3 PulseGen CLB Tile 1 10
4 PTO API Functions 12

Trademarks

1 Introduction

1.1 PTO – QepDiv

The QepDiv PTO function can be used to generate a divided pulse stream from QEP inputs. Figure 1 shows the QepDiv input and output diagram.

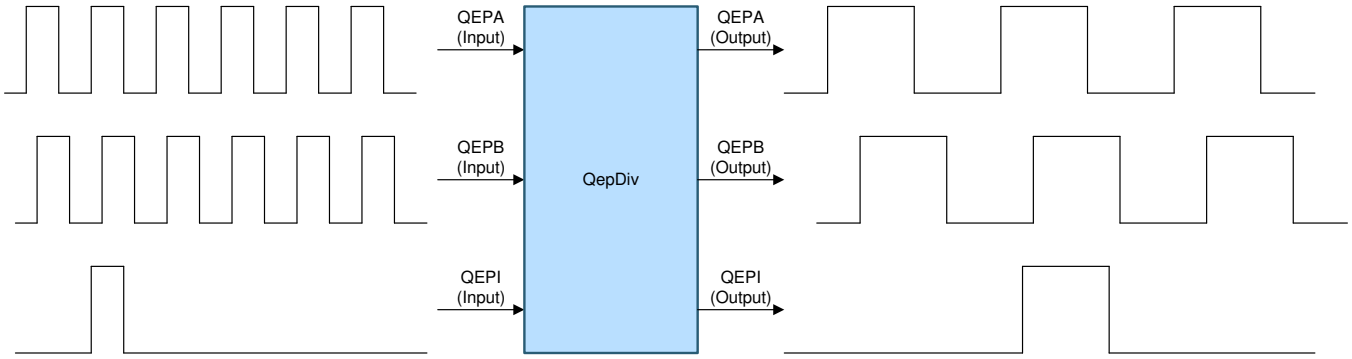

Figure 1. QepDiv Input and Output Diagram

Figure 2 shows the CLB interconnect diagram.

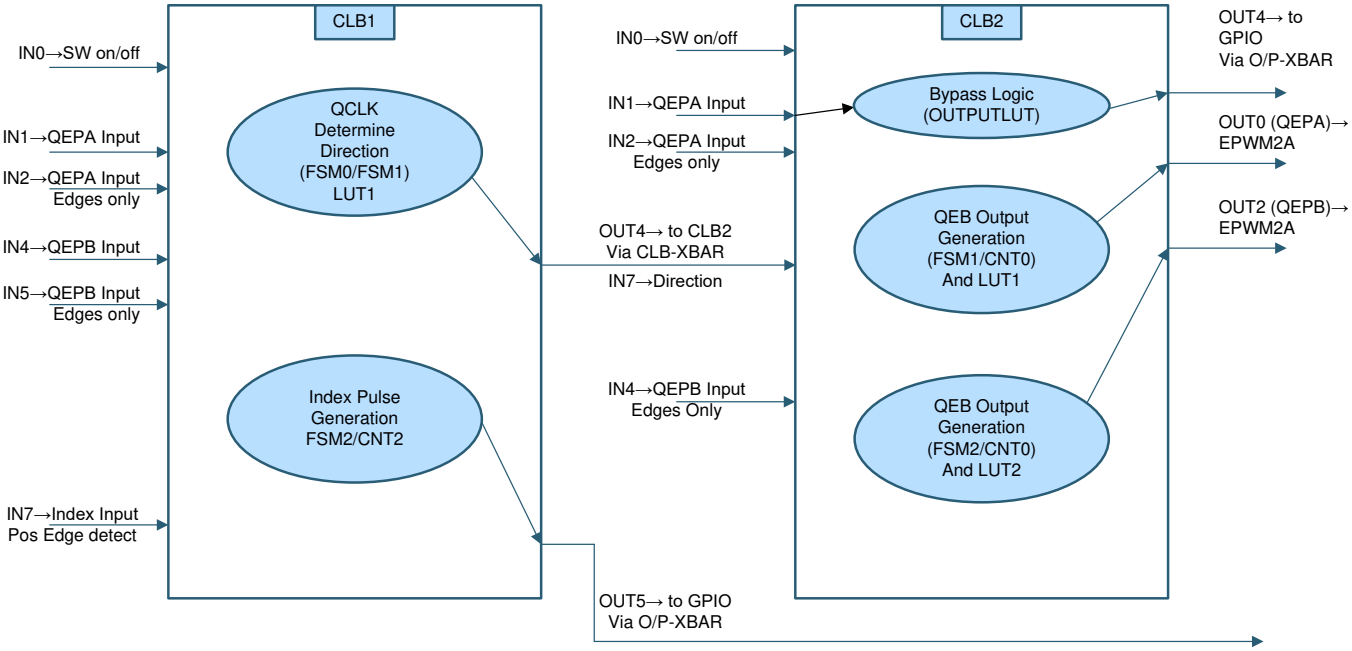

Figure 2. QepDiv Interconnect Diagram

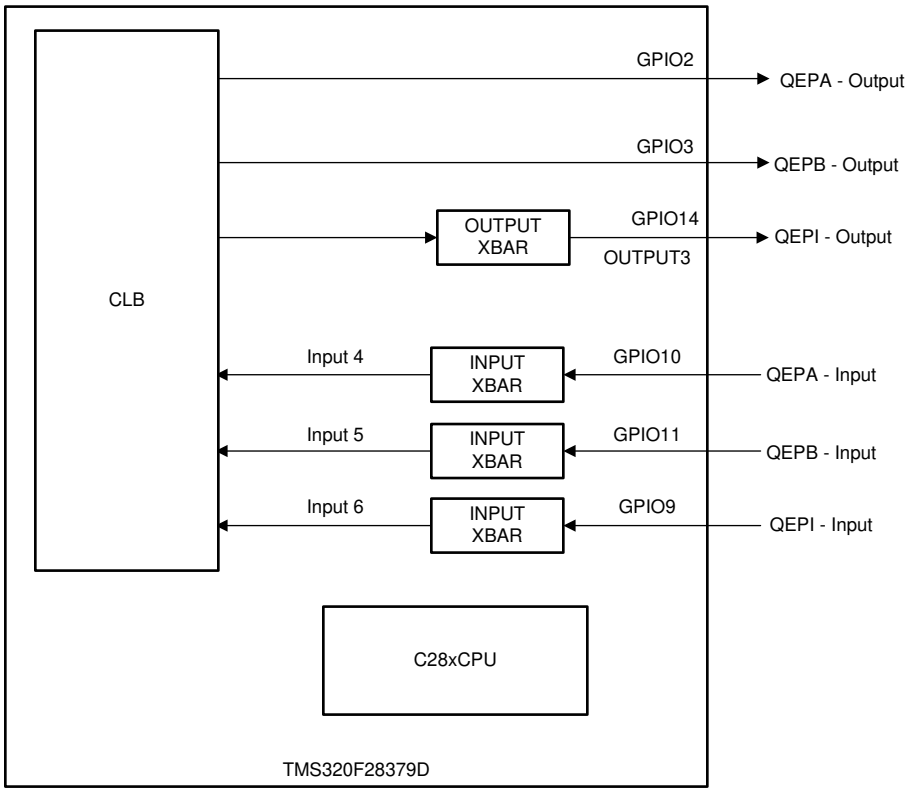

Figure 3. Implementation Diagram

Figure 3 shows the implementation diagram of the QepDiv interface.

NOTE: For the TMS320F280049C and TMS320F28388D devices, the outputs can be observed on GPIO2 (QEPA), GPIO3 (QEPB) and GPIO5 (QEPI).

Chip-level inputs to the PTO-QepDiv interface:

- QEPA input
- QEPB input
- QEPI input

Chip-level outputs to the PTO-QepDiv interface:

- QEPA output
- QEPB output
- QEPI output

1.1.1 Implementation of PTO QepDiv Interface

This section provides an overview of how the PTO QepDiv interface is implemented on TMS320F28379D, TMS320F28388D, and TMS320F280049C devices. This interface is primarily achieved by using the following components:

- C28x CPU
- Configurable logic block (CLB)
- Device interconnect (XBARs)

The following functions are implemented within the CLB module. For more details on the implementation of these functions see the source located in [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\source:

- Monitors to QEPA, QEPB, and index inputs connected to the GPIO
- Detects the direction of the motion and any changes in direction
- Detects the edges of the input signals
- Implements the division function and generates the QEPA, QEPB, and index outputs

Input and output XBARs are used as input and output signal routing to and from the CLB as applicable. The CPU initializes the function and configuration of the CLB, XBARs, and GPIOs as applicable.

The QepDiv interface implements division factors of /1, /2, /4, /8, and continues this pattern up to /1024 and /2048.

The PTO-QepDiv operation has the following usage limitations:

- The maximum frequency of the input signals (QEPA and QEPB) is limited to 5 MHz.
- The index pulse is generated on the index output when a rising edge is detected on the index input signal.
- The width of the index pulse can be user defined. See the pto_qepdiv_config function and the corresponding example provided in C2000Ware MotorControl SDK.
- The divider values work as follows:
  - The frequency of the output QEPA or QEPB = frequency of input QEPA or QEPB / (2 × divider)

1.1.2 Input-Output and Inter CLB Connectivity

X-Bar configuration: see the file pto_qepdiv.c and more specifically to the function pto_qepdiv_initCLBXBAR.

GPIO41->INPUTXBAR4->CLBXBAR-OUT0->CLB1_Input1 -> CLB1_OUT5-> GPIO4 (via EPWM3A)or GPIO14 via OUTPUTXBAR3 (F2837x only)

GPIO42->INPUTXBAR5->CLBXBAR-OUT1->CLB1_Input4 -> CLB1_OUT4-> OUTPUTXBAR1-> GPIO58

```
//
// QEPA - InputXbar INPUT4
//
XBAR_setCLBMuxConfig(XBAR_AUXSIG0, XBAR_CLB_MUX07_INPUTXBAR4);
XBAR_enableCLBMux(XBAR_AUXSIG0, XBAR_MUX07);

//
// QEPB - InputXbar INPUT5
//
XBAR_setCLBMuxConfig(XBAR_AUXSIG1, XBAR_CLB_MUX09_INPUTXBAR5);
XBAR_enableCLBMux(XBAR_AUXSIG1, XBAR_MUX09);

//
// QEPI - InputXbar INPUT6
//
XBAR_setCLBMuxConfig(XBAR_AUXSIG2, XBAR_CLB_MUX11_INPUTXBAR6);
XBAR_enableCLBMux(XBAR_AUXSIG2, XBAR_MUX11);

//
// DIR - CLB1_4.1
//
XBAR_setCLBMuxConfig(XBAR_AUXSIG3, XBAR_CLB_MUX01_CLB1_OUT4);
XBAR_enableCLBMux(XBAR_AUXSIG3, XBAR_MUX01);
```

1.1.3 Divider Settings and Initialization

Divider initialization is done via the function below.

COUNTER0 in CLB2 is used for divider*4 for Match2 value of the counter.

COUNTER0 in CLB2 is used for divider*2 for Match1 value of the counter.

Index pulse width is controlled using COUNTER2 in CLB1 for Match1 value setting.

```
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);
}
```

1.1.4 API Implementation

The PTO API implementation source files are located under [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\source.

The following resources are used inside the CLB tile to achieve the desired function detailed in Section 1.1.1.

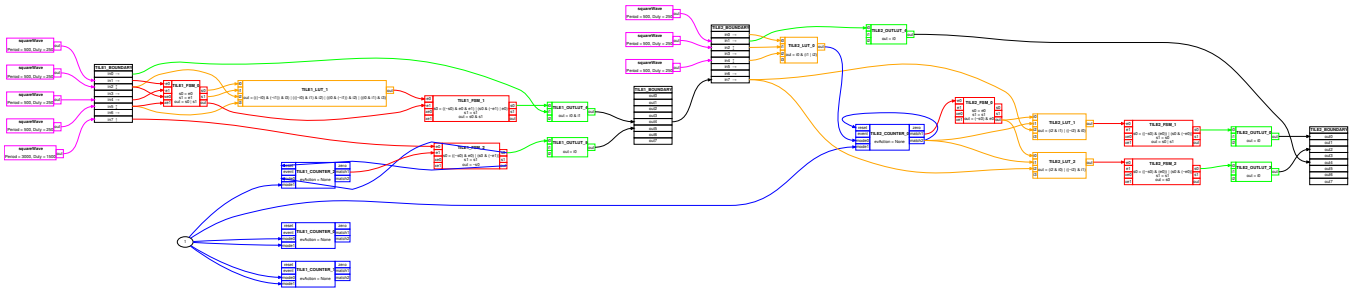

Figure 4. CLB Tile Diagram for PTO QepDiv

NOTE: You can import and build the QepDiv API reference project for each respective device, located in [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\ccs. By rebuilding the compiled object, it will regenerate the CLB tile diagram (clb.svg or clb.html). and object (.lib) The CLB tile diagram will be located in the RELEASE/syscfg directory.

Implementation is described in detail, below and visualized in Figure 4.

Table 1. QepDiv CLB Tile 1

Resource	Function	Notes
Inputs		
In0	On/Off Control via GPREG	Enable CLB
In1	On/Off Control via GPREG	QEPA via EPWM4A
In2	Edge Detect	QEPA via EPWM4A
In3	Not used	Not used
In4	On/Off Control via GPREG	QEPB via EPWM5A
In5	Edge Detect	QEPB via EPWM5A
In6	Not used	Not used
In7	Edge Detect	QEPI via EPWM4B
Outputs		
Out0	Not used	Not used
Out1	Not used	Not used
Out2	Not used	Not used
Out3	Not used	Not used
Out4	Transmit Enable	PTO Direction Via OUTPUT XBar; Input for CLB 2
Out5	Transmit Enable	QEPI output via OUTPUTXBAR3
Out6	Not used	Not used
Out7	Not used	Not used
Logic Resources		
LUT0	Not used	Not used
LUT1	Determines QCLK direction in combo with FSM0 and FSM1	Provides input to FSM1 for external input 0
LUT2	Not used	Not used
FSM0	Alternate inputs between QEPA and QEPB	This state machine checks the QEP signals and alternates between the different signals
FSM1	Set QCLK direction	Uses output of LUT1 and FSM0 to set up the QCLK, which in turn sets the direction. The output will be routed to CLB2 as the input direction
FSM2	Index pulse generation	Takes the QEPI input and uses CNT2 Match2 value to set the QEPI output period and duty cycle.
CNT0	Set index pulse width value	Load indexWidth-1 value set via CLB_writeInterface function
CNT1	Set divider value	Load divider*4 value set via CLB_writeInterface function
CNT2	Set divide value	Load divider*2 value set via CLB_writeInterface function
High Level Controller		
HLC	Not used	Not used

Table 2. QepDiv CLB Tile 2

Resource	Function	Notes
		Inputs
In0	On/Off Control via GPREG	Enable CLB
In1	On/Off Control via GPREG	QEPA via EPWM4A
In2	Edge Detect	QEPA via EPWM4A
In3	Not used	Not used
In4	Edge Detect	QEPB via EPWM5A
In5	Not used	Not used
In6	Not used	Not used
In7	On/Off Control via GPREG	PTO direction routed from CLB1 out4
		Outputs
Out0	Transmit Enable	QEPA Output via EPWM2A
Out1	Not used	Not used
Out2	Transmit Enable	QEPB Output via EPWM2B
Out3	Not used	Not used
Out4	Transmit Enable	Bypass Logic
Out5	Not used	Not used
Out6	Not used	Not used
Out7	Not used	Not used
		Logic Resources
LUT0	QEPA/QEPB signal input	When the tile is on, send the selected QEP signal to CNT0 as mode0 input
LUT1	Generate high and low values for	Alternate between high and low
LUT2	Not used	Not used
FSM0	QEP Pulse width generation	This state machine together with CNT0 will generate a number of hi and low pulse widths for LUT1 and LUT2.
FSM1	QEPA signal generation	Generates QEPA output using CNT0 and LUT1.
FSM2	QEPB signal generation	Generates QEPB output using CNT0 and LUT2.
CNT0	Counter for output QEP signal generation	Counter Match1 value is the external input for FSM0. Match2 value is the reset value for the counter. The match2 value is passed to LUT1 and LUT2 .
CNT1	Not used	Not used
CNT2	Not used	Not used
		High Level Controller
HLC	Not used	Not used

1.2 PTO – PulseGen

The PTO PulseGen function can be used to generate pulse and direction outputs as required by the application. Figure 5 shows the PulseGen output diagram.

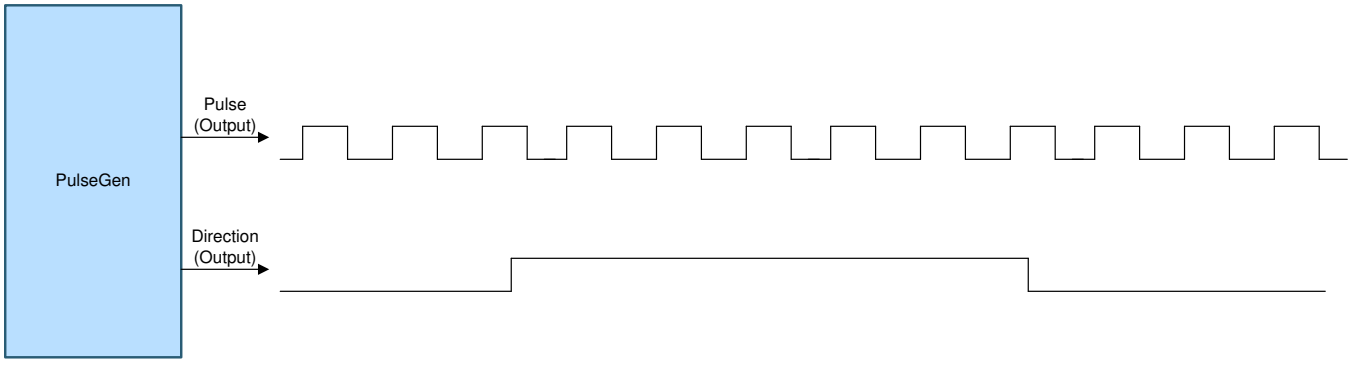

Figure 5. PulseGen Output Diagram

Figure 6 shows the implementation diagram of the PulseGen interface.

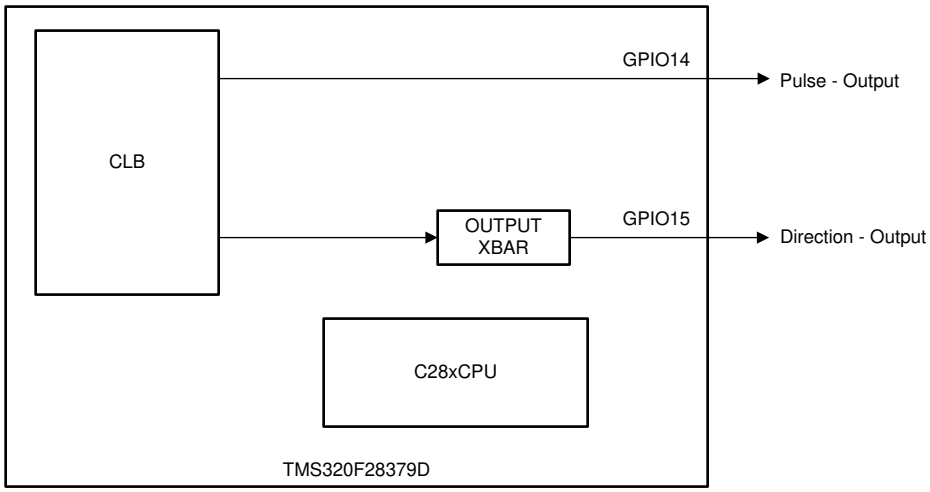

Figure 6. Implementation Diagram

Figure 6 shows the implementation diagram of the PulseGen interface.

NOTE: The TMS320F280049C and TMS320F28388D devices, the outputs can be observed on GPIO26 (Pulse) and GPIO15 (Direction).

Chip-level inputs to the PTO-PulseGen interface: none.

Chip-level outputs to the PTO-PulseGen interface:

- Pulse output
- Direction output

The PTO-PulseGen operation has the following usage limitations:

- The minimum number of cycles must be 1000 cycles for the PTO period (at 200-MHz system clock, this corresponds to 10 µs [for example, 100 KHz]).
- The number of cycles must be between 40% to 60% of the PTO period for the interrupt time to avoid conflicts with PTO updates.
- The maximum frequency of the PTO-PulseGen output is 5 MHz at a 200-MHz CPU CLk. See the example provided in C2000Ware MotorControl SDK.

1.2.1 Implementation of PTO-PulseGen Interface

This section provides an overview of how the PTO-PulseGen interface is implemented on TMS320F28379D, TMS320F28388D, and TMS320F280049C devices. This interface is achieved primarily by the following components:

- C28x CPU
- Configurable Logic Block (CLB)
- Device Interconnect (XBARs)

The following functions are implemented within the CLB module. For more details on the implementations of these function see the source located in [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\source:

- Software to provide the number of pulses and the duration of each pulse.
- The CLB generates the pulses and is defined by the software interface function.
- Direction input is applied as defined by the software interface function.

Output XBARs are used for output-signal routing to and from the CLB, as applicable. The CPU is used at the start to initialize the PulseGen interface initialization function, configuration of the CLB, XBARs, and GPIOs as required.

1.2.2 API Implementation

The PTO PulseGen API implementation source files are located under [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\source.

The following resources are used inside the CLB tile to achieve the desired function detailed in Section 1.2.1

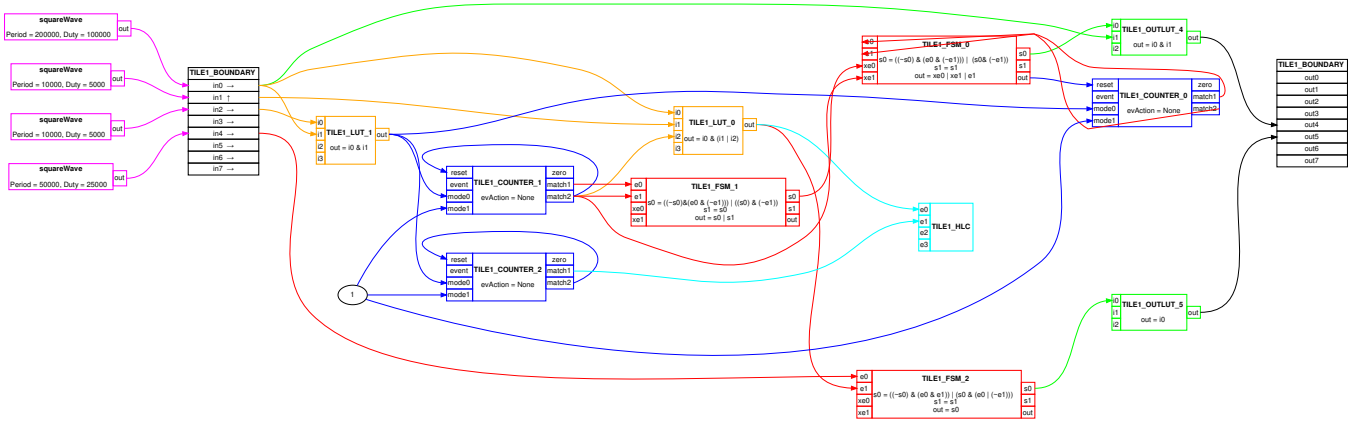

Figure 7. CLB Tile Diagram for PTO PulseGen

NOTE: You can import and build the PulseGen CLB project for each respective device, located in [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\ccs. By building the project, it will regenerate the CLB tile diagram (clb.svg or clb.html). and object (.lib) The CLB tile diagram will be located in the RELEASE/syscfg directory.

Implementation is described in detail, below, and visualized in Figure 7.

Table 3. PulseGen CLB Tile 1

Resource	Function	Notes
	Inputs	
In0	On/Off Control via GPREG	Enable CLB
In1	Rising Edge Detect	Via EPWM1A
In2	On/Off Control via GPREG	Run signal (start/stop of PTO)
In3	Not used	Not used
In4	On/Off Control via GPREG	Sets the PTO direction
In5	Not used	Not used
In6	Not used	Not used
In7	Not used	Not used
	Outputs	
Out0	Not used	Not used
Out1	Not used	Not used
Out2	Not used	Not used
Out3	Not used	Not used
Out4	Transmit Enable	Via OUTPUT XBar; PTO pulse output
Out5	Transmit Enable	Via OUTPUT XBar; PTO direction output
Out6	Not used	Not used
Out7	Not used	Not used
	Logic Resources	
LUT0	Input for Event0 in HLC	Edge detection on encoder input with either in1 or CNT1 match value. Triggers event in HLC to load new values into HLC registers
LUT1	Mode0 input for CNTs 1,2,3	Logic to determine the selected modes for CNT1, CNT2, and CNT3. Starts all three counters.
LUT2	Not used	Not used
FSM0	Pulse width generation	This state machine together with CNT0 will generate a number of hi and low pulse widths. The output sets the reset value of CNT0.
FSM1	Active and Full Period generation	Sets the values for the active and full period based on match1 and match2 outputs of CNT1. Outputs number of pulses in active period duration and none in between the difference of the full and active periods
FSM2	PTO output direction generation	Generates the PTO output direction. The output direction is held until the end of the full period set by FSM1.
CNT0	Pulse width generation	Counter Match1 and Match2 values determine triggers for hi and low pulse widths. The match values are loaded to FSM0 inputs, e0 and e1.
CNT1	Active and Full Period Clock generation	Generates inputs needed for FSM1 and FSM2. Match1 determines trigger for active period. Match2 determines trigger for full period. Match events are used by FSM1 to generate active and full periods. Match2 is used as extra external input in FSM0 to determine how long to hold PTO output direction.
CNT2	Counter for full period	Match1 event used to trigger interrupt in HLC. Counter is reset when full period of signal is reached
	High Level Controller	
HLC	Event0 used to trigger taskEvent1 used to trigger interrupt	Event0 used to load new options for the PTO from C28 core into CLBEvent1 used to generate an interrupt based on match1 event of CNT2, which corresponds to the full period. New PTO options take effect after this event.

2 PTO Software

The PTO APIs and source code are located at:

- <base> install directory is [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto

The following sub-directory structure is used:

- <base>\lib Contains generated compiled object (.lib)
- <base>\ccs Projectspec example using for PTO Reference API project
- <base>\include Contains include file for generated .lib
- <base>\source Contains all the source code of the PTO application including CLB configuration

To rebuild the CLB tool based object, follow this procedure:

NOTE: You will need to ensure that you have the appropriate tools installed in order to build CLB based projects. For more information, see CLB Tool User's Guide.

1. In CCS v9.2 or higher, click "Project -> Import CCS Projects…"
2. Navigate to the PTO libraries directory. The path is:
   a. [C2000Ware_MotorControl_SDK]\ libraries\ position_sensing \pto\ccs\f2837x
   b. [C2000Ware_MotorControl_SDK]\ libraries\ position_sensing \pto\ccs\f2838x

NOTE: The f2837x projects are applicable for the f28004x device and the object generated will run on both devices.

3. Select the project of choice "pto_pulsegen_libproj" and/or "pto_qepdiv_libproj", and click "Finish".
4. In the CCS Project Explorer window, expand the selected project and open the file corresponding SysConfig file (i.e. "pto_pulsegen.syscfg").
5. Inspect the configuration of the tile and observe the logical expressions in the LUTs and FSMs, and output LUTs.
6. From the CCS menu, select "Project -> Build Project".
7. At this point, an output object (.lib) is generated and located in the <base>\lib folder. This file will be included in PTO example projects. This object is a compiled object of the PTO source files.
8. [Optional] – for instructions on how to run a simulation of the CLB based project, see the *Running the Simulation* section in the CLB Tool User's Guide.

The PTO example projects are located at <base> install directory is:

- [C2000Ware_MotorControl_SDK]\solutions\boostxl_posmgr\f2837x
- [C2000Ware_MotorControl_SDK]\solutions\boostxl_posmgr\f2838x
- [C2000Ware_MotorControl_SDK]\solutions\boostxl_posmgr\f28004x

The following sub-directory structure is used:

- <base>\ccs Projectspec for examples using PTO APIs
- <base>\source Source files of the Example projects
- <base>\include Include files of the Example projects
- <base>\cmd Linker command files of the Example project

To run the examples, follow this procedure:

1. In CCS v9.2 or higher, click "Project -> Import CCS Projects…"
2. Navigate to the PTO solutions directory. The path is:
   a. [C2000Ware_MotorControl_SDK]\ solutions\ boostxl_posmgr\f2837xd\ccs
   b. [C2000Ware_MotorControl_SDK]\ solutions\ boostxl_posmgr\f28004x\ccs
   c. [C2000Ware_MotorControl_SDK]\ solutions\ boostxl_posmgr\f2838x\ccs

   In the description that follows, it is assumed the C2000Ware_MotorControl_SDK directory above is in use.

3. Select the project of choice “pto_pulsegen” and/or “pto_qepdiv”, and click “Finish”.
4. From the CCS menu, select “Project -> Build Project”.
5. Once the build completes without errors, execute the project by selecting “Run –>Debug”.

   If running the PTO PulseGen examples, verify and monitor the output waveform signals on GPIO14 via OutputXBAR3 (f2837xd) or GPIO26 (f2838x, f28004x) for the pulse output and GPIO15 via OutputXBAR4 for the output direction.

   If running the PTO QepDiv examples, verify and monitor the output waveform signals on GPIO2, GPIO3 for QEPA, QEPB. The QEPI signal can be viewed on GPIO14 (f2837xd) or GPIO5 via OutputXBAR3, (f2838x, f28004x) for the pulse output and GPIO15 for the output direction.

3 Module Summary

This section details the contents of the PTO API functions.

For more details on the function and their usage, see Section 3.2 and Section 3.3.

pto_pulsegen.h – the include file for using PTO PulseGen API functions.

pto_qepdiv.h – the include file for using PTO QepDiv API functions.

3.1 PTO API Functions

Table 4 lists the functions existing in the PTO package and a summary of cycles taken for execution. For more details of the functions and their uses, see Section 3.2 and Section 3.3.

Table 4. PTO API Functions

Name	Description	Type
PulseGen		
pto_pulsegen_reset	Used to reset the pulsegen parameters set by earlier configuration and start a fresh setup. This function needs to be called in case the pulse generation needs to be reset and started again at a later stage.	Initialization time
pto_pulsegen _startOperation	This function will initiate the pulse generation on the interface. To be called after pto_pulsegen_setupPeriph. Performs the transaction set up by earlier function. Note that the setup up and start operation are separate function calls. User can setup the peripherals when needed and start the actual pulse generation using this function call, as needed, at a different time.	Run time
pto_pulsegen _runPulseGen	A runtime function to be called periodically for dynamically configuring and changing the pulse generation requirements as needed by the application. This function needs to be called periodically with appropriate parameters like the number of pulses, period, duration etc. Details in the later section.	Run time
pto_pulsegen _setupPeriph	Setup for CLB and other interconnect XBARs is performed with this function during system initialization. This function needed to be called after every system reset. No transactions will be performed until the setup peripheral function is called.	Initialization time
QepDiv		
pto_qepdiv_reset	Used to reset the qepdiv parameters set by earlier configuration and start a fresh setup. This function needs to be called in case the pulse generation needs to be reset and started again at a later stage.	Initialization time
pto_qepdiv _startOperation	This function will initiate the pulse generation on the interface. To be called after pto_qepdiv_setupPeriph. Performs the transaction set up by earlier function. Note that the setup up and start operation are separate function calls. User can setup the peripherals when needed and start the actual pulse generation using this function call, as needed, at a different time.	Run time
pto_qepdiv_config	This function configures the divider, the divider value cannot be changed dynamically. User needs to reset the module using pto_qepdiv_reset before reconfiguring the functionality.	Run time
pto_qepdiv _setupPeriph	Setup for CLB and other interconnect XBARs is performed with this function during system initialization. This function needed to be called after every system reset. No transactions will be performed until the setup peripheral function is called.	Initialization time

3.2 Details of Function Usage (PulseGen)

3.2.1 pto_pulsegen_runPulseGen

Description

A runtime function to be called periodically for dynamically configuring and changing the pulse generation requirements as required by the application. This function must be called periodically with appropriate parameters like the number of pulses, period, duration, and more.

Definition

```
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
        );
```

Parameters

Input:

- pulseLo – Low pulse width
- pulseHi – High pulse width
- ptoActivePeriod – Period the pulses are sent out; less than ptoFullPeriod
- ptoFullPeriod – Full PTO period
- ptoInterruptTime – Time when that the interrupt is generated to the CPU
- ptoDirection – Direction output; latched as it is on direction output at the beginning of new period
- run – Value indicting 1-run and 0-stop. Sampled at the beginning of the new period to determine to continue or halt the pulse generation

Return:

- Val – If the function is executed successfully, the function will return ptoFullPeriod as the return value

Usage

In pto_pulsegen.c, a sample configuration function called pto_setOptions is provided as an example to assist with the pto_pulsegen_runPulseGen function and to perform the intermediate calculations. See the following code sample calculation that illustrates how various parameters for this function can be generated. For more details, see the pto_pulsegen.c and pto_pulsegen.h files.

```
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);
}
```

3.2.2 pto_startOperation

Description

This function initiates the pulse generation. This function must be called after pto_pulsegen_setupPeriph. Hence, the pto_pulsegen_startOperation kick starts the pulse generation that was set up earlier.

NOTE: The setup and start operations are separate function calls. Users can set up the transfer and start the pulse generation by using this function call, as required, at a different time.

Definition

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

Parameters

Input: none

Return: none

Usage

Example code:

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

3.2.3 pto_pulsegen_setupPeriph

Description

This function performs the setup for the CLB and other interconnect XBARs during system initialization. This function must be called after every system reset. No transactions will be performed until the setup peripheral function is called.

Definition

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

Parameters

Input: none

Return: none

Usage

Example code:

```
pto_pulsegen_setupPeriph();
```

3.2.4 pto_pulsegen_reset

Description

This function resets the pulsegen parameters set by the earlier configuration (PulseGen function calls) and starts a new setup. This function must be called in case the pulse generation must be reset and started again at a later stage.

Definition

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

Parameters

Input: none

Return: none

Usage

Example code:

```
pto_pulsegen_reset();
```

3.3 Details of Function Usage (QepDiv)

3.3.1 pto_qepdiv_config

Description

This function configures the divider. The divider value cannot be changed dynamically. Users must reset the module using pto_qepdiv_reset before reconfiguring the functionality.

Definition

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

Parameters

Input:

- Divider – Value of the divider
- Index width – Number of cycles for which the index pulse output is kept on

Return:

- Val – If the function is executed successfully, it will return ptoFullPeriod as the return value

Usage

Example code:

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

3.3.2 pto_startOperation

Description

This function initiates the pulse generation. This function must only be called after pto_qepdiv_setupPeriph. Hence, the pto_qepdiv_startOperation function kick starts the pulse generation that was set up earlier.

NOTE: The setup and start operations are separate function calls. Users can set up the transfer and start the pulse generation by using this function call, as required, at a different time.

Definition

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

Parameters

Input (parameters to be passed to start or stop the function):

- Start = 1
- Stop = 0

Return: none

Usage

Example code:

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

3.3.3 pto_qepdiv_setupPeriph

Description

Setup for the CLB and other interconnect XBARs is performed with the pto_qepdiv_setupPeriph function during system initialization. This function must be called after every system reset. No transactions will be performed until the setup peripheral function is called.

Definition

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

Parameters

Input: none

Return: none

Usage

Example code:

```
pto_qepdiv_setupPeriph();
```

3.3.4 pto_qepdiv_reset

Description

Used to reset the qepdiv parameters set by earlier configurations and to begin a fresh setup. This function must be called in case the pulse generation must be reset and started again at a later stage.

Definition

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

Parameters

Input: none

Return: none

Usage

Example code:

```
pto_qepdiv_reset();
```

4 Using the PTO Reference APIs in projects

4.1 Adding PTO Support to a Project

Use the following instructions to add the PTO APIs to a project.

1. Include the PTO path in {ProjectName}.h.

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

2. Navigate to Project Properties → Build → C2000 Compiler → Include Options, then add the PTO compiled object file path in the include paths (see Figure 8).

   The path for the PTO headers is [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\include.

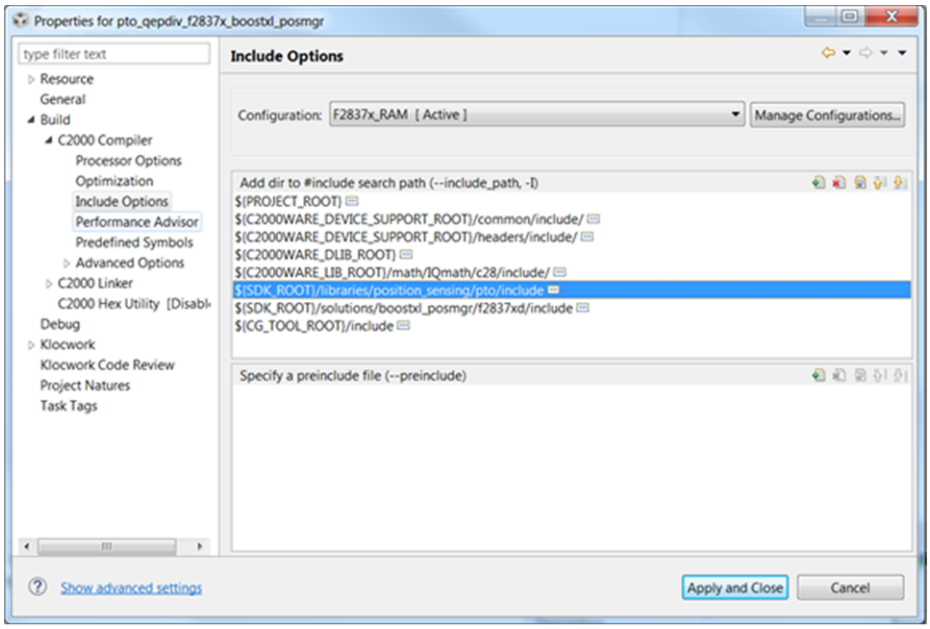

Figure 8. Compiler Include Options for Projects Using PTO Reference APIs

NOTE: The exact location may vary depending on where C2000Ware_MotorControl_SDK is installed and which other libraries the project is using.

3. The PTO compiled object file, pto_pulsegen.lib or pto_pulsegen_f2838x.lib, is located at: [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\lib. This file needs to be copied into the project and referenced in the linker options.

   Figure 9 and Figure 10 show the changes to the linker options that are required to include the PTO APIs compiled object file.

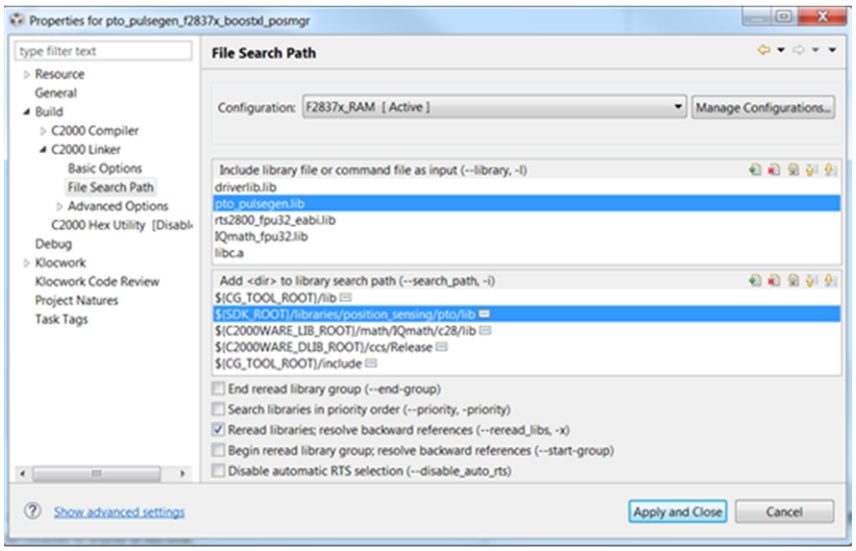

Figure 9. C2000 Linker Options – PulseGen

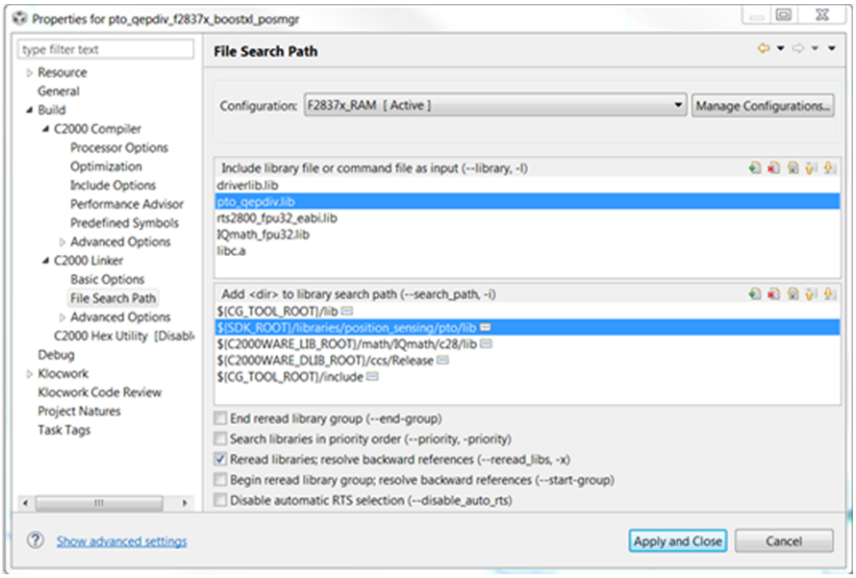

Figure 10. C2000 Linker Options – QepDiv

NOTE: The exact location may vary depending on where C2000Ware_MotorControl_SDK is installed and which other libraries the project is using.

4.2 Initialization Steps

4.2.1 PTO-PulseGen APIs

The following steps are required for initialization and proper function of the PTO PulseGen API functions. For more details, see the examples provided with the PTO APIs.

1. Initialize and set up the peripheral configuration by calling the pto_pulsegen_setupPeriph function.

```
pto_pulsegen_setupPeriph();
```

2. Set up the GPIOs required for configuration. This step is for the F28379D device.

```
GPIO_setMasterCore(14, GPIO_CORE_CPU1);
GPIO_setPinConfig(GPIO_14_OUTPUTXBAR3);

GPIO_setMasterCore(15, GPIO_CORE_CPU1);
GPIO_setPinConfig(GPIO_15_OUTPUTXBAR4);
```

3. To set the pulse generation configuration, see the pto_setOptions function.

   ptolsr is used as the primary ISR to update the PTO configuration in the example.

4.2.2 PTO-QepDiv APIs

The following steps are required for initialization and proper function of the PTO QepDiv API functions. For more details, see the examples provided with the PTO APIs.

1. Initialize and set up the peripheral configuration by calling the pto_qepdiv_setupPeriph function.

```
pto_qepdiv_setupPeriph();
```

2. Set up the GPIOs required for configuration. This step is for the F28379D device.

```
//
// QEP inputs to be tapped from GPIO10/11/9 - via InputXBar Input4/5/6
//
XBAR_setInputPin(XBAR_INPUT4, 10);  // GPIO10 = QEPA
XBAR_setInputPin(XBAR_INPUT5, 11);  // GPIO11 = QEPB
XBAR_setInputPin(XBAR_INPUT6, 9);  // GPIO9 = QEPI

//
// QEP outputs available on GPIO2/3 - QEPA/B (over EPWM2A/B)
//
GPIO_setPinConfig(GPIO_2_EPWM2A);  // Configure GPIO2 (Pulse out A)
GPIO_setDirectionMode(2, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(2, GPIO_PIN_TYPE_STD);

GPIO_setPinConfig(GPIO_3_EPWM2B);  // Configure GPIO3 (Pulse out B)
GPIO_setDirectionMode(3, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(3, GPIO_PIN_TYPE_STD);

//
// QEP outputs Index on OUTPUTXBAR3 - on GPIO14
// (users can choose any GPIO with OUTPUTXBAR3
//
GPIO_setMasterCore(14, GPIO_CORE_CPU1);
GPIO_setPinConfig(GPIO_14_OUTPUTXBAR3);
```

3. To set the configuration, see the pto_qepdiv_config function.
4. Run pto_qepdiv_startOperation to kick start the qepdiv configuration in the example.
5. Test the setup used in the example.

   In the example provided for the pto_qepdiv, spare EPWMs are used to provide QEP inputs. Customers must connect these EPWM outputs to the QEP inputs or feed external signals directly.

```
EPWM4A (GPIO6)-> EQEPA (GPIO10)
EPWM5A (GPIO8)-> EQEPB (GPIO11)
EPWM4B (GPIO7)-> Index (GPIO9)
// These are just for test purposes and do not correspond to real time usage
```

Below are the connections that need to be made:

- EPWM4A (GPIO6)-> EQEPA (GPIO10)
  - LAUNCHXL-F280049C: pin 78 to pin 40
  - LAUNCHXL-F28379D: pin 80 to pin 76
  - TMDSCNCD28388D: pin 54 to pin 61
- EPWM5A (GPIO8)-> EQEPB (GPIO11)
  - LAUNCHXL-F280049C: pin 38 to pin 39
  - LAUNCHXL-F28379D: pin 78 to pin 75
  - TMDSCNCD28388D: pin 57 to pin 63
- EPWM4B (GPIO7)-> Index (GPIO9)
  - LAUNCHXL-F280049C: pin 77 to pin 37
  - LAUNCHXL-F28379D: pin 79 to pin 77
  - TMDSCNCD28388D: pin 56 to pin 59

5 Hardware, Software, Testing Requirements

5.1 Hardware

This section describes the hardware specifics of the TMDSCNCD28388D, LAUNCHXL-F28379D and LAUNCHXL-F280049C devices. It then describes how to get started with the CCS PTO examples.

To experiment with the TMDSCNCD28388D, the following components are required:

- F28388D controlCARD evaluation module and docking station
- USB-B to A cable

PC with CCS (CCS v9.2.0 or greater) installed

To experiment with the LAUNCHXL-F28379D, the following components are required:

- F28379D LaunchPad development kit (LAUNCHXL-F28379D)
- USB-B to A cable
- PC with CCS (CCS v9.2.0 or greater) installed

To experiment with the LAUNCHXL-F280049C, the following components are required:

- F28379D LaunchPad development kit (LAUNCHXL-F280049C)
- USB-C to A cable
- PC with CCS (CCS v9.2.0 or greater) installed

5.1.1 PTO QepDiv Jumper Configuration

In order to understand how to connect the QEP inputs if external signals are not in use, see Section 4.2.2.

5.2 Software

This section describes how to configure the software environment for the F28388D control CARD, F28379D and F280049C LaunchPad.

5.2.1 Installing Code Composer Studio™ and C2000WARE-MOTORCONTROL-SDK™

1. Install CCS v9.2.0 or later, if it is not already installed on the PC
2. Install MotorControl SDK v2.01.00.00 or later, if it is not already installed on the PC
3. For more information on the CLB tool, see the *CLB Tool User's Guide*.
4. After installation, see Section 3 for more information on the PTO API Implementation

5.3 Import and Run Example Project

1. The PTO example projects are available under:
   [C2000Ware_MotorControl_SDK]\solutions\boostxl_posmgr\f2837xd\ccs
   [C2000Ware_MotorControl_SDK]\solutions\boostxl_posmgr\f2838x\ccs
   [C2000Ware_MotorControl_SDK]\solutions\boostxl_posmgr\f28004x\ccs
2. In case the PTO API (with CLB) project needs to be modified, it can be imported from here:
   [C2000Ware_MotorControl_SDK]\libraries\position_sensing\pto\ccs
3. After importing the project(s), right-click on the Project Name and select Rebuild Project, then watch the Console window. Any errors in the project are displayed in the Console window.
4. On successful completion of the build, click the Debug button , (located in the top-left side of the screen).
5. Run the code by pressing the Run button , in the Debug tab. The project should run and the designated waveforms should now be viewable on an oscilloscope.

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (March 2017) to A Revision **Page**

- Updates were made in Section 1.1. 2
- Updates were made in Section 1.1.4. 5
- Updates were made in Section 1.2. 7
- Updates were made in Section 1.2.1. 8
- Updates were made in Section 1.2.2. 9
- Updates were made in Section 2. 11
- Updates were made in Section 3. 12
- Updates were made in Section 3.1. 12
- Updates were made in Section 3.2.1. 13
- Updates were made in Section 3.2.2. 14
- Updates were made in Section 3.2.3. 14
- Updates were made in Section 3.2.4. 15
- Updates were made in Section 3.3.1. 15
- Updates were made in Section 3.3.2. 16
- Updates were made in Section 3.3.3. 16
- Updates were made in Section 4.1. 17
- Updates were made in Section 4.2.1. 19
- Updates were made in Section 4.2.2. 19
- Added new Section 5. 20

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265