Application Note

C2000 ePWM Developer's Guide

Marlyn Rosales and Nima Eskandari

ABSTRACT

The enhanced pulse width modulator (ePWM) peripheral is a key element in controlling many of the power electronic systems found in both industrial and automotive applications. C2000™ real-time microcontrollers offer many differentiated features within the ePWM peripheral that allow for advanced control techniques. This application report focuses on an application use case described within the introduction, demonstrating each of the features of the ePWM along with how to use SysConfig system configuration tool to set-up and program the desired outputs. SysConfig is a tool that exists integrated in Code Composer Studio™ (CCS) or as a standalone program that allows you to generate C header and code files using a graphical user interface (GUI). This application report was done using the TMS320F28388D device. However, the content in this application report is applicable to devices with a Type-4 ePWM module. For more information on peripheral types, see the *C2000 Real-Time Control MCU Peripherals Guide*.

Table of Contents

1 Introduction....3
2 SysConfig....4
3 Time-Base (TB) Submodule....4
3.1 Setting the Frequency....4
3.2 Applying a Phase Shift....6
3.3 Setting up the Synchronization (Sync) Scheme....6
4 Counter-Compare (CC) and Action-Qualifier (AQ) Submodules....8
4.1 Calculating the Duty Cycle....8
5 Deadband (DB) Submodule....10
5.1 Setting up Signal Pairs....10
6 Verifying the Output....12
6.1 Checking the Duty Cycle and Dead-Time Insertion....12
6.2 Checking the Phase Shift Applied....13
7 Trip-Zone (TZ) and Digital Compare (DC) Submodules....14
7.1 Drive Outputs Low for an ePWM Cycle Upon Trip Condition Set Through CMPSS....14
7.2 Drive Outputs Low Until Cleared Through Software Upon Trip Condition set Through GPIO....19
8 Event-Trigger (ET) Submodule....22
8.1 Setting Up Time-Base Interrupts....22
9 Global Load....22
9.1 Applying Global Loading and One-Shot Load Feature....22
9.2 Linking the ePWM Modules....23
9.3 Updating Action Qualifier Settings and Counter Compare Values Through Global Loading....23
10 Summary....25
11 References....25

List of Figures

Figure 1-1. Block Diagram for the EPWM Module....3
Figure 2-1. SysConfig- EPWM Module: Global Parameters....4
Figure 3-1. EPWM Time Base: Setting TBPRD and Count Mode....5
Figure 3-2. Synchronization Scheme....6
Figure 3-3. EPWM Time Base: Configuration for EPWM1 as the Sync Source....7
Figure 3-4. EPWM Time Base: Configuration for EPWM2 Sync Receiver Setup....7
Figure 4-1. EPWM Action-Qualifier: Setting Global Load and Output Events....8
Figure 4-2. EPWM Counter-Compare: Setting the Counter Compare Values....9

Figure 5-1. Active High Complementary Signal Pairs....10
Figure 5-2. Configuration Options for the Deadband Submodule....10
Figure 5-3. EPWM Deadband: Active High Complementary....11
Figure 6-1. Scope Capture of the EPWM Output Positive Duty....12
Figure 6-2. Scope Capture of EPWM Output With Dead-Time Insertion....13
Figure 6-3. Scope Capture of the EPWM Output With Phase Shift....13
Figure 7-1. Comparator Block Diagram....14
Figure 7-2. Comparator Pins....15
Figure 7-3. CMPSS Configuration....15
Figure 7-4. Partial Example of an EPWM X-BAR Mux Configuration Table....17
Figure 7-5. Input X-BAR Block Diagram....17
Figure 7-6. SysConfig Configuration for EPWMXBAR....18
Figure 7-7. EPWM Digital Compare: Setting up DCAEVT2....18
Figure 7-8. EPWM Trip-Zone: Configuration for CBC Trip Based on DCAEVT2....19
Figure 7-9. GPIO Setup in SysConfig....20
Figure 7-10. Input X-BAR in SysConfig....20
Figure 7-11. Trip Zone Configuration Including One-Shot Configuration....21
Figure 8-1. EPWM Event-Trigger: Setting up an Interrupt for When TBCTR=ZRO....22
Figure 9-1. Global Loading SysConfig Setup....23
Figure 9-2. Scope Capture of EPWM Output Positive Duty After AQ and CC Updates....25

Trademarks

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

1 Introduction

The ePWM module is separated into submodules, each with their own functionality. Figure 1-1 shows how the submodules are connected to each other. Throughout this application report, each of the different submodules is explained in detail.

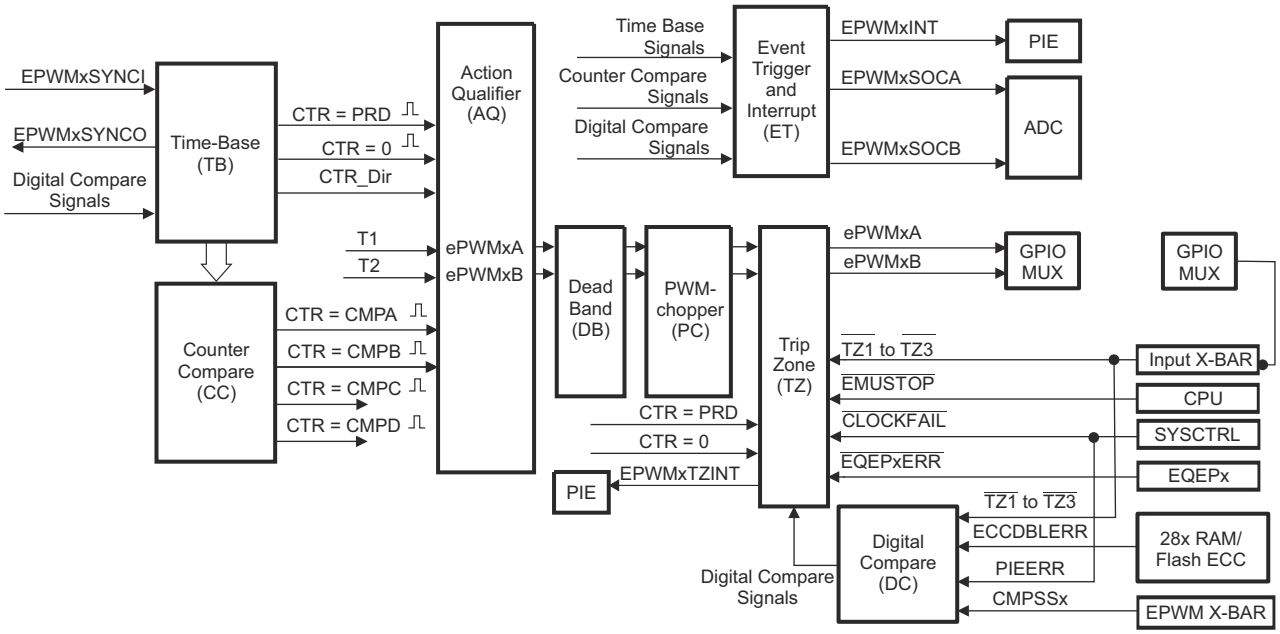

Figure 1-1. Block Diagram for the EPWM Module

The application use case that is discussed throughout this application report has the following criteria:

Use Case

- Output frequency of 400 kHz for EPWM1/2/3
- Phase shift of 120° for EPWM2 with respect to EPWM1
- Phase shift of 240° for EPWM3 with respect to EPWM1
- Duty cycle of 45% for EPWM1/2/3
- Active high complementary signal pairs with 200 nsec of rising/falling edge delay for EPWM1/2/3
- Cycle-by-Cycle trip zone protection through comparator signal on EPWM2
- One-Shot trip protection through general-purpose input/output (GPIO) on EPWM3
- Interrupt generation every time the time-base counter equals zero on EPWM1
- Global loading to support asynchronous updates of action qualifier settings
- Link CMPA/CMPB of EPWM1 to EPWM2 and EPWM3

2 SysConfig

As mentioned in the abstract, this application report utilizes SysConfig in order to configure the EPWM module along with the other required components. Within SysConfig, each submodule of the EPWM module has its own grouping, including a Global Load and HRPWM grouping. If more general information about SysConfig is required, see the items listed in Section 11.

Aside from being able to configure each of the submodules, the EPWM modules also have a "Global Parameters" section at the very top. These settings are used to enable information/warnings as well as affect how the code is being generated from the EPWM module.

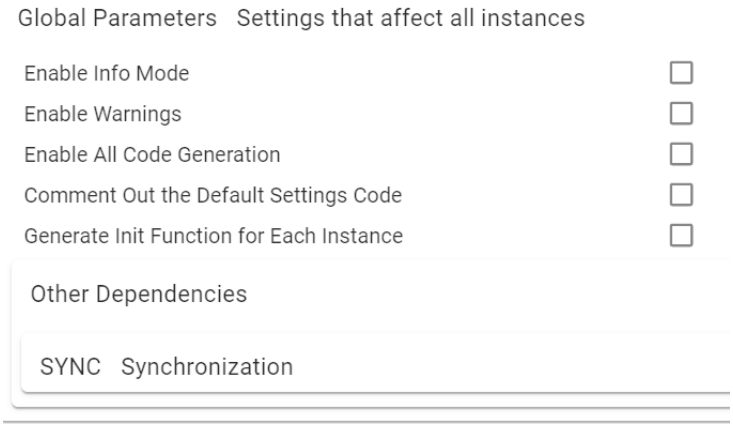

Figure 2-1. SysConfig- EPWM Module: Global Parameters

Note

By default, SysConfig only generates a predefined set of code along with code from any settings that are not in the default state for the EPWM module. However, these settings can be changed within the Global Parameters section.

3 Time-Base (TB) Submodule

3.1 Setting the Frequency

The Time-Base (TB) submodule is used to setup the ePWM modules with a frequency of 400 kHz. Each ePWM module has a time-base counter (TBCTR). There are three modes of the time-base counter: UP, DOWN, and UP-DOWN. The frequency of the PWM events (F_{PWM}) is controlled by the time-base period (TBPRD) register and the mode of the time-base counter.

For up and down count modes:

$$T_{PWM} = (TBPRD + 1)T_{TBCLK} \quad (1)$$

$$F_{PWM} = \frac{1}{T_{PWM}} \quad (2)$$

For up-down count mode:

$$T_{PWM} = 2*TBPRD*T_{TBCLK} \quad (3)$$

$$F_{PWM} = \frac{1}{T_{PWM}} \quad (4)$$

Where, T_{PWM} is the period of the PWM events and T_{TBCLK} is the period of the time-base clock. The time-base clock (TBCLK) is a prescaled version of the ePWM clock (EPWMCLK). This clock determines the rate at which the time-base counter increments and decrements.

For the use-case discussed in this application report, the up-down count mode is utilized as it provides more configurability options due to the symmetry of the count mode. Start by finding T_{PWM}:

$$F_{PWM} = \frac{1}{T_{PWM}} \rightarrow T_{PWM} = \frac{1}{F_{PWM}} \quad (5)$$

$$T_{PWM} = \frac{1}{400k} \rightarrow 2.5\ \mu sec \quad (6)$$

Now that you know what T_{PWM} is, you can start calculating the value for TBPRD:

$$TBPRD = \frac{T_{PWM}}{2*T_{TBCLK}} \quad (7)$$

The time-base clock is defined through the following formula:

$$TBCLK = \frac{EPWMCLK}{HSPCLKDIV*CLKDIV} \quad (8)$$

The maximum EPWMCLK is defined in the device-specific data sheet. EPWMCLK is usually the same as SYSCLK but some devices may have a clock divider, EPWMCLKDIV, which reduces the frequency of the ePWM clock. The high-speed clock divider (HSPCLKDIV) and the clock divider (CLKDIV) are programmable dividers used to help achieve a desired ePWM time-base clock. For this use case, both clock dividers are set to divide by 1 and EPWMCLK is 100 MHz.

$$TBCLK = \frac{100M}{1*1} \rightarrow 100\ MHz \quad (9)$$

$$T_{TBCLK} = \frac{1}{TBCLK} = \frac{1}{100M} = 10\ nsec \quad (10)$$

Now that you have all of the needed parameters, you can calculate the TBPRD value:

$$TBPRD = \frac{T_{PWM}}{2*T_{TBCLK}} \rightarrow \frac{2.5\mu\ sec}{2*10n\ sec} = 125 \quad (11)$$

What this means is that the time-base counter counts from zero to 125 and then back to zero. This counts as one period of the ePWM output, resulting in a 400 kHz output frequency for EPWM1, EPWM2, and EPWM3.

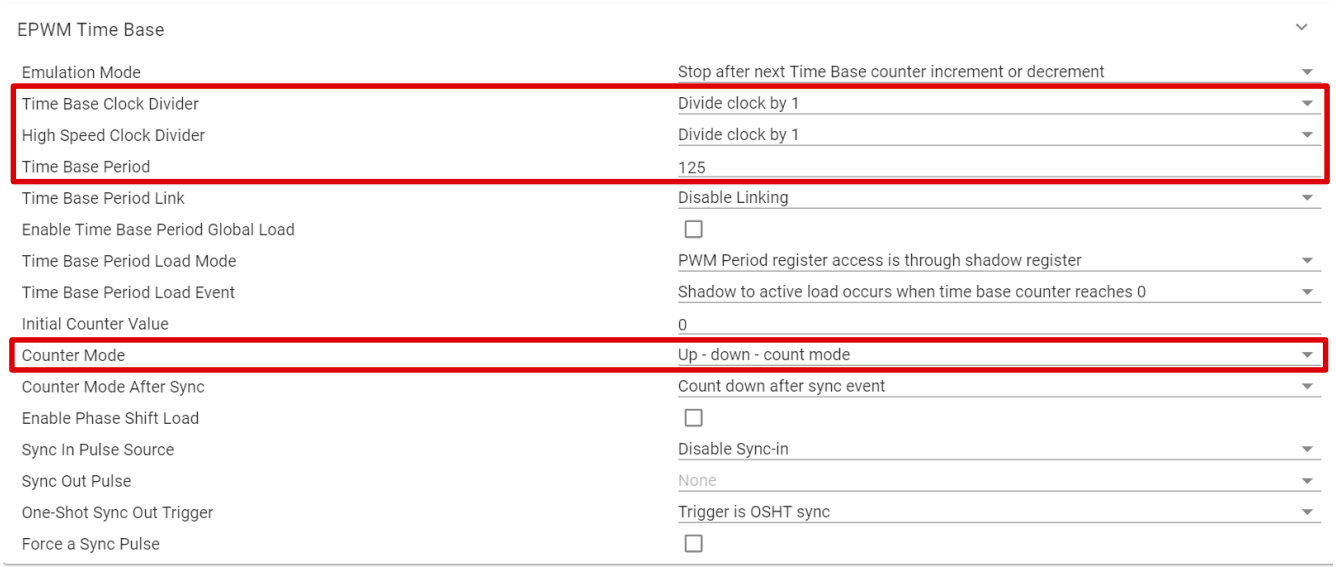

Figure 3-1. EPWM Time Base: Setting TBPRD and Count Mode

The following code is generated from SysConfig:

```
// EPWM1 (the code is the same for EPWM2 and EPWM3, except for the base address)
EPWM_setClockPrescaler(myEPWM1_BASE, EPWM_CLOCK_DIVIDER_1, EPWM_HSCLOCK_DIVIDER_1);
EPWM_setTimeBasePeriod(myEPWM1_BASE, 125);
EPWM_setTimeBaseCounterMode(myEPWM1_BASE, EPWM_COUNTER_MODE_UP_DOWN);
```

3.2 Applying a Phase Shift

The next area of interest is applying a phase shift between ePWM1, ePWM2, and ePWM3. This is also accomplished through the TB submodule.

In order to calculate the time-base phase shift (TBPHS) value, use the formula shown in Equation 12:

$$\text{TBPHS} = \frac{\text{TBPRD*Desired Phase Degree}}{360°} \tag{12}$$

The sync source is ePWM1, meaning it drives the sync signal to ePWM2 and ePWM3. ePWM2 has a phase shift of 120° from ePWM1.

$$\text{TBPHS} = \frac{125*120°}{360°} = \sim 42 \tag{13}$$

ePWM3 has a 240° phase shift from ePWM1:

$$\text{TBPHS} = \frac{125*240°}{360°} = \sim 83 \tag{14}$$

3.3 Setting up the Synchronization (Sync) Scheme

Section 3.3 shows how to set the phase shift value within SysConfig.

Now that the required time-base phase shift values are known, you can setup the synchronization scheme between the three ePWM modules. The ePWM type-4 module has two different sync schemes. Focus on the latest sync scheme for the type-4 module. For this sync scheme, each ePWM module has a synchronization input (SYNCIN), a synchronization output (SYNCO), and a peripheral synchronization output (SYNCPER).

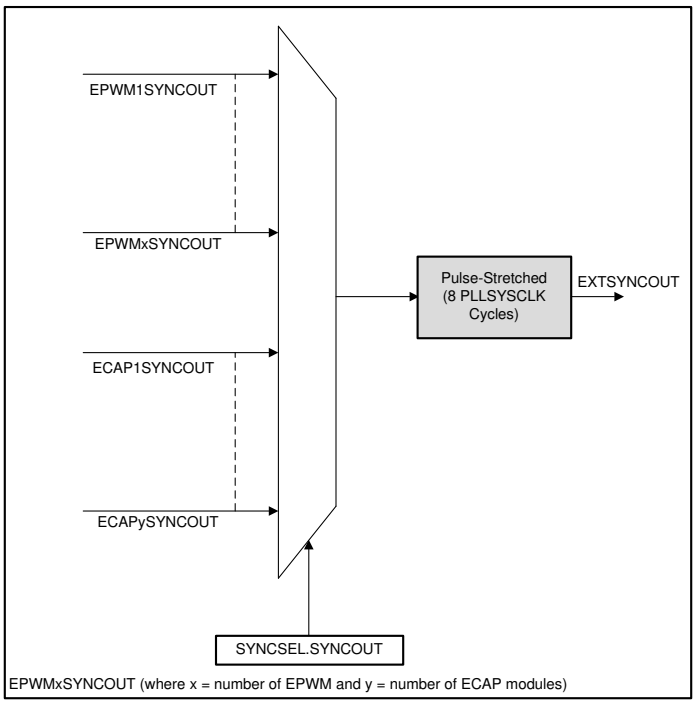

Figure 3-2. Synchronization Scheme

Since ePWM1 is the sync source, all that needs to be setup is the SYNCO, which is what drives the sync chain. There are many options to choose as the SYNCO of ePWM1, such as when the time-base counter is equal to zero or period. In this use-case, you can generate the SYNCO when the time-base counter of ePWM1 is equal to zero.

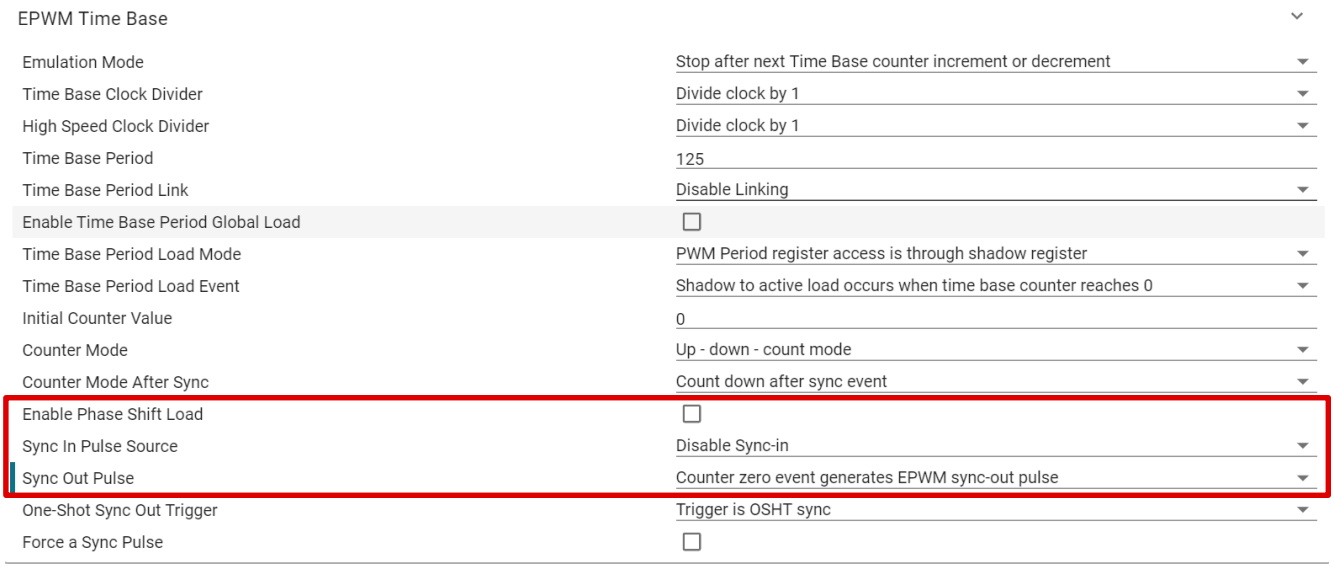

Figure 3-3. EPWM Time Base: Configuration for EPWM1 as the Sync Source

The following code is generated from SysConfig for EPWM1:

```
EPWM_enableSyncOutPulseSource(myEPWM1_BASE, EPWM_SYNC_OUT_PULSE_ON_CNTR_ZERO);
```

For ePWM2 and ePWM3, the SYNCIN comes from ePWM1's SYNCO. The SYNCO occurs whenever the time-base counter is equal to zero. This way, both EPWM2 and EPWM3 receive the SYNCO signal from EPWM1.

For sync receivers such as EPWM2 and EPWM3, it is important to enable the phase shift and provide a phase shift value. An example of this configuration is shown in Figure 3-4.

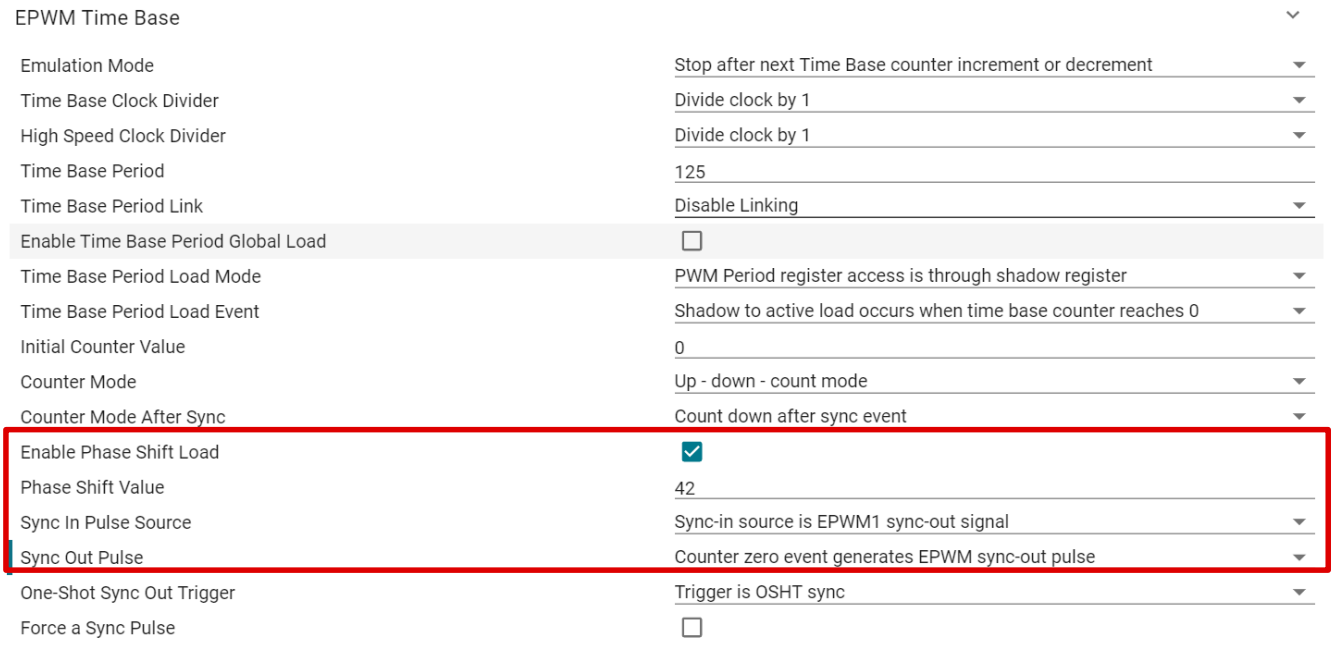

Figure 3-4. EPWM Time Base: Configuration for EPWM2 Sync Receiver Setup

The following code is generated from SysConfig for EPWM2 and EPWM3:

```
// EPWM 2
EPWM_enablePhaseShiftLoad(myEPWM2_BASE);
EPWM_setPhaseShift(myEPWM2_BASE, 42);
EPWM_setSyncInPulseSource(myEPWM2_BASE, EPWM_SYNC_IN_PULSE_SRC_SYNCOUT_EPWM1);
EPWM_enableSyncOutPulseSource(myEPWM2_BASE, EPWM_SYNC_OUT_PULSE_ON_CNTR_ZERO);
// EPWM 3
EPWM_enablePhaseShiftLoad(myEPWM3_BASE);
EPWM_setPhaseShift(myEPWM3_BASE, 83);
EPWM_setSyncInPulseSource(myEPWM3_BASE, EPWM_SYNC_IN_PULSE_SRC_SYNCOUT_EPWM1);
EPWM_enableSyncOutPulseSource(myEPWM3_BASE, EPWM_SYNC_OUT_PULSE_ON_CNTR_ZERO);
```

4 Counter-Compare (CC) and Action-Qualifier (AQ) Submodules

4.1 Calculating the Duty Cycle

Part of the requirements for this use case is having an initial 45% duty cycle. The duty cycle of each ePWM is based on the counter compare values and the actions taken whenever there is a counter compare match. As the time-base counter is incrementing or decrementing, it is constantly checking for a zero, counter compare, or period match. If there is a match, then the ePWM outputs can be programmed to go low (clear), go high (set), toggle, or do nothing.

In this case, the action qualifier events are defined in the following way:

For EPWMxA:

- CMPA UP -> Set
- CMPA DOWN -> Clear

For EPWMxB:

- CMPB UP -> Set
- CMPB DOWM -> Clear

Section 4.1 shows how to configure the AQ submodule. The global load feature is explained in Section 9.

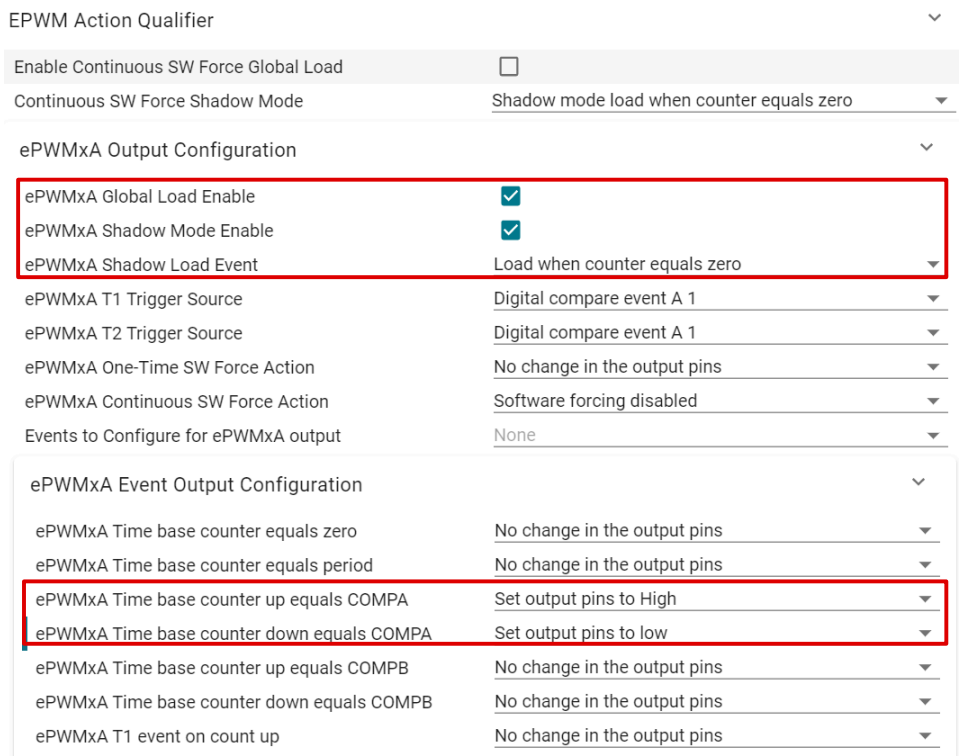

Figure 4-1. EPWM Action-Qualifier: Setting Global Load and Output Events

In order to get an initial 45% duty cycle, you need to determine what value to set for CMPA and CMPB.

Our time-base counter counts a total of 250 (2*125) times in one period since it counts both in the up and down direction. Due to the chosen settings, the waveform is “ON” for the duration of 2*(TBPRD-CMPA) for output A or 2*(TBPRD-CMPB) for output B. Based on this information, you can figure out the CMPA and CMPB values, abbreviated to CMPX.

In order to achieve an initial 45% duty cycle, CMPA and CMPB need to be set to 69 based upon the desired action qualifier events.

Figure 4-2. EPWM Counter-Compare: Setting the Counter Compare Values

The following code is generated from SysConfig:

```
// EPWM 1
EPWM_setCounterCompareValue(myEPWM1_BASE, EPWM_COUNTER_COMPARE_A, 69);
EPWM_enableGlobalLoadRegisters(myEPWM1_BASE, EPWM_GL_REGISTER_CMPA_CMPAHR);
EPWM_setCounterCompareValue(myEPWM1_BASE, EPWM_COUNTER_COMPARE_B, 69);
EPWM_enableGlobalLoadRegisters(myEPWM1_BASE, EPWM_GL_REGISTER_CMPB_CMPBHR);
// EPWM 2 (the code is the same for EPWM3, except for the base address)
EPWM_setCounterCompareValue(myEPWM2_BASE, EPWM_COUNTER_COMPARE_A, 69);
EPWM_enableGlobalLoadRegisters(myEPWM2_BASE, EPWM_GL_REGISTER_CMPA_CMPAHR);
EPWM_setupEPWMLinks(myEPWM2_BASE, EPWM_LINK_WITH_EPWM_1, EPWM_LINK_COMP_A);
EPWM_setCounterCompareValue(myEPWM2_BASE, EPWM_COUNTER_COMPARE_B, 69);
EPWM_enableGlobalLoadRegisters(myEPWM2_BASE, EPWM_GL_REGISTER_CMPB_CMPBHR);
EPWM_setupEPWMLinks(myEPWM2_BASE, EPWM_LINK_WITH_EPWM_1, EPWM_LINK_COMP_B);
```

5 Deadband (DB) Submodule

5.1 Setting up Signal Pairs

For this use case, all three ePWM modules are setup for active high complementary signals. Meaning ePWMxA remains as is with a rising edge delay (RED) and ePWMB has the same source signal as ePWMA, but it is inverted with a falling edge delay (FED) instead of a rising edge delay. Figure 5-1 illustrates these two signals.

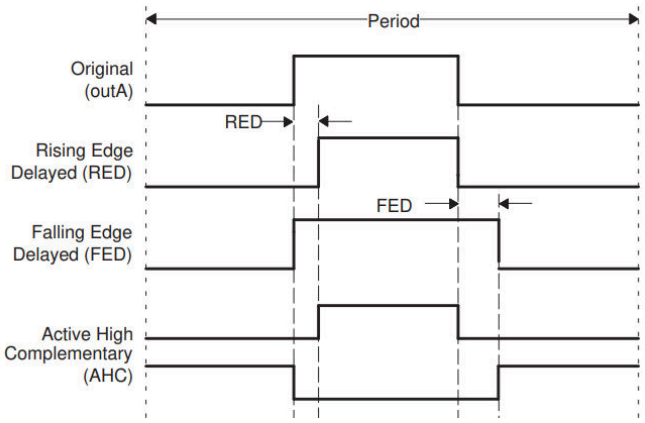

Figure 5-1. Active High Complementary Signal Pairs

In order to setup these signal pairs, utilize the deadband submodule as it allows you to apply this scheme of outputs along with many other signal pairs.

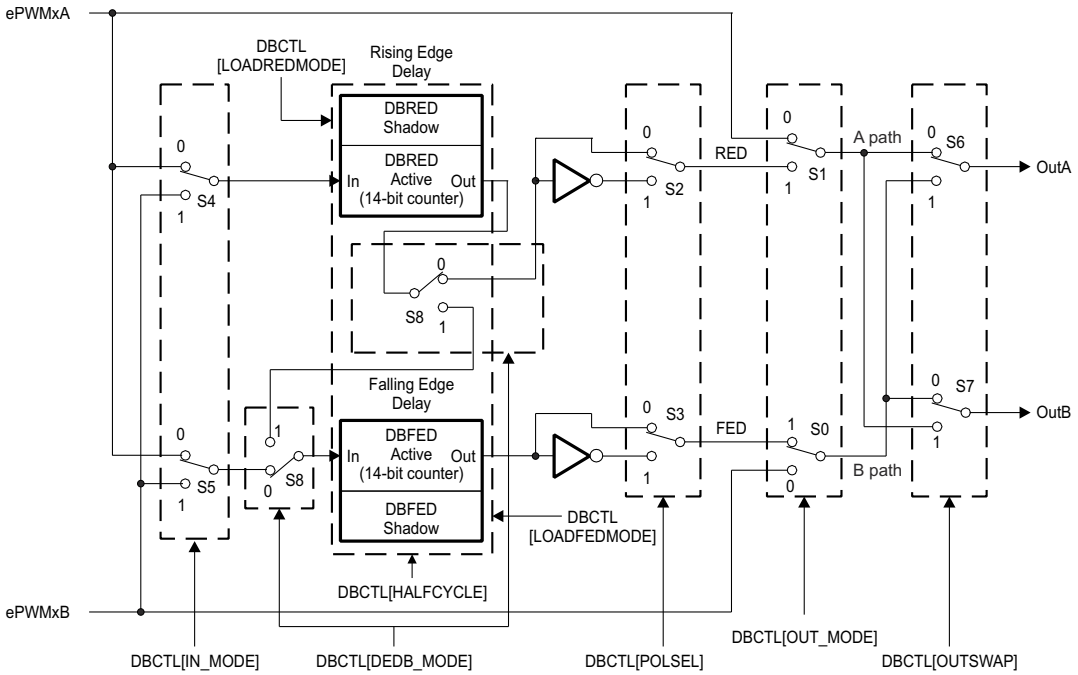

Figure 5-2. Configuration Options for the Deadband Submodule

To begin, take a look at the deadband submodule, which is essentially a set of switches that work together to achieve the desired output. S4 and S5 decide what the source signal is for the rest of the submodules: ePWMA or ePWMB. Since you want the same source signal for both resulting ePWMA and ePWMB outputs, set both switches to the ePWMA input. Next is S8, which is responsible for deciding which of the outputs will have RED and FED or both applied to it. Only one channel, either A or B, can have both RED and FED.

For this case, ePWMA will have RED and ePWMB will have FED so the switch is set to the zero position. Since ePWMB is inverted, S3 is in the 1 position, while S2 remains in the 0 position. The deadband module is being utilized, so S0 and S1 are both set to 1. The last step is deciding whether to swap the current outputs. For this use-case, leave S7 and S6 as-is, meaning the current setup is what is applied and not the swapped output versions.

Figure 5-3 shows this configuration within SysConfig. By pressing the 'SETUP THE DEAD-BAND MODULE' button next to the 'Active High Complementary' option, the SysConfig tool automatically sets up the module in the correct way to achieve active high complementary pairs.

In order to calculate the required dead-band rising edge delay (DBRED) and dead-band falling edge delay (DBFED), the below calculations are done:

$$\text{RED} = \text{DBRED} * \text{T}_{\text{TBCLK}} \tag{15}$$

Re-arrange the equation to find the value of DBRED, which is the value needed:

$$\text{DBRED} = \frac{\text{RED}}{\text{T}_{\text{TBCLK}}} \tag{16}$$

The period of the time-base clock is known from prior equations done in this application report:

$$\text{DBRED} = \frac{200\ \text{nsec}}{10\ \text{nsec}} = 20 \tag{17}$$

Note

The above method is the same for finding the value of DBFED.

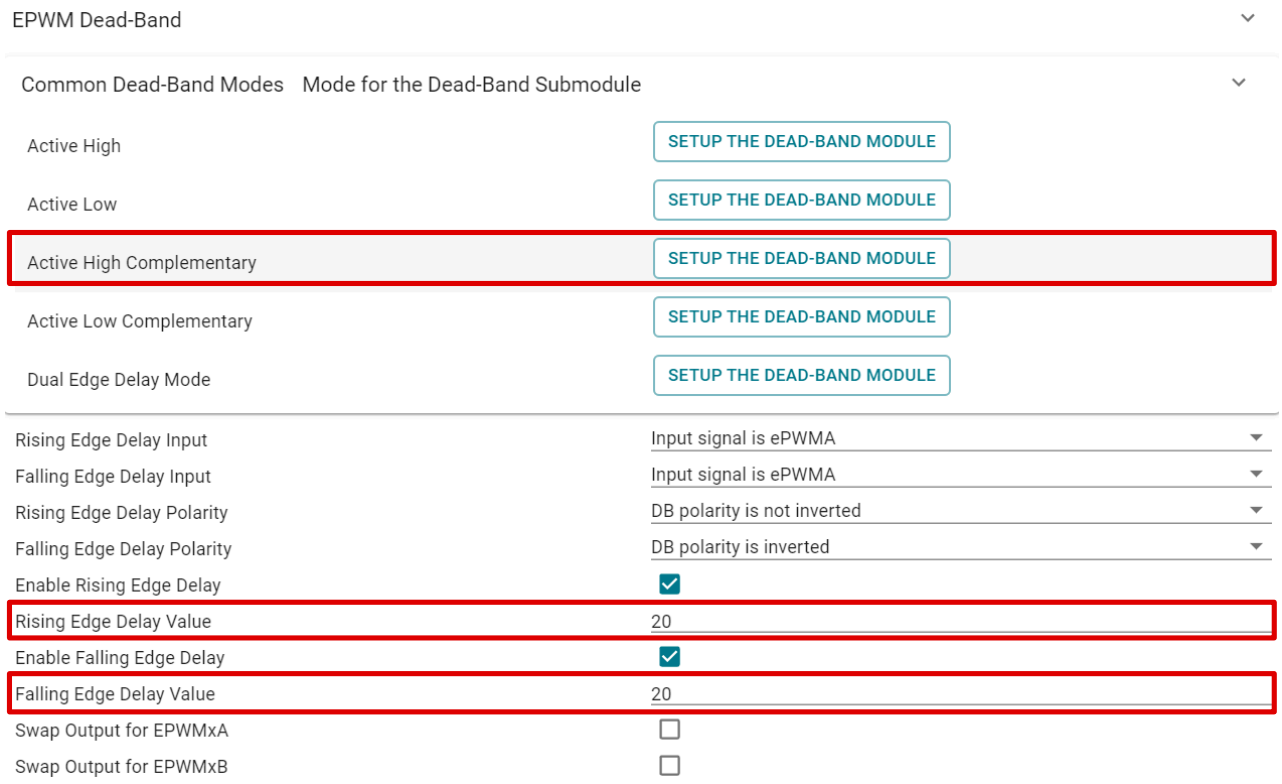

Figure 5-3. EPWM Deadband: Active High Complementary

```
// EPWM 1 (the code is the same for EPWM2 and EPWM3, except for the base address)
EPWM_setDeadBandDelayPolarity(myEPWM1_BASE, EPWM_DB_FED, EPWM_DB_POLARITY_ACTIVE_LOW);
EPWM_setDeadBandDelayMode(myEPWM1_BASE, EPWM_DB_RED, true);
EPWM_setRisingEdgeDelayCount(myEPWM1_BASE, 20);
```

```
EPWM_setDeadBandDelayMode(myEPWM1_BASE, EPWM_DB_FED, true);
EPWM_setFallingEdgeDelayCount(myEPWM1_BASE, 20);
```

6 Verifying the Output

At this point in the application, the initial settings of the PWM can be verified, specifically the duty cycle, dead-band time, and phase shift.

6.1 Checking the Duty Cycle and Dead-Time Insertion

To begin, the positive duty of the dead-band is verified. The desired frequency is 400 kHz for EPWM1, EPWM2, and EPWM3. As seen in Equation 6 the period of the EPWM outputs should be 2.5 µsec. Based on Equation 11, the TBPRD value is 125, meaning the time-base counter of the EPWM module counts from 0 to 125 and then back down to 0, since the counter mode is set to up-down. This yields a total of 250 counts over one period. Therefore, each count of the time-base counter is 10 nsec based on Equation 18.

$$\text{Time for one count of the time base counter} = \frac{T_{PWM}}{TBPRD*2} = \frac{2.5\ \mu sec}{250} = 10\ nsec \tag{18}$$

The counter compare values are set at 69. For the A output, when the time-base counter equals the counter compare while it is counting up, the output is set to go high. When the time-base counter is counting down and there is a match with the counter compare value, then the output is supposed to go low. However, the dead-band submodule has also been setup to incorporate dead-time in the output. As seen through Equation 17, the DBRED and DBFED are set to 20 in order to have a dead-time of 200 nsec, shown in Figure 6-2. Therefore, for EPWMxA, the output does not go high when the time-base counter equals 69 (CMPA value), but rather 89 since there are 20 counts of dead-time inserted for the rising edge. Equation 20 shows how to calculate the duty cycle based on this information.

$$\text{Duty Cycle} = \frac{\text{ON Time}}{\text{ON Time} + \text{OFF Time}} \tag{19}$$

$$\text{Duty Cycle} = \frac{(TBPRD - (CMPA + DBRED)) + (TBPRD - CMPA)}{TBPRD*2} \tag{20}$$

$$\text{Duty Cycle} = \frac{(125 - 89) + (125 - 69)}{TBPRD*2} = \frac{92}{250} = 36.8\ \% \tag{21}$$

Putting it in terms of time, 92 * 10 nsec is 920 nsec, so the positive pulse width should be 920 nsec as seen in Figure 6-1.

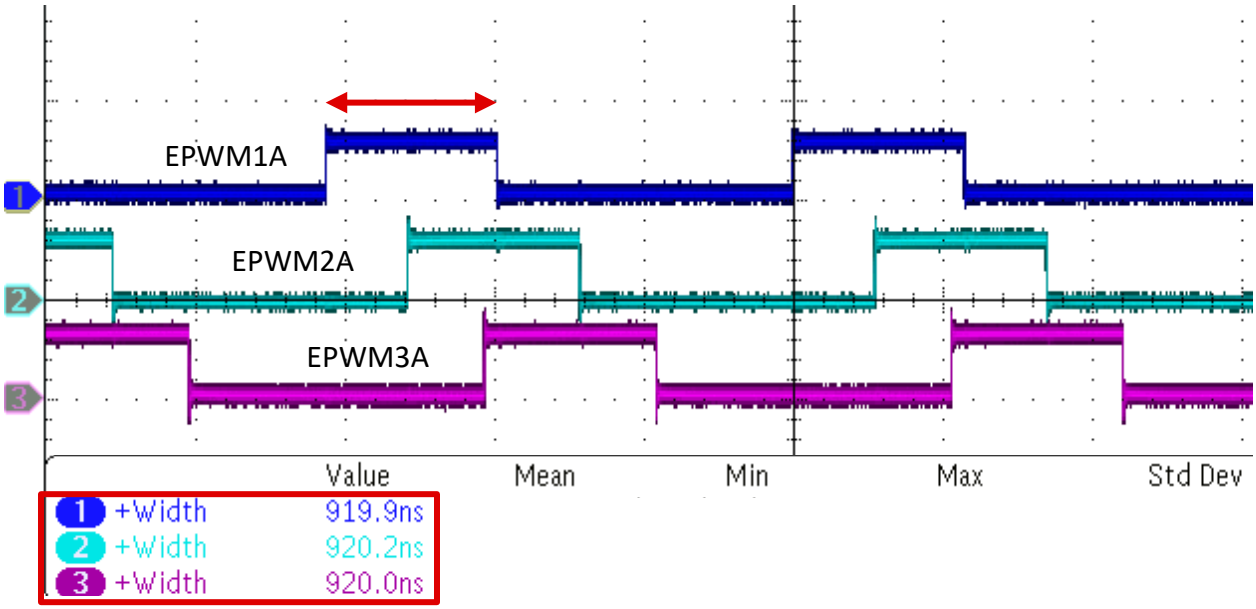

Figure 6-1. Scope Capture of the EPWM Output Positive Duty

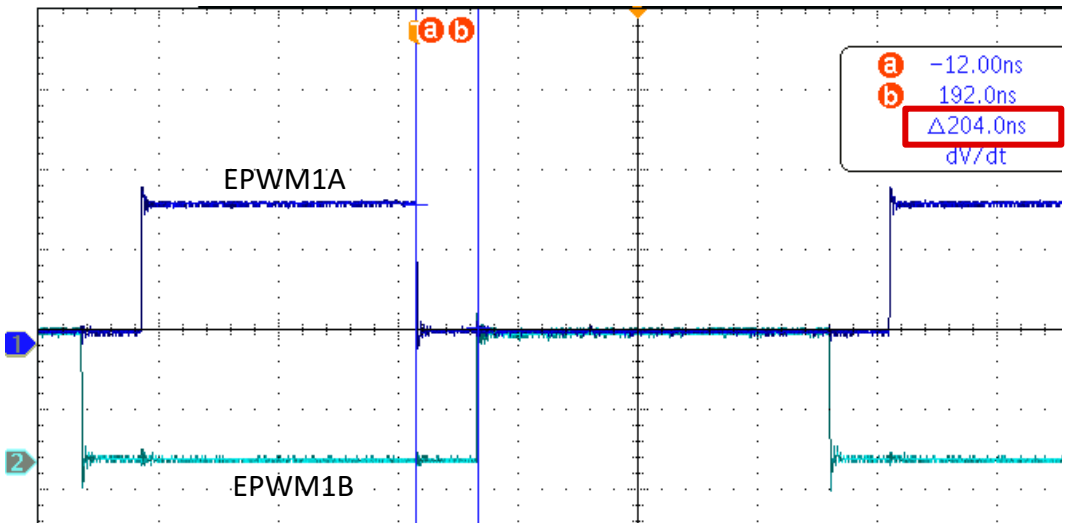

Figure 6-2. Scope Capture of EPWM Output With Dead-Time Insertion

6.2 Checking the Phase Shift Applied

The three EPWM outputs are phase shifted. EPWM2 has a 120° phase shift with respect to EPWM1, and EPWM3 has a 240° phase shift with respect to EPWM1. As seen from Equation 13 and Equation 14, TBPHS is set to 42 for EPWM2 and 83 for EPWM3. Based on the synchronization scheme that was setup, every time EPWM1's time-base counter is equal to 0, a synchronization pulse is generated in which the time-base counter of EPWM2 and EPWM3 is set to the value in the TBPHS register.

Note

The delay from the internal synchronization source module to a synchronization receive module is given by 2 x EPWMCLK if TBCLK = EPWMCLK.

Based on the note, EPWM2 and EPWM3's time-base counter is not set to the TBPHS value, but rather TBPHS + 2. For example, for EPWM2, the TBPHS is 42 counts, which means 42 * 10 nsec = 420 nsec, but accounting for the additional cycles, this is 2 * 10 nsec = 20 nsec, 440 nsec in total. Figure 6-3 shows the time delay between the rising edge of EPWM1A and EPWM2A to showcase the phase difference.

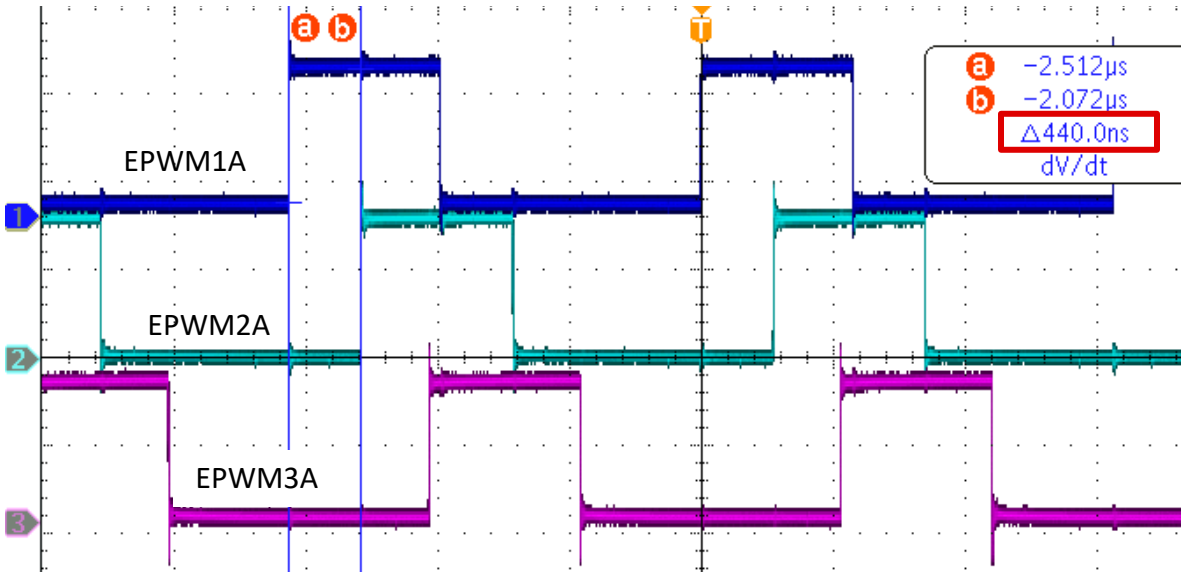

Figure 6-3. Scope Capture of the EPWM Output With Phase Shift

7 Trip-Zone (TZ) and Digital Compare (DC) Submodules

7.1 Drive Outputs Low for an ePWM Cycle Upon Trip Condition Set Through CMPSS

A cycle-by-cycle (CBC) trip is used for current limiting operations. When a cycle-by-cycle trip is detected, the trip-zone submodule drives EPWMxA and EPWMxB to a certain specified state. The outputs go back to their pre-trip state at the next ZRO, PRD, or ZRO/PRD events.

A comparator from the comparator subsystem (CMPSS) is set up to cause a trip event that then generates a CBC trip on EPWM2.

The CMPSS consists of analog comparators and supporting circuits that are useful for power applications. The comparators have a positive and negative input. A high digital output is generated when the voltage on the positive input is greater than the voltage on the negative input, and a low digital output when the voltage on the positive input is less than the voltage on the negative input.

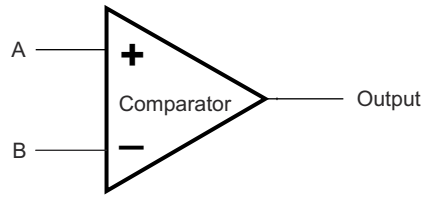

Figure 7-1. Comparator Block Diagram

In order to cause a high digital output, a trip condition needs to occur when the positive input (voltage A) is greater than the negative input (voltage B). Therefore, the positive input is connected to an external voltage source that you need to monitor to make sure it does not exceed a certain state. The negative input can come from the reference digital-to-analog convert (DAC) on the device. For this application, a value of 2500 is written to DACVALA, the value for which the negative terminal will be.

The ideal output of the reference 12-bit DAC can be calculated as:

$$\text{DACOUT} = \frac{\text{DACVALA*DACREF}}{4096} \tag{22}$$

Applying the know values, you get a DACOUT value of 2 V. Therefore, whenever the voltage applied to the positive input of the comparator exceeds 2 V, the trip signal is asserted.

$$\text{DACOUT} = \frac{2500*3.3}{4096} = 2.01\ \text{V} \tag{23}$$

Comparator 1 is utilized for this use case. Figure 7-3 displays the setup for the CMPSS1 module based on the above calculation.

Note

Comparator pins are typically shared with other analog functionality. To determine the analog pin for the positive input of Comparator 1, see the *Analog Subsystem* section within the device-specific Technical Reference Manual. As an example on the F2838x device, CMPIN1P is shared with ADCINA2, so the voltage would be applied to the A2 pin of the device.

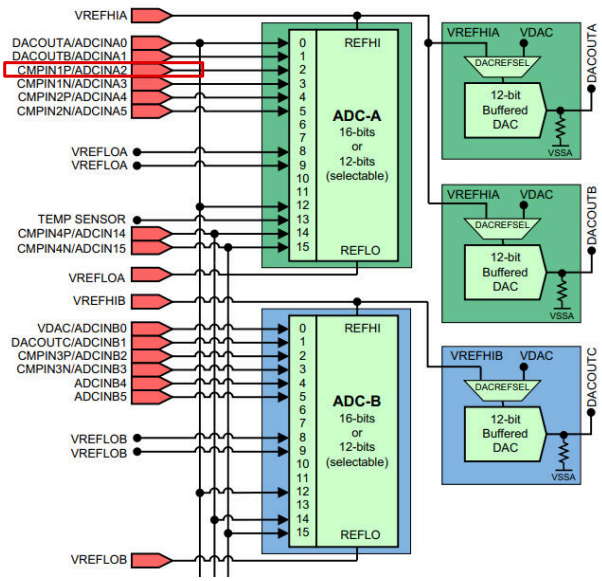

Figure 7-2. Comparator Pins

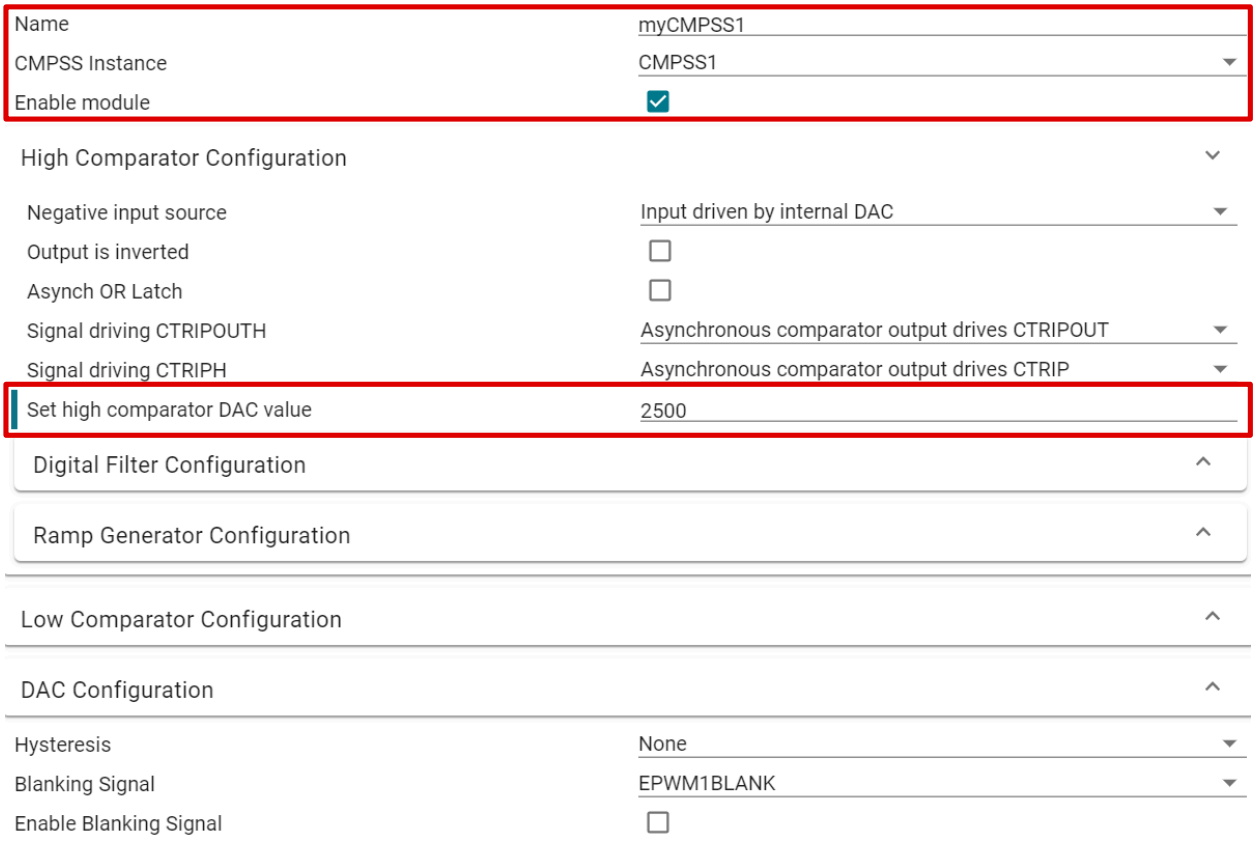

Figure 7-3. CMPSS Configuration

The following code is generated by SysConfig for the CMPSS configuration:

```
void CMPSS_init(){
// myCMPSS1 initialization
// Sets the configuration for the high comparator.
CMPSS_configHighComparator(myCMPSS1_BASE,(CMPSS_INSRC_DAC));
// Sets the configuration for the high comparator.
CMPSS_configLowComparator(myCMPSS1_BASE,(CMPSS_INSRC_DAC));
// Sets the configuration for the internal comparator DACs.
CMPSS_configDAC(myCMPSS1_BASE,(CMPSS_DACVAL_SYSCLK | CMPSS_DACREF_VDDA | CMPSS_DACSRC_SHDW));
// Sets the value of the internal DAC of the high comparator.
CMPSS_setDACValueHigh(myCMPSS1_BASE,2500U);
// Sets the value of the internal DAC of the low comparator.
CMPSS_setDACValueLow(myCMPSS1_BASE,0U);
// Configures the digital filter of the high comparator.
CMPSS_configFilterHigh(myCMPSS1_BASE, 0U, 1U, 1U);
// Configures the digital filter of the low comparator.
CMPSS_configFilterLow(myCMPSS1_BASE, 0U, 1U, 1U);
// Sets the output signal configuration for the high comparator.
CMPSS_configOutputsHigh(myCMPSS1_BASE,(CMPSS_TRIPOUT_ASYNC_COMP | CMPSS_TRIP_ASYNC_COMP));
// Sets the output signal configuration for the low comparator.
CMPSS_configOutputsLow(myCMPSS1_BASE,(CMPSS_TRIPOUT_ASYNC_COMP | CMPSS_TRIP_ASYNC_COMP));
// Sets the comparator hysteresis settings.
CMPSS_setHysteresis(myCMPSS1_BASE,0U);
// Configures the comparator subsystem's ramp generator.
CMPSS_configRamp(myCMPSS1_BASE,0U,0U,0U,1U,true);
// Disables reset of HIGH comparator digital filter output latch on PWMSYNC
CMPSS_disableLatchResetOnPWMSYNCHigh(myCMPSS1_BASE);
// Disables reset of LOW comparator digital filter output latch on PWMSYNC
CMPSS_disableLatchResetOnPWMSYNCLow(myCMPSS1_BASE);
// Sets the ePWM module blanking signal that holds trip in reset.
CMPSS_configBlanking(myCMPSS1_BASE,1U);
// Disables an ePWM blanking signal from holding trip in reset.
CMPSS_disableBlanking(myCMPSS1_BASE);
// Configures whether or not the digital filter latches are reset by PWMSYNC
CMPSS_configLatchOnPWMSYNC(myCMPSS1_BASE,false,false);
// Enables the CMPSS module.
CMPSS_enableModule(myCMPSS1_BASE);
// Delay for CMPSS DAC to power up.
DEVICE_DELAY_US(500);
}
```

It is important to note that the external signal applied to the positive input of the comparator needs to remain for a minimum of 3*TBCLK for the signal to be properly detected and cause a trip condition.

There is no direct path from the comparator subsystem to the trip zone submodule of the ePWM submodule, therefore the digital compare submodule must be utilized to route the signal. In order to figure out how to properly route the CMPSS signal through the digital compare submodule, you must first start with the ePWM X-BAR. The TRM of each device has a table that describes the mux positioning of the ePWM X-BAR, known as “ePWM X-BAR Mux Configuration Table”. Section 7.1 shows an example of an ePWM X-BAR configuration table. From Figure 7-4, you can see that `CMPSS1.CTRIPH` is in position 0 of Mux 0. From the Input X-BAR configuration, you can route this to any trip signal between 4 and 12; for this application, Trip 4 is chosen.

Mux	0	1	2	3
0	CMPSS1.CTRIPH	CMPSS1.CTRIPH_OR_CTRIPL	ADCAEVT1	ECAP1.OUT
1	CMPSS1.CTRIPL	INPUTXBAR1	CLB1_4	ADCCEVT1
2	CMPSS2.CTRIPH	CMPSS2.CTRIPH_OR_CTRIPL	ADCAEVT2	ECAP2.OUT
3	CMPSS2.CTRIPL	INPUTXBAR2	CLB1_5	ADCCEVT2
4	CMPSS3.CTRIPH	CMPSS3.CTRIPH_OR_CTRIPL	ADCAEVT3	ECAP3.OUT
5	CMPSS3.CTRIPL	INPUTXBAR3	CLB2_4	ADCCEVT3
6	CMPSS4.CTRIPH	CMPSS4.CTRIPH_OR_CTRIPL	ADCAEVT4	ECAP4.OUT
7	CMPSS4.CTRIPL	INPUTXBAR4	CLB2_5	ADCCEVT4
8	CMPSS5.CTRIPH	CMPSS5.CTRIPH_OR_CTRIPL	ADCBEVT1	ECAP5.OUT
9	CMPSS5.CTRIPL	INPUTXBAR5	CLB3_4	ADCDEVT1
10	CMPSS6.CTRIPH	CMPSS6.CTRIPH_OR_CTRIPL	ADCBEVT2	ECAP6.OUT
11	CMPSS6.CTRIPL	INPUTXBAR6	CLB3_5	ADCDEVT2

Figure 7-4. Partial Example of an EPWM X-BAR Mux Configuration Table

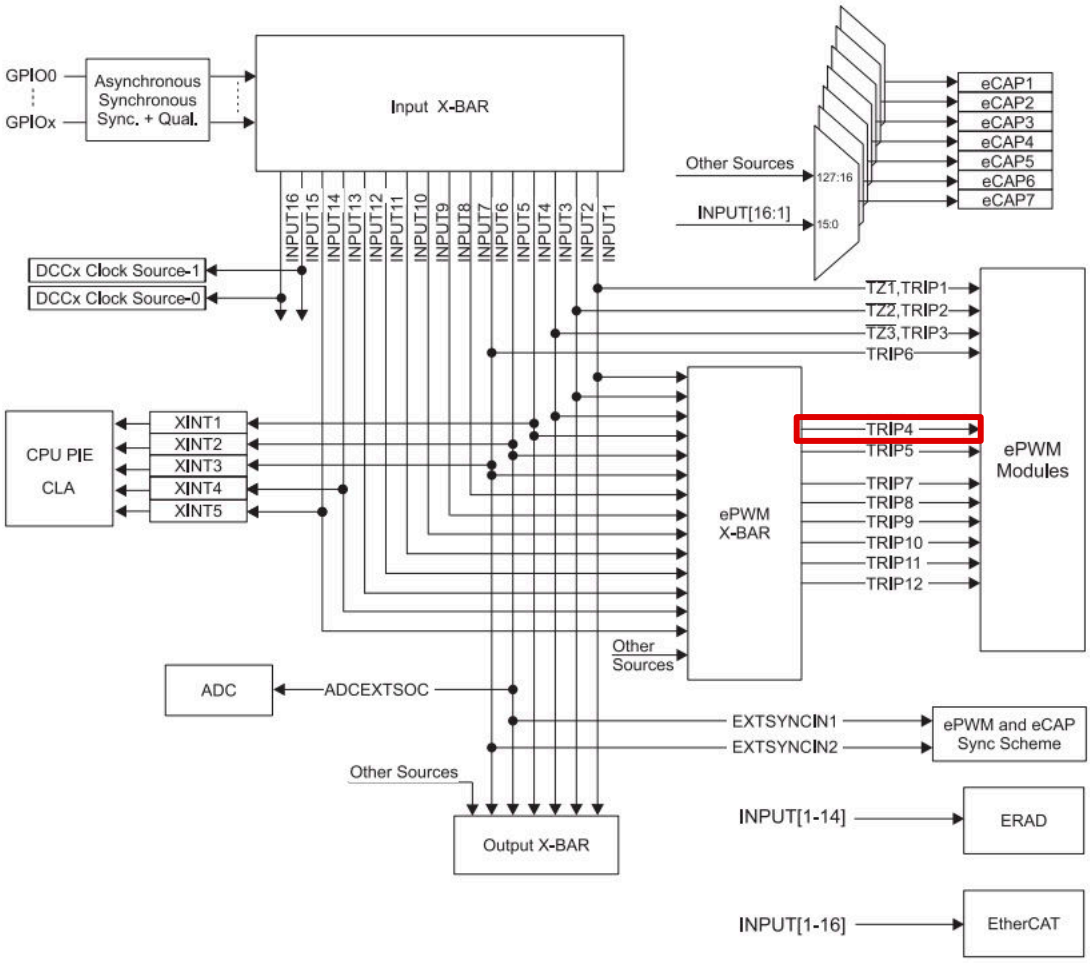

Figure 7-5. Input X-BAR Block Diagram

Figure 7-6 shows how to configure the 'EPWMXBAR' within the system options of SysConfig.

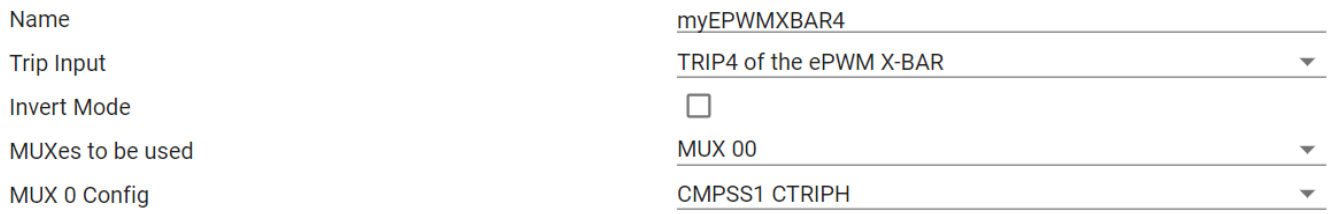

Figure 7-6. SysConfig Configuration for EPWMXBAR

The following code is generated from SysConfig:

```
void EPWMXBAR_init(){
//myEPWMXBAR4 initialization
XBAR_setEPWMMuxConfig(XBAR_TRIP4, XBAR_EPWM_MUX00_CMPSS1_CTRIPH);
XBAR_enableEPWMMux(XBAR_TRIP4, XBAR_MUX00);
}
```

The DC submodule can generate up to four events DCAEVT1, DCAEVT2, DCBEVT1, and DCBEVT2. Only the event 2 can cause a cycle-by-cycle trip, so you need to focus on DCAEVT2 for this application use-case. The digital compare event has a high (DCAH) and low (DCAL) signal. These two signals can be used to cause DCAEVT2. For this case, route the CMPSS1 output to any of the two signals since you only need one. DCAH was chosen. The condition in which the event is generated is when this signal goes high, meaning the value applied to the comparator's positive input is above the voltage at the negative input.

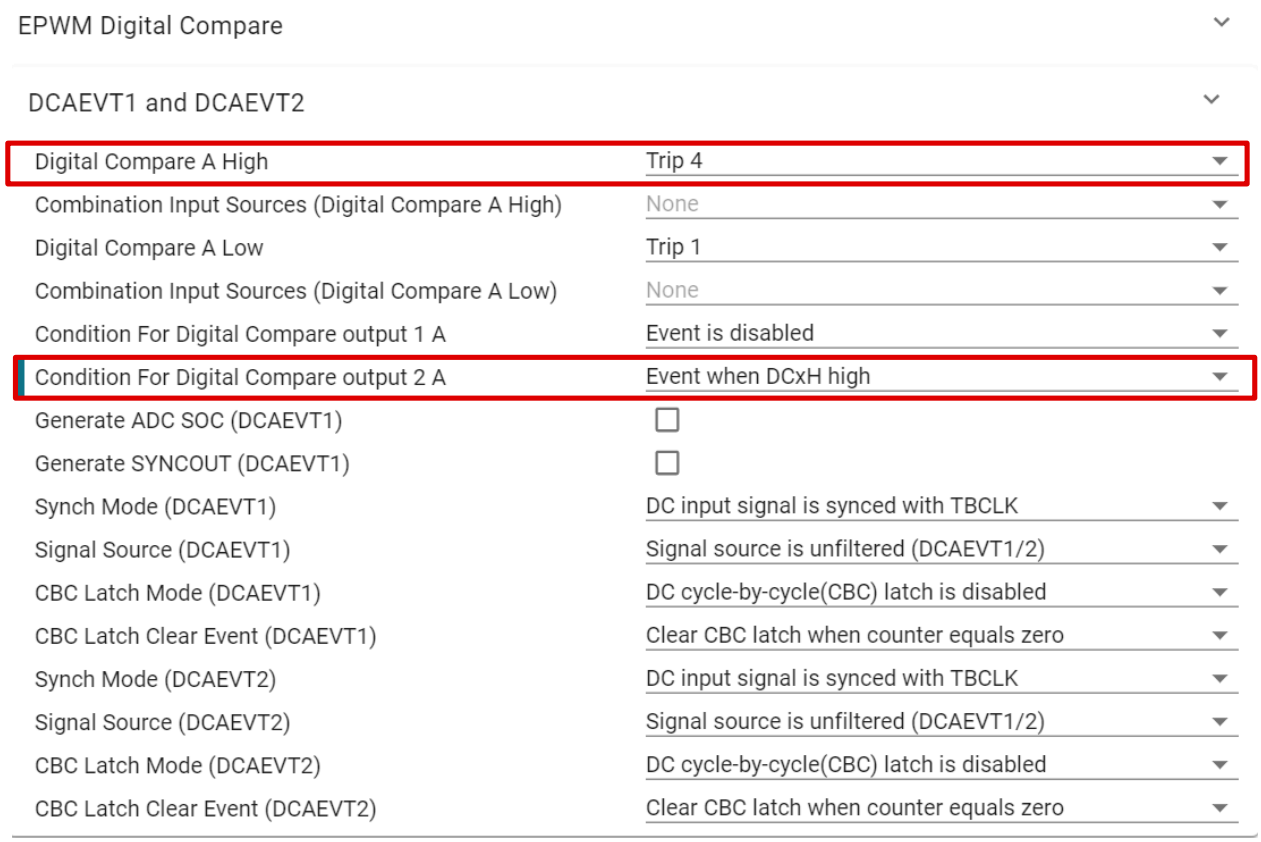

Figure 7-7. EPWM Digital Compare: Setting up DCAEVT2

The following code is generated from SysConfig:

```
EPWM_selectDigitalCompareTripInput(myEPWM2_BASE, EPWM_DC_TRIP_TRIPIN4, EPWM_DC_TYPE_DCAH);
EPWM_setTripZoneDigitalCompareEventCondition(myEPWM2_BASE, EPWM_TZ_DC_OUTPUT_A2,
EPWM_TZ_EVENT_DCXH_HIGH);
```

You can now setup the trip-zone settings to drive both A and B outputs to a low state through a cycle by cycle trip whenever there is a DCAEVT2 event. An interrupt is also generated whenever the trip occurs to allow you to clear the flags.

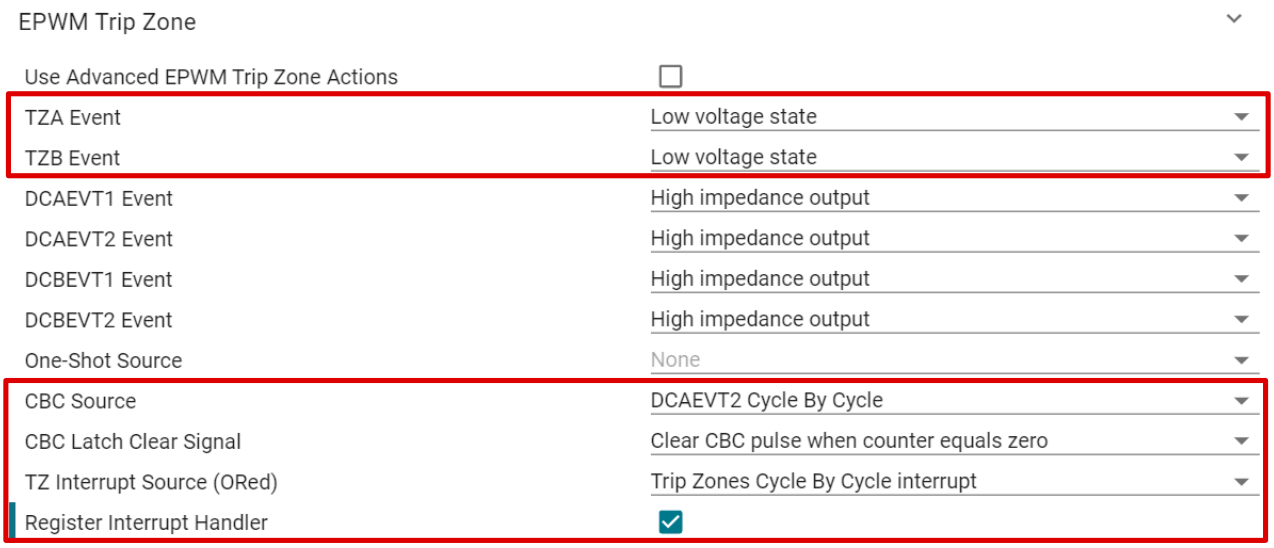

Figure 7-8. EPWM Trip-Zone: Configuration for CBC Trip Based on DCAEVT2

The following code is generated from SysConfig:

```
EPWM_setTripZoneAction(myEPWM2_BASE, EPWM_TZ_ACTION_EVENT_TZA, EPWM_TZ_ACTION_LOW);
EPWM_setTripZoneAction(myEPWM2_BASE, EPWM_TZ_ACTION_EVENT_TZB, EPWM_TZ_ACTION_LOW);
EPWM_enableTripZoneSignals(myEPWM2_BASE, EPWM_TZ_SIGNAL_DCBEVT2);
```

Note

To find the correct interrupt group, see the *PIE Channel Mapping* table within the device-specific TRM.

```
void epwm2TZISR(void){
    epwm2TZIntCount++;

    // Clear the flags
    EPWM_clearTripZoneFlag(myEPWM2_BASE, (EPWM_TZ_INTERRUPT | EPWM_TZ_FLAG_CBC));

    // Acknowledge this interrupt to receive more interrupts from group 2
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP2);
}
```

After configuring EPWM2 to have a trip condition when a positive voltage of greater than 2 V is applied to the positive input on comparator 1, EPWM2 should be driven low.

7.2 Drive Outputs Low Until Cleared Through Software Upon Trip Condition set Through GPIO

Another popular type of trip is a one-shot trip (OSHT). A one-shot trip is used for major short circuit or over-current conditions. When a one-shot trip is detected, the trip-zone submodule drives EPWMxA and EPWMxB to a certain specified state. The outputs remain in that state until the trip is manually cleared.

A GPIO is used to indicate a trip condition on EPWM3 within our application. In order to connect a GPIO to a trip source of the ePWM modules, you need to route it through the Input X-BAR. The Input X-BAR can be used to map any GPIO to the ePWM X-BAR or the ePWM module's trip sources themselves, depending on the desired signal.

For this use-case, use GPIO 12 and route it through the Input X-BAR's Input 1, which feeds directly to TZ1 (Trip Zone input 1).

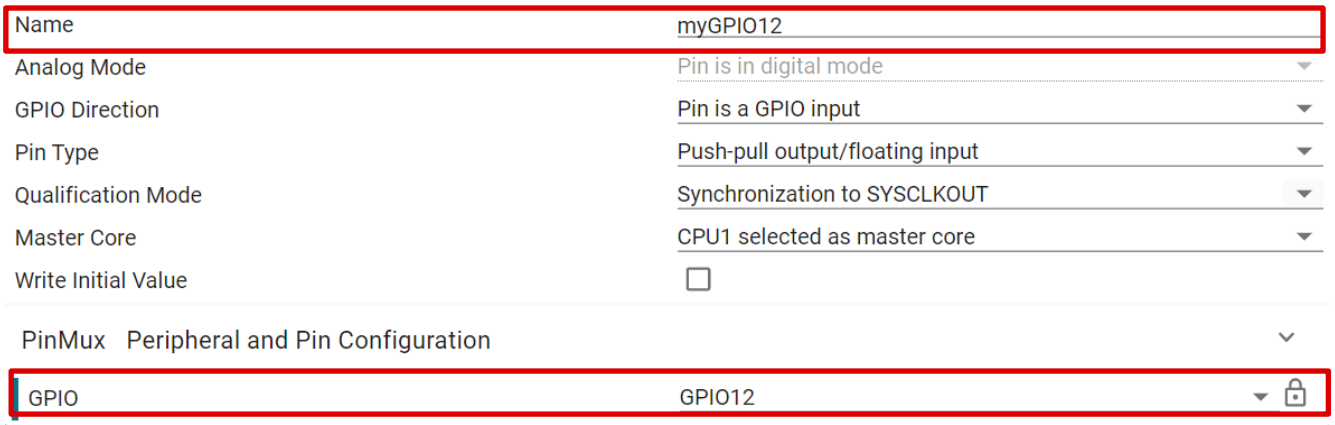

Figure 7-9. GPIO Setup in SysConfig

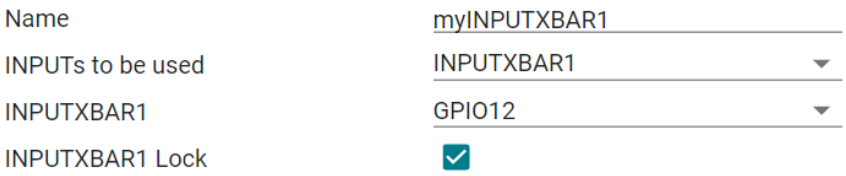

Figure 7-10. Input X-BAR in SysConfig

The following code is generated from SysConfig:

```
Void GPIO_init(){
//myGPIO12 initialization
GPIO_setDirectionMode(myGPIO12, GPIO_DIR_MODE_IN);
GPIO_setPadConfig(myGPIO12, GPIO_PIN_TYPE_STD);
GPIO_setMasterCore(myGPIO12, GPIO_CORE_CPU1);
GPIO_setQualificationMode(myGPIO12, GPIO_QUAL_SYNC);
GPIO_writePin(myGPIO12, 1);
}
void INPUTXBAR_init(){
//myINPUTXBAR1 initialization
XBAR_setInputPin(INPUTXBAR_BASE, XBAR_INPUT1, 12);
XBAR_lockInput(INPUTXBAR_BASE, XBAR_INPUT1);
}
```

Once the GPIO has been routed, you can setup the event. The first step is choosing Trip 1 as the one-shot trip condition. Every time there is a trip condition that occurs on GPIO 12 (the state of the GPIO goes from high to low), the ePWM outputs go low. The action taken, clear, is specified through the TZA (for ePWMXA) and TZB (for ePWMXB) bits of the TZCTL register.

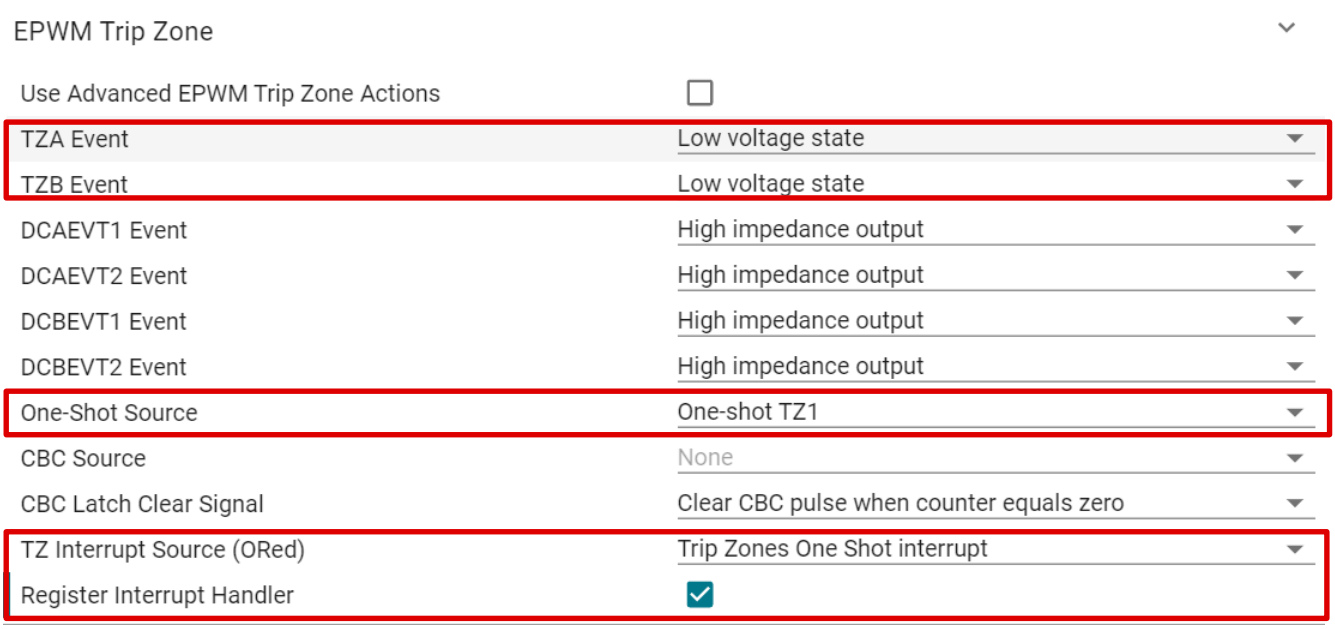

Figure 7-11. Trip Zone Configuration Including One-Shot Configuration

The following code is generated from SysConfig:

```
EPWM_setTripZoneAction(myEPWM3_BASE, EPWM_TZ_ACTION_EVENT_TZA, EPWM_TZ_ACTION_LOW);
EPWM_setTripZoneAction(myEPWM3_BASE, EPWM_TZ_ACTION_EVENT_TZB, EPWM_TZ_ACTION_LOW);
EPWM_enableTripZoneSignals(myEPWM3_BASE, EPWM_TZ_SIGNAL_OSHT1);
EPWM_enableTripZoneInterrupt(myEPWM3_BASE, EPWM_TZ_INTERRUPT_OST);
```

Similar to the cycle-by-cycle trip case, the one-shot trip configuration is also setup to generate an interrupt whenever the trip occurs so that the interrupt flag can be cleared.

```
void epwm3TZISR(void){
epwm3TZIntCount++
//
// Re-enable the OST Interrupt
//
EPWM_clearTripZoneFlag(myEPWM3_BASE, (EPWM_TZ_INTERRUPT | EPWM_TZ_FLAG_OST));
//
// Acknowledge this interrupt to receive more interrupts from group 2
//
Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP2);
}
```

Now that GPIO12 has been setup to do a one-shot trip on EPWM3, drive GPIO12 low and EPWM3 should go low until the GPIO is no longer low and the one-shot trip flag has been cleared.

8 Event-Trigger (ET) Submodule

8.1 Setting Up Time-Base Interrupts

The event trigger submodule manages the events generated by the TB, CC, and DC submodules to generate an interrupt to the CPU or a start of conversion pulse to the analog-to-digital converter (ADC) when a selected event occurs. For this example, interrupts are generated whenever there is a trip event that occurs, which was previously explained in Section 7.1 and Section 7.2 sections. In order to setup an interrupt when the time-base counter reaches zero for EPWM1, see Section 8.1.

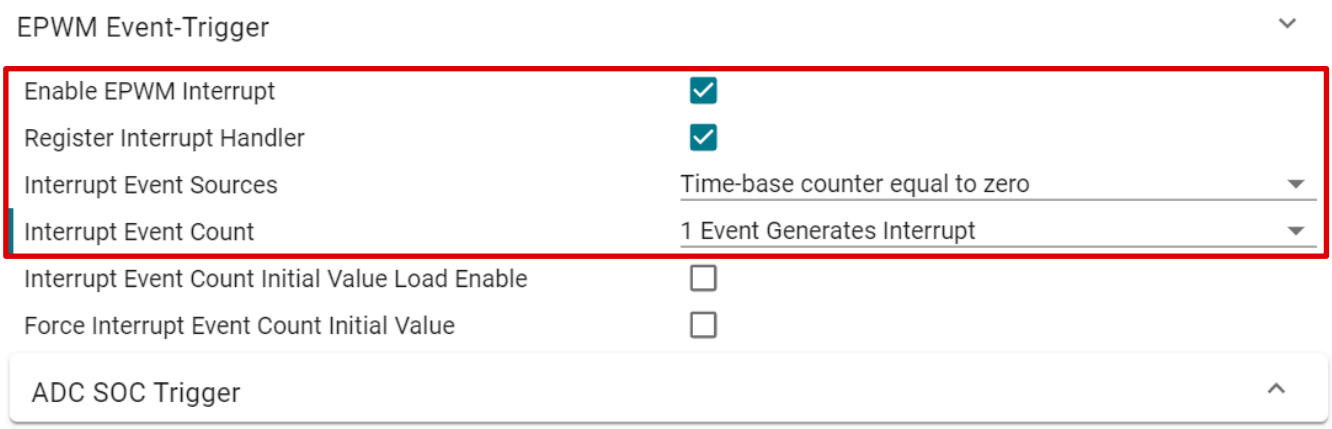

Figure 8-1. EPWM Event-Trigger: Setting up an Interrupt for When TBCTR=ZRO

The following code is generated from SysConfig:

```
// EPWM1
EPWM_enableInterrupt(myEPWM1_BASE);
EPWM_setInterruptSource(myEPWM1_BASE, EPWM_INT_TBCTR_ZERO);
EPWM_setInterruptEventCount(myEPWM1_BASE, 1);
```

9 Global Load

9.1 Applying Global Loading and One-Shot Load Feature

Global loading is a feature within the ePWM module that allows a single event to be the source to update multiple key parameters such as TBPRD, CMPA, CMPB, CMPC, CMPD, DBRED, DBFED, DBCTL, AQCTLA, AQCTLB, and AQCSFRC. These registers have shadow registers, which allow contents written to that register to be held in a “shadow” register until a specified event, such as TBCTR=ZRO, PRD, or ZRO/PRD, for which the content in the shadow register is transferred to the active register.

The one-shot load feature works in conjunction with the global-load feature. With normal global loading, the singular event that causes all of the active registers to update is occurring periodically. However, often times the new values that are required to update key ePWM parameters are not always ready by the next periodic event that is generating the global load signal. Therefore, the one-shot feature is used in order to update the active registers one-time only whenever all of the content is ready to be updated. This is extremely useful in applications where the control loop is happening in an asynchronous ISR such as a timer or ADC ISR, which is not running at a frequency multiple of the ePWM.

In this application report's use case, the global-load/one-shot load feature is used to update the action qualifier settings as well as the counter compare values. The global-load settings for the action qualifier and counter compare submodules have already been configured. For more details, see Section 4.

However, the *Global Load* section within SysConfig still needs to be configured to setup the overall Global Load and One-Shot Load feature.

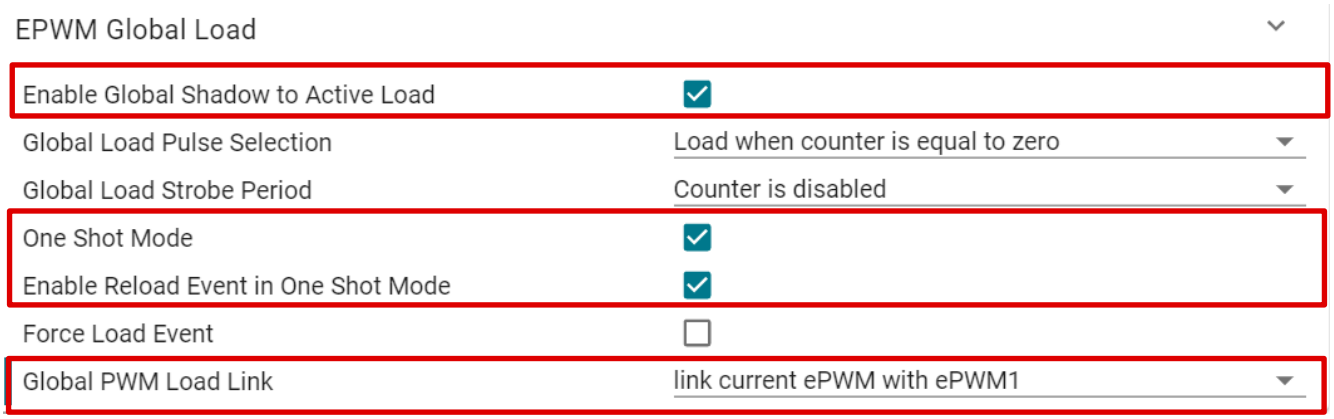

Figure 9-1. Global Loading SysConfig Setup

The following code is generated from SysConfig. For the Action Qualifer and Counter Compare submodules, see Section 4.

```
// EPWM1
EPWM_enableGlobalLoad(myEPWM1_BASE); EPWM_enableGlobalLoadOneShotMode(myEPWM1_BASE);
EPWM_setGlobalLoadOneShotLatch(myEPWM1_BASE);
// EPWM2 (the code is the same for EPWM3, except for the base address)
EPWM_enableGlobalLoad(myEPWM2_BASE);  EPWM_enableGlobalLoadOneShotMode(myEPWM2_BASE);
EPWM_setGlobalLoadOneShotEvent(myEPWM2_BASE);
EPWM_setupEPWMLinks(myEPWM2_BASE, EPWM_LINK_WITH_EPWM_1, EPWM_LINK_GLDCTL2);
```

9.2 Linking the ePWM Modules

In many applications such as a three-phase interleaved LLC implementation, it is useful to be able to update key ePWM parameters simultaneously across different modules. C2000's ePWM Type-4 achieves this though its linking feature. Multiple ePWM modules can be linked together in order to update TBPRD, CMPA, CMPB, CMPC, CMPD, and GLDCTL2 (Global Load control 2) registers simultaneously.

For this case, link to CMPA and CMPB of ePWM2 and ePWM3 to ePWM1, meaning every time you update the value of CMPA or CMPB for ePWM1, the same value is updated for CMPA/CMPB of ePWM2/3.

The linking setting for these parameters is shown in Figure 4-2 and Figure 9-1.

In the main application code, you can utilize the linking feature by updating only CMPA and CMPB of EPWM1. EPWM2 and EPWM3's CMPA/CMPB values are updated simultaneously to the same value.

The action qualifier settings do not have a linking feature capability, so settings for all three ePWM modules needs to be updated independently; changes are being updated in the shadow registers. However, the one-shot load across all three modules is linked so the effect of the change in action qualifier settings occurs at the same time. That is, for all three ePWM modules, the shadow to active load of the AQCTLA settings occurs simultaneously.

9.3 Updating Action Qualifier Settings and Counter Compare Values Through Global Loading

Now that global loading and linking have been setup, the next step is writing the application code to modify the action qualifier settings and counter compare values. For this application report, the counter compare values are changed from 69 to 100, and the action qualifier settings are reversed. For EPWMxA, the settings are now clear on CMPA up and set on CMPA down. The following code shows these new settings as well as the global load one-shot latch being set after the settings are updated. The code should be integrated into a control loop that is asynchronous to the EPWM cycle. However, to simply show functionality, the code is placed within the main for loop of the program. To enable the new settings, set the variable "perform_one_shot_load" to one through the expression window of CCS.

```
if(perform_one_shot_load==1)
  {
    //
    // EPWM1, EPWM2, EPWM3 are linked so only EPWM1 needs to be updated
    //
    EPWM_setCounterCompareValue(myEPWM1_BASE, EPWM_COUNTER_COMPARE_A, 100);
    EPWM_setCounterCompareValue(myEPWM1_BASE, EPWM_COUNTER_COMPARE_B, 100);
    //
    // Change the Action Qualifier Settings for EPWM1, EPWM2, and EPWM3
    //
    EPWM_setActionQualifierAction(myEPWM1_BASE,EPWM_AQ_OUTPUT_A,EPWM_AQ_OUTPUT_LOW,
                                          EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
    EPWM_setActionQualifierAction(myEPWM1_BASE,EPWM_AQ_OUTPUT_A,EPWM_AQ_OUTPUT_HIGH,
                                          EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPA);
    EPWM_setActionQualifierAction(myEPWM2_BASE,EPWM_AQ_OUTPUT_A,EPWM_AQ_OUTPUT_LOW,
                                          EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
    EPWM_setActionQualifierAction(myEPWM2_BASE,EPWM_AQ_OUTPUT_A,EPWM_AQ_OUTPUT_HIGH,
                                          EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPA);
    EPWM_setActionQualifierAction(myEPWM3_BASE,EPWM_AQ_OUTPUT_A,EPWM_AQ_OUTPUT_LOW,
                                          EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
    EPWM_setActionQualifierAction(myEPWM3_BASE,EPWM_AQ_OUTPUT_A,EPWM_AQ_OUTPUT_HIGH,
                                          EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPA);
    //
    // EPWM1, EPWM2, EPWM3 are linked
    //
    EPWM_setGlobalLoadOneShotLatch(myEPWM1_BASE);
    //
    // Clear the setting
    //
    perform_one_shot_load = 0;
}
```

Note

Remember to declare 'perform_one_shot_load' as part of the global variables of the program.

Note

For projects utilizing C2000Ware version 4.01 or below, the following workaround is required to ensure the action qualifier settings are not cleared after enabling the global loading feature within SysConfig. Insert the code below within the main function after the 'board_init()' function call.

```
EPWM_setActionQualifierAction(myEPWM1_BASE, EPWM_AQ_OUTPUT_A, EPWM_AQ_OUTPUT_HIGH,
EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
EPWM_setActionQualifierAction(myEPWM1_BASE, EPWM_AQ_OUTPUT_A, EPWM_AQ_OUTPUT_LOW,
EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPA);
EPWM_setActionQualifierAction(myEPWM1_BASE, EPWM_AQ_OUTPUT_B, EPWM_AQ_OUTPUT_LOW,
EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPB);
EPWM_setActionQualifierAction(myEPWM1_BASE, EPWM_AQ_OUTPUT_B, EPWM_AQ_OUTPUT_HIGH,
EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPB);

EPWM_setActionQualifierAction(myEPWM2_BASE, EPWM_AQ_OUTPUT_A, EPWM_AQ_OUTPUT_HIGH,
EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
EPWM_setActionQualifierAction(myEPWM2_BASE, EPWM_AQ_OUTPUT_A, EPWM_AQ_OUTPUT_LOW,
EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPA);
EPWM_setActionQualifierAction(myEPWM2_BASE, EPWM_AQ_OUTPUT_B, EPWM_AQ_OUTPUT_LOW,
EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPB);
EPWM_setActionQualifierAction(myEPWM2_BASE, EPWM_AQ_OUTPUT_B, EPWM_AQ_OUTPUT_HIGH,
EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPB);

EPWM_setActionQualifierAction(myEPWM3_BASE, EPWM_AQ_OUTPUT_A, EPWM_AQ_OUTPUT_HIGH,
EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
EPWM_setActionQualifierAction(myEPWM3_BASE, EPWM_AQ_OUTPUT_A, EPWM_AQ_OUTPUT_LOW,
EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPA);
EPWM_setActionQualifierAction(myEPWM3_BASE, EPWM_AQ_OUTPUT_B, EPWM_AQ_OUTPUT_LOW,
EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPB);
EPWM_setActionQualifierAction(myEPWM3_BASE, EPWM_AQ_OUTPUT_B, EPWM_AQ_OUTPUT_HIGH,
EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPB);

// EPWM1, EPWM2, EPWM3 are linked so this will update both
EPWM_setGlobalLoadOneShotLatch(myEPWM1_BASE);
```

Redoing the calculations from earlier, the new duty cycle should be as shown in Equation 24:

$$\text{Duty Cycle} = \frac{\text{CMPA} + (\text{CMPA} - \text{DBRED})}{\text{TBPRD*2}} = \frac{(100 + 80)}{250} = \frac{180}{250} = 72\ \% \quad (24)$$

Putting it in terms of time, 180 * 10 nsec is 1.8 µsec, so the positive pulse width should be 1.8 µsec as seen in Section 9.3.

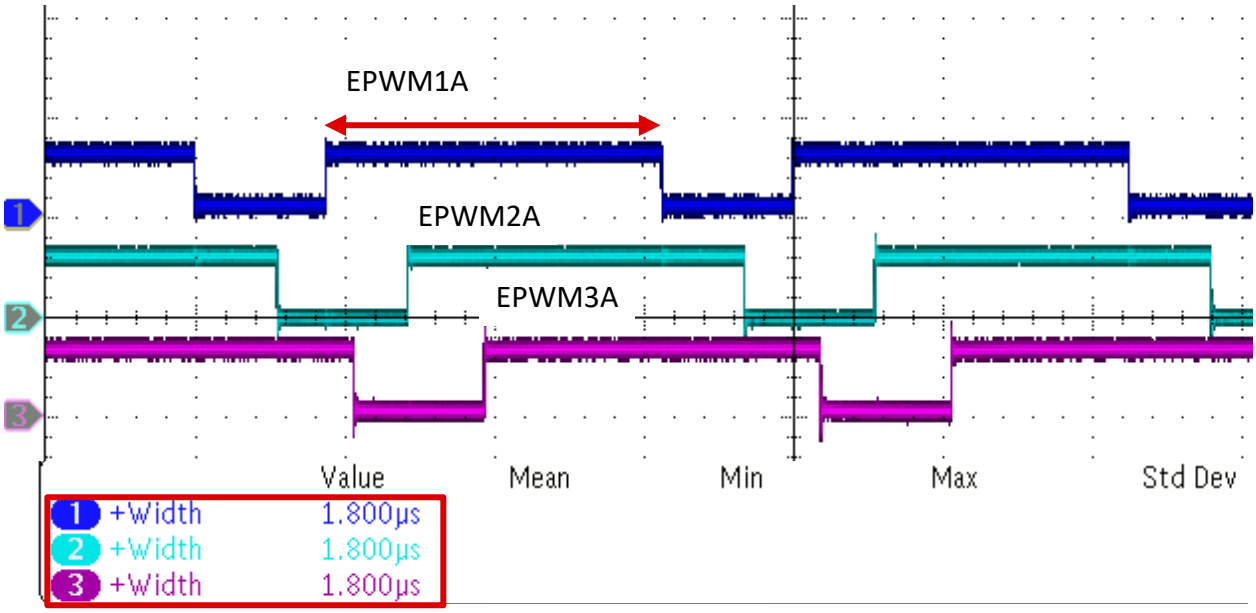

Figure 9-2. Scope Capture of EPWM Output Positive Duty After AQ and CC Updates

10 Summary

This application report highlights all of the major features of the EPWM Type-4 module and how they can be utilized in an application. All of the required calculations were explained along with how to use SysConfig to configure the module and the code generated from SysConfig. For more specific guidance on registers or the EPWM module, see the device-specific Technical Reference Manual.

11 References

- EPWM Video Series
- EPWM Section of C2000 Academy
- Texas Instruments: *Leverage New Type ePWM Features for Multiple Phase Control*
- Texas Instruments: *CRM/ZVS PFC Implementation Based on C2000™ Type-4 PWM Module*
- Texas Instruments: *Implement Three-Phase Interleaved LLC on C2000 Type-4 PWM*
- SysConfig Video Series
- Texas Instruments: *C2000 Real-Time Control MCU Peripherals Guide*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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, regulatory 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 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.

TI objects to and rejects any additional or different terms you may have proposed.

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