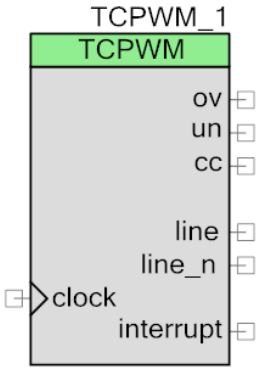
PSoC 4 Timer Counter Pulse Width Modulator (TCPWM)

1.10

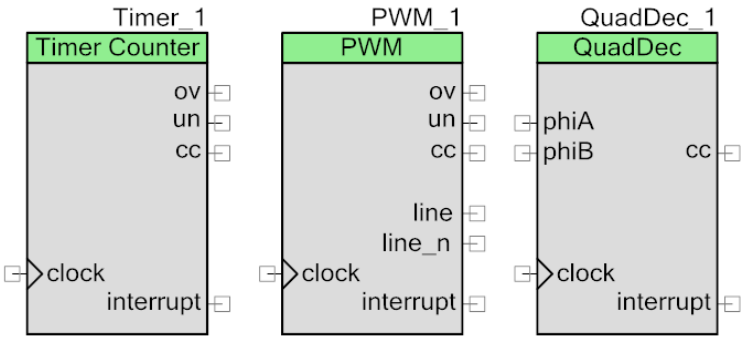


Features

- 16-bit fixed-function implementation
- Timer/Counter functional mode
- Quadrature Decoder functional mode
- Pulse Width Modulation (PWM) mode
- PWM with configurable dead time insertion
- Pseudo random PWM
- Run-time customization

General Description

The TCPWM component is a multifunction component that implements core microcontroller functionality, including Timer/Counter, PWM, and Quadrature Decoder using the PSoC 4 TCPWM block. Each is available as a pre-configured schematic macro in the PSoC Creator Component Catalog, labeled as “TCPWM Mode.”



The base component in the catalog is setup in the unconfigured mode. The unconfigured component is configured at run-time through function calls to perform the operation of any of the modes.

The component is based on a hardware structure designed to share the same hardware across all the various modes of operation. This structure allows the same hardware to provide a flexible set of functions with little increase in silicon use. You can define the functionality at build time to match one of the major modes of operation supported by the hardware. You can also keep the

component as an unconfigured TCPWM and the specific configuration setup at runtime with the API interface.

- The TCPWM has a 16 bit counter that supports Up, Down, and Up/Down counting modes. Rising edge, falling edge, combined rising/falling edge detection or pass-through on all HW input signals can be used to derive counter events. Three routed output signals are available to indicate underflow, overflow. and counter/compare match events.
- The component has double buffered compare/capture and period registers that allow for reconfiguration or register switching through the APIs at run time. The start, reload, stop, count, and capture events can be derived from any HW input signal, and can also be generated by software.
- You can set the PWM mode to either PWM, PWM with dead time insertion, or Pseudo random PWM mode. Two PWM complementary output lines are available. Dead time insertion of 0 to 255 counter cycles is supported.

When to use a TCPWM

TCPWM can be configured to any one of these modes using the customizer:

- Timer with Compare
- Timer with Capture
- PWM
- PWM with Dead time
- PWM with Pseudo Random output
- Quadrature Decoder

Using the customizer to configure the component is the typical use case for anyone using PSoC Creator as their development environment since it is the simplest configuration method. Alternatively, the TCPWM can be unconfigured at build time and configured with software APIs at run-time. The unconfigured method can be used to create designs for multiple applications and where the specific usage of the TCPWM in the design is not known when the Creator hardware design is developed. All configuration settings can be made at run-time except for the connection of signals, clock, and interrupts.

Input/Output Connections

This section describes the various input and output connections for the TCPWM component. The mode field indicates the modes in which the I/Os are visible.

Inputs

Input	Mode	Description
clock	All	The clock input defines the operating frequency of this component. The maximum frequency is 48 MHz.
reload	Timer, PWM	This input allows a reload event to initialize and start the counter. In up and up/down counting mode, the counter is initialized with “0”. In down counting mode, the counter is initialized with the period value. When in the PWM pseudo random mode, the reload signal performs the same function as the start signal.
index	QuadDec	The index input detects a reference position for the Quadrature Decoder. An event on this signal generates a terminal count (TC) event and initializes the counter with the mid-point counter value “0x8000”.
count	Timer, PWM	Depending on the configuration, the count signal increments or decrements the counter value.
phiA	QuadDec	One of the two counting inputs that control the count value, increment, and decrement, depending on their relationship and the mode.
start	Timer, PWM	The start signal does not initialize the counter, but continues counting from the current counter value. If the Start terminal is present, then the counter will start based on this signal.
phiB	QuadDec	One of the two counting inputs that control the count value, increment, and decrement, depending on their relationship and the mode.
stop	All	The stop signal halts the counter. The event does not erase the current counter value. The stop event has a higher priority than the reload and start events.
capture	Timer	An assertion on this input captures the current counter value. So a capture event copies the counter register value to the capture register, and copies the capture register value to the buffer capture register.
switch	PWM	This signal swaps the period (period, periodBuf) and/or compare (compare, compareBuf) registers at the next TC event (this swap depends on the GUI settings or APIs swap parameters).

Note All inputs are double synchronized in the TCPWM. The synchronizer is run at HFCLK speed. After that, these signals are synchronized with the component clock. This results in a delay between when these signals are applied and when they take effect. The delay depends on the ratio between HFCLK and the clock that runs the TCPWM component. All waveforms shown for the TCPWM component show the signal after it has been synchronized.

The actual HFCLK frequency can be observed in the Design-Wide Resources Clock Editor.

Outputs

Output	Mode	Description
ov	Timer, PWM	This output Indicates the status of the counter overflow. It is high when an overflow occurs. Not applicable to PWM in pseudo random mode.
un	Timer, PWM	When this output goes high, it indicates that a counter underflow has occurred. Not applicable to PWM in pseudo random mode.

Output	Mode	Description
cc	All	Comparison or capture output.
line	PWM	PWM output value.
line_n	PWM	Inverted PWM output value. In Dead Time Insertion mode, the line and line_n each have their rising edges delayed, resulting in a period where both are low.
interrupt	All	An Interrupt can be triggered by any of the following sources: • The count has hit its TC. • A hardware capture has been performed. • Compare signal has a rising edge. • The component doesn't have buried ISRs. Use Derived Interrupt type of the ISR component. Please care about clear happened interrupt by the TCPWM_ClearInterrupt() API in the own interrupt handler.

Note The overflow (ov), underflow (un), and compare/capture (cc) output signals have two HFCLK cycle pulse width for PSoC 4100/PSoC 4200 devices and two SYSCLK cycle pulse width for PSoC4000 devices.

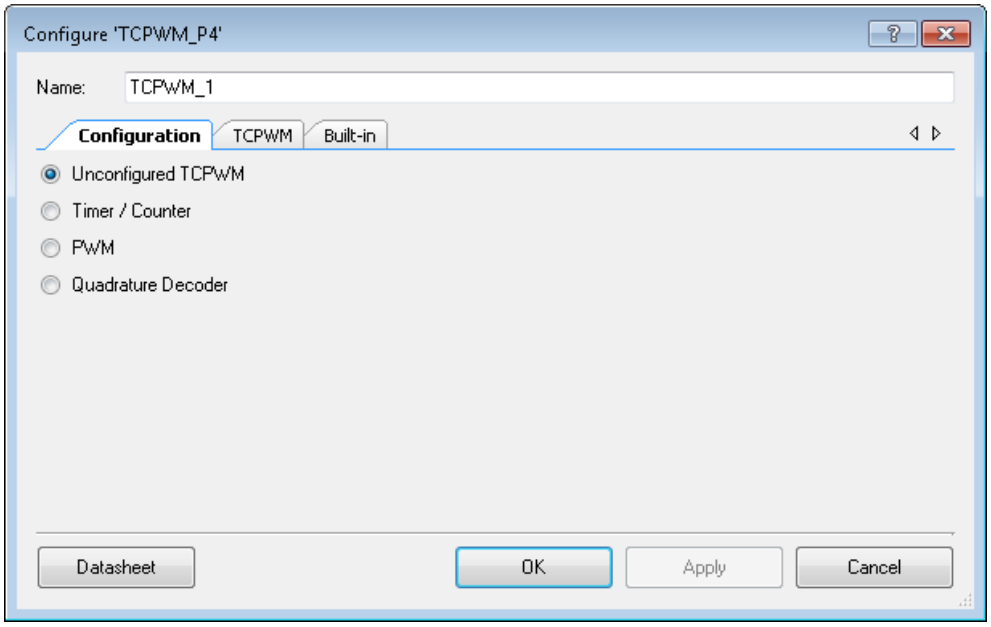
The actual HFCLK and SYSCLK frequencies can be observed in the Design-Wide Resources Clock Editor.

Component Parameters

Drag a TCPWM component onto your design and double-click it to open the **Configure** dialog. This dialog contains the following tabs to guide you through the process of setting up the TCPWM component:

- **Configuration:** Configures the TCPWM mode.
- **TCPWM:** Provides options to display input signals for the Unconfigured TCPWM.
- **Timer/Counter:** Provides configuration for the Timer/Counter mode. This tab is visible only when the Timer/Counter mode has been selected.
- **PWM:** Provides configuration for the PWM mode. It is visible only when the PWM mode has been selected.
- **Quadrature Decoder:** Provides configuration for the Quadrature Decoder mode. Visible only when the Quadrature Decoder mode has been selected.

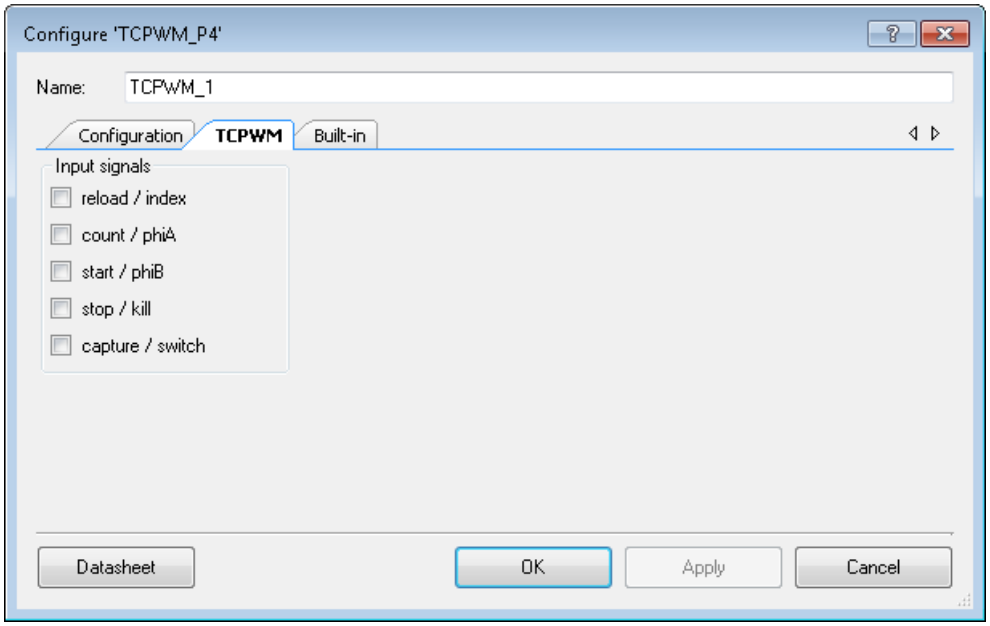
Configuration Tab



The **Configuration** tab provides options for configuring the TCPWM mode. The available modes are as follows:

- **Unconfigured TCPWM –** This is the default mode. The TCPWM component must be configured at run-time when configured in this mode.
- **Timer/Counter –** Configured to be in Timer/Counter mode. The Timer/Counter tab is available when you select this mode.
- **PWM –** Configured to be in PWM mode. The PWM Tab is available when you select this mode.
- **Quadrature Decoder –** Configured to be in Quadrature Decoder mode. The Quadrature Decoder tab is available when you select this mode.

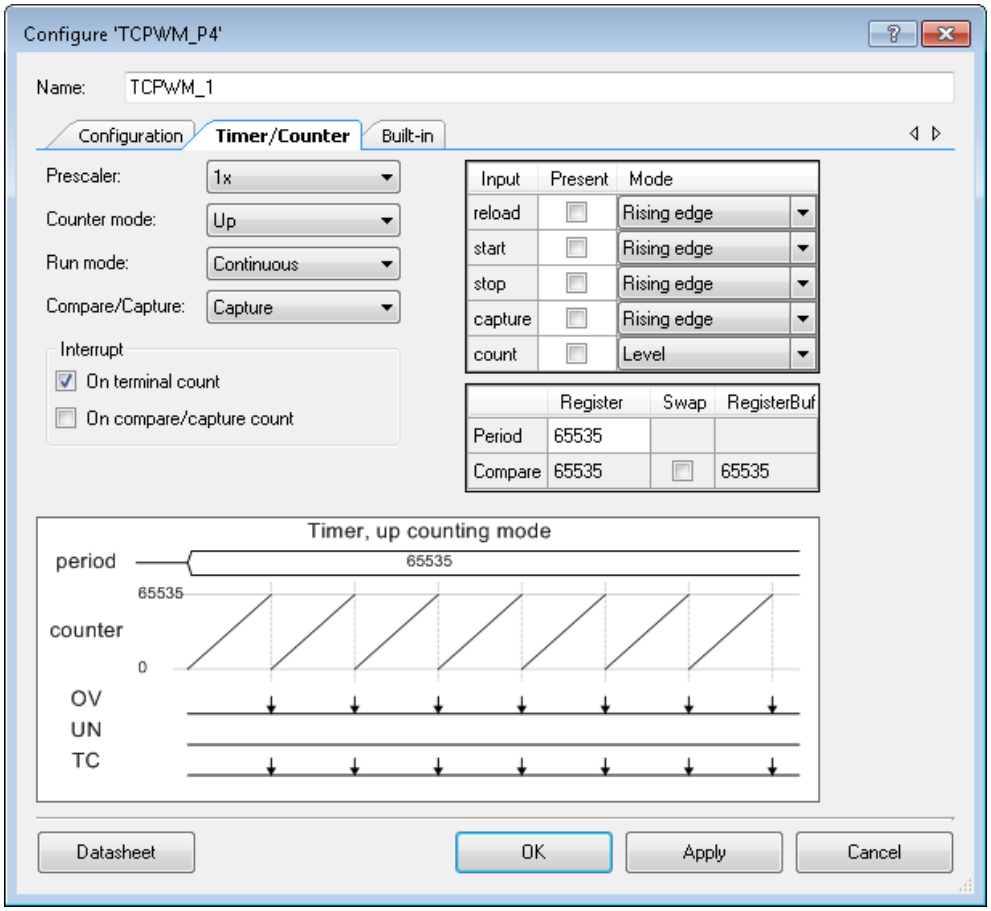
TCPWM Tab



Input signals

Use the **Input Signals** parameter to select the visibility for each of the five input signals: reload/index, count/phiA, start/phiB, stop/kill, and/or capture/switch. These inputs are not visible on the symbol by default. Select the appropriate check box for each input needed in the design to make it visible on the symbol for connection.

Timer/Counter Tab



Prescaler

The **Prescaler** parameter selects which prescaler value to apply to the clock. The available values range from 1 to 128 in power of 2 increments. The default is to not prescale the clock (1x).

Counter mode

The **Counter Mode** parameter selects the direction of the counter. It can be configured to count Up, Down, Up/Down 0, and Up/Down 1. Up/Down 0 only triggers a terminal count (TC) on underflow and Up/Down 1 triggers a TC on both an underflow and an overflow.

Capture / Compare

This parameter selects between the capability to Capture or to Compare. Only one of these two functions is available at the same time.

Run mode

You select the **Run mode** to run continuously or in one shot. One shot mode causes the counter to stop when TC is reached.

Input configuration

The **Input Configuration** parameters select the visibility and mode for each of the five input signals (Reload, Start, Stop, Capture, and Count). These inputs are not visible on the symbol by default. Check the Present checkbox for each input that is needed in the design to make it present on the symbol for connection. Each of these signals can be configured to be triggered by a Rising edge, Falling edge, Either edge, or based on the signal Level.

Period

The **Period** parameter determines the initial value for the period register. Valid **Period** values range from 0 to 65535. The default value for the **Period** is set to 65535.

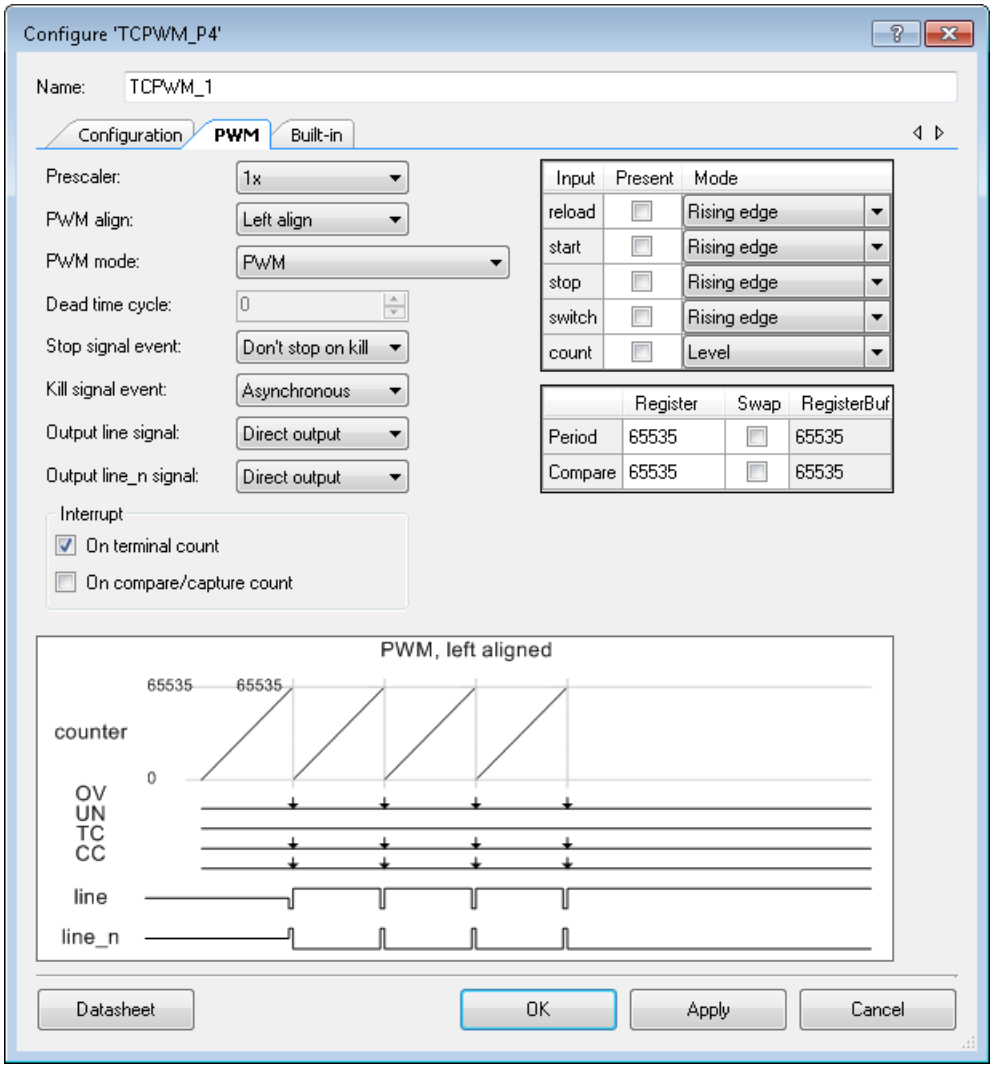
Compare

The **Compare** parameter sets the initial value for the comparison registers and the swap check box selects whether one or two comparison values are used. You cannot access the **Compare** parameter unless the **Capture/Compare** mode is set to “compare.” By default, the swap check box is unchecked. The first compare value is always present and the second compare value (compareBuf) is present only when you select the swap check box. The swap selection causes the two compare values to swap at each TC event. Valid **Compare** values range from 0 to 65535. The default value for the **Compare** is set to 65535.

Interrupt

The **Interrupt** parameter determines which events cause interrupts. It can be configured to be triggered on TC, or on compare/capture count.

PWM Tab



Prescaler

The **Prescaler** parameter selects the prescaler value to apply to the clock. The available values range from 1 to 128 in power of 2 increments. This feature is not available in dead time mode.

PWM align

The alignment of the PWM waveform is selected using the **PWM align** parameter. This can be set to Left align, Right align, Center align, or Asymmetric.

PWM mode

The **PWM mode** can be set to either PWM, PWM with dead time insertion, or Pseudo random PWM mode. The default setting is PWM.

Dead time cycles

The **Dead time cycles** are configurable when the PWM with dead time insertion mode is selected as the **PWM mode**. This parameter sets the number of cycles of dead time insertion. The value can range from 0 to 255. The default is set to 0.

Run mode

The selection is only present in Pseudo Random PWM mode. The **Run mode** can be selected to run continuously or in one shot. One shot mode causes the counter to stop when TC is reached.

Stop signal event

The **Stop signal event** determines the type of action taken when a stop signal is asserted. The stop event will cause a kill operation (line and line_n outputs will go inactive). This selection determines whether in addition to the kill operation the stop event also stops the counter. It can be set to either Stop on kill or Don't stop on kill. The default setting is Don't stop on kill.

Kill signal event

The **Kill signal event** parameter selects whether or not a kill is Synchronous or Asynchronous. A synchronous Kill kills the output as long as the Kill signal is high. Once Kill signal goes to low again, the PWM output will not be re-enabled until TC event comes. For this mode the stop signal must be configured as a rising edge triggered signal. An asynchronous kill will kill the output only while the stop signal is held high. For this mode the stop signal must be configured as a level triggered signal. The default setting is Asynchronous.

Output line signal

The **Output line signal** selects whether inversion is performed on the line signal to give either a Direct output or the Inverse output. It is configured to give the Direct output by default.

Output line_n signal

The **Output line_n** signal selects whether inversion is performed on the line_n signal to give either a Direct output or the Inverse output. It is configured to give the Direct output by default.

Input configuration

The **Input Configuration** parameters select the visibility and mode for each of the five input signals (Reload, Start, Stop, Switch, and Count). These inputs are not visible on the symbol by default. Check the Present checkbox for each input that is needed in the design to make it present on the symbol for connection. Each of these signals can be configured to be triggered by a Rising edge, Falling edge, Either edge, or based on the signal Level.

Period

The **Period** parameter determines the initial value for the period registers. The swap checkbox selects whether or not to use one or two period values. By default, the swap checkbox is unchecked. The first period value is always present and the second period value (periodBuf) is present only when the swap checkbox is checked. The swap selection causes the two period values to swap at each TC event. Valid **Period** values range from 0 to 65535. The default value for the **Period** is set to 65535.

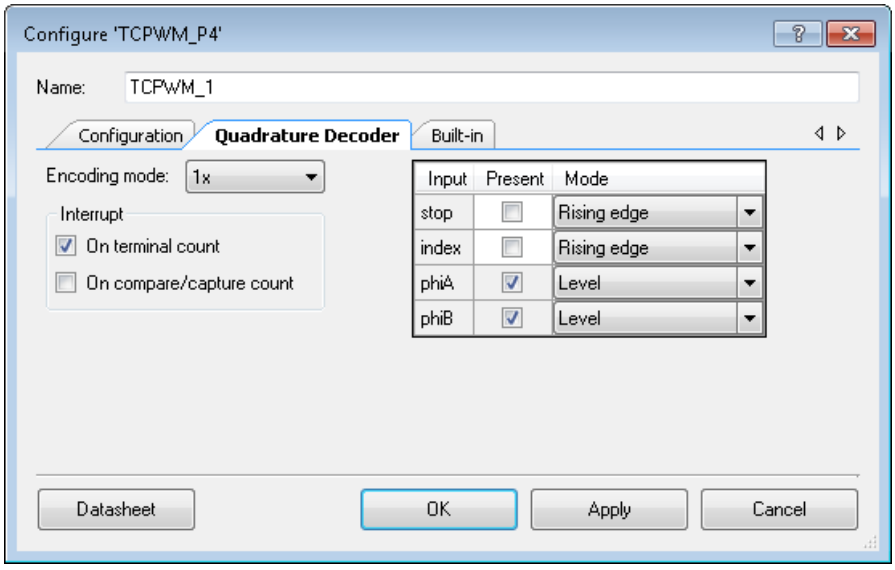
Compare

The **Compare** parameter sets the initial value for the comparison registers and the swap checkbox selects whether one or two comparison values are used. By default, the swap checkbox is unchecked. Note that the first compare value is always present and the second compare value (compareBuf) is present only when the swap checkbox is checked. The swap selection causes the two compare values to swap at each TC event. Valid **Compare** values range from 0 to 65535. The default value for the **Compare** is set to 65535.

Interrupt

The **Interrupt** parameter determines which events cause interrupts. It can be configured to be triggered on TC, or on compare/capture count.

Quadrature Decoder Tab



Encoding mode

The quadrature decoder **Encoding mode** can be set to one of three modes: 1x, 2x, or 4x. This determines the resolution of the counter measuring the transitions. The higher the resolution, the more accurate a position can be encoded at the cost of counter size.

Input configuration

The **Input Configuration** parameters select the visibility and mode for each of the four input signals (Stop, Index, PhiA, and PhiB). The Stop and Index inputs are not visible on the symbol by default. Check the Present checkbox for each input that is needed in the design to make it present on the symbol for connection. Each of these signals can be configured to be triggered by a Rising edge, Falling edge, Either edge, or based on the signal Level. The PhiA and PhiB signals are always present and should be configured in Level mode

Interrupt

The **Interrupt** parameter determines which events cause interrupts. It can be configured to be triggered on TC, or on compare/capture count.

Clock Selection

The clock is provided using the clock terminal. This clock must be from the global clock generation logic. Clock prescaler functionality is available within the TCPWM component.

Placement

The TCPWM component will be placed into one of the available TCPWM resources. The specific TCPWM used does not impact the operation of the component.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. This table lists and describes the interface for each function. The following sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "TCPWM_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "TCPWM". Table with function appliance is presented below.

Function	Description
TCPWM_Init()	Initialize/Restore default TCPWM configuration
TCPWM_Enable()	Enables the TCPWM. TCPWM will be started if the Start terminal is not present
TCPWM_Start()	Initializes the TCPWM with default customizer values when called the first time and enables the TCPWM. TCPWM will be started if the Start terminal is not present
TCPWM_Stop()	Disables the TCPWM

Function	Description
TCPWM_SetMode()	Sets the operational mode of the TCPWM
TCPWM_SetPrescaler()	Sets the prescaler value that is applied to the clock input
TCPWM_TriggerCommand()	Triggers the designated command to occur on the designated TCPWM instances
TCPWM_SetOneShot()	Writes the register that controls whether the TCPWM runs continuously or stops after TC is reached
TCPWM_SetPWMMode()	Writes the control register that determines what mode of operation the PWM output lines are driven in
TCPWM_SetPWMSyncKill()	Writes the register that controls whether the PWM kill signal (stop input) causes an asynchronous or synchronous kill operation
TCPWM_SetPWMStopOnKill()	Writes the register that controls whether the PWM kill signal (stop input) causes the PWM counter to stop
TCPWM_SetPWMDeadTime()	Writes the dead time control value
TCPWM_SetPWMInvert()	Writes the bits that control whether the line and line_n outputs are inverted from their normal output values
TCPWM_SetQDMode()	Sets the Quadrature Decoder to one of 3 supported modes
TCPWM_SetInterruptMode()	Sets the interrupt mask to control which interrupt requests generate the interrupt signal
TCPWM_GetInterruptSourceMasked()	Gets the interrupt requests masked by the interrupt mask
TCPWM_GetInterruptSource()	Gets the interrupt requests (without masking)
TCPWM_ClearInterrupt()	Clears the interrupt request
TCPWM_SetInterrupt()	Sets a software interrupt request
TCPWM_WriteCounter()	Writes a new 16 bit counter value directly into the counter register
TCPWM_ReadCounter()	Reads the current counter value
TCPWM_SetCounterMode()	Sets the counter mode
TCPWM_SetPeriodSwap()	Writes the register that controls whether the period registers are swapped
TCPWM_SetCompareSwap()	Writes the register that controls whether the compare registers are swapped
TCPWM_ReadCapture()	Reads the captured counter value
TCPWM_ReadCaptureBuf()	Reads the capture buffer register
TCPWM_WritePeriod()	Writes the 16 bit period register with the new period value
TCPWM_ReadPeriod()	Reads the 16 bit period register
TCPWM_WritePeriodBuf()	Writes the 16 bit period buffer register with the new period value
TCPWM_ReadPeriodBuf()	Reads the 16 bit period buffer register

Function	Description
TCPWM_WriteCompare()	Writes the 16 bit compare register with the new compare value
TCPWM_ReadCompare()	Reads the compare register
TCPWM_WriteCompareBuf()	Writes the 16 bit compare buffer register with the new compare value
TCPWM_ReadCompareBuf()	Reads the compare buffer register
TCPWM_SetCaptureMode()	Sets the capture trigger mode
TCPWM_SetReloadMode()	Sets the reload trigger mode
TCPWM_SetStartMode()	Sets the start trigger mode
TCPWM_SetStopMode()	Sets the stop trigger mode
TCPWM_SetCountMode()	Sets the count trigger mode
TCPWM_ReadStatus()	Reads the status of the TCPWM
TCPWM_Sleep()	This is the preferred API to prepare the component for sleep.
TCPWM_Wakeup()	This is the preferred API to restore the component to the state when TCPWM_Sleep() was called.
TCPWM_SaveConfig()	Saves the configuration of the component.
TCPWM_RestoreConfig()	Restores the configuration of the component.

Global Variables

Variable	Description
TCPWM_initVar	Indicates whether or not the TCPWM has been initialized. The variable is initialized to 0 and set to 1 the first time TCPWM_Start() is called. This allows the component to restart without reinitialization after the first call to the TCPWM_Start() routine. If reinitialization of the component is required, call TCPWM_Init() before calling TCPWM_Start(). Alternatively, the TCPWM can be reinitialized by calling the TCPWM_Init() and TCPWM_Enable() functions

Function Applicability

Function Name	Timer/Counter (Capture)	Timer/Counter (Compare)	PWM	PWM DT	PWM PR	Quad Dec
Init	+	+	+	+	+	+
Enable	+	+	+	+	+	+
Start	+	+	+	+	+	+
Stop	+	+	+	+	+	+
SetMode	+	+	+	+	+	+

Function Name	Timer/Counter (Capture)	Timer/Counter (Compare)	PWM	PWM DT	PWM PR	Quad Dec
SetPrescaler	+	+	+	-	+	-
TriggerCommand	+	+	+	+	+	+
SetOneShot	+	+	-	-	+	-
SetPWMMode	-	-	+	+	+	-
SetPWMSyncKill	-	-	+	+	-	-
SetPWMStopOnKill	-	-	+	+	+	-
SetPWMDeadTime	-	-	-	+	-	-
SetPWMInvert	-	-	+	+	+	-
SetQDMode	-	-	-	-	-	+
SetInterruptMode	+	+	+	+	+	+
GetInterruptSourceMasked	+	+	+	+	+	+
GetInterruptSource	+	+	+	+	+	+
ClearInterrupt	+	+	+	+	+	+
SetInterrupt	+	+	+	+	+	+
WriteCounter	+	+	+	+	+	+
ReadCounter	+	+	+	+	+	+
SetCounterMode	+	+	+	+	-	-
SetPeriodSwap	-	-	+	+	+	-
SetCompareSwap	-	+	+	+	+	-
ReadCapture	+	-	-	-	-	+
ReadCaptureBuf	+	-	-	-	-	+
WritePeriod	+	+	+	+	+	-
ReadPeriod	+	+	+	+	+	-
WritePeriodBuf	-	-	+	+	+	-
ReadPeriodBuf	-	-	+	+	+	-
WriteCompare	-	+	+	+	+	-
ReadCompare	-	+	+	+	+	-
WriteCompareBuf	-	+	+	+	+	-
ReadCompareBuf	-	+	+	+	+	-
SetCaptureMode	+	-	+(switch)	+(switch)	+(switch)	-
SetReloadMode	+	+	+	+	+	+(index)

Function Name	Timer/Counter (Capture)	Timer/Counter (Compare)	PWM	PWM DT	PWM PR	Quad Dec
SetStartMode	+	+	+	+	+	+(phiB)
SetStopMode	+	+	+	+(kill)	+	+
SetCountMode	+	+	+	+	+	+(phiA)
ReadStatus	+	+	+	+	+	+
Sleep	+	+	+	+	+	+
Wakeup	+	+	+	+	+	+
SaveConfig	+	+	+	+	+	+
RestoreConfig	+	+	+	+	+	+

void TCPWM_Init(void)

Description: Initialize/Restore the default TCPWM configuration.

Parameters: None

Return Value: None

Side Effects: None

void TCPWM_Enable(void)

Description: Enables the TCPWM. TCPWM will be started if the Start terminal is not present. If the Start terminal is present then the counter will start based on this signal.

Parameters: None

Return Value: None

Side Effects: None

void TCPWM_Start(void)

Description: Initializes the TCPWM with default customizer values when called the first time and enables the TCPWM. For subsequent calls the configuration is left unchanged and the component is simply enabled. TCPWM will be started if the Start terminal is not present. If the Start terminal is present then the counter will start based on this signal.

Parameters: None

Return Value: None

Side Effects: None

void TCPWM_Stop(void)

Description: Disables the TCPWM.

Parameters: None

Return Value: None

Side Effects: None

void TCPWM_SetMode(uint32 mode)

Description: Sets the operational mode of the TCPWM. This function is used when the component is configured as an unconfigured TCPWM, and the actual mode of operation is set at run-time. The mode must be set while the component is disabled.

Parameters: uint32 mode: Mode for the TCPWM to operate in.

Value	Description
TCPWM_MODE_TIMER_COMPARE	Timer / Counter with compare capability
TCPWM_MODE_TIMER_CAPTURE	Timer / Counter with capture capability
TCPWM_MODE_QUAD	Quadrature decoder
TCPWM_MODE_PWM	PWM
TCPWM_MODE_PWM_DT	PWM with dead time
TCPWM_MODE_PWM_PR	PWM with pseudo random capability

Return Value: None

Side Effects: None

void TCPWM_SetPrescaler(uint32 prescaler)

Description: Sets the prescaler value that is applied to the clock input. It is not applicable to PWM with dead time mode or Quadrature Decoder mode.

Parameters: uint32 prescaler: Prescaler divider value.

Value	Description
TCPWM_PRESCALE_DIVBY1	Divide by 1 (no prescaling)
TCPWM_PRESCALE_DIVBY2	Divider by 2
TCPWM_PRESCALE_DIVBY4	Divider by 4
TCPWM_PRESCALE_DIVBY8	Divider by 8
TCPWM_PRESCALE_DIVBY16	Divider by 16
TCPWM_PRESCALE_DIVBY32	Divider by 32
TCPWM_PRESCALE_DIVBY64	Divider by 64
TCPWM_PRESCALE_DIVBY128	Divider by 128

Return Value: None

Side Effects: None

void TCPWM_TriggerCommand(uint32 mask, uint32 command)

Description: Triggers the designated command to occur on the designated TCPWM instances. Use the mask to apply this command simultaneously to more than one instance. This allows multiple TCPWM instances to be synchronized.

Parameters: uint32 mask: Combination of mask bits for each instance of the TCPWM that the command should apply to. A function from one instance can be used to apply the command to any of the instances in the design. The mask value for a specific instance is available with the TCPWM_MASK define for that instance.

uint32 command: Enumerated command values.

Value	Description
TCPWM_CMD_CAPTURE	Trigger Capture command
TCPWM_CMD_RELOAD	Trigger Reload command
TCPWM_CMD_STOP	Trigger Stop command
TCPWM_CMD_START	Trigger Start command

Return Value: None

Side Effects: None

void TCPWM_SetPWMMode(uint32 modeMask)

Description: Writes the control register that determines what mode of operation the PWM output lines are driven in. There is a setting for what to do on a comparison match (CC_MATCH), on an overflow (OVERFLOW) and on an underflow (UNDERFLOW). The value for each of the 3 must be ORed together to form the mode.

Parameters: uint32 modeMask: Combination of the 3 mode settings. Mask must include a value for each of the 3 or use one of the preconfigured PWM settings.

Value	Description
TCPWM_CC_MATCH_SET	Set on comparison match
TCPWM_CC_MATCH_CLEAR	Clear on comparison match
TCPWM_CC_MATCH_INVERT	Invert on comparison match
TCPWM_CC_MATCH_NO_CHANGE	No change on comparison match
TCPWM_OVERLOW_SET	Set on overflow
TCPWM_OVERLOW_CLEAR	Clear on overflow
TCPWM_OVERLOW_INVERT	Invert on overflow
TCPWM_OVERLOW_NO_CHANGE	No change on overflow
TCPWM_UNDERFLOW_SET	Set on underflow
TCPWM_UNDERFLOW_CLEAR	Clear on underflow
TCPWM_UNDERFLOW_INVERT	Invert on underflow
TCPWM_UNDERFLOW_NO_CHANGE	No change on underflow
TCPWM_PWM_MODE_LEFT	Setting for left aligned PWM. Should be combined with up counting mode
TCPWM_PWM_MODE_RIGHT	Setting for right aligned PWM. Should be combined with down counting mode
TCPWM_PWM_MODE_CENTER	Setting for center aligned PWM. Should be combined with up/down 0 mode.
TCPWM_PWM_MODE_ASYM	Setting for asymmetric PWM. Should be combined with up/down 1 mode.

Return Value: None

Side Effects: None

void TCPWM_SetOneShot(uint32 oneShotEnable)

Description: Writes the register that controls whether the TCPWM runs continuously or stops after the TC is reached. By default, the TCPWM operates in continuous mode. It is applicable to Timer/Counter or Pseudo Random PWM.

Parameters: uint32 oneShotEnable: 0 = Continuous; 1 = Enable One Shot.

Return Value: None

Side Effects: None

void TCPWM_SetPWMSyncKill(uint32 syncKillEnable)

Description: Writes the register that controls whether the PWM kill signal (stop input) causes an asynchronous or synchronous kill operation. For Synchronous mode, a synchronous Kill kills the output as long as the Kill signal is high. Once the Kill signal goes to low again, the PWM output will not be re-enabled until the TC event occurs. This mode requires rising edge mode for the stop input. For Asynchronous mode the kill signal disables both the line and line_n signals when the kill signal is present. This mode should only be used when the kill signal (stop input) is configured in pass through mode (Level sensitive signal). By default the kill operation is asynchronous. This functionality is only applicable to PWM and PWM with dead time modes.

Parameters: uint32 syncKillEnable: 0 = Asynchronous; 1 = Synchronous.

Return Value: None

Side Effects: None

void TCPWM_SetPWMStopOnKill(uint32 stopOnKillEnable)

Description: Writes the register that controls whether the PWM kill signal (stop input) causes the PWM counter to stop. By default the kill operation does not stop the counter. This functionality is only applicable to the three PWM modes

Parameters: uint32 stopOnKillEnable: 0 = Don't stop; 1 = Stop.

Return Value: None

Side Effects: None

void TCPWM_SetPWMDeadTime(uint32 deadTime)

Description: Writes the dead time control value. This value delays the rising edge of both the line and line_n signals for the Direct signals mode and delays the falling edge of both the line and line_n signals for the Inverse signals mode by the designated number of clock cycles. This results in both signals being inactive for that many cycles. This functionality is only applicable to the PWM in dead time mode.

Parameters: uint32 deadTime: Dead time to insert. Range from 0 to 255.

Return Value: None

Side Effects: None

void TCPWM_SetPWMInvert(uint32 mask)

Description: Writes the bits that control whether the line and line_n outputs are inverted from their normal output values. This functionality is only applicable to the three PWM modes

Parameters: uint32 mask: Mask of outputs to invert. Outputs not included in the mask are set to normal non-inverted operation.

Value	Description
TCPWM_INVERT_LINE	Inverts the line output
TCPWM_INVERT_LINE_N	Inverts the line_n output

Return Value: None

Side Effects: None

void TCPWM_SetQDMode(uint32 qdMode)

Description: Sets the Quadrature Decoder to one of 3 supported modes. This functionality is only applicable to Quadrature Decoder operation.

Parameters: uint32 qdMode: Quadrature Decoder mode.

Value	Description
TCPWM_MODE_X1	Counts on phi A rising
TCPWM_MODE_X2	Counts on both edges of phiA (2x faster)
TCPWM_MODE_X4	Counts on both edges of phiA and phiB (4x faster)

Return Value: None

Side Effects: None

void TCPWM_SetInterruptMode(uint32 interruptMask)

Description: Sets the interrupt mask to control which interrupt requests generate the output interrupt signal

Parameters: uint32 interruptMask: Mask of bits to be enabled.

Value	Description
TCPWM_INTR_MASK_TC	Terminal count mask
TCPWM_INTR_MASK_CC_MATCH	Compare count / capture mask

Return Value: None

Side Effects: None

uint32 TCPWM_GetInterruptSourceMasked(void)

Description: Gets the interrupt requests masked by the interrupt mask.

Parameters: None

Return Value: uint32 Masked interrupt source.

Value	Description
TCPWM_INTR_MASK_TC	Terminal count mask
TCPWM_INTR_MASK_CC_MATCH	Compare count / capture mask

Side Effects: None

uint32 TCPWM_GetInterruptSource(void)

Description: Gets the interrupt requests (even without masking). This API returns all occurred interrupt requests in the component.

Parameters: None

Return Value: uint32 Interrupt request value.

Value	Description
TCPWM_INTR_MASK_TC	Terminal count mask
TCPWM_INTR_MASK_CC_MATCH	Compare count / capture mask

Side Effects: None

void TCPWM_ClearInterrupt(uint32 interruptMask)

Description: Clears the interrupt request.

Parameters: uint32 interruptMask: Mask of interrupts to clear.

Value	Description
TCPWM_INTR_MASK_TC	Terminal count mask
TCPWM_INTR_MASK_CC_MATCH	Compare count / capture mask

Return Value: None

Side Effects: None

void TCPWM_SetInterrupt(uint32 interruptMask)

Description: Sets a software interrupt request.

Parameters: uint32 interruptMask: Mask of interrupts to set.

Value	Description
TCPWM_INTR_MASK_TC	Terminal count mask
TCPWM_INTR_MASK_CC_MATCH	Compare count / capture mask

Return Value: None

Side Effects: None

void TCPWM_WriteCounter(uint32 count)

Description: Writes a new 16-bit counter value directly into the counter register, thus setting the counter (not the period) to the value written. For reliable operation, the counter should first be stopped before entering a new counter value.

Parameters: uint32 count: value to write.

Return Value: None

Side Effects: None

uint32 TCPWM_ReadCounter(void)

Description: Reads the current counter value.

Parameters: None

Return Value: uint32 Current counter value.

Side Effects: None

void TCPWM_SetCounterMode(uint32 counterMode)

Description: Sets the counter mode. It is applicable to all modes except Quadrature Decoder and PWM with pseudo random output.

Parameters: uint32 counterMode: Enumerated counter type values.

Value	Description
TCPWM_COUNT_UP	Counts up
TCPWM_COUNT_DOWN	Counts down
TCPWM_COUNT_UPDOWN0	Counts up and down. TC generated when counter reaches 0
TCPWM_COUNT_UPDOWN1	Counts up and down. TC generated both when counter reaches 0 and period

Return Value: None

Side Effects: None

void TCPWM_SetPeriodSwap(uint32 swapEnable)

Description: Writes the register that controls whether the period registers are swapped. When enabled in PWM mode the swap occurs at the next TC event following a hardware switch event. Not applicable in Quadrature Decoder and Timer/Counter modes.

Parameters: uint32 swapEnable: 0 = Disable swap; 1 = Enable swap.

Return Value: None

Side Effects: None

void TCPWM_SetCompareSwap(uint32 swapEnable)

Description: Writes the register that controls whether the compare registers are swapped. When enabled in Timer/Counter mode (without capture) the swap occurs at a TC event. In PWM mode the swap occurs at the next TC event following a hardware switch event. Not applicable for Timer/Counter with Capture or in Quadrature Decoder modes.

Parameters: uint32 swapEnable: 0 = Disable swap; 1 = Enable swap.

Return Value: None

Side Effects: None

uint32 TCPWM_ReadCapture(void)

Description: Reads the captured counter value. This API is applicable only for Timer/Counter with capture mode and Quadrature Decoder modes.

Parameters: None

Return Value: uint32 Captured value.

Side Effects: None

uint32 TCPWM_ReadCaptureBuf(void)

Description: This API reads the capture buffer register. It should be used to read the capture value in the capture register when the component is configured as a Timer/Counter in capture mode, or as a Quadrature Decoder.

Parameters: None

Return Value: uint32 Capture buffer value

Side Effects: None

void TCPWM_WritePeriod(uint32 period)

Description: Writes the 16-bit period register with the new period value. To cause the counter to count for N cycles this register should be written with N-1 (counts from 0 to period inclusive). This API is not applicable for QuadratureDecoder mode.

Parameters: uint32 period: Period value.

Return Value: None

Side Effects: In PWM mode, when the **PWM align** parameter is set to Left align, the counter value increments from zero to the period value. After that, the counter resets to zero and starts to count again. However, when a new period value is set that is lower than the current counter value, the PWM will not function as expected. The counter will count up to 65535 before resetting to zero and continuing to operate normally. To avoid this situation, call TCPWM_WriteCounter(0) whenever the period is changed to a smaller value while running in Left align mode.

uint32 TCPWM_ReadPeriod(void)

Description: Reads the 16 bit period register.

Parameters: None

Return Value: uint32 Period value.

Side Effects: None

void TCPWM_WritePeriodBuf(uint32 periodBuf)

Description: Writes the 16 bit period buffer register with the new period value. This API is applicable for PWM modes.

Parameters: uint32 periodBuf: Period value.

Return Value: None

Side Effects: None

uint32 TCPWM_ReadPeriodBuf(void)

Description: Reads the 16 bit period buffer register. This API is applicable for PWM modes.

Parameters: None

Return Value: uint32 Period buffer value.

Side Effects: None

void TCPWM_WriteCompare(uint32 compare)

Description: Writes the 16 bit compare register with the new compare value. Not applicable for Timer/Counter with Capture or in Quadrature Decoder modes.

Parameters: uint32 compare: Compare value.

Return Value: None

Side Effects: PSoC 4000 devices write the 16 bit compare register with the decremented compare value in the Up counting mode, and the incremented compare value in the Down counting mode.

uint32 TCPWM_ReadCompare(void)

Description: Reads the compare register. Not applicable for Timer/Counter with Capture or in Quadrature Decoder modes.

Parameters: None

Return Value: uint32 Compare value.

Side Effects: PSoC 4000 devices read the incremented compare register value in the Up counting mode, and the decremented value in the Down counting mode.

void TCPWM_WriteCompareBuf(uint32 compareBuf)

Description: Writes the 16 bit compare buffer register with the new compare value. Not applicable for Timer/Counter with Capture or in Quadrature Decoder modes.

Parameters: uint32 compareBuf: Compare value.

Return Value: None

Side Effects: PSoC 4000 devices write the 16 bit compare buffer register with the decremented compare value in the Up counting mode, and the incremented compare value in the Down counting mode.

uint32 TCPWM_ReadCompareBuf(void)

Description: Reads the compare buffer register. Not applicable for Timer/Counter with Capture or in Quadrature Decoder modes.

Parameters: None

Return Value: uint32 Compare buffer value.

Side Effects: For PSoC 4000 devices read the incremented compare buffer register value in the Up counting mode, and the decremented value in the Down counting mode.

void TCPWM_SetCaptureMode(uint32 triggerMode)

Description: Sets the capture trigger mode. For PWM mode this is the switch input. This input is not applicable to the Timer/Counter without Capture. For PWM modes this is the switch input.

Parameters: uint32 triggerMode: Enumerated trigger mode value.

Value	Description
TCPWM_TRIG_LEVEL	Level
TCPWM_TRIG_RISING	Rising edge
TCPWM_TRIG_FALLING	Falling edge
TCPWM_TRIG_BOTH	Both rising and falling edge

Return Value: None

Side Effects: None

void TCPWM_SetReloadMode(uint32 triggerMode)

Description: Sets the reload trigger mode. For Quadrature Decoder mode this is the index input.

Parameters: uint32 triggerMode: Enumerated trigger mode value.

Value	Description
TCPWM_TRIG_LEVEL	Level
TCPWM_TRIG_RISING	Rising edge
TCPWM_TRIG_FALLING	Falling edge
TCPWM_TRIG_BOTH	Both rising and falling edge

Return Value: None

Side Effects: None

void TCPWM_SetStartMode(uint32 triggerMode)

Description: Sets the start trigger mode. For Quadrature Decoder mode this is the phiB input.

Parameters: uint32 triggerMode: Enumerated trigger mode value.

Value	Description
TCPWM_TRIG_LEVEL	Level
TCPWM_TRIG_RISING	Rising edge
TCPWM_TRIG_FALLING	Falling edge
TCPWM_TRIG_BOTH	Both rising and falling edge

Return Value: None

Side Effects: None

void TCPWM_SetStopMode(uint32 triggerMode)

Description: Sets the stop trigger mode. For PWM mode this is the kill input.

Parameters: uint32 triggerMode: Enumerated trigger mode value.

Value	Description
TCPWM_TRIG_LEVEL	Level
TCPWM_TRIG_RISING	Rising edge
TCPWM_TRIG_FALLING	Falling edge
TCPWM_TRIG_BOTH	Both rising and falling edge

Return Value: None

Side Effects: None

void TCPWM_SetCountMode(uint32 triggerMode)

Description: Sets the count trigger mode. For Quadrature Decoder mode this is the phiA input.

Parameters: uint32 triggerMode: Enumerated trigger mode value.

Value	Description
TCPWM_TRIG_LEVEL	Level
TCPWM_TRIG_RISING	Rising edge
TCPWM_TRIG_FALLING	Falling edge
TCPWM_TRIG_BOTH	Both rising and falling edge

Return Value: None

Side Effects: None

uint32 TCPWM_ReadStatus(void)

Description: Reads the status of the TCPWM.

Parameters: None

Return Value: uint32 Status.

Value	Description
STATUS_DOWN	Set if counting down
STATUS_RUNNING	Set if counter is running

Side Effects: None

void TCPWM_Sleep(void)

Description: Stops the component operation and saves the user configuration.

Parameters: None

Return Value: None

Side Effects: None

void TCPWM _Wakeup(void)

Description: Restores the user configuration and restores component state.

Parameters: None

Return Value: None

Side Effects: Calling the TCPWM_Wakeup() function without first calling the TCPWM_Sleep() function may produce unexpected behavior.

void TCPWM _SaveConfig(void)

Description: This function saves the component configuration and non-retention registers. This function is called by the TCPWM_Sleep() function.

Parameters: None

Return Value: None

Side Effects: None

void TCPWM _RestoreConfig(void)

Description: This function restores the component configuration and non-retention registers. This function is called by the TCPWM_Wakeup() function.

Parameters: None

Return Value: None

Side Effects: None

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined: project deviations – deviations that are applicable for all PSoC Creator components and specific deviations – deviations that are applicable only for this component. This section provides information on component specific deviations. The project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The TCPWM component does not have any specific deviations.

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples,

open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

Functional Description

Some event functionalities are redefined in certain modes:

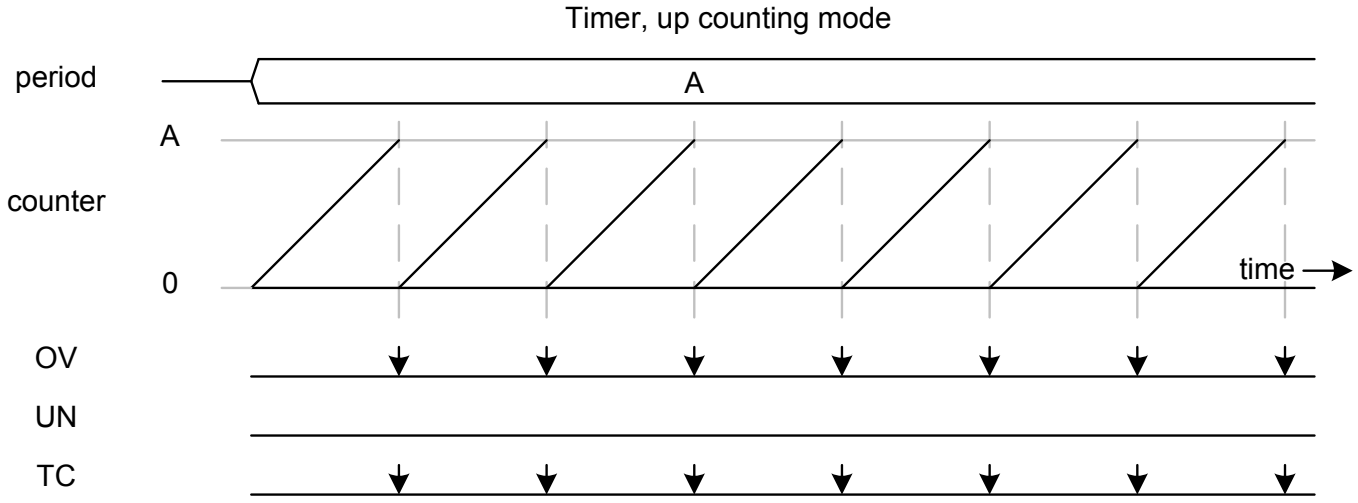
- In quadrature mode, the reload event acts as a quadrature index event. An index/reload indicates a completed rotation.
- In quadrature mode, the count event acts as quadrature phase phiA.
- In quadrature mode, the start event acts as quadrature phase phiB.
- In the PWM modes, the stop event acts as a kill event. A kill or stop event disables PWM output lines.
- In the PWM modes, the capture event acts as a switch event. It switches the values of the compare and buffered compare registers. You use this feature to modulate the pulse width.

Timer/Counter

The following three figures illustrate the timer behavior in the four counting modes: Up, Down, Up/Down 0, and Up/Down 1. The figures show the period register (contains the maximum counter value), the counter value with respect to time, and the times at which ‘ov’ and ‘un’ pins are triggered.

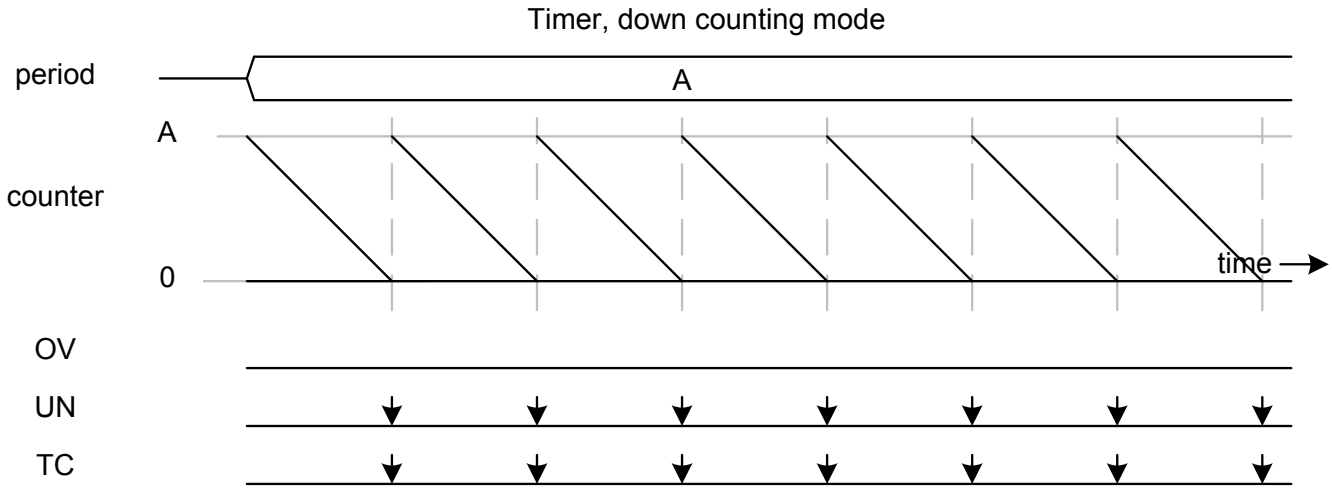
Up Mode

In the Up counting mode with a period register value of A, the counter starts from 0 and counts up to A resulting in A+1 counter values. The counter returns to 0 and starts to count back up to A in the next cycle. When the counter reaches this point, the overflow signal (ov) is triggered.



Down Mode

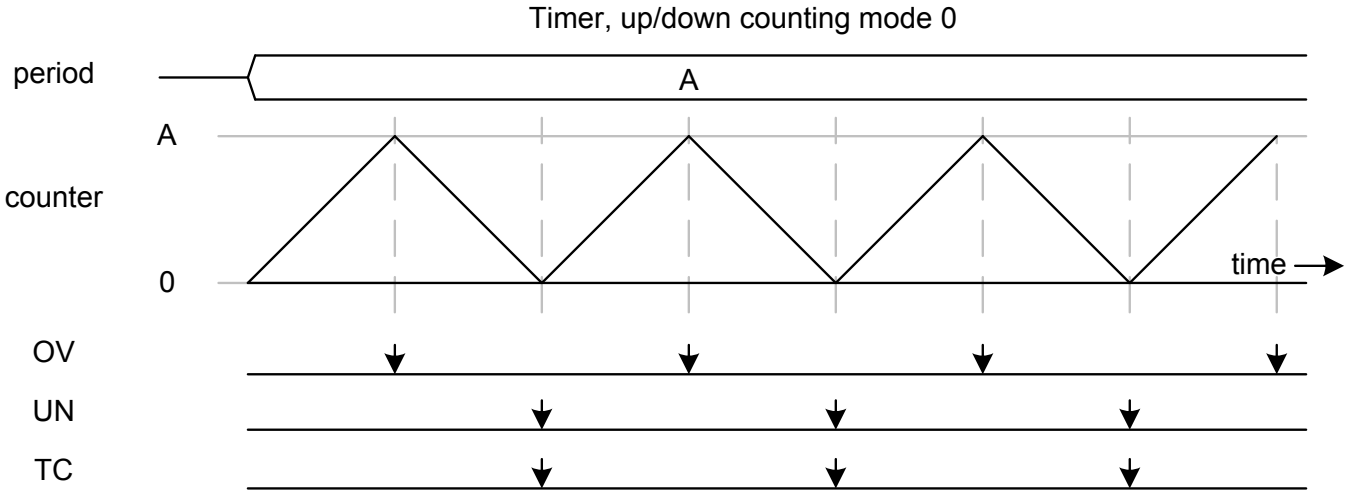
In the Down counting mode with a period register value of A, the counter begins from A and counts down to 0 for a total of A+1 counter values. The counter then resets back to A and triggers the underflow signal, ‘un’.

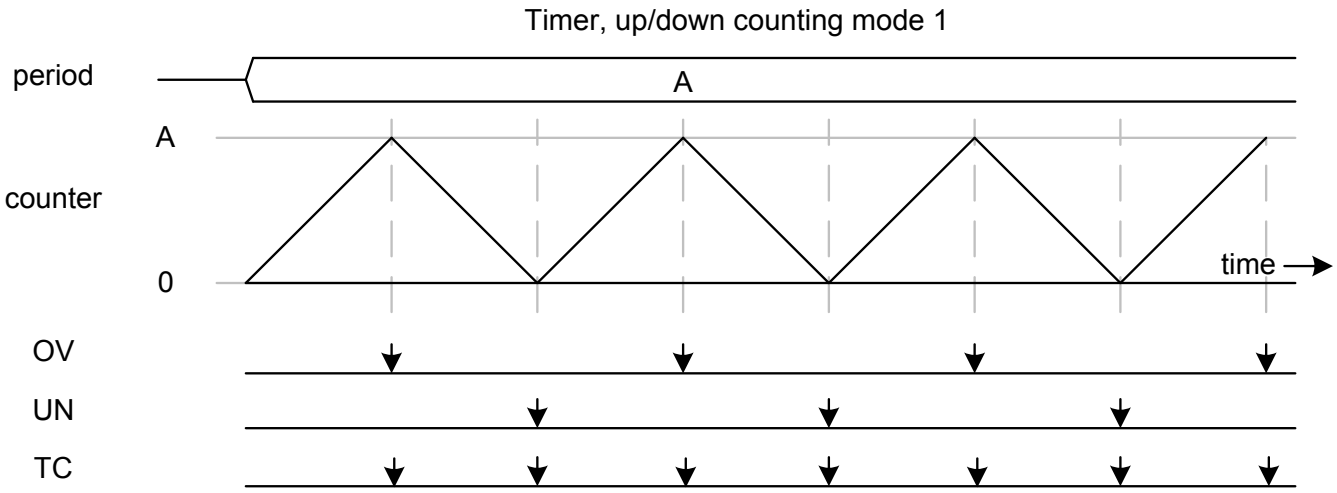


Up/Down Modes

The Up/Down counting modes 0 and 1 with a period register value of A begin counting from 0 up to A and then count down to 0. Therefore 2*A counter values are observed for a full cycle. The ‘ov’ signal is triggered when the counter reaches A, and the ‘un’ signal is triggered when the counter reaches 0.

In the up/down counting mode 0, a TC condition is generated when the counter reaches “0”; no TC condition is generated when the counter value reaches the period value. In the up/down counting mode 1, a TC condition is generated when the counter reaches “0” and when the counter value reaches the period value.

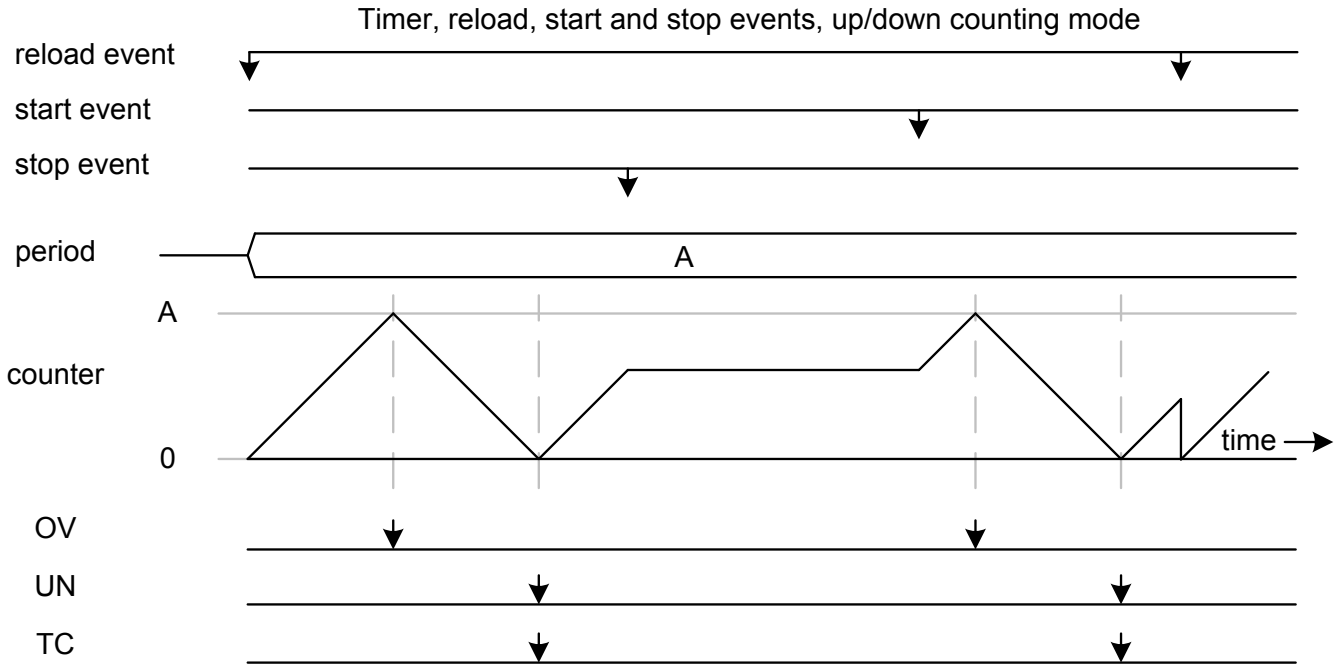




Operation

This first example illustrates how reload, start and stop events are used to control the counter. Consider a Timer in Up/Down counting mode 0. Asserting a reload event initializes the counter to

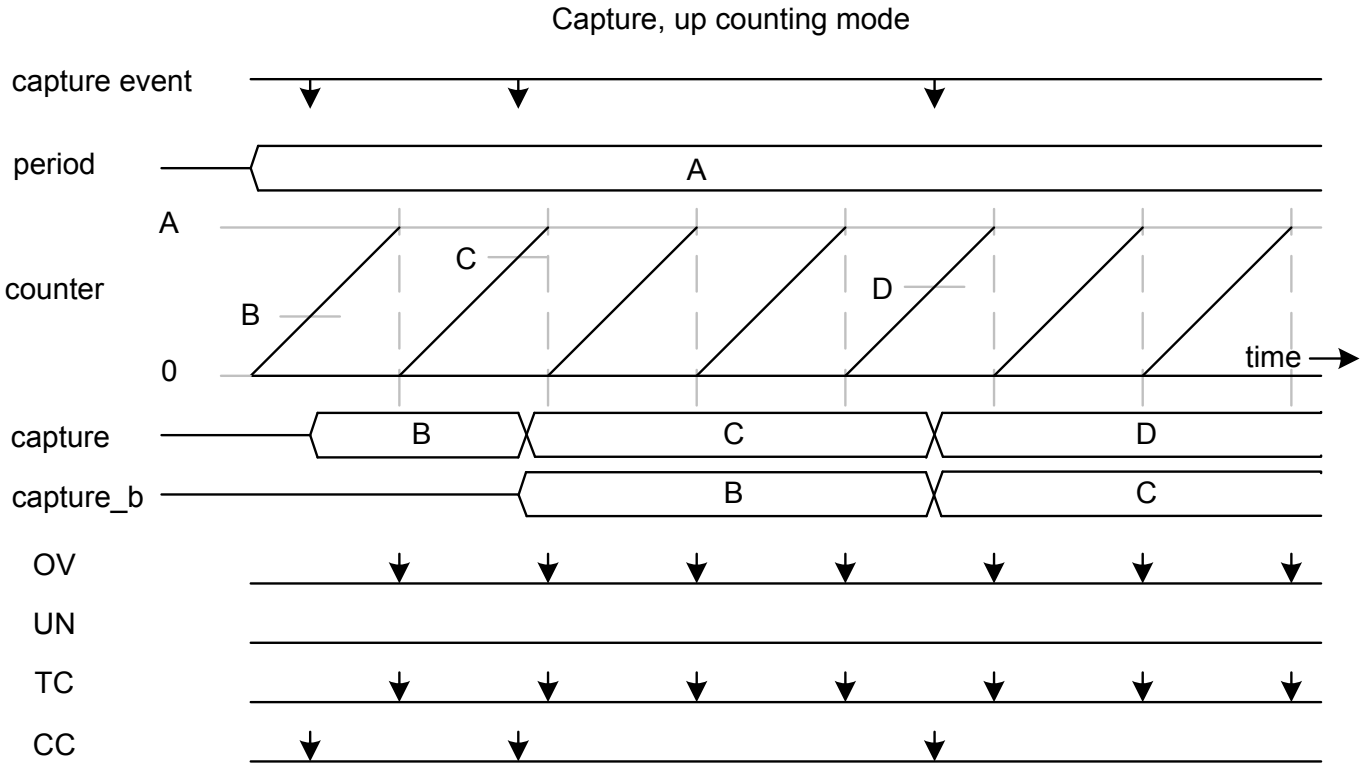
0 and starts the counter. A start event does not initialize the counter, but continues counting from the current counter's value. A stop event stops counting and leaves the current counter's value unaffected. A stop event has a higher priority than reload and start events. These operations are shown below in graphical form.



Capture

The second example illustrates the capture behavior of the Timer. In capture mode, the counter value can be captured at any time either through a firmware trigger (TCPWM_TriggerCommand() API) or a capture trigger input. This mode is used for period and pulse-width measurement.

The counter can be set to different modes: count up, down, up/down0, and up/down1. Consider a Timer configured in the Up counting mode. When a capture event occurs, the counter value is copied into the capture register. The capture value is copied to the buffered capture register (capture_b). A CC condition is generated when the counter value is captured. This condition can be used to generate an interrupt.

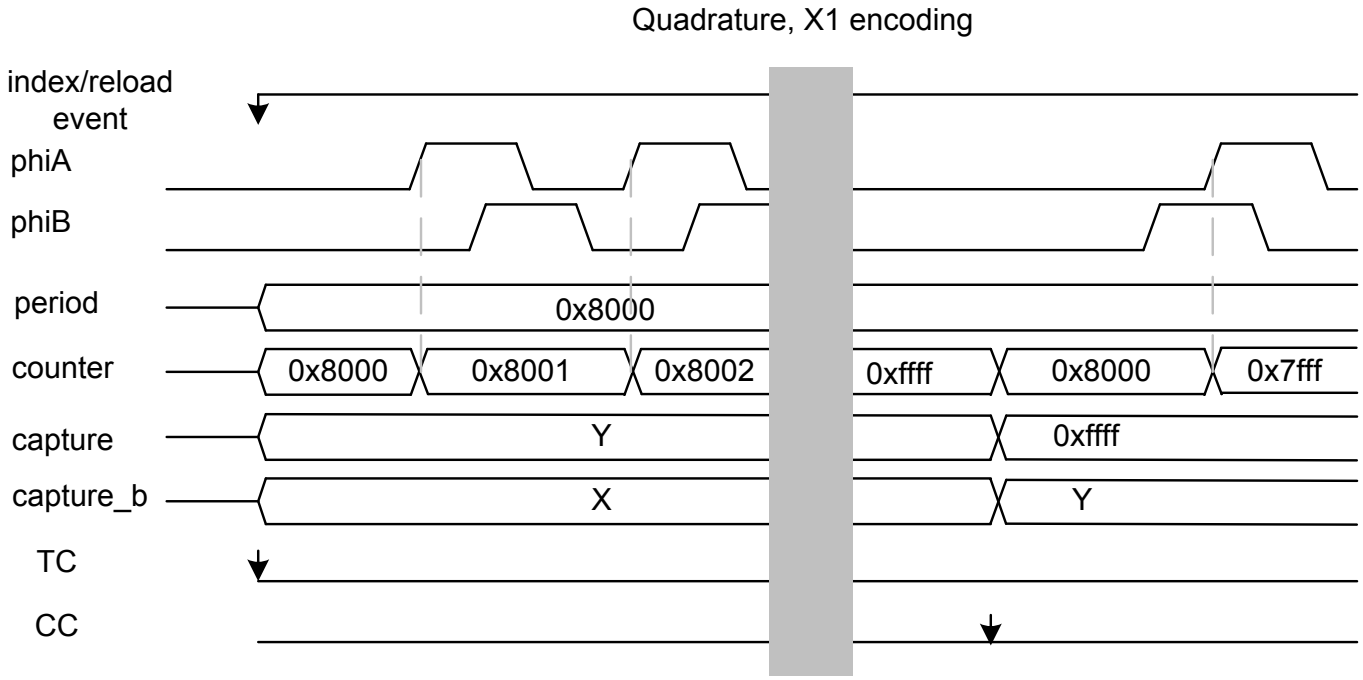


Incrementing and decrementing the counter is under the control of the count event and the counter clock enable signal. Typical operations use 1 as a count event and use the TCPWM clock as the counter clock without any clock pre-scaling. However advanced operations are possible using the count event configuration. For example, the count event can be configured to count the rising edges of a trigger input from the DSI.

Events affect the counter at the counter clock frequency. Events are detected on the counter frequency. When the event frequency exceeds the counter clock frequency, events may be lost. This is more likely to happen for high frequency events (in the counter frequency domain) and a timer configuration in which a pre-scaled counter clock is used.

Quadrature Decoder

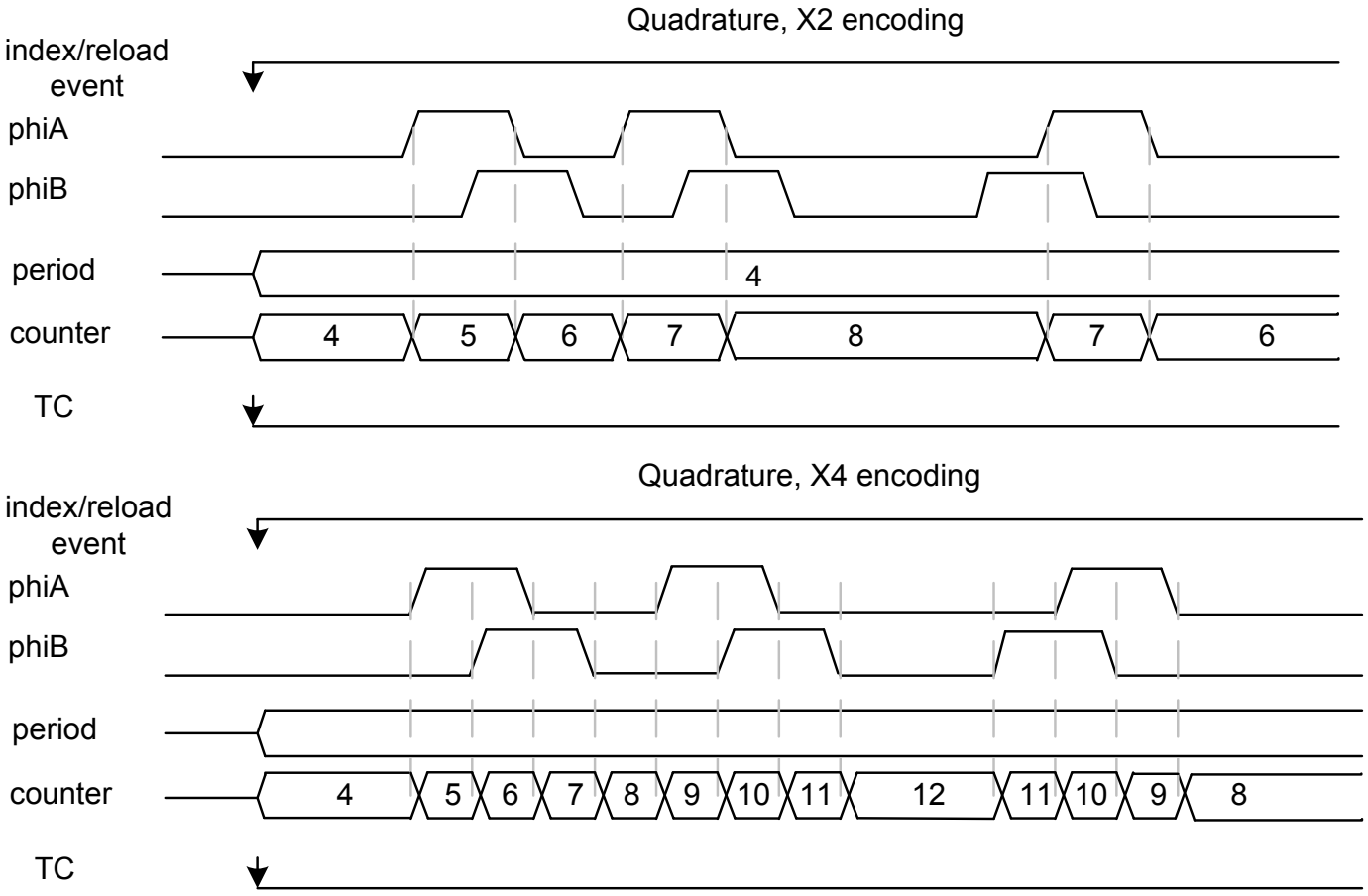
The following figure illustrates the behavior of the Quadrature Decoder in the 1x encoding mode: A positive edge on phiA increments the counter when phiB is 0 and decrements the counter when phiB is 1. The counter is initialized with the mid-point counter value “0x8000” on an index event. A TC condition is also generated when the counter is initialized. This condition can be used to generate an interrupt.



When the counter reaches “0xFFFF” (the maximum counter value), the counter value is copied to the capture register and the counter is initialized to “0x8000” (the mid-point counter value). A CC condition is generated when the counter value is copied. This condition can be used to generate an interrupt.

Some mechanical and low cost optical quadrature encoders may exhibit noise near the transition from one state to the next. This noise may cause the encoder to count invalid transitions and result in an encoder position error. To avoid this error, the encoder outputs should be filtered, either with a passive RC circuit or with the use of a digital filter. The Debouncer component that is available on PSoC devices with UDBs may be used to filter the encoder outputs. This should eliminate the noise on most low cost quadrature decoders.

The 2x and 4x Quadrature encoding modes count twice and four times as fast as the 1x Quadrature encoding mode. The following two figures illustrate the Quadrature Decoder operation in the 2x and 4x encoding modes. Note the higher resolution achieved when using these modes.

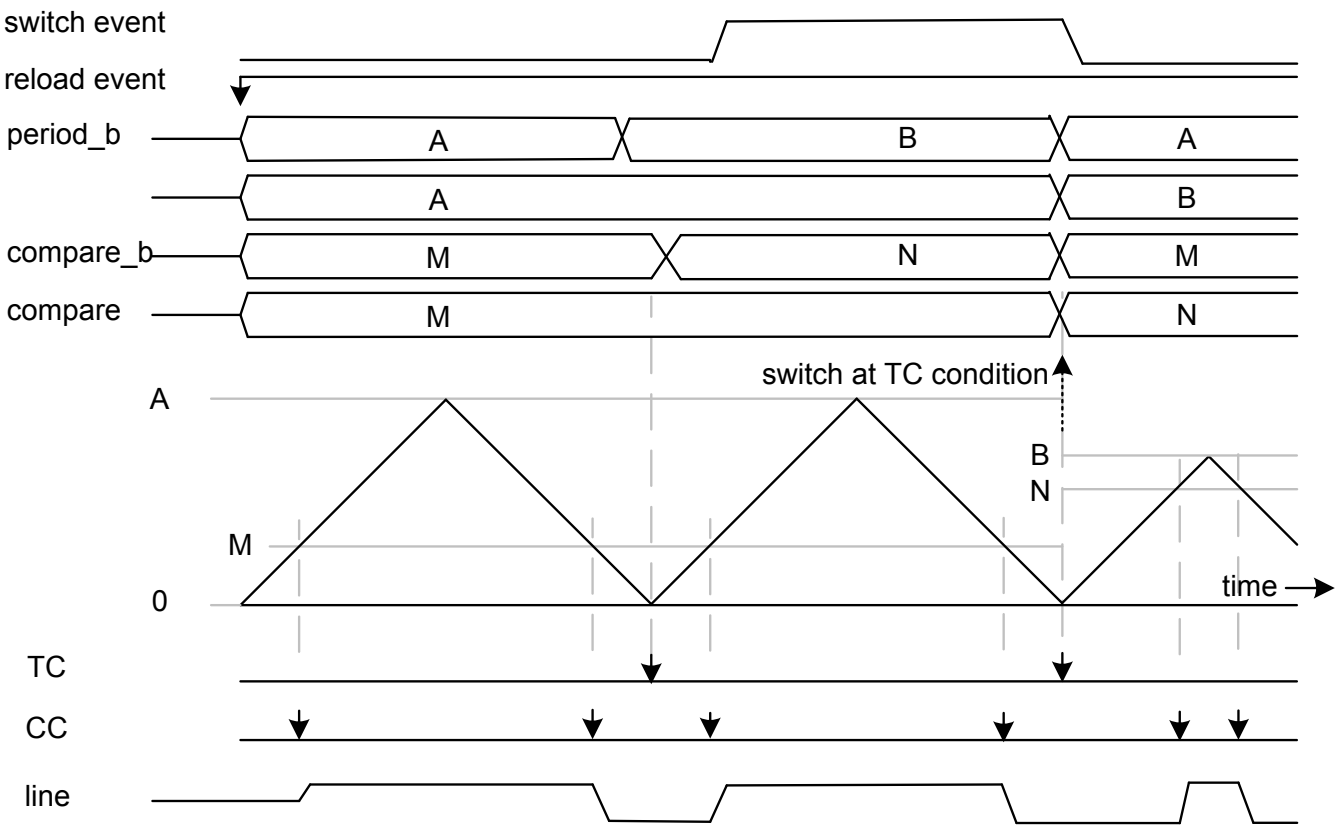


PWM

The mode of the PWM can be set to either PWM, PWM with dead time insertion, or Pseudo random PWM mode. The default setting is PWM mode that is left-aligned and gives direct output.

The PWM's period and compare registers can be swapped automatically on a TC event when the swap check box in the customizer is selected and a switch event is triggered. The switch event can be triggered with a hardware signal or by a software triggered switch event. In each case the actual swap occurs at a TC event. The following figure illustrates center aligned PWM with software generated switch events. Software generates a switch event only after both the period buffer and compare buffer registers are updated. As the updates of the second PWM pulse come late (after the TC condition), the first PWM pulse is repeated. The switch event is automatically cleared by hardware at the TC condition after the swap takes effect.

PWM, center aligned, buffered (software switch event)

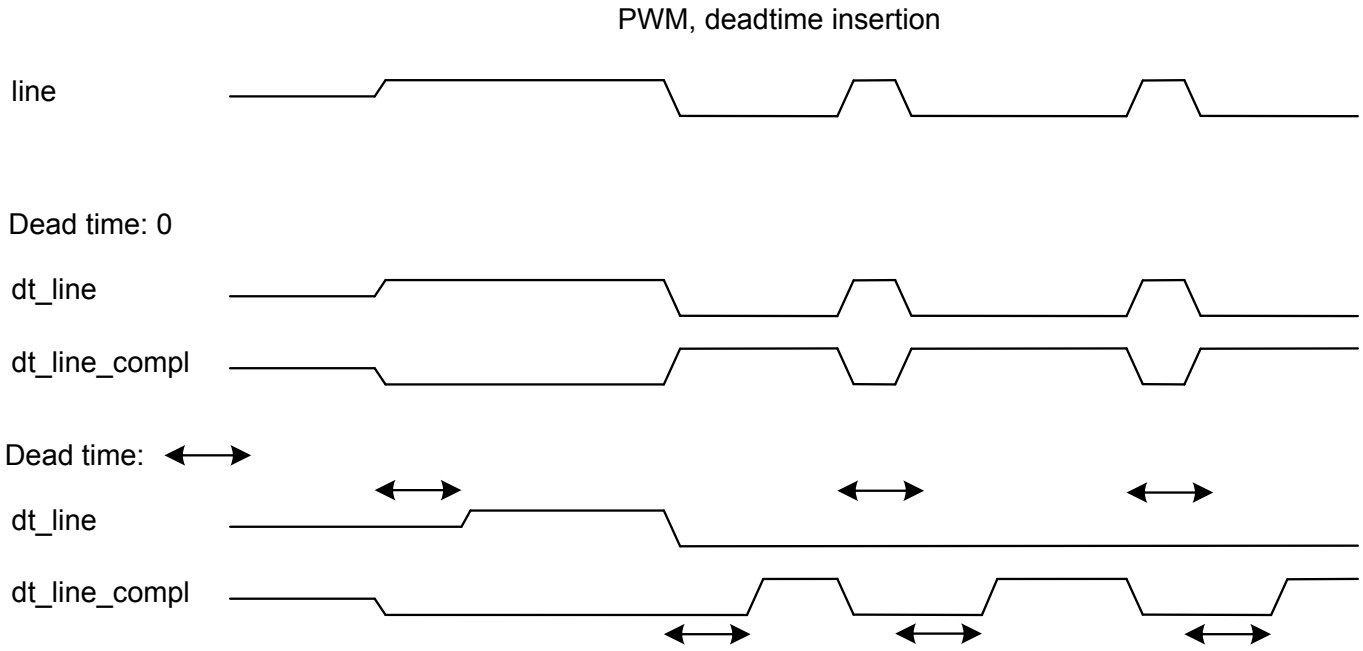


The switch/capture event functionality can be used to allow for DSI controlled modulation of the PWM high phase to create a PWM high phase accuracy that exceeds the accuracy of a single PWM period. This higher accuracy is achieved by modulating between two compare values over multiple PWM periods. The intended functionality is illustrated by the following example.

Modulation example: Assume a PWM period of 100 cycles. Both period and buffered period registers are set to 99 to achieve this 100 cycle PWM period. Without modulation, a compare value of 50 would generate a 50 cycle PWM high phase. The PWM period of 100 cycles allows for a pulse accuracy of 1%. To create a higher accuracy, with the same PWM period of 100 cycles, modulation between two compare values is supported. Within a PWM period, the accuracy is still 1%. However, over multiple PWM periods, a larger accuracy can be achieved. To achieve a 50.5 cycle PWM high phase (0.5% accuracy), the compare register is set to 50 and the buffered compare register is set to 51. At a switch/capture event (possibly generated by the DSI) the compare and buffered compare values are switched. When both compare values are active an equal amount of times, a 50.5 cycle PWM high phase is achieved over multiple 100 cycle PWM periods.

PWM with dead time insertion (PWM_DT)

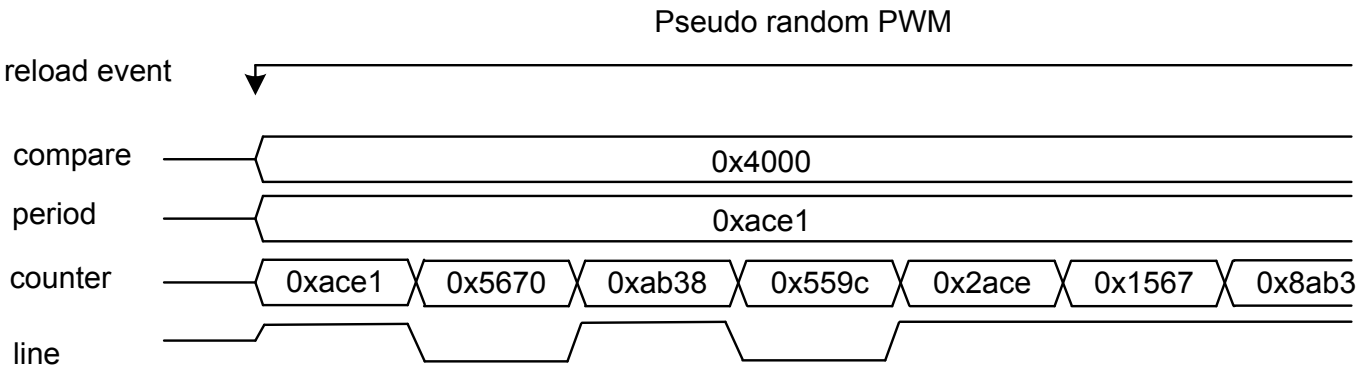
Dead time insertion mode allows the insertion of dead time into the PWM. The following figure illustrates how the complementary output lines “dt_line” and “dt_line_compl” are generated from the PWM output line, “line”. The top example shows the output lines with no dead time. The bottom example shows the output with a short dead time. For Direct output signal mode, the rising edges are delayed by the dead time, but the negative edges are not. For Inverse output signal mode, the negative edges are delayed by the dead time, but the rising edges are not.



To implement a “kill” (the ability to disable both output lines immediately) based on the value of a DSI input signal, the value of the specified DSI input signal is chosen as the stop event. No edge detection is performed on the trigger signal, as it is in pass through mode. The generated stop event is used to disable both output lines.

PWM Pseudorandom Mode (PWM_PR)

The Pseudorandom mode allows the output line to toggle when the pseudorandom value of the counter passes the value specified in the compare register. The counter value changes in a pseudo random sequence. The lower 15 bits of the counter value are compared against the compare register to determine the line value. The line is active when the lower 15 bits of the counter are less than the compare value and inactive otherwise. The following figure illustrates the pseudo random noise behavior with a compare value of 0x4000, resulting in roughly 50% duty cycle. This is possible since only the lower 15 bits of the 16 bits of the counter are used to compare with the compare register value.



Registers

See the chip Technical Reference Manual (TRM) for more information about registers.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. This table shows the memory usage for all APIs available in a given component configuration.

The measurements were done with the associated compiler configured in release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration	PSoC 4 (GCC)			
	PSoC 4000		PSoC 4100/PSoC 4200	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Unconfigured	1144	2	1036	2
Timer / Counter	752	2	752	2
Quadrature Decoder	572	2	572	2
PWM	1128	2	1020	2

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

TCPWM DC Specifications PSoC 4000

Parameter	Description	Min	Typ	Max	Units	Conditions
I_{TCPWM1}	Block current consumption at 3 MHz	–	–	45	µA	All modes (Timer/Counter/PWM)
I_{TCPWM2}	Block current consumption at 8 MHz	–	–	145	µA	All modes (Timer/Counter/PWM)
I_{TCPWM3}	Block current consumption at 16 MHz	–	–	160	µA	All modes (Timer/Counter/PWM)

TCPWM DC Specifications PSoC 4100 / PSoC 4200

Parameter	Description	Min	Typ	Max	Units	Conditions
I_{TCPWM1}	Block current consumption at 3 MHz	–	–	19	µA	All modes (Timer/Counter/PWM)
I_{TCPWM2}	Block current consumption at 12 MHz	–	–	66	µA	All modes (Timer/Counter/PWM)
I_{TCPWM3}	Block current consumption at 48 MHz	–	–	285	µA	All modes (Timer/Counter/PWM)

TCPWM AC Specifications

Parameter	Description	Min	Typ	Max	Units	Conditions
$T_{TCPWMFREQ}$	Operating frequency	–	–	F_C	MHz	F_C max = F_{CPU}.
$T_{TCPWMENEXT}$	Input Trigger Pulse Width for all Trigger Events	$2/F_C$	–	–	ns	Trigger Events can be Stop, Start, Reload, Count, Capture, or Kill depending on which mode of operation is selected.
$T_{TCPWMEXT}$	Output Trigger Pulse widths	$2/F_C$	–	–	ns	Minimum possible width of Overflow, Underflow, and CC (Counter equals Compare value) trigger outputs.
T_{CRES}	Counter resolution	$1/F_C$	–	–	ns	Minimum time between successive counts.
T_{PWMRES}	PWM resolution	$1/F_C$	–	–	ns	Minimum pulse width of PWM Output
T_{QRES}	Quadrature inputs resolution	$1/F_C$	–	–	ns	Minimum pulse width between Quadrature phase inputs.

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.10	Added support for PSoC 4000 devices. Removed interrupt generation when the Down counter or the right-aligned PWM is started.	Clarified description for input and output signals. Clarified functional description. Added side effect description for the Compare like APIs. Clarified Kill signal behavior.
1.0.c	Datasheet edits.	Clarified description for the TCPWM_ReadCaptureBuf API. Clarified description for the TCPWM_WriteCounter API. Clarified description for the interrupt APIs and interrupt terminal.
1.0.b	Datasheet edits.	Added a footnote for Output signals in the Input/Output Connections section. Added the TCPWM tab description to the Component Parameters section.
1.0.a	Datasheet edits.	Updated the description of the PWM Dead Time functionality for Direct and Inverse output signal modes. Clarified the behavior of TCPWM_WritePeriod() function.
1.0	New Component	

© Cypress Semiconductor Corporation, 2014. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC® Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A HALFICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.