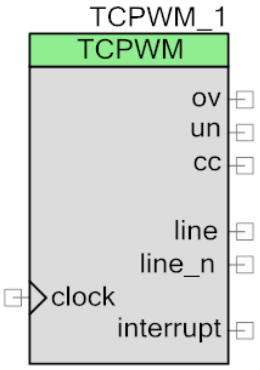
PSoC 4 Timer Counter Pulse Width Modulator (TCPWM)

2.10

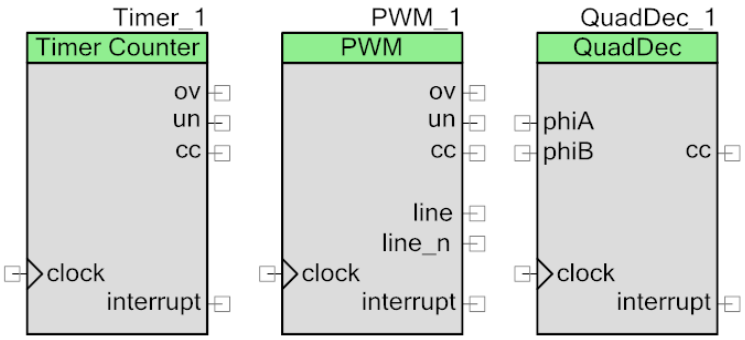


Features

- 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 all of them (except count) can also be generated by software (TCPWM_TriggerCommand() API).
- 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 (clock cycles for PSoC 4000/PSoC 4100/PSoC 4200) 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[1]	Timer, PWM	This input allows a reload event to initialize and start the counter. In up and up/down counting modes, the counter is initialized with “0” (for PSoC 4000, PSoC 4100, PSoC 4200 devices). For other devices, the counter is initialized with “1” for up/down counting modes. 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. For all devices, except PSoC 4000, PSoC 4100, PSoC 4200, it should only be used when the counter is not running.
index [1]	QuadDec	The index input detects a reference position for the Quadrature Decoder. An event on this signal generates a terminal count (TC [2]) and CC events, and initializes the counter with the mid-point counter value “0x8000”. If the Index terminal is present, then the counter will reload based on this signal, otherwise the TCPWM_TriggerCommand() with reload event will be used during call the TCPWM_Enable() or TCPWM_Start() APIs.
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 [1]	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, otherwise the TCPWM_TriggerCommand() with start event will be used during call the TCPWM_Enable() or TCPWM_Start() APIs. Should only be used when the counter is not running.
phiB [1]	QuadDec	One of the two counting inputs that control the count value, increment, and decrement, depending on their relationship and the mode.
stop [1]	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.

1. For all devices, except PSoC 4000, PSoC 4100, PSoC 4200, the input event will take effect in the next counter clock when the count event becomes active (it depends on the edge detection mode of count event). In the other words, besides for counter, other HW writable register, interrupt and output will not change state when count event is low. For example, if you are using an external count signal and an external start signal, the first Active count signal won't be counted; it will be used to start the counter, so your counts will be off by one.
2. A tc (terminal count) event is only an internal signal that can only be used for triggering an ISR. It is the logical OR of the underflow and overflow events. In Quadrature mode, index will generate tc and CC events. For all devices, except PSoC 4000, PSoC 4100, PSoC 4200, the exception is that a reload event will generate underflow or overflow, but not generate a tc event.

Input	Mode	Description
capture [1]	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 [1]	PWM	This signal swaps the period (period, periodBuf) and/or compare (compare, compareBuf) registers at the next TC [2] event (this swap depends on the GUI settings or APIs swap parameters). A switch event requires rising, falling, or either edge mode. Level mode is not supported. Note: period and periodBuf are swaps on second TC [2] for PSoC 4100/PSoC 4200 in the right align PWM mode.

Note All inputs are double synchronized in the TCPWM. The synchronizer is run at HFCLK speed. After that (just for PSoC 4000, PSoC 4100, PSoC 4200, (Timer/Counter, PWM modes)), 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. For PWM and PWM_DT modes, waveforms reflect line, line_n outputs behavior before "start" event ("reload" (if it is checked) for all devices, except PSoC 4000, PSoC 4100, PSoC 4200,) during counting, and after "stop" event.

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. An overflow event indicates that in up counting mode the COUNTER has changed from a state where it equaled PERIOD, to a state where it does not equal PERIOD. For all devices, except PSoC 4000, PSoC 4100, PSoC 4200, Reload will also generate overflow event in up counting 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. An underflow event indicates that in a down counting mode the COUNTER has changed from a state where it equaled “0”, to a state where it does not equal “0”. For all devices, except PSoC 4000, PSoC 4100, PSoC 4200, Reload will also generate underflow event in down counting mode and up/down counting modes.

<table>
<tr><th>Output</th><th>Mode</th><th>Description</th></tr>
<tr><td>cc[3]</td><td>All</td><td>Comparison or capture output. Comparison behavior: A cc event indicates that the COUNTER register equals COMPARE register.
For all devices, except PSoC 4000, PSoC 4100, PSoC 4200, this event is either generated when the match is about to occur (COUNTER does not equal COMPARE, and is changed to COMPARE, in the up or down counting modes), or when the match is about to not occur (COUNTER equals COMPARE, and is changed to a different value, in the up/down counting modes).
The Comparison signal behavior on different devices:
<table>
<tr><th>Device/ Mode</th><th>Timer / Counter Up Mode</th><th>Timer / Counter Down Mode</th><th>Timer / Counter Up/Down 0 Mode</th><th>Timer / Counter Up/Down 1 Mode</th><th>PWM_PR</th><th>QuadDec</th></tr>
<tr><td>Other Devices</td><td colspan="5">Counter changes from a state in which Counter equals Compare.</td><td rowspan="2">Counter is running and (Counter = 0x0000 or Counter = 0xffff) or Index event</td></tr>
<tr><td>PSoC 4100, PSoC 4200</td><td colspan="5">Counter = Compare</td></tr>
</table>
<table>
<tr><th>Device/ Mode</th><th>PWM, PWM_DT Left align (Up Mode)</th><th>PWM, PWM_DT Right align (Down Mode)</th><th>PWM, PWM_DT Center align (Up/Down 0 Mode)</th><th>PWM, PWM_DT Asymmetric align (Up/Down 1 Mode)</th></tr>
<tr><td>Other Devices</td><td colspan="2">Counter changes to a state in which Counter equals Compare.</td><td colspan="2">Counter changes from a state in which Counter equals Compare.</td></tr>
<tr><td>PSoC 4100, PSoC 4200</td><td colspan="4">Counter = Compare</td></tr>
</table></td></tr>
<tr><td>line [3, 4]</td><td>PWM</td><td>PWM output value. This signal is generated by the cc, un and ov internal events only.
It depends on PWM align settings or TCPWM_SetPWMMode() API parameters.
For Pseudo random PWM mode line reflects: COUNTER[14:0] < COMPARE[15:0]. As a result, for COMPARE register greater or equal to 0x8000, “line” is always 0. The counter value COUNTER is changed based on the LFSR polynomial: x^16 + x^14 + x^13 + x^11 + 1. COUNTER is initialized with “1” in Init API.</td></tr>
<tr><td>line_n</td><td>PWM</td><td>Inverted PWM output value (during counting). In Dead Time Insertion mode, the line and line_n each have their rising edges delayed, resulting in a period where both are low.</td></tr>
</table>

3. For PSoC 4000/PSoC 4100/PSoC 4200 devices, the CC output toggles each clock edge in which the event is true. Thus if the event is true for multiple clock edges multiple pulses will be present on CC. CC is also used for toggling the line and line_n outputs, thus this can cause undesired toggling on those outputs.

4. The Output Line signal parameter determines how the "line" signal is formed. The “line” signal forming is depends on Output line signal parameter (it defines output signal default value). In all devices, except PSoC 4000, PSoC 4100, PSoC 4200, : For 100% PWM output duty cycle (Left align (Up counting mode)), the COMPARE register should be greater than “1” from PERIOD. For 0% PWM output duty cycle (Right align (Down counting mode)), the COMPARE register should be “0xFFFFu”. These modes are not applicable if PERIOD = 0xFFFFu.

Output	Mode	Description
ìnterrupt	All	An Interrupt can be triggered by any of the following sources: • The count has hit its TC [2]. • A hardware capture has been performed. • Compare signal has a rising edge. This signal goes high while any of the enabled interrupt sources are true. The “interrupt” output remains asserted until the software clears the interrupt source by writing a ‘1’ to the interrupt bit of the Interrupt request register using the TCPWM_ClearInterrupt() API. The component doesn’t have any buried ISRs. Use Derived Interrupt type of the ISR component. In order to receive subsequent interrupts, the interrupt should be cleared by using the TCPWM_ClearInterrupt() API in the interrupt handler of the ISR component.

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 other devices.

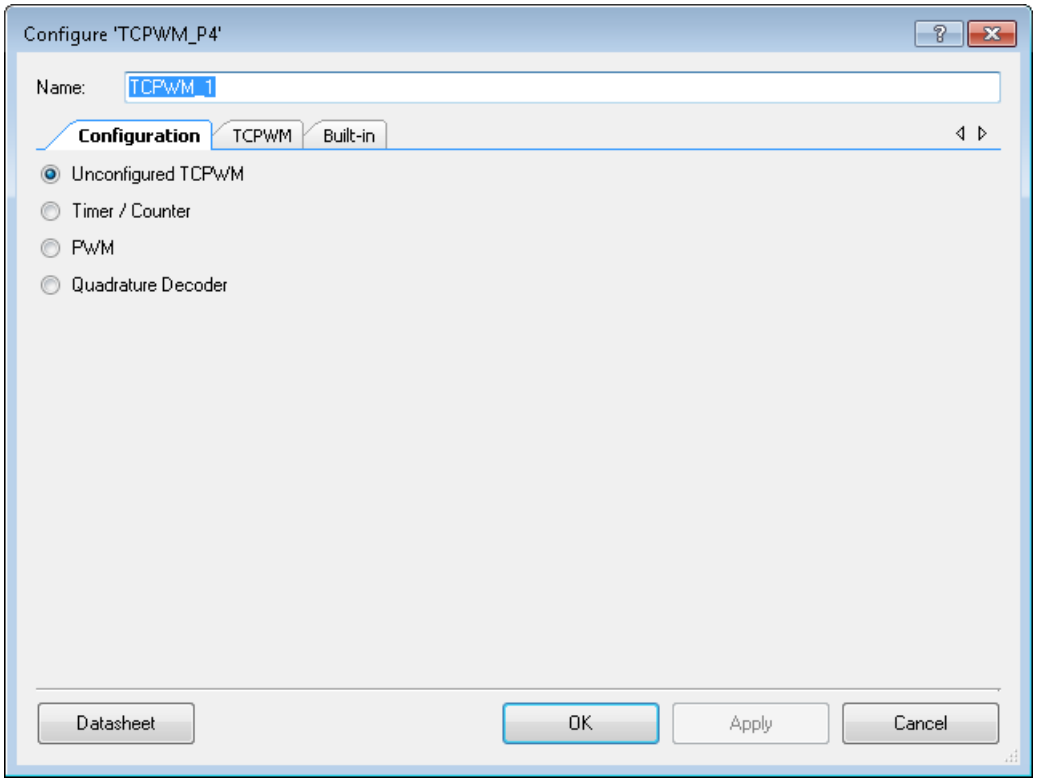
The actual HFCLK and SYSCLK frequencies can be observed in the Design-Wide Resources Clock Editor.

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.

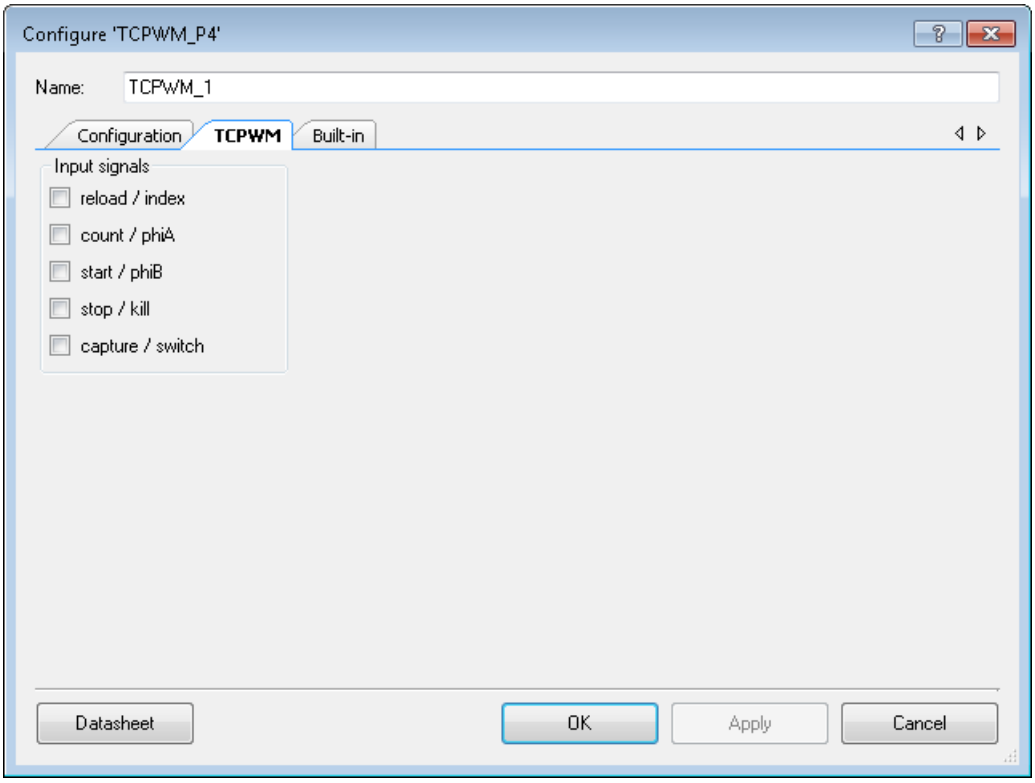
Configuration Tab



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.

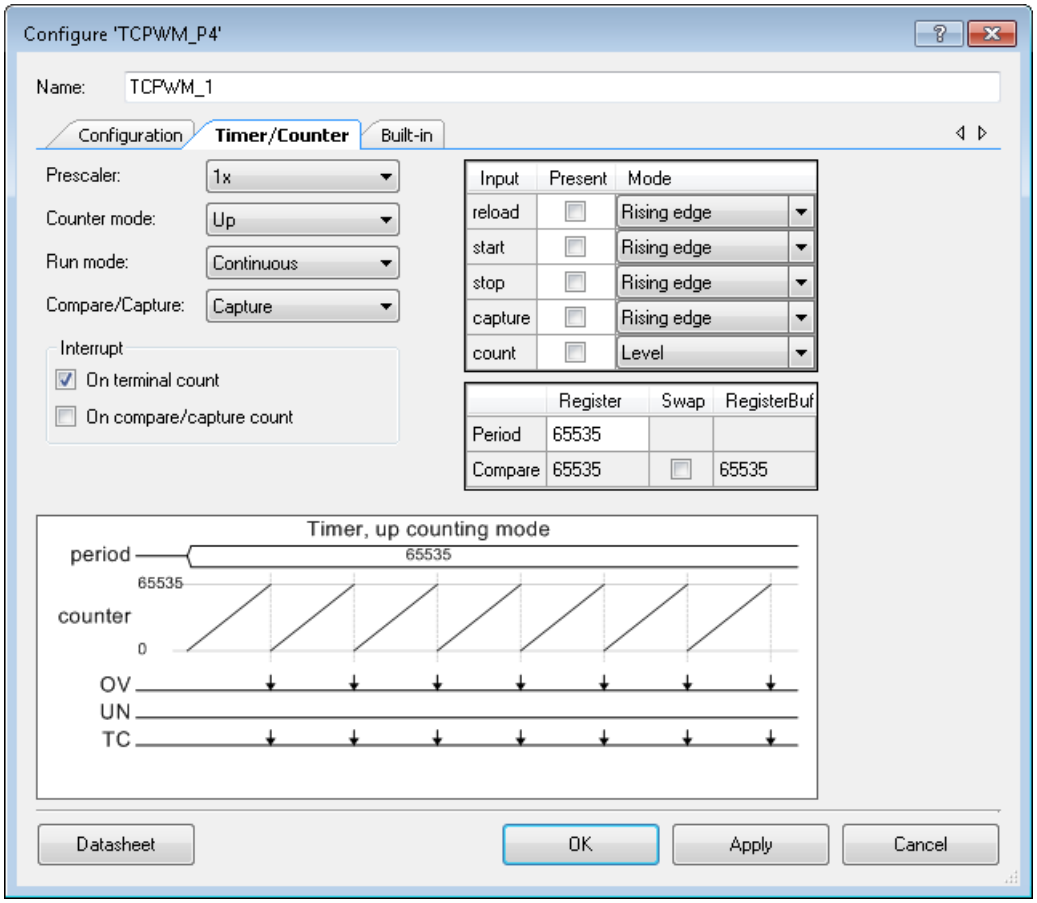
TCPWM Tab



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.

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.

Capture / Compare

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

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.

Note To cause the counter to count for N cycles, this register should be written with N-1 (counts from 0 to period inclusive).

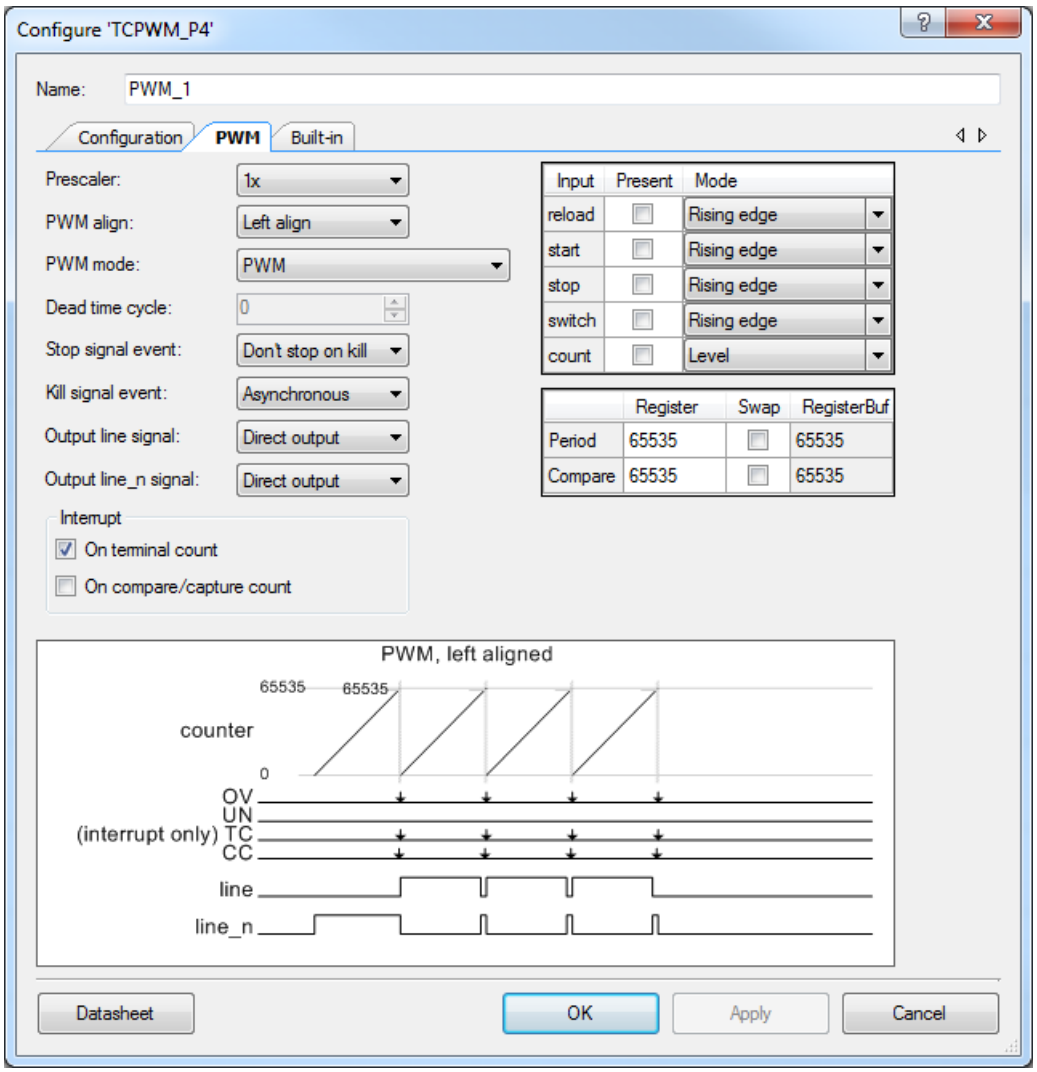
Compare

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 capture/compare 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.

Note For All devices, except PSoC 4000, PSoC 4100, PSoC 4200, in Asymmetric mode, ensure that the PERIOD and PERIOD_BUFF values are the same at a tc event that coincides with an overflow event.

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.

Note To cause the counter to count for N cycles, this register should be written with N-1 (counts from 0 to period inclusive).

The period and periodBuf are swaps on second TC for PSoC 4100/PSoC 4200 in the right align PWM mode.

The period value has an effect on tc signal generation in Pseudo random mode. If counter value = period value, the tc is generated.

Compare

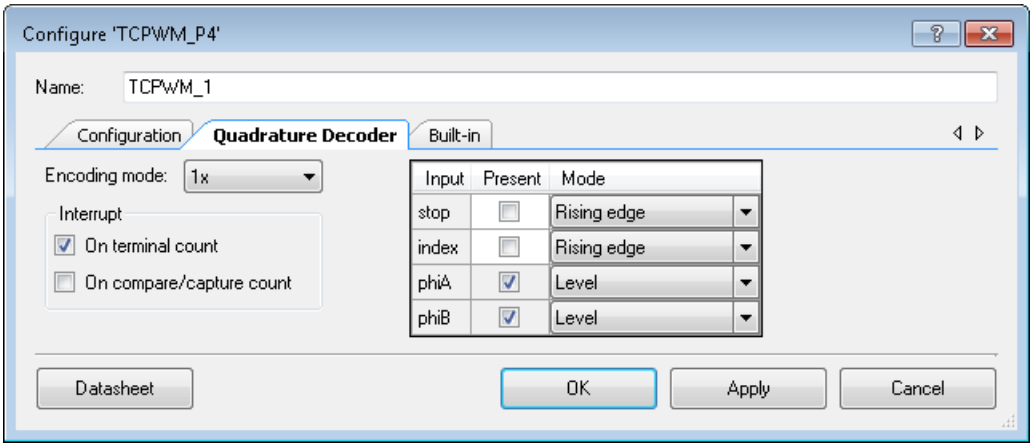
The **Compare** parameter sets the initial value for the comparison registers and the swap checkbox selects whether one or two comparison values are used. By default, the swap checkbox is unchecked. Note that the first compare value is always present and the second compare value (compareBuf) is present 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.

Note It is not recommended to use a value equal to "0" or equal to "period value" in Center or Asymmetric align modes on the PSoC 4000, PSoC 4100, and PSoC 4200 devices.

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 (changing the mode does not have any impact on these signals).

Interrupt

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

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
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

Function	Description
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
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.

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. If the TCPWM is configured in the Quadrature Decoder mode, the counter will be forced to start counting.

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. If the TCPWM is configured in the Quadrature Decoder mode, the counter will be forced to start counting.

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/Switch command
TCPWM_CMD_RELOAD	Trigger Reload/Index command
TCPWM_CMD_STOP	Trigger Stop/Kill command
TCPWM_CMD_START	Trigger Start/phiB 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 (ORed values). Should be combined with up counting mode: TCPWM_CC_MATCH_CLEAR, TCPWM_OVERLOW_SET, TCPWM_UNDERFLOW_NO_CHANGE.
TCPWM_PWM_MODE_RIGHT	Setting for right aligned PWM (ORed values). Should be combined with down counting mode TCPWM_CC_MATCH_SET, TCPWM_OVERLOW_NO_CHANGE, TCPWM_UNDERFLOW_CLEAR.
TCPWM_PWM_MODE_CENTER	Setting for center aligned PWM (ORed values): Should be combined with up/down 0 mode. PSoC 4000/PSoC 4100/PSoC 4200: TCPWM_CC_MATCH_INVERT, TCPWM_OVERLOW_NO_CHANGE, TCPWM_UNDERFLOW_CLEAR All devices, except PSoC 4000, PSoC 4100, PSoC 4200, : TCPWM_CC_MATCH_INVERT, TCPWM_OVERLOW_SET, TCPWM_UNDERFLOW_CLEAR.
TCPWM_PWM_MODE_ASYM	Setting for asymmetric PWM. (ORed values). Should be combined with up/down 1 mode. See Up/Down Modes section for differences in TC between up/down modes. TCPWM_CC_MATCH_INVERT, TCPWM_OVERLOW_SET, TCPWM_UNDERFLOW_CLEAR.

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 counter cycles (clock cycles for PSoC 4000/PSoC 4100/PSoC 4200). 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: The period and periodBuf are swaps on second TC for PSoC 4100/PSoC 4200 in the right align PWM mode.

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 compare/capture 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.

Note It is not recommended to use the value equal to "0" or equal to "period value" in Center or Asymmetric align PWM modes on the PSoC 4100/PSoC 4200 devices.

Note PSoC 4000 devices write the 16-bit compare register with the decremented compare value in the Up counting mode (except 0x0u), and the incremented compare value in the Down counting mode (except 0xFFFFu).

Parameters: uint32 compare: Compare value.

Return Value: None

Side Effects: None

uint32 TCPWM_ReadCompare(void)

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

Note PSoC 4000 devices read the incremented compare register value in the Up counting mode (except 0xFFFFu), and the decremented value in the Down counting mode (except 0x0u).

Parameters: None

Return Value: uint32 Compare value.

Side Effects: None

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.

Note It is not recommended to use the value equal to "0" or equal to "period value" in Center or Asymmetric align PWM modes on the PSoC 4100/PSoC 4200 devices.

Note PSoC 4000 devices write the 16 bit compare buffer register with the decremented compare value in the Up counting mode (except 0x0u), and the incremented compare value in the Down counting mode (except 0xFFFFu).

Parameters: uint32 compareBuf: Compare value.

Return Value: None

Side Effects: None

uint32 TCPWM_ReadCompareBuf(void)

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

Note For PSoC 4000 devices read the incremented compare buffer register value in the Up counting mode (except 0xFFFFu), and the decremented value in the Down counting mode (except 0x0u).

Parameters: None

Return Value: uint32 Compare buffer value.

Side Effects: None

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

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	+	+	+	+	+	+
SetPrescaler	+	+	+	-	+	-
TriggerCommand	+	+	+	+	+	+
SetOneShot	+	+	-	-	+	-

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

Function Name	Timer/Counter (Capture)	Timer/Counter (Compare)	PWM	PWM DT	PWM PR	Quad Dec
ReadStatus	+	+	+	+	+	+
Sleep	+	+	+	+	+	+
Wakeup	+	+	+	+	+	+
SaveConfig	+	+	+	+	+	+
RestoreConfig	+	+	+	+	+	+

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.

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.

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 4000 (GCC)		Other PSoC 4 devices (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Unconfigured	1208	2	1032	2
Timer / Counter	764	2	764	2

Configuration	PSoC 4000 (GCC)		Other PSoC 4 devices (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Quadrature Decoder	644	2	644	2
PWM	1204	2	1028	2

Functional Description [5]

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.

The capture/compare (‘cc’) condition is generated by the TCPWM when the counter is running and one of the following conditions occur:

- For PSoC 4100, PSoC 4200, the COUNTER value equals the COMPARE value.
- For All devices, except PSoC 4000, PSoC 4100, PSoC 4200, this event is either generated when the match is about to occur (COUNTER does not equal COMPARE, and is changed to COMPARE, in the up or down counting modes), or when the match is about

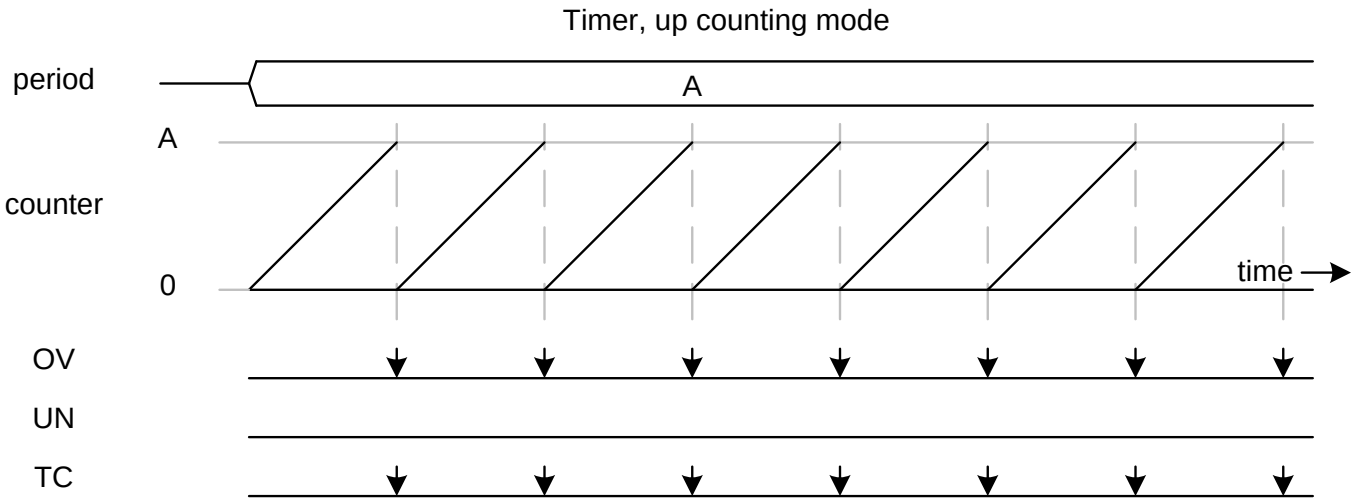
[5] Refer to the “Fixed Function Digital” section of the device datasheet and “Timer, Counter, and PWM” section of the Architecture TRM for more information. You can find links to the appropriate documentation on the **Datasheets** tab (in the Workspace Explorer of the PSoC Creator project) or from the **Help** menu.

to not occur (COUNTER equals COMPARE, and is changed to a different value, in the up/down counting modes).

- A capture event occurs.

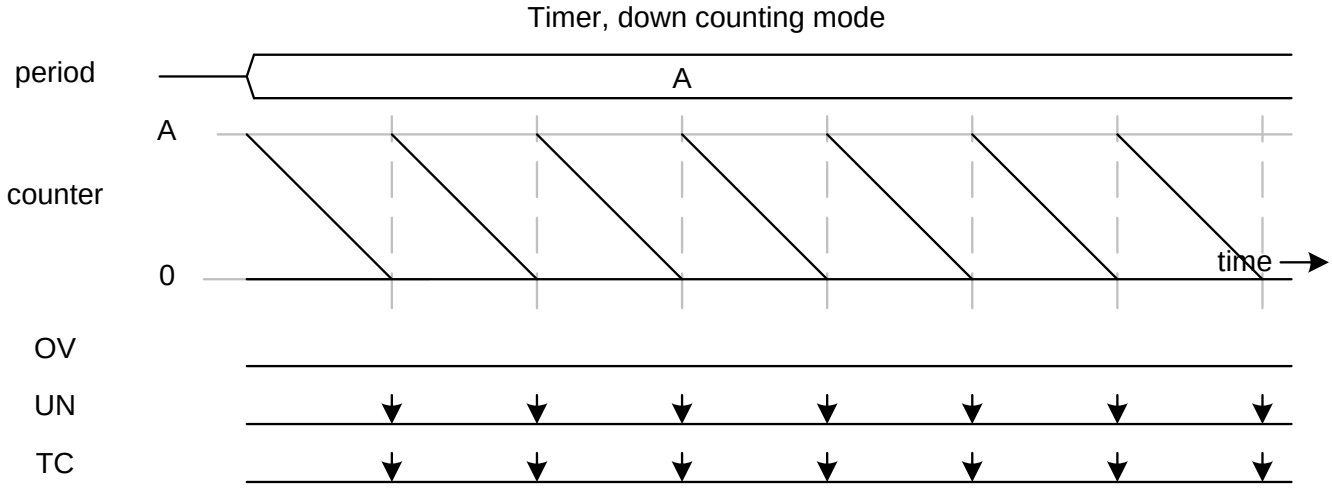
Up Mode

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’.



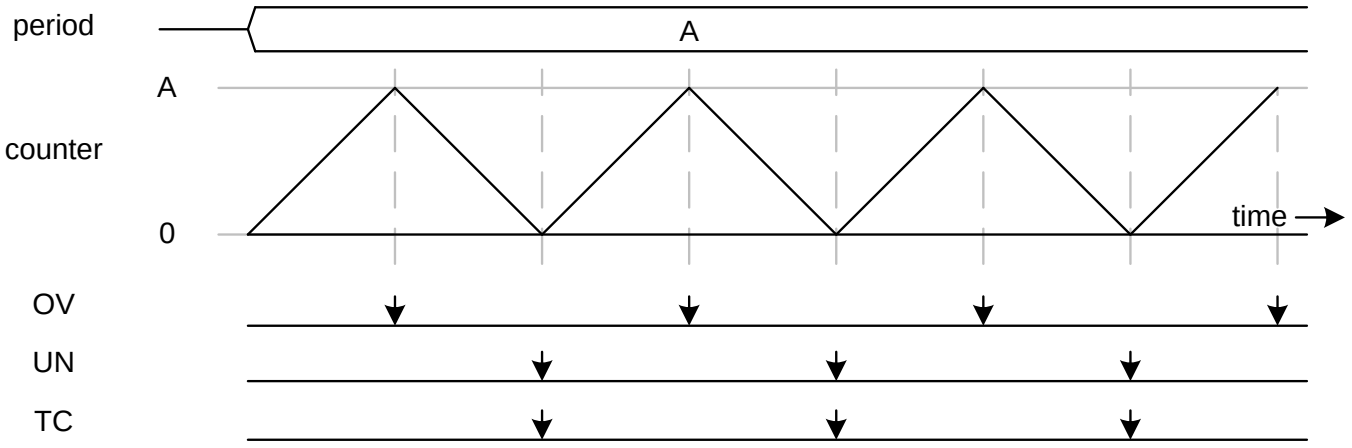
Up/Down Modes

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.

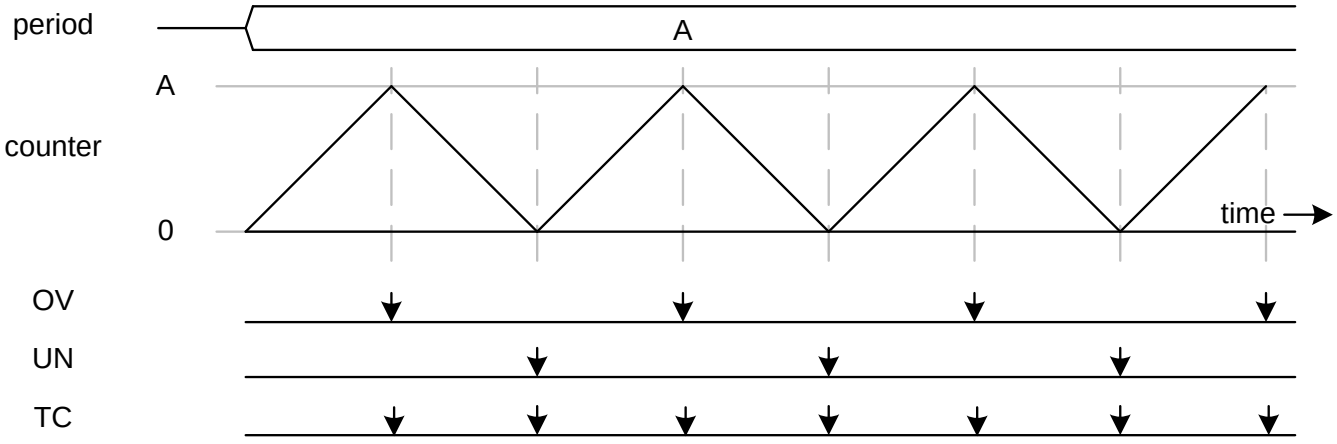
Note For All devices, except PSoC 4000, PSoC 4100, PSoC 4200, the counter register is initialized with value“1”. The Reload will also generate overflow (in up counting mode) / underflow (in down counting mode and up/down counting modes) event.

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.

Timer, up/down counting mode 0



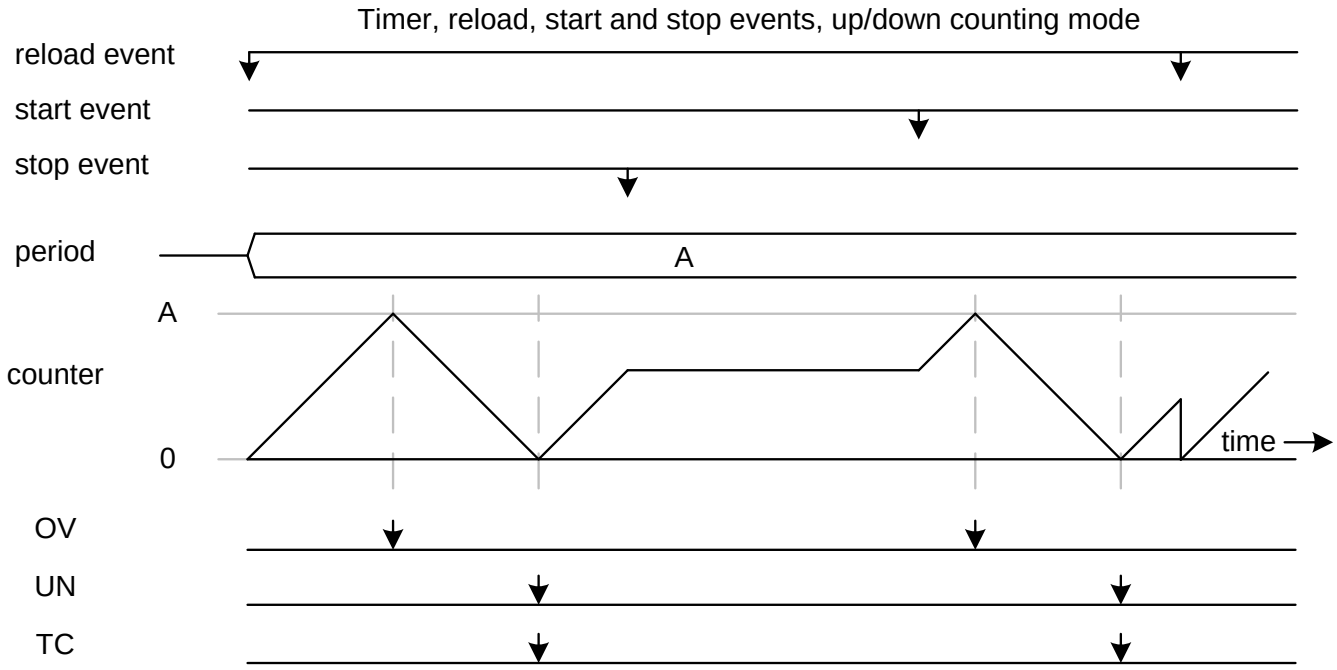
Timer, up/down counting mode 1



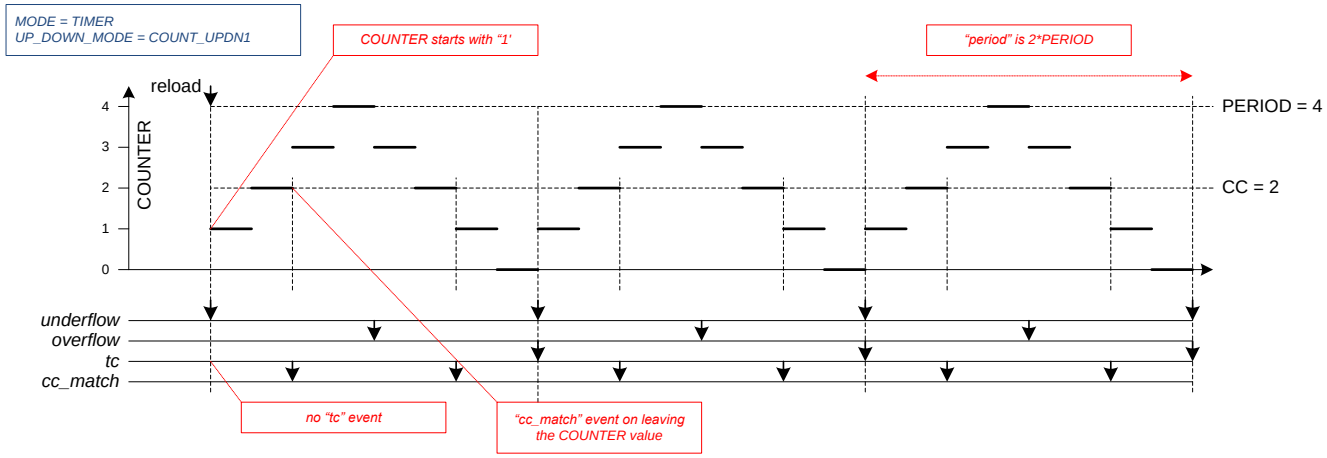
Operation

The first two examples illustrate 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.

The following figure shows a timer in up/down counting mode 0 for PSoC 4000 / PSoC 4200.



The following figure shows a timer in up/down counting mode 0 for all devices, except PSoC 4000, PSoC 4100, PSoC 4200, .



The counter is initialized (to 1) and started with a SW based reload event. Note:

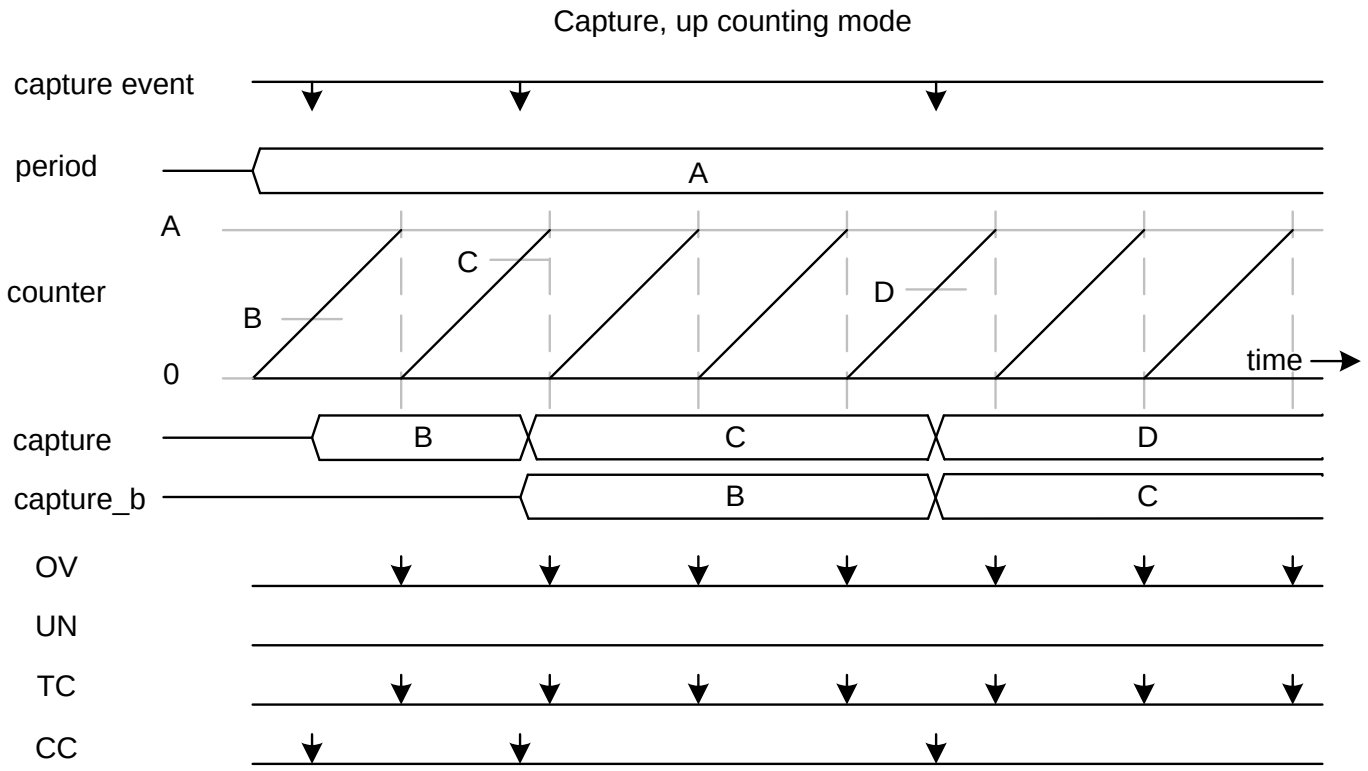
- When the counter changes from a state in which COUNTER is 4, an overflow is generated.
- When the counter changes from a state in which COUNTER is 0, an underflow and tc event are generated.
- When the counter changes from a state in which COUNTER equals 2, a cc_match event is generated.
- PERIOD is 4, resulting in an effective repeating counter pattern of 2*4 = 8 counter clock periods.

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.

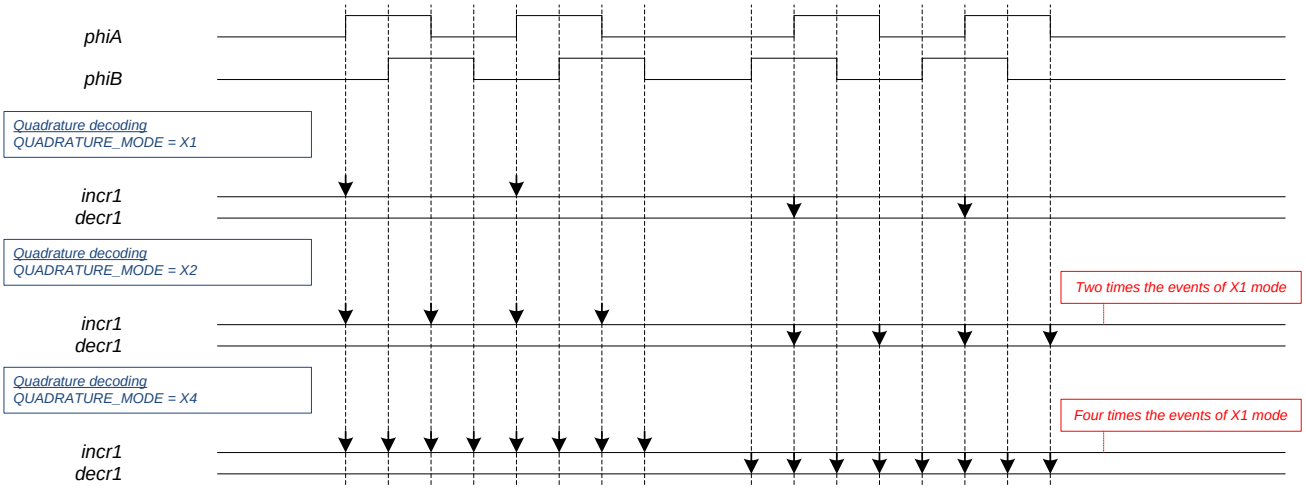


Incrementing and decrementing the counter is under the control of the count event and the counter clock enable signal. Typical operations use 1 as a count event and use the TCPWM clock as the counter clock without any clock pre-scaling. However advanced operations are possible using the count event configuration. 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 (PSoC 4100/PSoC 4200) or high frequency clock (other devices). 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.

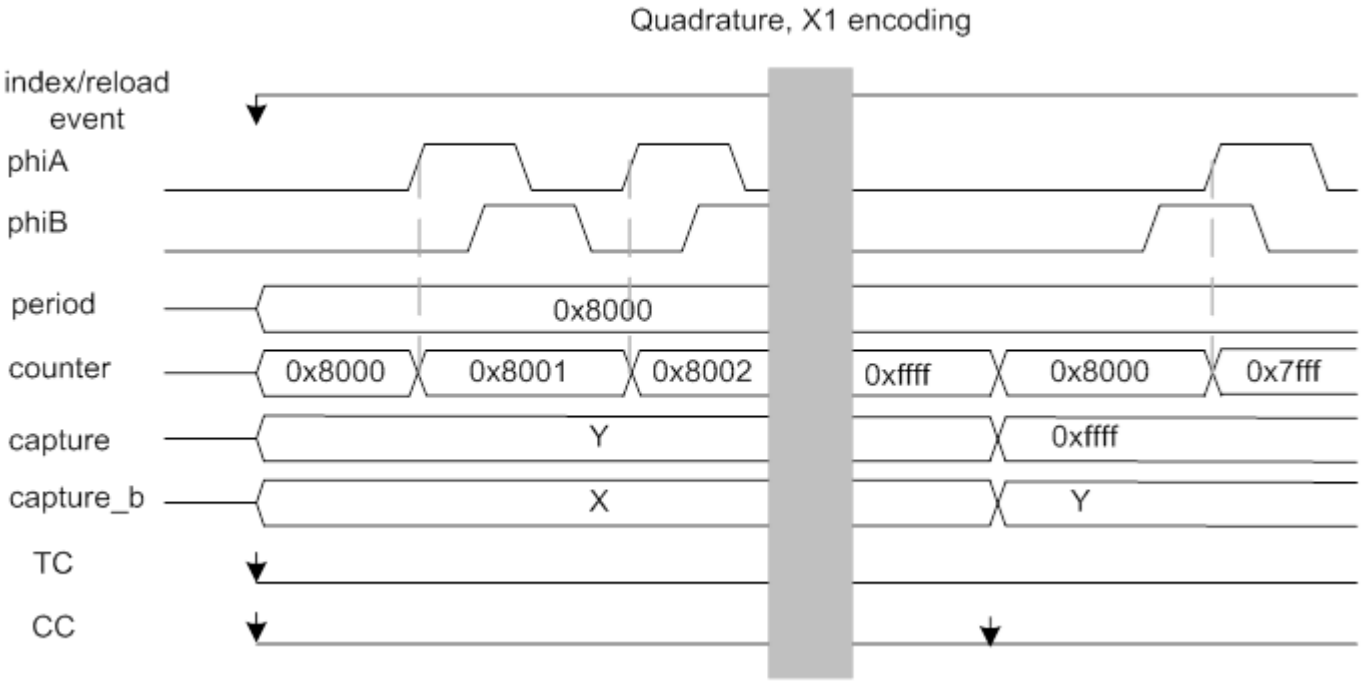
Quadrature Decoder

Counter increments (incr1 event) and decrements (decr1 event) are determined by the quadrature encoding scheme as illustrated by the following Figure.



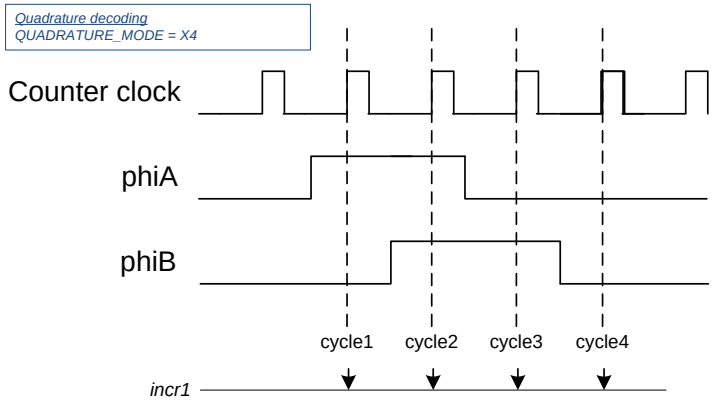
Note A higher resolution is achieved when using 2x or 4x modes.

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.

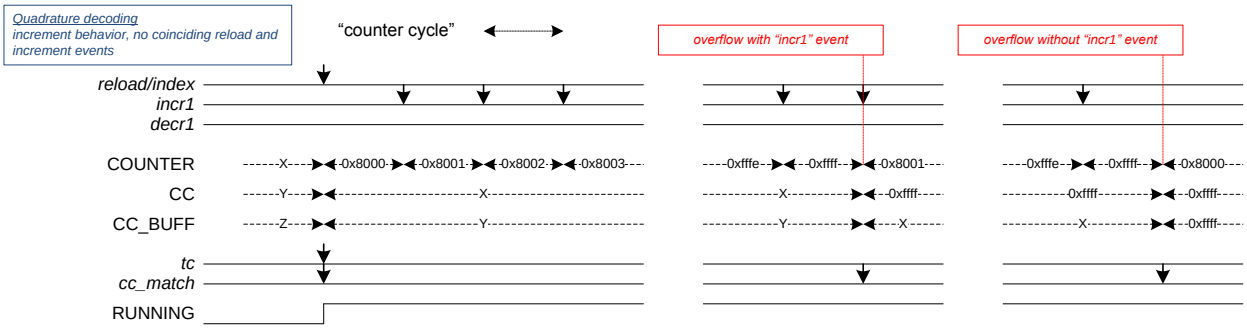
The state of phiA/phiB determines the increment/decrement per settings of encoding mode, and the phiA/phiB is detected by counter clock. The increment/decrement occurs at next counter clock rising edge after edges of phiA/phiB. E.g. the condition of increment in X1 mode is, at 1st counter clock rising edge, the phiA is low, at 2nd counter clock rising edge, phiA is high (now the counter will recognize that rising edge occurs on phiA). In the meanwhile (2nd counter clock rising edge), phiB is low. Then the counter will do an increment. Hence to get correct quadrature encoding as above figure, the rising/falling edge of phiA/phiB must be detected in different counter clock cycle. Following figure shows the phiA/phiB detection at different counter clock cycle.



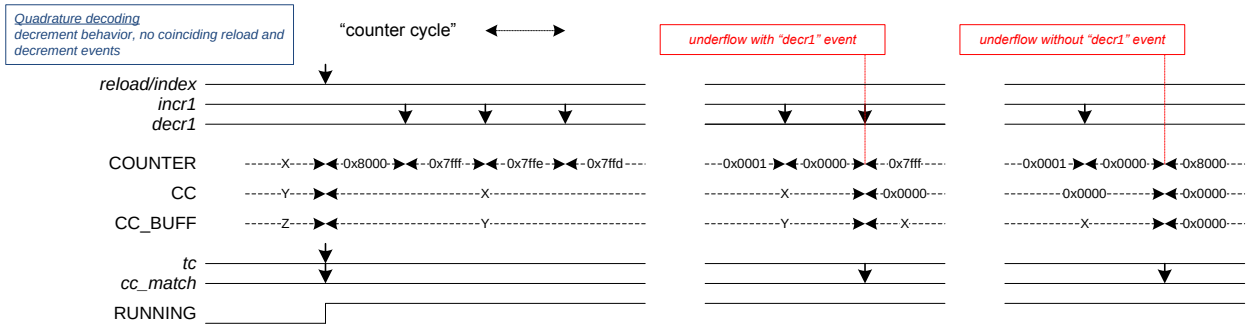
Some mechanical and low cost optical quadrature encoders may exhibit noise near the transition from one state to the next. This noise may cause the encoder to count invalid transitions and result in an encoder position error. To avoid this error, the encoder outputs should be filtered, either with a 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.

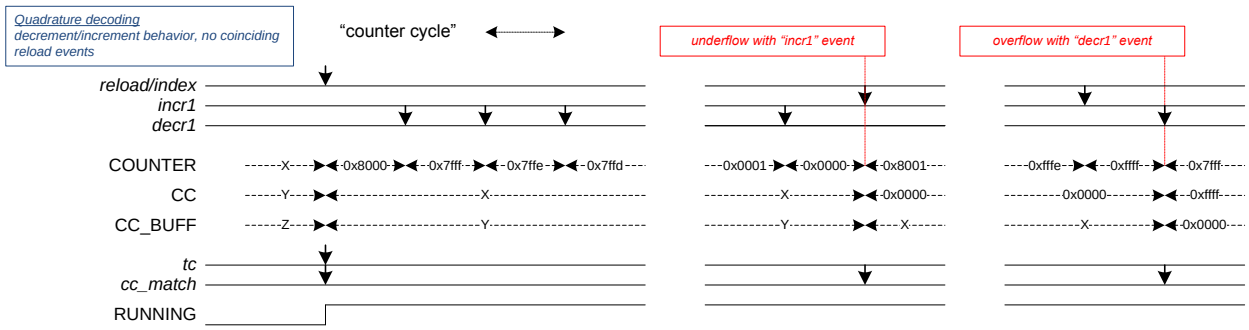
All devices, except PSoC 4000, PSoC 4100, PSoC 4200, Quadrature

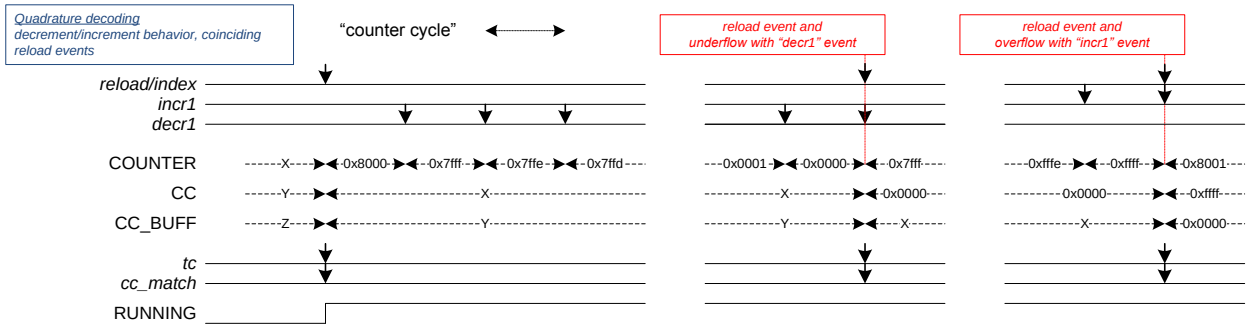
The first reload/index event copies the counter value COUNTER to CC.

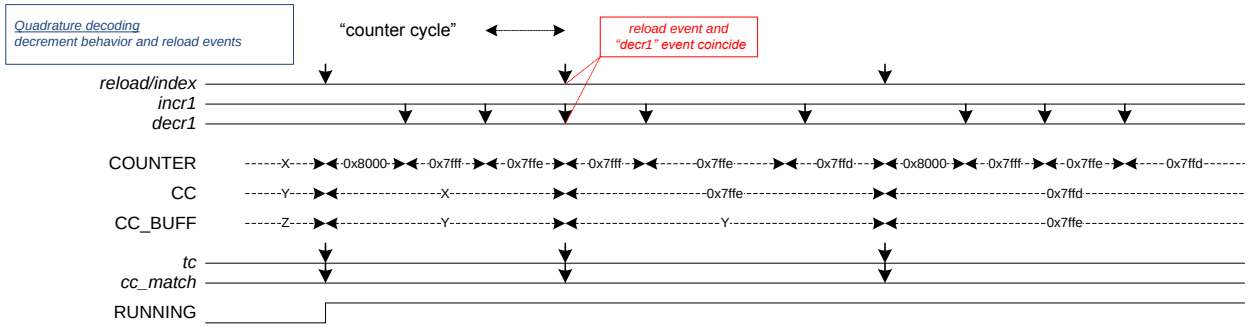


The following figures show quadrature functionality for different event scenarios (including scenarios with coinciding events). In all scenarios, the first reload/index event is generated by SW when the counter is not yet running.


Quadrature decoding
decrement behavior, no coinciding reload and decrement events
"counter cycle"
underflow with "decr1" event
underflow without "decr1" event
reload/index
incr1
decr1
COUNTER
CC
CC_BUFF
tc
cc_match
RUNNING
X
Y
Z
0x8000
0x7fff
0x7ffe
0x7ffd
0x0001
0x0000
0x8000
0x0000


Quadrature decoding
decrement/increment behavior, no coinciding reload events
"counter cycle"
underflow with "incr1" event
overflow with "decr1" event
reload/index
incr1
decr1
COUNTER
CC
CC_BUFF
tc
cc_match
RUNNING
X
Y
Z
0x8000
0x7fff
0x7ffe
0x7ffd
0x0001
0x0000
0x8001
0x0000
0xfffe
0xffff
0x7fff
0x0000
0xffff
0x0000


Quadrature decoding
decrement/increment behavior, coinciding reload events
"counter cycle"
reload event and underflow with "decr1" event
reload event and overflow with "incr1" event
reload/index
incr1
decr1
COUNTER
CC
CC_BUFF
tc
cc_match
RUNNING
X
Y
Z
0x8000
0x7fff
0x7ffe
0x7ffd
0x0001
0x0000
0x7fff
0x0000
0xfffe
0xffff
0x8001
0x0000
0xffff
0x0000


Quadrature decoding
decrement behavior and reload events
"counter cycle"
reload event and "decr1" event coincide
reload/index
incr1
decr1
COUNTER
CC
CC_BUFF
tc
cc_match
RUNNING
X
Y
Z
0x8000
0x7fff
0x7ffe
0x7ffd
0x7ffe
0x7ffd
0x8000
0x7fff
0x7ffe
0x7ffd
0x7ffe
0x7ffd
0x7ffe

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 line signal is generated by the cc, un and ov internal events only. It depends on **PWM align** settings, see **TCPWM_SetPWMMode()** API parameters description for additional clarification. For Pseudo random PWM mode line reflects: COUNTER[14:0] < COMPARE[15:0]. For PSoC 4100/PSoC 4200 devices configured in the Center and Asymmetric PWM modes, the cc event toggles line output for each clock in which the event is true.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.

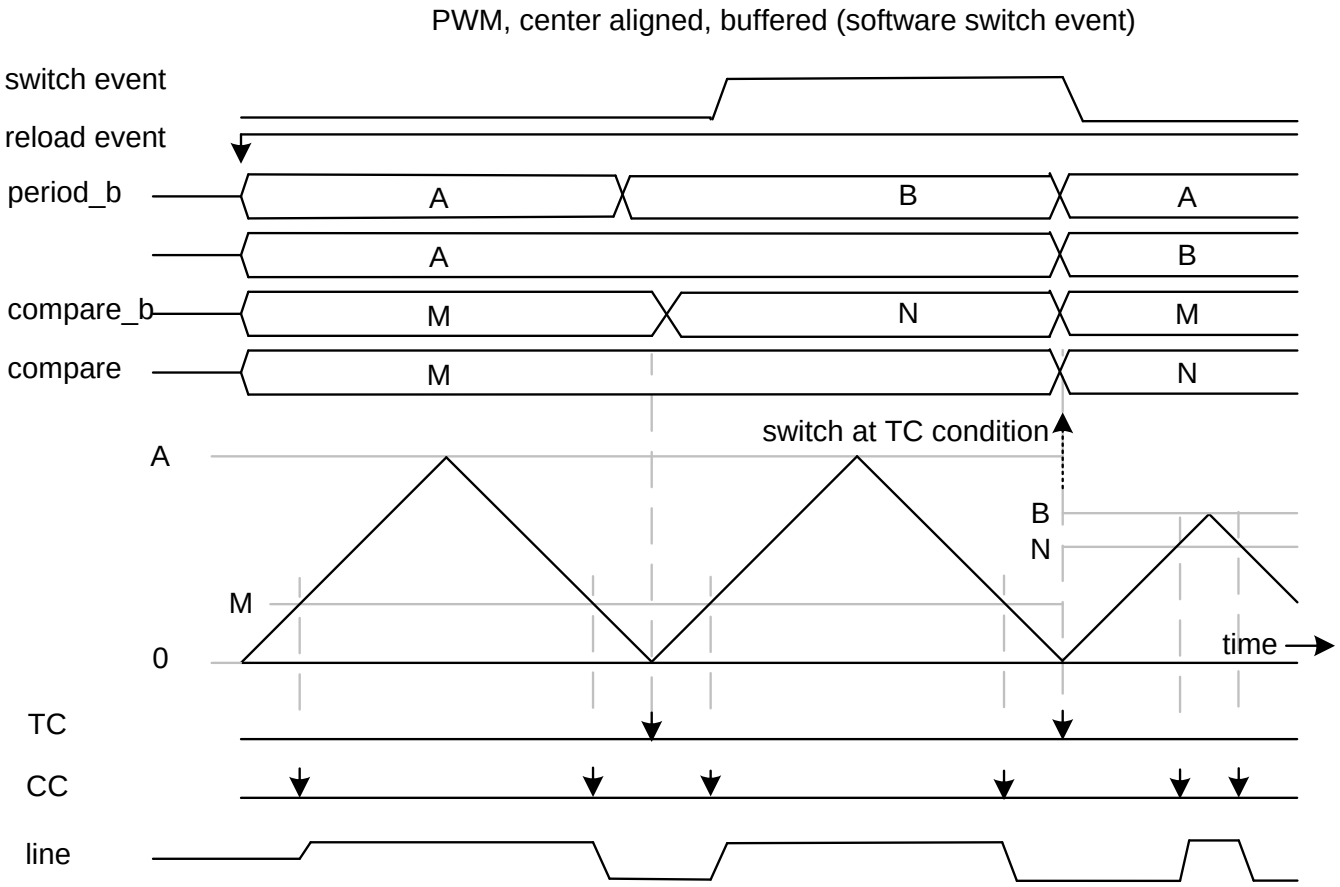
For an Asymmetric mode, the Line output pulse is not necessarily centered in the middle of the period (Center mode). A tc event is generated for both an underflow and overflow events. The tc event is used to exchange the COMAPRE and COMPARE_BUFF values. This functionality is realized by having a different Compare value when counting up and when counting down.

Note For non PSoC 4100/PSoC 4200 devices, the COMPARE and COMPARE_BUFF values are exchanged on an ov (overflow) event, that this restricts the asymmetry of the generated Line output pulse.

The user should update COMPARE_BUFF and PERIOD_BUFF in an interrupt handler on the tc event (and overwrites the HW updated values from the COMPARE/COMPARE_BUFF and PERIOD/PERIOD_BUFF exchanges).

The Asymmetric PWM mode should use the same period value when counting up and counting down. When PERIOD and PERIOD_BUFF are switched on a tc event (overflow or underflow event), care should be taken to ensure that:

- Within a PWM period (tc event coincides with an overflow event), the period values are the same (an overflow switch of PERIOD and PERIOD_BUFF should NOT change the period value; i.e. PERIOD_BUFF should be PERIOD)
- Between PWM periods (tc event coincides with an underflow event), the period value can change (an underflow switch of PERIOD and PERIOD_BUFF may change the period value; i.e. PERIOD_BUFF may be different from PERIOD).

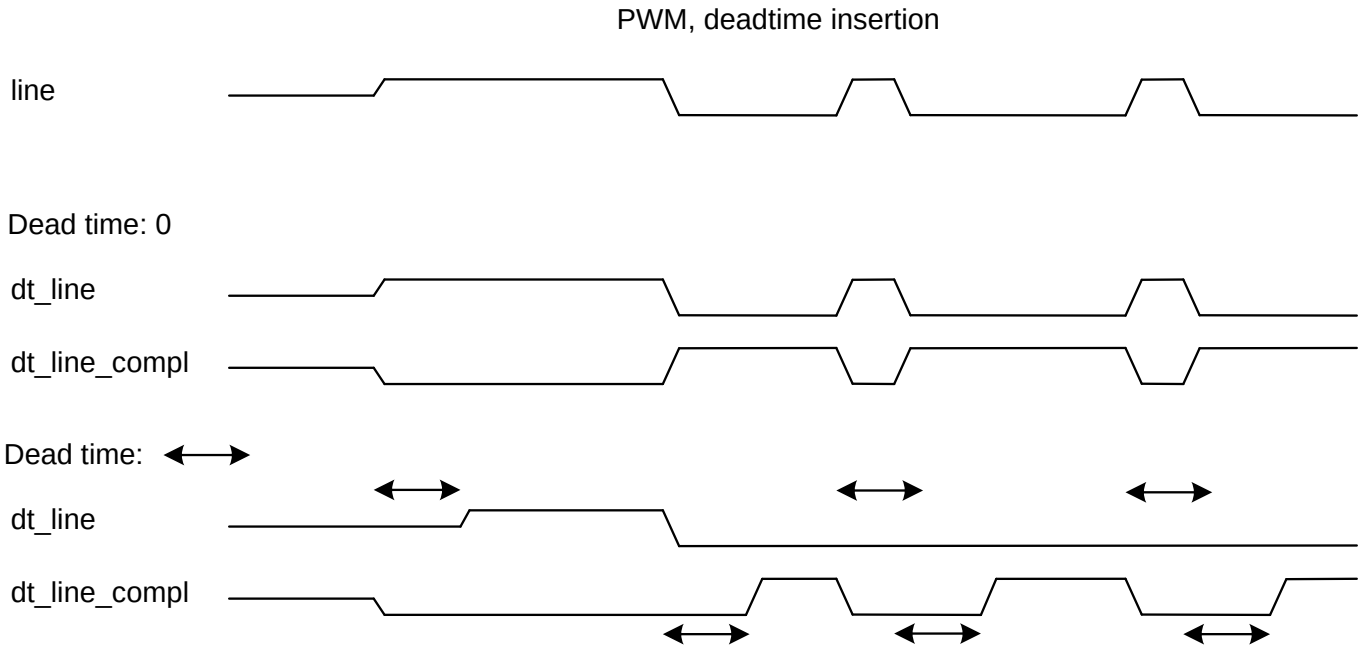


The switch/capture event functionality can be used to allow for DSI controlled modulation of the PWM high phase to create a PWM high phase accuracy that exceeds the accuracy of a single PWM period. This higher accuracy is achieved by modulating between two compare values over multiple PWM periods. The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

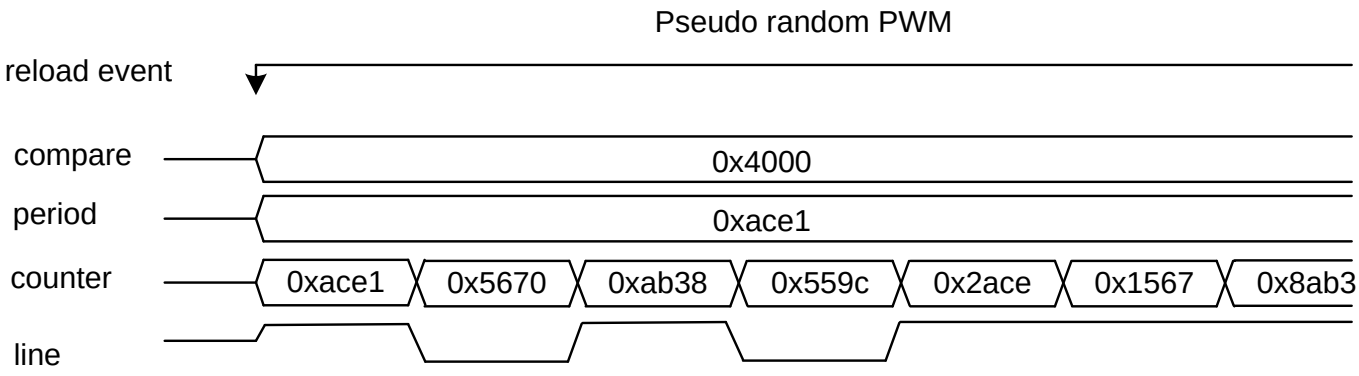
Dead time insertion mode allows the insertion of dead time into the PWM. The following figure illustrates how the complementary output lines “dt_line” and “dt_line_compl” are generated from the PWM output line, “line”. The top example shows the output lines with no dead time. The bottom example shows 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.



Clock Selection

There is no internal clock in this component. You must attach a clock source. This clock must be from the global clock generation logic. Clock prescaler functionality is available within the TCPWM component.

DMA Support

The DMA component can be used to transfer data from the component registers to RAM or another component.

Name of DMA Source / Destination	Width	Direction	DMA Req Signal	DMA Req Type	Description
TCPWM_COUNTER_PTR	32	Source / Destination	N/A	N/A	Count register. It is not advised to write to this register when the counter is running.
TCPWM_COMP_CAP_PTR	32	Source / Destination	N/A	N/A	Counter compare/capture register.
TCPWM_COMP_CAP_BUF_PTR	32	Source / Destination	N/A	N/A	Counter buffered compare/capture register.
TCPWM_PERIOD_PTR	32	Source / Destination	N/A	N/A	Period register. To cause the counter to count for N cycles this register should be written with N-1 (counts from 0 to period inclusive).
TCPWM_PERIOD_BUF_PTR	32	Source / Destination	N/A	N/A	Period buffered register. To cause the counter to count for N cycles this register should be written with N-1 (counts from 0 to period inclusive).

N/A – The component does not have any specific requirements.

Note The TCPWM has a DMA bus interface that supports 32-bit (word) transfers only. If a data element size used for DMA transfer is less than a word, set the DMA descriptor with the correct width; for example, data element size is halfword (2 bytes). If a component register is used as Source, make sure the DMA descriptor is configured as "Word to Halfword." If a component

register is used as Destination, then the DMA descriptor should be configured as "Halfword to Word."

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.

Registers

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

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.

Note Final characterization data for PSoC Analog Coprocessor device is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.

TCPWM DC Specifications

PSoC 4000

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

PSoC 4100 / PSoC 4200 / PSoC 4200U

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

PSoC 4100 BLE/PSoC 4200 BLE

Parameter	Description	Min	Typ	Max	Units	Conditions
I_{TCPWM1}	Block current consumption at 3 MHz	–	–	42	µA	All modes
I_{TCPWM2}	Block current consumption at 12 MHz	–	–	132	µA	All modes
I_{TCPWM3}	Block current consumption at 48 MHz	–	–	535	µA	All modes

Other devices

Parameter	Description	Min	Typ	Max	Units	Conditions
I_{TCPWM1}	Block current consumption at 3 MHz	–	–	45	µA	All modes
I_{TCPWM2}	Block current consumption at 12 MHz	–	–	155	µA	All modes
I_{TCPWM3}	Block current consumption at 48 MHz	–	–	650	µA	All modes

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
2.10.c	Updated datasheet.	Added final characterization data for PSoC 4100S device. Updated waveforms for quadrature decoder mode in Functional description section. Updated note related to the Input/Output Connections section. Minor datasheet edits with supported devices info.
2.10.b	Updated datasheet.	Added final characterization data for PSoC 4000S device. Updated information to include the PSoC 4000S, PSoC 4100S, PSoC Analog Coprocessor devices.
2.10.a	Updated datasheet.	Final characterization data for PSoC 4000S, PSoC 4100S and PSoC Analog Coprocessor devices is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.
2.10	Added auto-start counting for QuadDec mode for the new placed component in the design.	QuadDec_TriggerCommand or hardware Index input signal is optional for newly placed components in designs. **Note** The updated component from a previous version will maintain previous behavior and not start until an index event (either hardware or firmware generated) occurs. Therefore, if a hardware index is not used, the TriggerCommand API must be called with CMD_RELOAD before the counter will start
	Datasheet edits.	Clarified Input/Output Connections, Component Parameters, Application Programming Interface, Functional Description sections. Clarified Enable(), Start(), WriteCompare(), ReadCompare(), WriteCompareBuf() and ReadCompareBuf() APIs descriptions. Updated information to include the PSoC 4200L device.
2.0.b	Datasheet edits.	Updated Input/Output Connections section. Added DMA Support section for PSoC 4100M and PSoC 4200M devices. Updated DC and AC Electrical Characteristics section with PSoC 4100M/PSoC 4200M data.
2.0.a	Datasheet edits.	Updated DC and AC Electrical Characteristics section to support PSoC 4100 BLE/PSoC 4200 BLE devices. Clarified TCPWM_SetPWMDeadTime() API description. Updated General Description section. Updated Input/Output Connections section.
2.0	Added support for Bluetooth Low Energy devices.	New device and features.
	Datasheet edits.	Clarified TCPWM_TriggerCommand () API description. Clarified TCPWM_SetPWMMode() API description.

Version	Description of Changes	Reason for Changes / Impact
	Changed configuration for PWM Asymmetric mode.	Improve the TCPWM performance.
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.
	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.
	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, 2015-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.