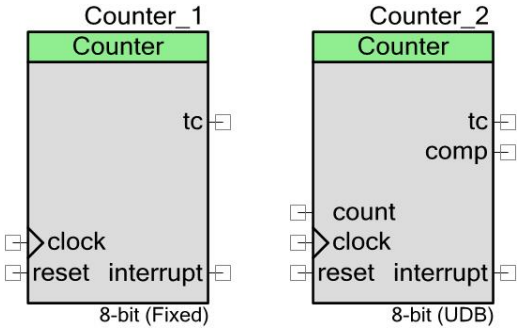
# Counter
**2.40**




## Features

- Fixed-function (FF) implementation for PSoC 3 and PSoC 5 devices
- 8-, 16-, 24-, or 32-bit counter
- Up, down, or up-and-down configurations
- Optional compare output
- Optional capture input
- Enable and reset inputs for synchronizing with other components
- Continuous or one-shot run modes

## General Description

The Counter component provides a method to count events. It can implement a basic counter function and offers advanced features such as capture, compare output, and count direction control.

For PSoC 3 and PSoC 5 devices component can be implemented using FF blocks or UDB. PSoC 4 devices support only UDB implementation. A UDB implementation typically has more features than an FF implementation. If your design is simple enough, consider using FF to conserve UDB resources for other purposes.

The following table shows the major differences between FF and UDB. For more details about FF resources in the devices, refer to the applicable device datasheet or Technical Reference Manual.

| Feature | FF | UDB |
|---|---|---|
| Number of bits | 8 or 16 | 8, 16, 24, or 32 |
| Run mode | Continuous or one-shot | Continuous or one-shot |
| Counter mode | Down only | Up, down, or up-and-down |
| Enable input | No (software enable only) | Yes (hardware or software enable) |
| Capture input | No | Yes |

| Feature | FF | UDB |
|---|---|---|
| Capture mode | None | Rising edge, falling edge, either edge, or software controlled |
| Capture FIFO | No (one capture register) | Yes (up to four captures) |
| Reset input | Yes | Yes |
| Terminal count output | Yes | Yes |
| Compare output | No | Yes |
| Compare mode | None | <, ≤, =, ≥, >, or software controlled |
| Interrupt output | Yes | Yes |
| Interrupt conditions | TC | TC, capture, and compare |
| Period register | Yes | Yes |
| Period reload | Y (always reload on reset or TC) | Y (reload on one or more of reset, TC, capture, compare) |
| Clock input | Limited to digital clocks in the clock system | Any signal |
| Sampling clock input | No | Requires an explicit clock signal (component clock) for sampling input signals of the component |

## When to Use a Counter

The default use of the Counter is to count the number of edge events on the count input. However, there are several other potential uses of the Counter:

- Clock divider: By driving a clock into the count input and using the compare or terminal count output as the divided clock output
- Frequency counter: By connecting a signal with a known period to the enable input of the counter while counting the signal to measure on the count input.
- Tool to measure complementary events such as the output of a quadrature decoder

**Note** A Timer component is better used in situations focused on measuring the time between events. A PWM component is better used in situations requiring multiple compare outputs with more control features such as center alignment, output kill, and deadband outputs.

# Input/Output Connections

This section describes the various input and output connections for the Counter component. Some I/Os may be hidden on the symbol under the conditions listed in the description of that I/O.

**Note** All signals are active high unless otherwise specified. All inputs to the counter must be synchronized outside of the counter.

| Input | May Be Hidden | Description |
|---|---|---|
| clock | N | The functional behavior of the clock input differs for the fixed-function Counter compared to the UDB Counter.<br>▪ For a fixed-function Counter, there is no count input. A fixed-function Counter updates (decrements its internal counter) on every rising edge of the clock input.<br>▪ For a UDB Counter, both the clock and count input appear on the Counter symbol. The clock is used to sample the inputs of the Counter component. The UDB Counter is implemented as a synchronous counter that uses the clock input only as a synchronization clock. All inputs of the Counter must be synchronized to the clock input to avoid setup violations. This also makes sure that the edge-detect circuitry in the UDB Counter implementation functions properly. |
| Count | Y | In a fixed-function Counter, there is no count input on the Counter symbol. A FF Counter updates its internal count on every rising edge of the clock input.<br>For a UDB Counter, the Counter updates its internal counter with an edge-detect logic to determine an update event. The source signal for the update event depends on the clock mode of the UDB Counter. The edge-detect logic is clocked using the clock input. Both edges of the count input must meet setup to the clock input; therefore, the maximum count input is one-half of the clock input frequency.<br>▪ For a UDB Counter in the **Up Counter** or **Down Counter** clock mode, the edge detect logic detects the rising edge of the count input synchronous to the clock input. Depending on whether the Counter is configured as an up counter or down counter, the edge detect event on the count input increments or decrements the Counter, respectively. See Figure 5 on page 24 for a functional description of the Up/Down Counter.<br>▪ For a UDB Counter in the **Count Input and Direction** clock mode, the edge-detect logic detects the rising edge of the count input synchronous to the clock input. Depending on the whether the "upndown" signal is a 1 or a 0, the edge-detect event on the count input increments or decrements the Counter, respectively. See Figure 7 on page 25 for a functional description of the Counter in **Count Input and Direction** clock mode.<br>▪ For a UDB Counter in the **Clock with UpCnt & DwnCnt** clock mode, there is no count input. The Counter update event is determined by a combination of the edge-detect logic on UpCnt and DwnCnt signals synchronous to the clock input. See Figure 9 on page 26 for a functional description of the Counter in **Clock with UpCnt & DwnCnt** clock mode. |

| Input | May Be Hidden | Description |
|---|---|---|
| upCnt | Y | Increment signal to the counter. When **Clock Mode** is set to **Clock With UpCnt & DwnCnt**, this input is used in conjunction with the dwnCnt input and the clock input to allow the Counter to be used as an encoder. A rising edge on this input increments the count value. Both edges of the upCnt input must meet setup to the clock input; therefore, the maximum count input is one-half of the clock input frequency. |
| dwnCnt | Y | Decrement signal to the counter. When **Clock Mode** of the counter is set to **Clock With UpCnt & DwnCnt**, this input is used in conjunction with the upCnt input and the clock input to allow the counter to be used as an encoder. A rising edge on this input decrements the count value. Both edges of the dwnCnt input must meet setup to the clock input; therefore, the maximum count input is one-half of the clock input frequency. |
| up_ndown | Y | Defines the counting direction of the counter. This input is available only if **Clock Mode** is set to enable direction control (**Count Input and Direction**) On a rising edge of the count input, a '1' on this input causes the counter to increment, and a '0' on this input causes the counter to decrement. |
| reset | N | The reset input resets the counter to the starting value.<br>▪ For the **Up Counter** configuration, the starting value is zero.<br>▪ For **Down Counter**, **Count Input and Direction**, and **Clock With UpCnt & DwnCnt** configurations, the starting value is set to the current period register value.<br>The reset input is sampled on the count/clock input.<br>For PSoC 3 Production or later silicon, the Terminal Count pin for the fixed-function counter is held low during reset. The reset input is used to reset the control register in PSoC 3 Production, PSoC 4 and PSoC 5 counter implementation, and also to implement the One Shot Run Mode feature.<br>UDB one-shot mode needs a reset pulse in order to start counting. If the Counter is only run in one-shot mode from power up, it will not start counting until reset is applied and removed. |
| enable | Y | Hardware enable of the counter. This input is visible when the **Enable Mode** parameter is set to **Hardware**. |
| capture | Y | Captures the current count value to a capture register or FIFO. This input is visible if the **Capture Mode** parameter is set to any mode other than **None**. Capture may take place on a rising edge, falling edge, or either edge applied to this input, depending on the **Capture Mode** setting.<br>The capture input is sampled on the clock input. |

| Output | May Be Hidden | Description |
|---|---|---|
| tc | N | Terminal count is a synchronous output that indicates that the count value is equal to the terminal count. The output is synchronous to the clock input of the Counter. The signal goes high one clock cycle after the count value matches the terminal count and stays high while the count value is equal to the terminal count. If the Counter is disabled when the count is at terminal count, the output stays high until the Counter is re-enabled. |

| Output | May Be Hidden | Description |
|---|---|---|
| comp | Y | The compare output indicates the counter value compared to the compare value based on the configuration in the **Compare Mode** parameter. The compare output goes high after a delay of one clock cycle following the compare event. |
| interrupt | N | The interrupt output is driven by the interrupt sources configured in the hardware. All sources are ORed together to create the final output signal. The sources of the interrupt can be: Compare, Terminal Count or Capture. |

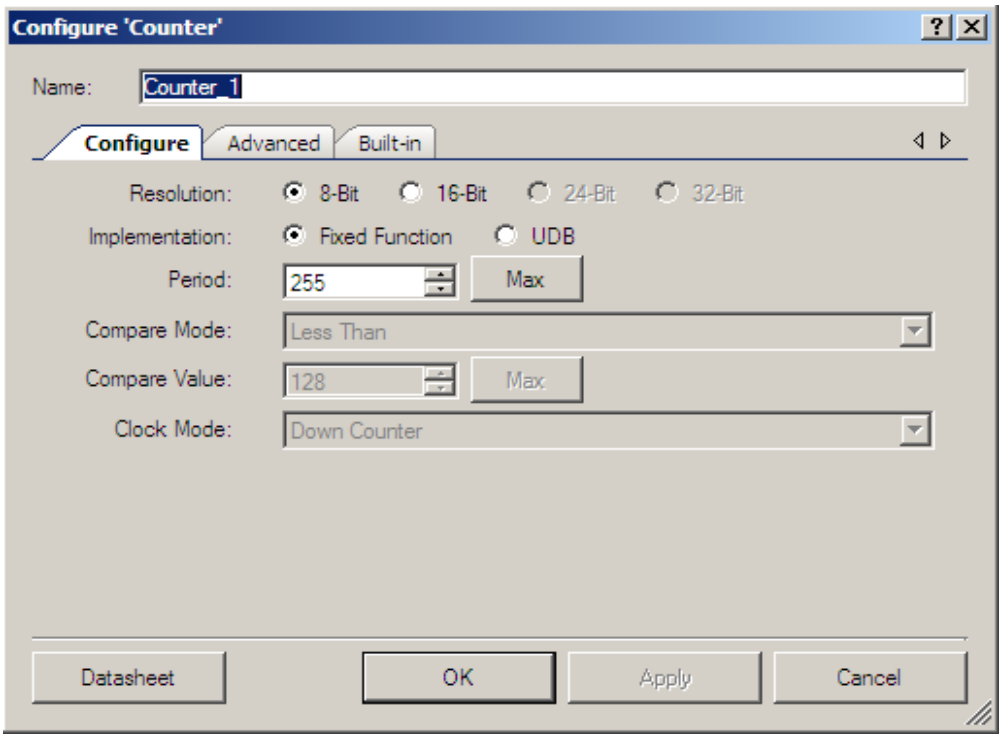
# Component Parameters

Drag a Counter onto your design and double-click it to open the **Configure** dialog.

## Hardware versus Software Configuration Options

Hardware configuration options change the way the project is synthesized and placed in the hardware. You must rebuild the hardware if you make changes to any of these options. Software configuration options do not affect synthesis or placement. When setting these parameters before build time you are setting their initial value, which may be modified at any time with the API provided. Most parameters described in the following sections are hardware options. The software options are noted as such.

## Configure Tab



### Resolution

The **Resolution** parameter defines the bit-width resolution of the Counter component. This value may be set to 8, 16, 24, or 32 for maximum count values of 255, 65535, 16777215, and 4294967295, respectively.

### Implementation

The **Implementation** parameter allows you to choose between a fixed-function block implementation and a UDB implementation of the Counter. If you select **Fixed Function**, UDB functions are disabled.

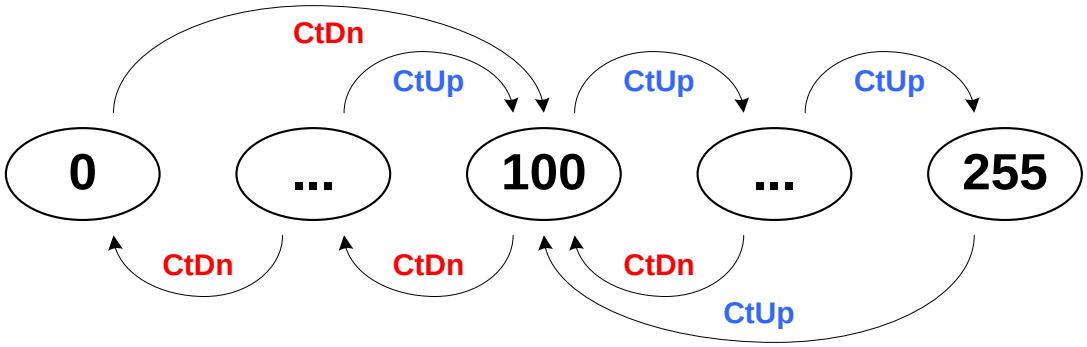
### Period (Software Option)

The **Period** parameter defines the max counts value (or rollover point) for the Counter component. This parameter defines the initial value loaded into the period register, which the software can change at any time with the Counter_WritePeriod() API.

The limits of this value are defined by the **Resolution** parameter. For 8-, 16-, 24-, and 32-bit **Resolution** parameters, the maximum value of the **Period** value is defined as (2^8) – 1, (2^16) – 1, (2^24) – 1, and (2^32) – 1 or 255, 65535, 16777215, and 4294967295, respectively.

When **Clock Mode** is configured as **Clock with UpCnt & DwnCnt** or **Count Input and Direction**, the counter is set to the period at start and any time the counter overflows or underflows. In these clock modes, do not set the period value to all 1s or all 0s. The Counter won't work if period is set to all 1s or all 0s. Instead, the normal practice is to keep the period value at the midpoint of the period range (for an 8-bit counter, 0x7F). Figure 1 shows **Clock Mode** set to **Count Input and Direction**.

**Figure 1. Clock Mode**



### Compare Mode (Software Option)

The **Compare Mode** parameter configures the operation of the comp output signal. This signal is the status of a compare between the compare value parameter and current counter value. This parameter defines the initial setting. You can change it at any time to reconfigure the compare operation of the Counter component.

**Compare Mode** can be set to any of the following values:

- **Less Than** – The counter value is less than the compare value.
- **Less Than Or Equal** – The counter value is less than or equal to the compare value.
- **Equal To** – The counter value is equal to the compare value.
- **Greater Than** – The counter value is greater than the compare value.
- **Greater Than Or Equal** – The counter value is greater than or equal to the compare value.
- **Software Controlled** – The compare mode can be set during run time with the Counter_SetCompareMode() API call to any one of the other five compare modes on this list.

### Compare Value (Software Option)

The **Compare Value** parameter defines the initial value loaded into the compare register of the counter. This value is used in conjunction with the **Compare Mode** parameter to define the operation of the compare output.

This value can be any unsigned integer from 0 to (2^Resolution – 1), but it must be less than or equal to the period value.

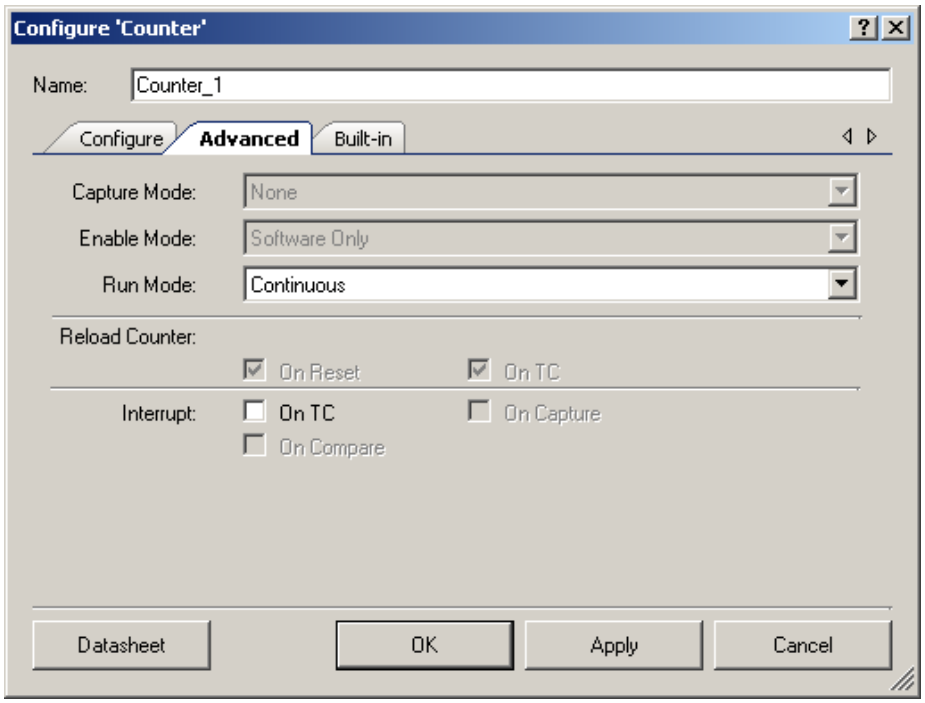
### Clock Mode

The **Clock Mode** parameter configures how the Counter will count. This mode can be set to any of the following values:

- **Count Input and Direction** – Counter is a bidirectional counter. It counts up while the up_ndown input is high on each rising edge of the input clock and counts down while up_ndown is low on each rising edge of the input clock.
- **Clock With UpCnt & DwnCnt** – Counter is a bidirectional counter. It increments the counter for each rising edge on the upCnt input and decrements the counter for each rising edge of the dwnCnt input, with respect to the clock input. In this mode, the clock input frequency must be at least 2x the frequency of the upCnt and dwnCnt inputs.
- **Up Counter** – Counter is an up counter only. It increments on a rising edge of the count input with respect to the clock signal while the counter is enabled.

- **Down Counter** – Counter is a down counter only. It decrements on a rising edge of the count input with respect to the clock signal while the counter is enabled.

## Advanced Tab




### Capture Mode

The **Capture Mode** parameter configures when a capture takes place. The capture input is sampled on the rising edge of the clock input. This mode can be set to any of the following values:

- **None** – No capture implemented. The capture input pin is hidden.
- **Rising Edge** – Capture the counter value on a rising edge of the capture input with respect to the clock input.
- **Falling Edge** – Capture the counter value on a falling edge of the capture input with respect to the clock input.
- **Either Edge** – Capture the counter value on either edge of the capture input with respect to the clock input.
- **Software Controlled** –Set the mode at run time by setting the capture mode bits in the control register Counter_CTRL_CAPMODE_MASK with the enumerated capture mode types. These are defined in the *Counter.h* header file.

### Enable Mode

The **Enable Mode** parameter configures the enable implementation of the counter. The enable input is sampled on the rising edge of the clock input. This mode can be set to any of the following values:

- **Software Only** – The Counter is enabled based on the enable bit of the control register only.
- **Hardware Only** – The Counter is enabled based on the enable input only.
- **Hardware and Software** – The Counter is enabled if both hardware and software enables are true.

### Run Mode

The **Run Mode** parameter allows you to configure the Counter component to run continuously or in one-shot mode:

- **Continuous** – The Counter runs continuously while it is enabled.
- **One Shot** – The Counter runs through a single period and stops at terminal count. After it is reset, it begins another single cycle. On stop, for a UDB counter, it reloads period into the count register; for a fixed-function counter, the count register remains at terminal count.

### Reload Counter

The **Reload Counter** parameters allow you to configure when the counter value is reloaded. The counter value is reloaded when one or more of the following selected events occur. The counter is reloaded with its start value on a reload event.

- **On Capture** – The counter value is reloaded when a capture event has occurred. By default, this parameter is cleared. This parameter is shown only when **UDB** is selected for **Implementation**.
- **On Compare** – The counter value is reloaded when a compare true event has occurred. By default, this parameter is cleared. This parameter is shown only when **UDB** is selected for **Implementation**.
- **On Reset** – The counter value is reloaded when a reset event has occurred. By default this parameter is selected. This parameter is always shown. For a fixed-function counter, it cannot be changed. For a UDB counter, it can be turned off.
- **On TC** – The counter value is reloaded when the counter has overflowed (in up count mode) or underflowed (in down count mode). By default, this parameter is selected. For a fixed-function counter, it cannot be changed. For a UDB counter, it can be turned off.

  When the clock mode is set to **Clock With UpCnt & DwnCnt** this option reloads to the period value when counter is 0x00 or all 0xFF.

The following table lists the reload and terminal count conditions for the various Clock Modes of the Counter component.

| Clock Mode | Terminal Count Condition | Counter Start Value and Reload Value on a Reload Event |
|---|---|---|
| Fixed Function Counter | When counter equals 0. | The start and the reload value of the Counter is the **Period** value of the component. |
| UDB Up Counter | When counter equals **Period** value | The start and the reload value of the Counter is 0. |
| UDB Down Counter | When counter equals 0. | The start and the reload value of the Counter is the **Period** value of the component. |
| UDB Count Input and Direction | When the Counter is all 1s (0xFF for 8-bit, 0xFFFF for 16-bit, 0xFFFFFF for 24-bit, or 0xFFFFFFFF for a 32 bit Counter) when it is counting up OR when the Counter equals 0 when it is counting down. | The start and reload value of the Counter is the **Period** value of the component. If there is no reload condition, the counter wraps around and continues counting without a counter register reload. |
| UDB Clock with UpCnt and DwnCnt | When the Counter is all 1s (0xFF for 8-bit, 0xFFFF for 16-bit, 0xFFFFFF for 24-bit, or 0xFFFFFFFF for a 32-bit Counter) when it is counting up OR when the Counter equals 0 when it is counting down. | The start and reload value of the Counter is the **Period** value of the component. If there is no reload condition, then the counter wraps around and continues counting without a counter register reload. |

### Interrupt

The **Interrupt** parameters allow you to configure the initial interrupt sources. An interrupt is generated when one or more of the following selected events occur. This parameter defines an initial configuration. The software can reconfigure this mode at any time.

- **On TC** –This parameter is always active; by default, it is not selected.
- **On Capture** – By default, this parameter is not selected. It is always shown, but it is only active when the **Implementation** parameter is set to **UDB**.
- **On Compare** – By default, this parameter is not selected. It is always shown, but it is only active when the **Implementation** parameter is set to **UDB**.

# Clock Selection

For the Counter component,

- When the **Clock Mode** parameter is set to **Up Counter** or **Down Counter**, the count input can be any signal for which the rising edges are counted. The clock input to the component samples the count input and both the rising and falling edges must meet the setup requirements to the clock.

- When the **Clock Mode** parameter is set to **Count Input and Direction**, the count input can be any signal for which the rising edges are counted. The clock input to the component samples the count input and both the rising and falling edges must meet the setup requirements to the clock. The counter will count up or down depending on the up_ndown input.
- When the **Clock Mode** parameter is set to **Clock With UpCnt & DwnCnt**, the upCnt and dwnCnt rising edges are sampled with respect to the clock input. The counter counts on a rising edge of the upCnt signal and down on the rising edge of the dwnCnt signal. Both edges of upCnt and dwnCnt must meet setup requirements to the clock.

See the Clock component datasheet and the appropriate device datasheet for more details about PSoC 3, PSoC 4 or PSoC 5 clocking systems..

### Fixed-Function Components

When configured to use the FF block in the device, the Counter component has the following restrictions:

- The count input must be a digital clock from the clock system.
- If the frequency of the clock is to be the same as bus clock, the clock must actually be the bus clock.

  Open the **Configure** dialog of the appropriate Clock component to configure the **Clock Type** parameter as **Existing** and the **Source** parameter as **BUS_CLK**. A clock at this frequency cannot be divided from any other source, such as the master clock, IMO, and so on.

### UDB-based Components

You can connect any digital signal from any source to the count/clock input. The frequency of that signal is limited to the frequency range defined in the DC and AC Electrical Characteristics (UDB Implementation) section in this datasheet. The count input must, at most, be half the frequency as that of the clock input in any of the Counter Clock modes.

## Placement

PSoC Creator places the Counter component in the device based on the **Implementation** parameter. If it is set to **Fixed Function**, this component is placed in any available FF counter/timer block. If it is set to **UDB**, this component is placed around the UDB array in the best possible configuration.

# Application Programming Interface

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

By default, PSoC Creator assigns the instance name “Counter_1” to the first instance of a component in a given design. You can then rename the instance 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 “Counter.”

| Function | Description |
|---|---|
| Counter_Start() | Sets the initVar variable, calls the Counter_Init() function, and then calls the Enable function |
| Counter_Stop() | Disables the Counter |
| Counter_SetInterruptMode() | Enables or disables the sources of the interrupt output |
| Counter_ReadStatusRegister() | Returns the current state of the status register |
| Counter_ReadControlRegister() | Returns the current state of the control register |
| Counter_WriteControlRegister() | Sets the bit-field of the control register |
| Counter_WriteCounter() | Writes a new value directly into the counter register |
| Counter_ReadCounter() | Forces a capture, and then returns the capture value |
| Counter_ReadCapture() | Returns the contents of the capture register or the output of the FIFO |
| Counter_WritePeriod() | Writes the period register |
| Counter_ReadPeriod() | Reads the period register |
| Counter_WriteCompare() | Writes the compare register |
| Counter_ReadCompare() | Reads the compare register |
| Counter_SetCompareMode() | Sets the compare mode |
| Counter_SetCaptureMode() | Sets the capture mode |
| Counter_ClearFIFO() | Clears the capture FIFO |
| Counter_Sleep() | Stops the Counter and saves the user configuration |
| Counter_Wakeup() | Restores and enables the user configuration |
| Counter_Init() | Initializes or restores the Counter according to the Configure dialog settings |
| Counter_Enable() | Enables the Counter |
| Counter_SaveConfig() | Saves the Counter configuration |
| Counter_RestoreConfig() | Restores the Counter configuration |

## Global Variables

| Variable | Description |
|---|---|
| Counter_initVar | Indicates whether the Counter has been initialized. The variable is initialized to 0 and set to 1 the first time Counter_Start() is called. This allows the component to restart without reinitialization after the first call to the Counter_Start() routine.<br>If reinitialization of the component is required, then the Counter_Init() function can be called before the Counter_Start() or Counter_Enable() function. |

## void Counter_Start(void)

**Description:** This is the preferred method to begin component operation. Counter_Start() sets the initVar variable, calls the Counter_Init() function, and then calls the Counter_Enable() function.

**Parameters:** None

**Return Value:** None

**Side Effects:** If the initVar variable is already set, this function only calls the Counter_Enable() function.

## void Counter_Stop(void)

**Description:** This function disables the Counter only in software enable modes.

**Parameters:** None

**Return Value:** None

**Side Effects:** If **Enable Mode** is set to **Hardware Only**, this function has no effect.

## void Counter_SetInterruptMode(uint8 interruptSource)

**Description:** This function enables or disables the sources of the interrupt output.

**Parameters:** uint8: interrupt sources. For bit definitions, sees the Status Register section of this datasheet.

**Return Value:** None

**Side Effects:** The bit locations are different between FF and UDB. Mask #defines are provided to encapsulate the differences.

## uint8 Counter_ReadStatusRegister(void)

**Description:** This function returns the current state of the status register.

**Parameters:** None

**Return Value:** uint8: Current status register value. The status register bits are:
[7]: Unused (0)
[6]: FIFO not empty
[5]: FIFO full
[4]: Capture status
[3]: Underflow status
[2]: Overflow status
[1]: A0 Zero status
[0]: Compare output
For bit definitions, see the Status Register section of this datasheet.

**Side Effects:** Some of these bits are cleared when the status register is read. Clear-on-read bits are defined in the Status Register section of this datasheet.

## uint8 Counter_ReadControlRegister(void)

**Description:** This function returns the current state of the control register. It is available only if one of the modes defined in the control register is actually used.

**Parameters:** None

**Return Value:** uint8: Current control register value. The control register bits are:
[7]: Counter Enable
[6:5]: Unused
[4:3]: Capture Mode select
[2:0]: Compare Mode select
For bit definitions, see the Control Register section of this datasheet.

**Side Effects:** None

## void Counter_WriteControlRegister(uint8 control)

**Description:** This function sets the bit field of the control register. It is available only if one of the modes defined in the control register is actually used.

**Parameters:** uint8: Control register bit field. The control register bits are:
[7]: Counter Enable
[6:5]: Unused
[4:3]: Capture Mode select
[2:0]: Compare Mode select
For bit definitions, see the Control Register section of this datasheet.

**Return Value:** None

**Side Effects:** None

## void Counter_WriteCounter(uint8/16/32 count)

**Description:** This function writes a new value directly into the counter register. This API is not supported for the fixed-function implementation on PSoC 5 silicon.

**Parameters:** uint8/16/32: New counter value. For 24-bit Counters, the parameter is uint32.

**Return Value:** None

**Side Effects:** Overwrites the counter value. This may cause unwanted behavior on the compare output, terminal count output, or period width. This is not an atomic write and the function may be interrupted. The Counter should be disabled before calling this function.

## uint8/16/32 Counter_ReadCounter(void)

**Description:** This function forces a capture then returns the capture value.
The capture that occurs is not considered a capture event and does not cause the counter to be reset or trigger an interrupt.

**Parameters:** None

**Return Value:** uint8/16/32: Current counter value. For 24-bit Counters, the return type is uint32.

**Side Effects:** Returns the contents of the capture register or the output of the FIFO (UDB only).

## uint8/16/32 Counter_ReadCapture(void)

**Description:** This function returns the contents of the capture register or the output of the FIFO (UDB only).

**Parameters:** None

**Return Value:** uint8/16/32: Current capture value. For 24-bit Counters, the return type is uint32.

**Side Effects:** None

## void Counter_WritePeriod(uint8/16/32 period)

**Description:** This function writes the period register.

**Parameters:** uint8/16/32: New period value. For 24-bit Counters, the parameter is uint32.

**Return Value:** None

**Side Effects:** The period of the counter output does not change until the Counter is reloaded.

## uint8/16/32 Counter_ReadPeriod(void)

**Description:** This function reads the period register.

**Parameters:** None

**Return Value:** uint8/16/32: Current period value. For 24-bit Counters, the return type is uint32.

**Side Effects:** None

## void Counter_WriteCompare(uint8/16/32 compare)

**Description:** This function writes the compare register. It is available only for the UDB implementation.

**Parameters:** uint8/16/32: New compare value. For 24-bit Counters, the parameter is uint32.

**Return Value:** None

**Side Effects:** The compare output may change immediately depending on the value written and the current value of the Counter.

## uint8/16/32 Counter_ReadCompare(void)

**Description:** This function reads the compare register. It is available only for UDB implementation.

**Parameters:** None

**Return Value:** uint8/16/32: Current compare value. For 24-bit Counters, the return type is uint32.

**Side Effects:** None

## void Counter_SetCompareMode(uint8 compareMode)

**Description:** This function sets the compare mode. It is available only for UDB implementation and when software compare mode is selected.

**Parameters:** uint8: Enumerated compare mode. Also see the Control Register section.

```
Counter__B_COUNTER__LESS_THAN
Counter__B_COUNTER__LESS_THAN_OR_EQUAL
Counter__B_COUNTER__EQUAL
Counter__B_COUNTER__GREATER_THAN
Counter__B_COUNTER__GREATER_THAN_OR_EQUAL
Counter__B_COUNTER__SOFTWARE
```

Return Value: None

**Side Effects:** The compare output may change immediately depending on the value written and the current value of the counter.

## void Counter_SetCaptureMode(uint8 captureMode)

**Description:** This function sets the capture mode. It is available only for UDB implementation and when the **Capture Mode** parameter is set to **Software Controlled**.

**Parameters:** uint8: Enumerated capture mode. Also see the Control Register section.

```
Counter__B_COUNTER__NONE
Counter__B_COUNTER__RISING_EDGE
Counter__B_COUNTER__FALLING_EDGE
Counter__B_COUNTER__EITHER_EDGE
Counter__B_COUNTER__SOFTWARE_CONTROL
```

Return Value: None

**Side Effects:** None

## void Counter_ClearFIFO(void)

**Description:** This function clears the capture FIFO. It is available only for UDB implementation.

See UDB FIFOs in the Functional Description section of this datasheet.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void Counter_Sleep(void)

**Description:** This is the preferred routine to prepare the component for sleep. The Counter_Sleep() routine saves the current component state. Then it calls the Counter_Stop() function and calls Counter_SaveConfig() to save the hardware configuration.

Call the Counter_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to the PSoC Creator *System Reference Guide* for more information about power-management functions.

**Parameters:** None

**Return Value:** None

**Side Effects:** For FF implementation, all registers are retained across low-power modes. For UDB implementation, the control register and counter value register are saved and restored.

Additionally, when calling Counter_Sleep, the enable state is stored in case you call Counter_Sleep() without calling Counter_Stop().

## void Counter_Wakeup(void)

**Description:** This is the preferred routine to restore the component to the state when Counter_Sleep() was called. The Counter_Wakeup() function calls the Counter_RestoreConfig() function to restore the configuration. If the component was enabled before the Counter_Sleep() function was called, the Counter_Wakeup() function also re-enables the component.

**Parameters:** None

**Return Value:** None

**Side Effects:** Calling the Counter_Wakeup() function without first calling the Counter_Sleep() or Counter_SaveConfig() function may produce unexpected behavior.

## void Counter_Init(void)

**Description:** Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call Counter_Init() because the Counter_Start() routine calls this function and is the preferred method to begin component operation.

**Parameters:** None

**Return Value:** None

**Side Effects:** All registers will be set to values according to the customizer Configure dialog.

## void Counter_Enable(void)

**Description:** Activates the hardware and begins component operation. It is not necessary to call Counter_Enable() because the Counter_Start() routine calls this function, which is the preferred method to begin component operation. This function enables the Counter for either of the software controlled enable modes.

**Parameters:** None

**Return Value:** None

**Side Effects:** If the **Enable Mode** parameter is set to **Hardware Only**, this function has no effect on the operation of the Counter.

## void Counter_SaveConfig(void)

**Description:** This function saves the component configuration and nonretention registers. It also saves the current component parameter values, as defined in the Configure dialog or as modified by appropriate APIs. This function is called by the Counter_Sleep() function.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void Counter_RestoreConfig(void)

**Description:** This function restores the component configuration and nonretention registers. It also restores the component parameter values to what they were before calling the Counter_Sleep() function.

**Parameters:** None

**Return Value:** None

**Side Effects:** Calling this function without first calling the Counter_Sleep() or Counter_SaveConfig() function may produce unexpected behavior.

## Conditional Compilation Information

The Counter component API files require two conditional compile definitions to handle the multiple configurations the Counter must support. The API files must conditionally compile on the **Resolution** and **Implementation** parameters chosen between the FF or UDB block. The two conditions defined are based on these parameters. The API files should never use these parameters directly but should use the two defines listed here.

### Counter_DataWidth

The DataWidth define is assigned to the **Resolution** value at build time. It is used throughout the API to compile in the correct data width types for the API functions relying on this information.

### Counter_UsingFixedFunction

The UsingFixedFunction define is used mostly in the header file to make the correct register assignments. This is necessary because the registers provided in the FF block are different than those used when the component is implemented in UDBs. In some cases, this define is also used with the DataWidth define because the FF block is limited to 16 bits maximum data width.

# 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
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Counter component does not have any specific deviations.

# Sample Firmware Source Code

PSoC Creator provides many 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

## General Operation

The Counter component can count in one direction (either up or down) or both directions, depending on the **Clock Mode** parameter setting.

- If set to **Up Counter** or **Down Counter**, the component counts in only one direction. The counter register increments or decrements once for each rising edge on the count input with respect to the clock input.

- If set to **Clock Input and Direction** or **Clock With UpCnt & DwnCnt**, the component can count in both directions, based on the upCnt, dwnCnt, and up_ndown inputs. These inputs are described in detail in the Input/Output Connections section of this datasheet.

### Counter Overflow/Underflow

Counter underflow and overflow can occur in any clock mode. Bits are available in the status register to indicate when the overflow or underflow has occurred. Bits in the mode register exist to control whether an int is generated on these conditions.

| Clock Mode | Overflow occurs when | Underflow occurs when |
|---|---|---|
| Down Counter | Not defined. Interrupt generation should be masked. | Counter register equals 0. |
| Up Counter | Counter register equals period register | Not defined. Interrupt generation should be masked. |
| Clock Input and Direction | Counter register equals 0xFF, 0xFFFF, 0xFFFFFF, or 0xFFFFFFFF | Counter register equals 0. |
| Clock With UpCnt & DwnCnt | Counter register equals 0xFF, 0xFFFF, 0xFFFFFF, or 0xFFFFFFFF | Counter register equals 0. |

### Counter Outputs

The counter register can be monitored and reloaded. Two outputs, tc and comp, are available to monitor the current value of the counter register and may be configured as reload events. See the Input/Output Connections section for more details.

The counter register is reloaded from the period register. The following table shows how terminal count and reload work for each of the **Clock Mode** settings:

| Clock Mode | tc Output is Active When | Counter is Reloaded with |
|---|---|---|
| Down Counter | One clock input cycle after the counter register is equal to 0 | Contents of the period register as soon as the counter register is equal to 0 |
| Up Counter | One clock input cycle after the counter register equals the period register | Counter is reset to 0 as soon as the counter register equals the period register |
| Clock Input and Direction | One clock input cycle after the counter register rolls over to 0 | None – counter rolls over |
| Clock With UpCnt & DwnCnt | One clock input cycle after the counter register is equal to 0 | None – counter rolls over |

The comp output continually indicates the counter value compared to the compare value. The **Compare Mode** parameter is configurable to all of the standard modes (for example, **Less Than Or Equal**, **Greater Than**). This can be used to create different output waveforms while the counter is counting. The comp output is synchronous to the clock input of the Counter.

### Counter Inputs

A capture operation can be done in either hardware or firmware. The current value in the counter register is copied into either a capture register or a FIFO. Firmware can then read the captured value at a later time.

Reset and enable features allow the Counter component to be synchronized to other components. The Counter component counts only when enabled and not held in reset. It may be reset or enabled by either hardware or firmware.

### Counter Interrupt

An interrupt output is available to communicate event occurrences to the CPU or to other components. The interrupt can be set to be active on a combination of one or more events. The interrupt handler should be designed with careful consideration for determining the source of the interrupt and whether it is edge- or level-sensitive, and clearing the source of the interrupt.

### Counter Registers

There are two registers: status and control. Refer to the Registers section.

## Configurations

The following sections describe a few of the different Clock component configurations.

### Default Configuration

When you drag a Counter component onto a PSoC Creator schematic, the default configuration is an 8-bit, FF counter that decrements the counter register on a rising edge at the clock input. Figure 2 shows the default component symbol and **Configure** dialog tabs.

**Figure 2. Default Configuration**

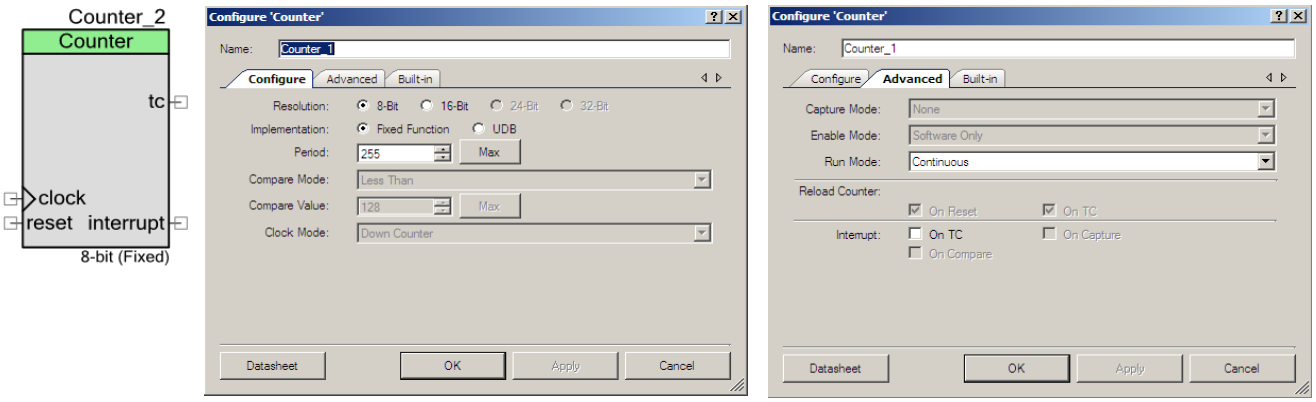
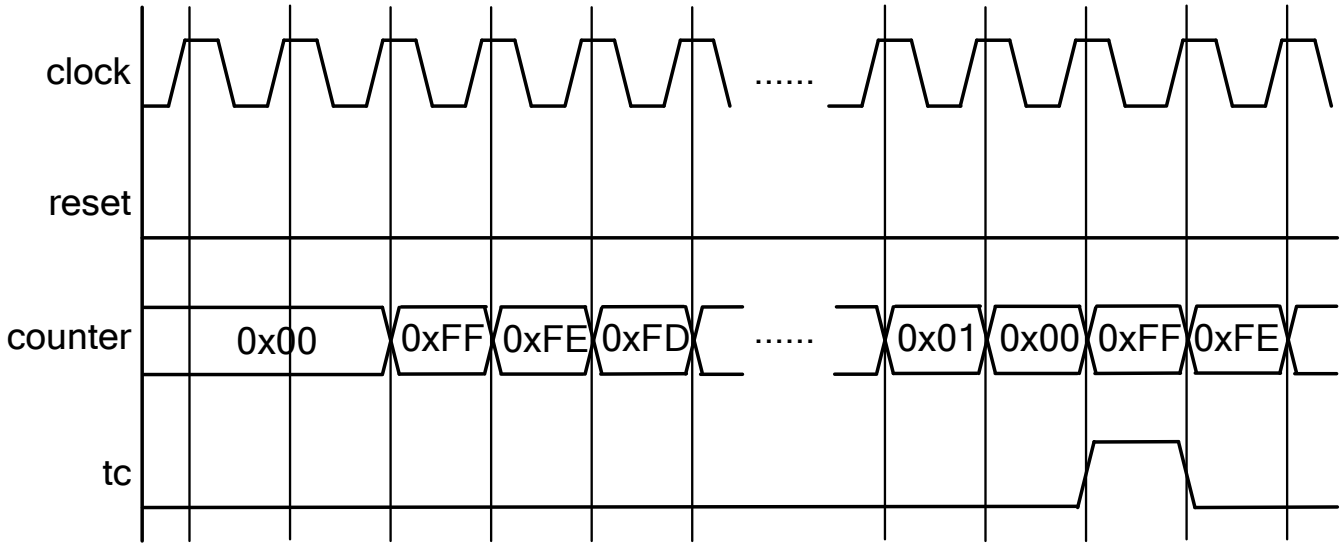



Figure 3 shows the timing diagram for the default configuration.

**Figure 3. Default Configuration Waveform**




### UDB 8-bit Up/Down Counter Configuration

In this configuration, the count register either increments or decrements on the rising edge at the clock input, depending on the clock mode selected. If the clock mode selected is **Up Counter**, the count register increments from 0 to period value. If the clock mode selected is **Down Counter**, then the count register decrements from period value to 0.

Figure 4 shows the UDB 8-bit Up/Down Counter symbol and **Configure** dialog tabs

**Figure 4. UDB 8-bit Up/Down Counter Configuration**

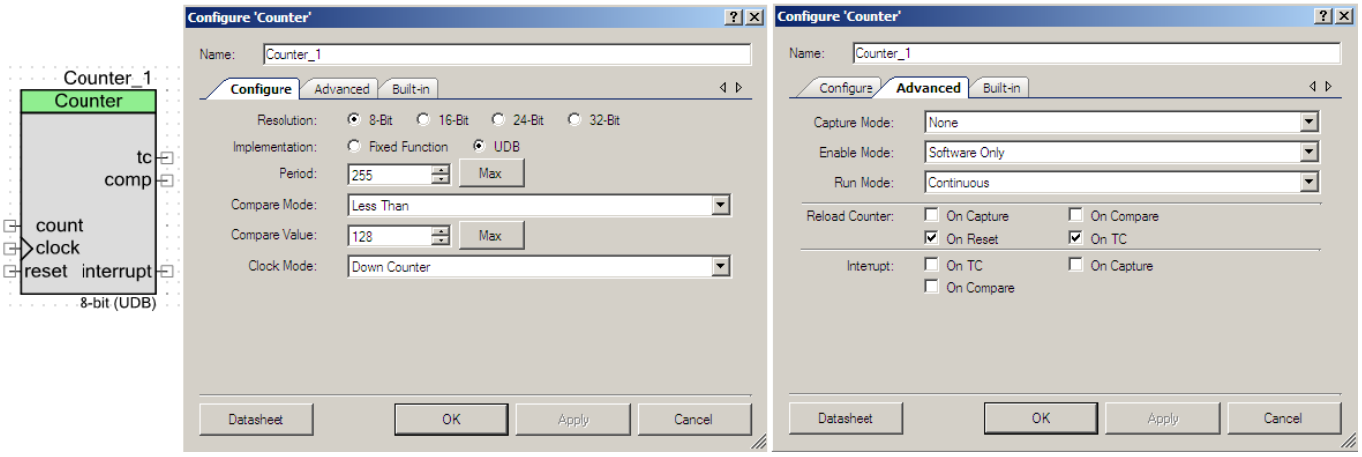
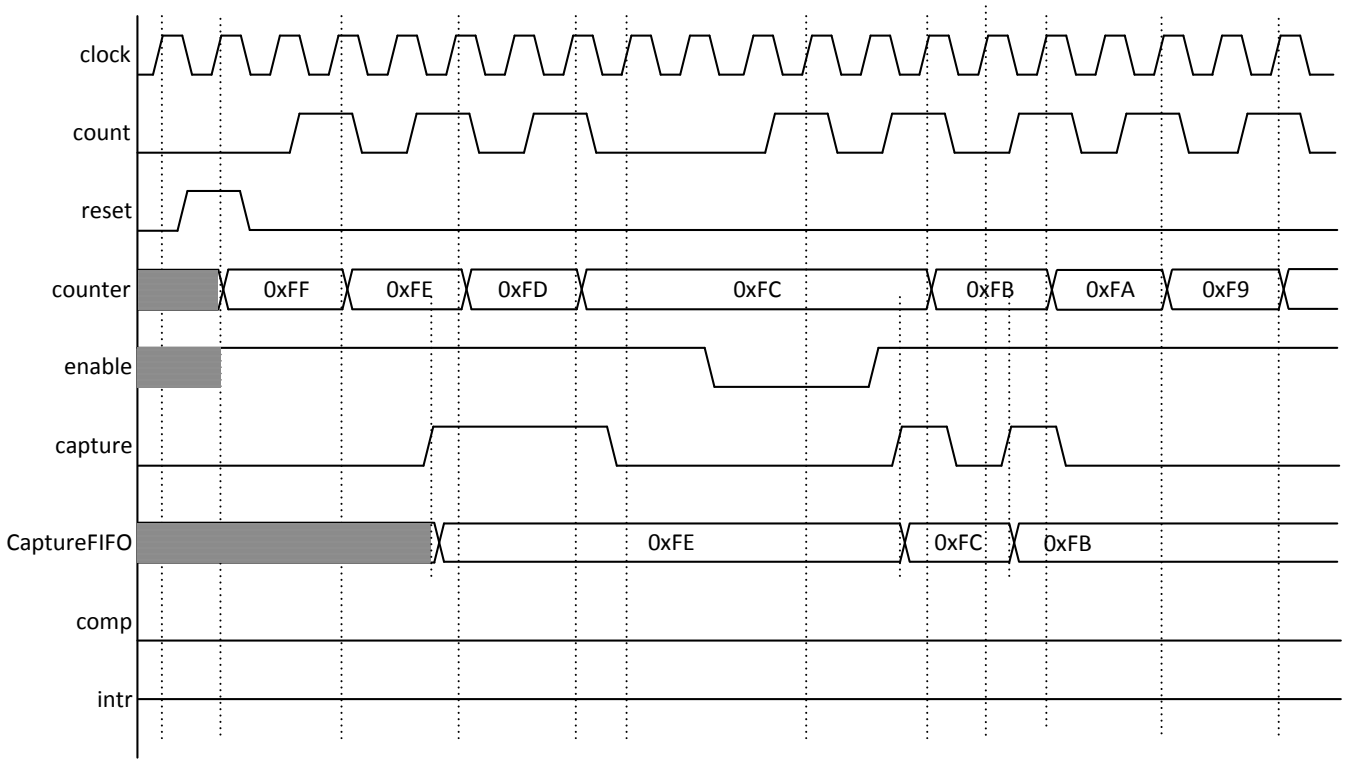



Figure 5 shows the timing diagram for the UDB 8-bit Up/Down Counter configuration.

**Figure 5. UDB 8-bit Up/Down Counter Configuration Waveform**




## Clock Input and Direction Configuration

In this configuration, the count register either increments or decrements, based on the signal to the up_ndown input terminal. When the up_ndown input receives a high signal, the counter increments on the rising edge of the count input. When the up_ndown input receives a low signal, the counter decrements on the rising edge of the count input.

Figure 6 shows the **Clock input and Direction** configuration symbol and **Configure** dialog tabs

### Figure 6. Clock Input and Direction Configuration

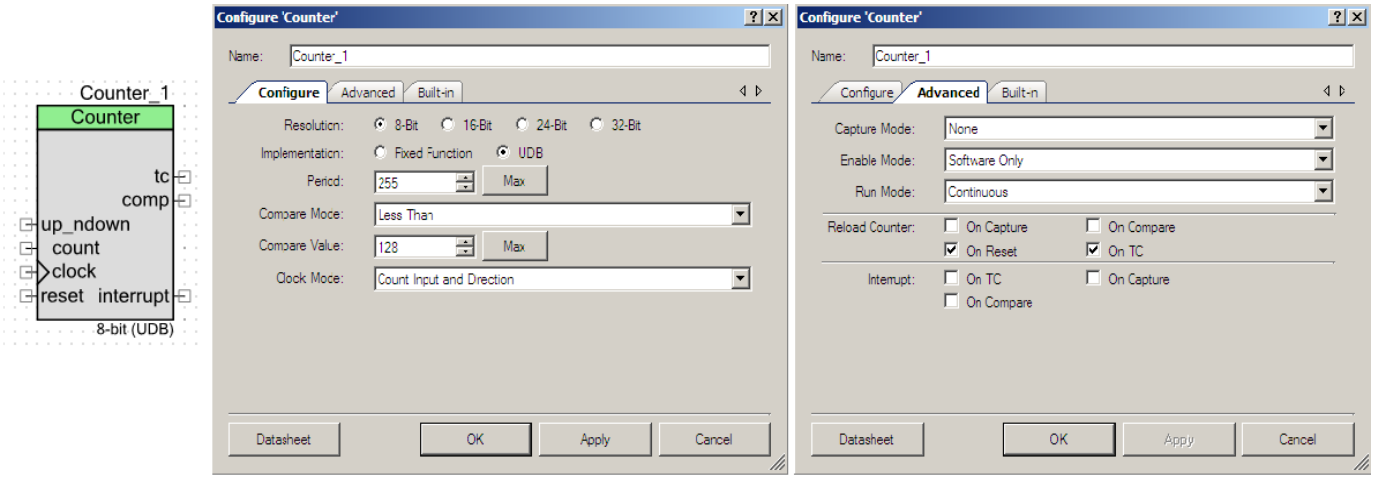
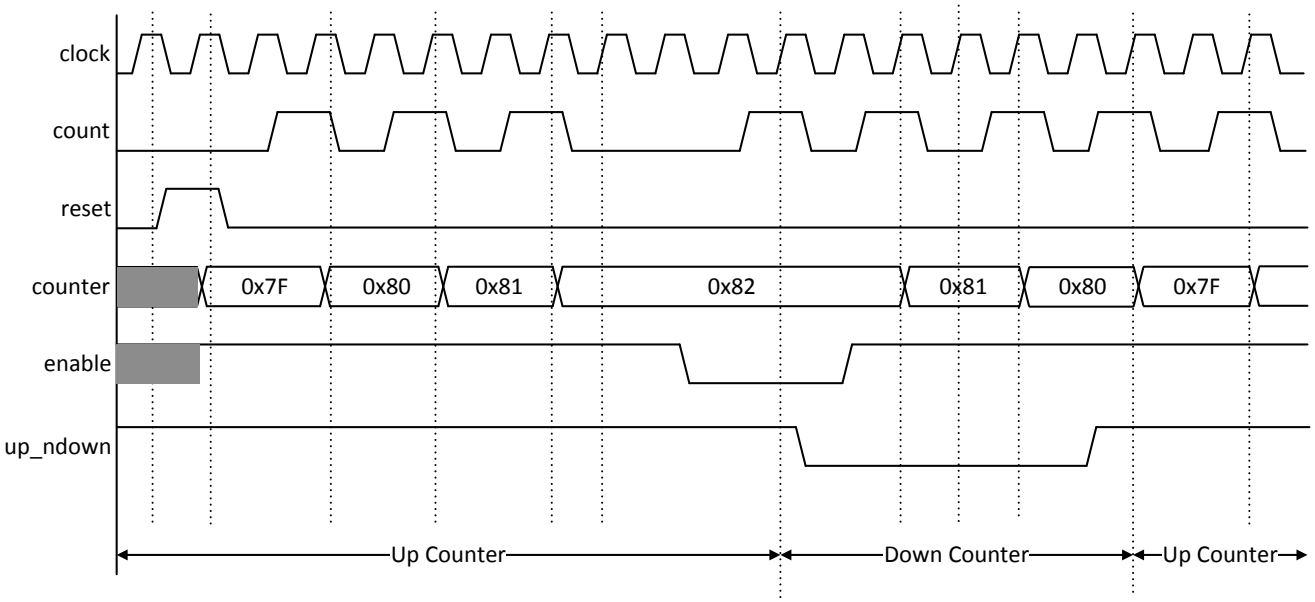



Figure 7 shows the timing diagram for the **Clock Input and Direction** mode configuration

### Figure 7. Clock Input and Direction Mode Configuration Waveform




Timing Diagram for Clock with Direction

### Clock with UpCnt and DwnCnt Configuration

In this configuration, the count input is absent. The Counter either increments or decrements, based on the signal to the upCnt and downCnt inputs.

Figure 8 shows the **Clock with UpCnt & DwnCnt** configuration symbol and **Configure** dialog tabs

## Figure 8. Clock with UpCnt and DwnCnt Mode Configuration

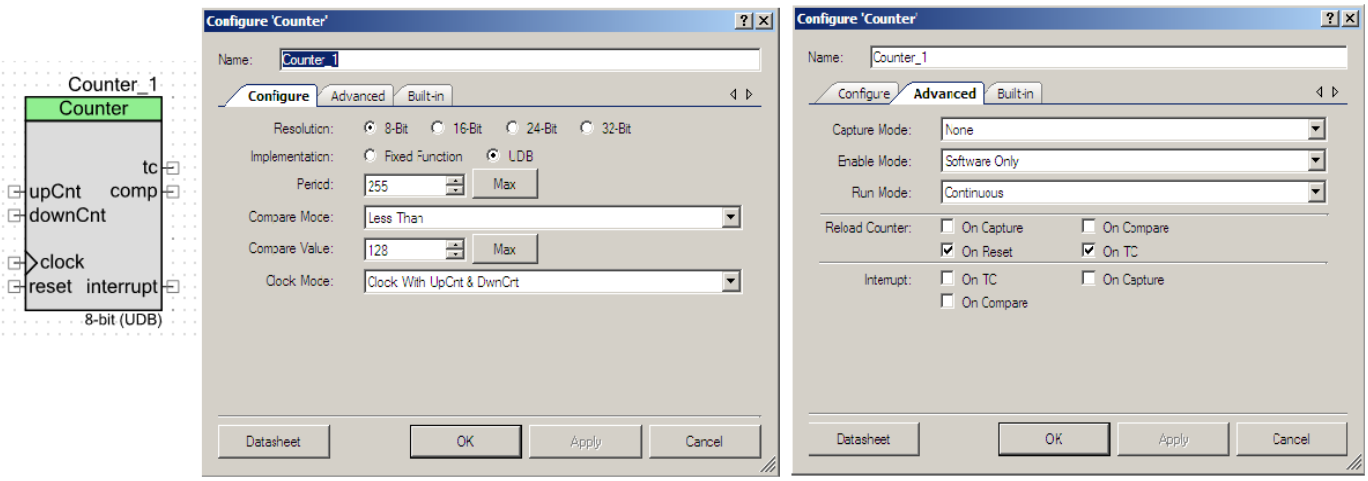
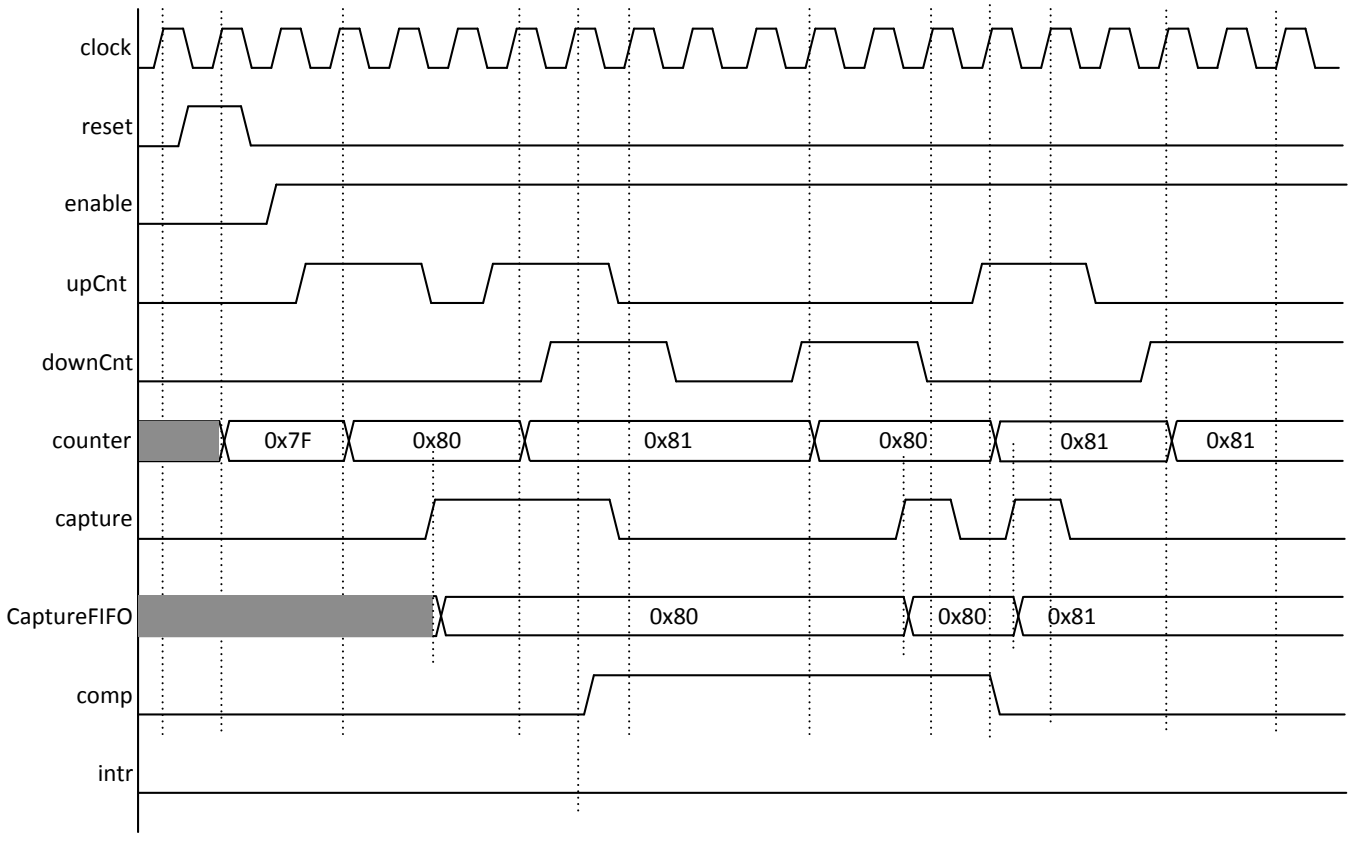



Figure 9 shows the timing diagram for the **Clock with UpCnt & DwnCnt** mode configuration

## Figure 9. Clock with UpCnt and DwnCnt Mode Configuration Waveform



### Event Counter Configuration

There are limitations on what signal can be applied to the count input when the **Implementation** parameter is set to **Fixed Function**. Therefore, setting the component to **UDB** can make it easier to create an event counter. In this configuration, intermittent asynchronous events can be detected and processed to generate a pulse. The clock input is used to sample this count input to produce a rising edge that causes the counter to increment or decrement, depending on the **Clock Mode** setting. The counter register can be captured and read by the CPU to determine the number of events that have occurred.

### Clock Divider Configuration

Changing the **Implementation** parameter setting to **UDB** also enables a comp output. This output can be used to create a clock divider with a programmable frequency and duty cycle. With the default configuration, the comp output is a ~50-percent duty cycle clock whose frequency is 1/256 the frequency of the input clock.

### Figure 10. Clock Divider Configuration

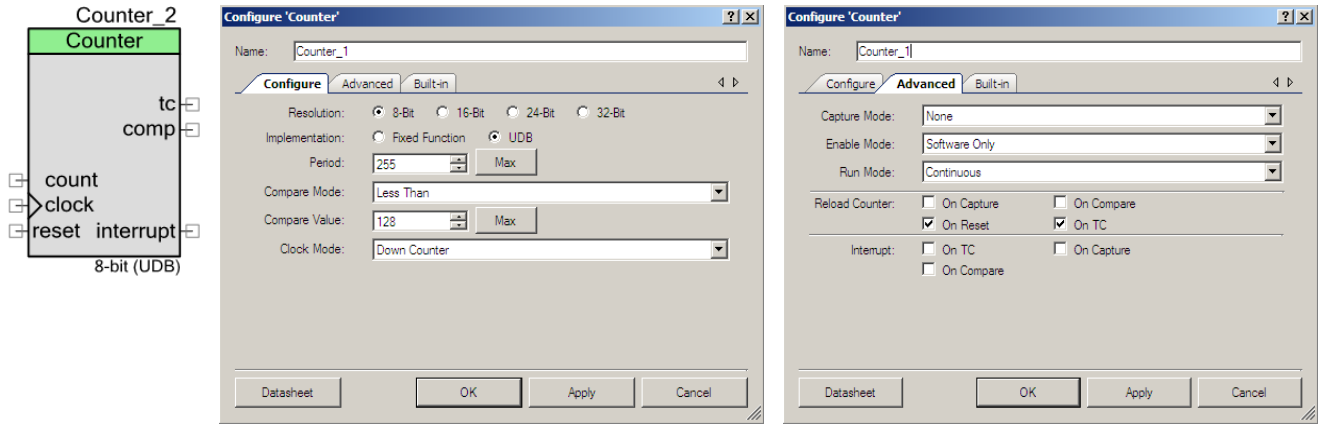
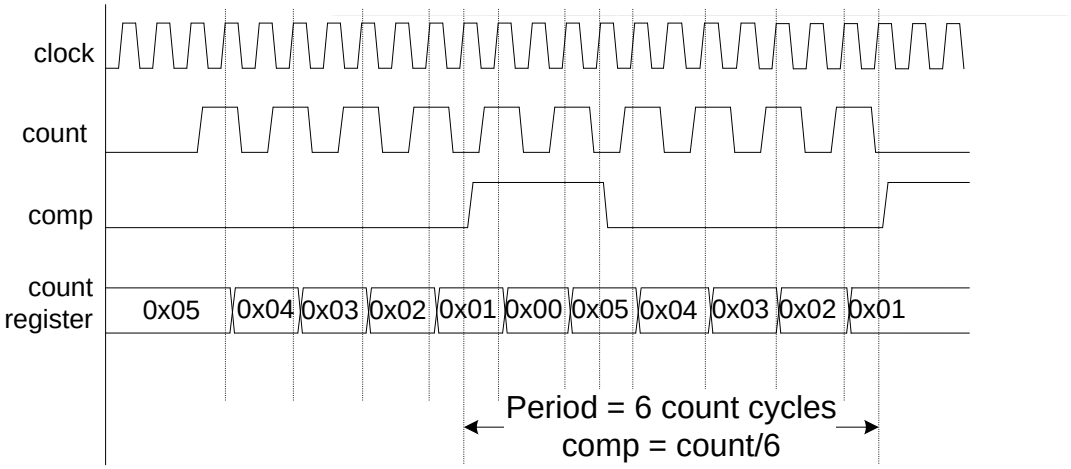



Figure 11 is an example waveform where the period is 6, the compare value is 2 and the **Compare Mode** parameter is set to **Less Than**.

### Figure 11. Clock Divider Configuration Example Waveform




## Frequency Counter Configuration

Adding hardware enable allows the Counter to implement a frequency counter function. If the enable input is driven by a known period signal, the frequency of a signal on the count input can be determined. The math can be simplified if the **Clock Mode** parameter is set to **Up Counter** instead of **Down Counter**.

### Figure 12. Frequency Counter Configuration

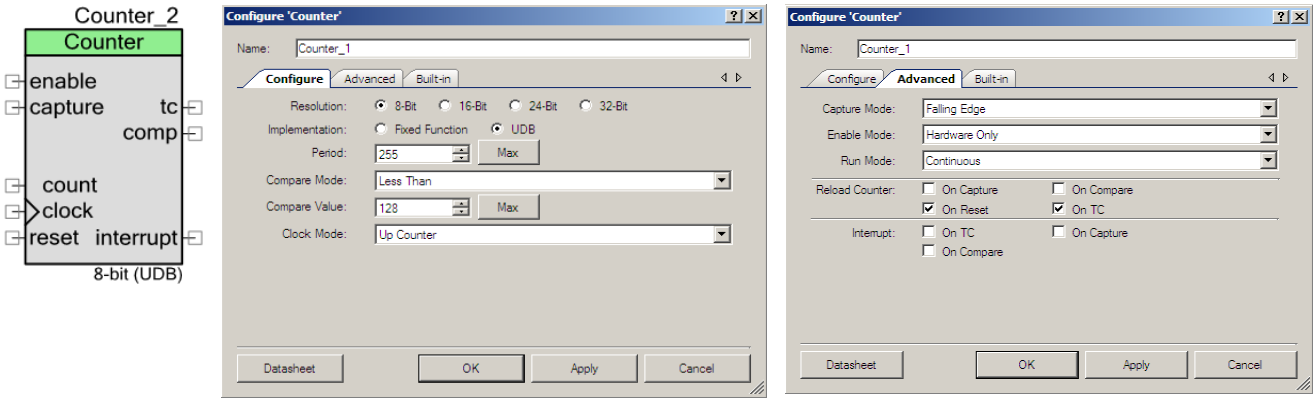
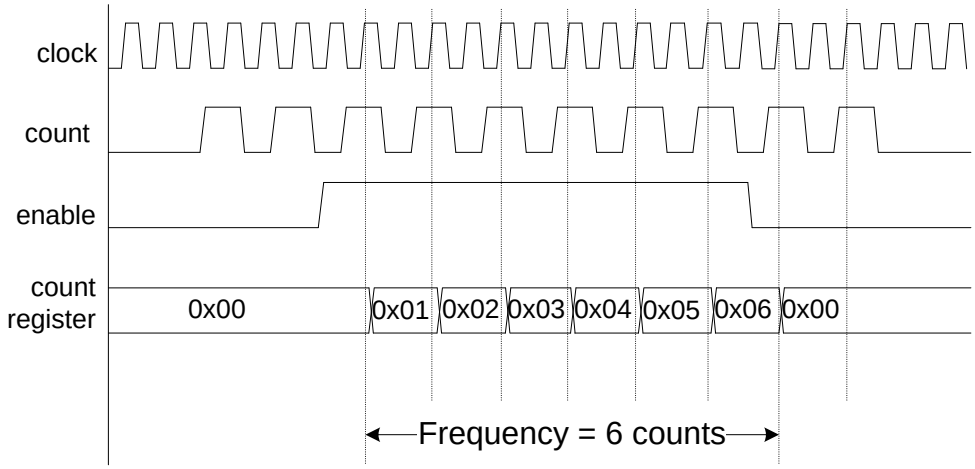
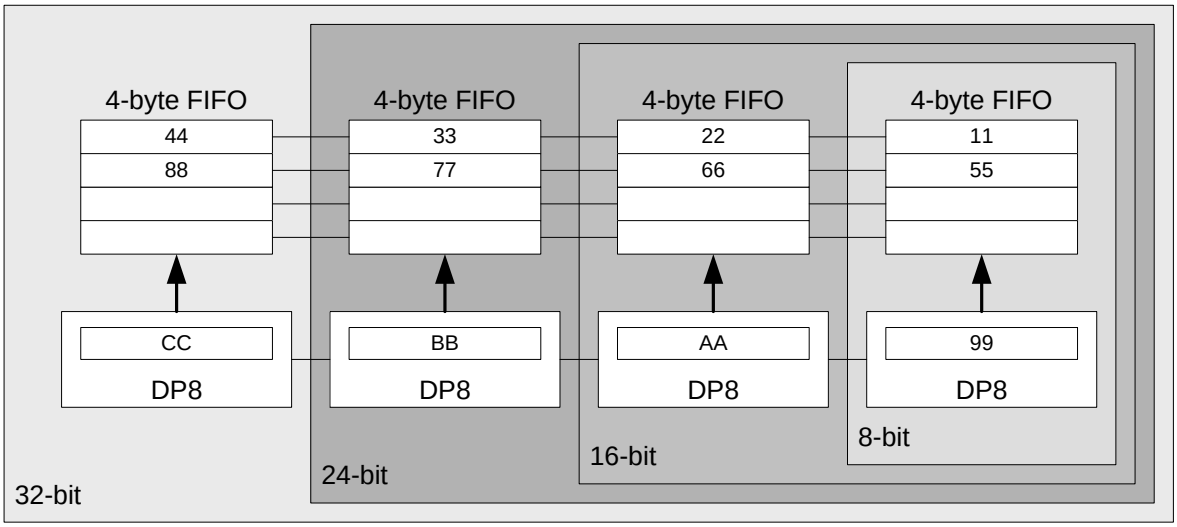


**Figure 13. Frequency Counter Configuration Example Waveform**




## UDB FIFOs

Each UDB datapath contains two 8-bit FIFO registers: F0 and F1 (see the applicable device datasheet or TRM for details). Each FIFO is four bytes deep. The Counter UDB implementation uses one of the FIFOs as a capture register. Additional FIFOs in other datapaths are used for 16-, 24- and 32-bit counters. Therefore, up to four captures can be done before the CPU must read the capture register to avoid losing data.




Capture Value #1 = 0x44332211
Capture Value #2 = 0x88776655
Accumulator = 0xCCBBAA99

# Registers

There are several constants defined to address all of the registers. Each of the register definitions requires either a pointer into the register data or a register address. Because different compilers have different endian settings, use the CY_GET_REGX and CY_SET_REGX macros for register accesses greater than 8 bits, with the _PTR definition for each of the registers. The _PTR definitions are provided in the generated header file.

## Status Register

The status register is a read-only register that contains the status bits defined for the counter. Use the Counter_ReadStatusRegister() function to read the status register value. All operations on the status register must use the following defines for the bit fields because these bit fields may be different between FF and UDB implementations.

Some bits in the status register are sticky, meaning that after they are set to 1, they retain that state until cleared when the register is read. The status data for sticky bits is registered at the input clock edge of the counter, giving all sticky bits the timing resolution of the counter. All nonsticky bits are transparent and read directly from the inputs to the status register.

### Counter_Status (UDB Implementation)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Name** | RSVD | FIFO Not Empty | FIFO Full | Capture | Underflow | Overflow | Zero | Cmp |
| **Sticky** | N/A | FALSE | FALSE | TRUE | TRUE | TRUE | TRUE | TRUE |

### Counter_Status (Fixed Function Implementation)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Name** | TC | Capture | Enable | Stop | RSVD | RSVD | RSVD | RSVD |
| **Sticky** | TRUE | TRUE | TRUE | TRUE | N/A | N/A | N/A | N/A |

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

| Bit Name | #define in header file | Description |
|---|---|---|
| Cmp | Counter_STATUS_CMP | This bit goes to 1 when the compare output is high. |
| Zero | Counter_STATUS_ZERO | This bit goes to 1 when the counter value is equal to zero. |
| Overflow | Counter_STATUS_OVERFLOW | This bit goes to 1 when the counter value is equal to the period value. |
| Underflow | Counter_STATUS_UNDERFLOW | This bit goes high when the counter value is equal to zero. |
| Capture | Counter_STATUS_CAPTURE | This bit goes to 1 whenever a valid capture event has been triggered. This does not include software capture. |

| Bit Name | #define in header file | Description |
|---|---|---|
| FIFO Full | Counter_STATUS_FIFOFULL | This bit goes to 1 when the UDB FIFO reaches the full state defined as four entries. |
| FIFO Not Empty | Counter_STATUS_FIFONEMP | This bit goes to 1 when the UDB FIFO contains at least one entry. |

## Mode Register

The mode register is a read/write register that contains the interrupt mask bits defined for the counter. Use the Counter_SetInterruptMode() function to set the mode bits. All operations on the mode register must use the following defines for the bit fields because these bit fields may be different between FF and UDB implementations.

The Counter component interrupt output is an OR function of all interrupt sources. Each source can be enabled or masked by the corresponding bit in the mode register.

### Counter_Mode (UDB Implementation)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | RSVD | FIFO Not Empty | FIFO Full | Capture | Underflow | Overflow | Zero | Cmp |

### Counter_Mode (Fixed Function Implementation)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | RSVD | RSVD | RSVD | RSVD | TC | Capture | Enable | Stop |

| Bit Name | #define in header file | Enables Interrupt Output On |
|---|---|---|
| Cmp | Counter_STATUS_CMP_INT_MASK | Compare |
| Zero | Counter_STATUS_ZERO_INT_MASK | Counter register equals 0 |
| Overflow | Counter_STATUS_OVERFLOW_INT_MASK | Counter register overflow |
| Underflow | Counter_STATUS_UNDERFLOW_INT_MASK | Counter register underflow |
| Capture | Counter_STATUS_CAPTURE_INT_MASK | Capture |
| FIFO Full | Counter_STATUS_FIFOFULL_INT_MASK | UDB FIFO full |
| FIFO Not Empty | Counter_STATUS_FIFONEMP_INT_MASK | UDB FIFO not empty |

## Control Register

The control register allows you to control the general operation of the counter. This register is written with the Counter_WriteControlRegister() function and read with the Counter_ReadControlRegister() function. All operations on the control register must use the following defines for the bit fields because these bit-fields may be different between FF and UDB implementations.

**Note** When writing to the control register, you must not change any of the reserved bits. All operations must be read-modify-write with the reserved bits masked.

### Counter_Control (UDB Implementation)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | Enable | RSVD | RSVD | Capture Mode[1:0] | | Compare Mode[2:0] | | |

### Counter_Control (Fixed Function Implementation)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | RSVD | RSVD | RSVD | RSVD | RSVD | Oneshot | RSVD | Enable |

| Bit Name | #define in header file | Description / Enumerated Type |
|---|---|---|
| Compare Mode | Counter_CTRL_CMPMODE_MASK | The compare mode control bits define the expected compare output operation. This bit field is configured at initialization with the compare mode defined in the **Compare Mode** parameter.<br>▪ Counter__B_COUNTER__CM_LESSTHAN<br>▪ Counter__B_COUNTER__CM_LESSTHANOREQUAL<br>▪ Counter__B_COUNTER__CM_EQUAL<br>▪ Counter__B_COUNTER__CM_GREATERTHAN<br>▪ Counter__B_COUNTER__CM_GREATERTHANOREQUAL |
| Capture Mode | Counter_CTRL_CAPMODE_MASK | The capture mode control bits are a two-bit field that defines the expected capture input operation. This bit field is configured at initialization with the capture mode defined in the **Capture Mode** parameter.<br>▪ Counter__B_COUNTER__CPTM_NONE<br>▪ Counter__B_COUNTER__CPTM_RISINGEDGE<br>▪ Counter__B_COUNTER__CPTM_FALLINGEDGE<br>▪ Counter__B_COUNTER__CPTM_EITHEREDGE |
| Enable | Counter_CTRL_ENABLE | This bit enables counting under software control. This bit is valid only if the **Enable Mode** parameter is set to **Software Only** or **Hardware and Software**. |

| Bit Name | #define in header file | Description / Enumerated Type |
|---|---|---|
| Oneshot | Counter_ONESHOT | This bit selects between one-shot and continuous run modes<br>▪ 1: One-shot mode<br>▪ 0: Continuous run mode |

### Counter (8-, 16-, 24-, or 32-bit based on Resolution)

The counter register contains the current counter value. This register is incremented or decremented in response to various count/clock inputs. This register may be read at any time with the Counter_ReadCounter() function.

### Capture (8-, 16-, 24-, or 32-bit based on Resolution)

The capture register contains the captured counter value. Any capture event copies the counter register to this register. In the UDB implementation, this register is actually a FIFO. See the UDB FIFOs section for details.

### Period (8-, 16-, 24-, or 32-bit based on Resolution)

The period register contains the period value set through the Counter_WritePeriod() function and defined by the **Period** parameter at initialization. The period register is copied into the counter register on a reload event.

### Compare (8-, 16-, 24-, or 32-bit based on Resolution)

The compare register contains the compare value used to determine the state of the compare (comp) output.

## Resources

Depending on the Implementation parameter Counter component uses one FF counter/timer block or is placed throughout the UDB array. The UDB Implementation utilizes the following resources.

| Configuration | Resource Type | | | | | |
|---|---|---|---|---|---|---|
| | Datapath Cells | Macrocells | Status Cells | Control Cells | DMA Channels | Interrupts |
| 8-bits Up Counter | 1 | 6 | 1 | 1 | – | – |
| 8-bits Counter with direction | 1 | 9 | 1 | 1 | – | – |
| 16-bits Up Counter | 2 | 7 | 1 | 1 | – | – |
| 16-bits Counter with direction | 2 | 9 | 1 | 1 | – | – |

| Configuration | Resource Type | | | | | |
|---|---|---|---|---|---|---|
| | Datapath Cells | Macrocells | Status Cells | Control Cells | DMA Channels | Interrupts |
| 24-bits Up Counter | 3 | 6 | 1 | 1 | – | – |
| 32-bits Up Counter | 4 | 7 | 1 | 1 | – | – |

# API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.
The measurements have been 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 3 (Keil_PK51) | | PSoC 4 (GCC) | | PSoC 5LP (GCC) | |
|---|---|---|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes |
| 8-bits FF Counter | 257 | 5 | N/A | N/A | 376 | 5 |
| 16-bits FF Counter | 261 | 6 | N/A | N/A | 376 | 9 |
| 8-bits UDB Up Counter | 265 | 5 | 412 | 5 | 460 | 5 |
| 8-bits UDB Counter with direction | 265 | 5 | 412 | 5 | 460 | 5 |
| 16-bits Up Counter | 316 | 6 | 412 | 5 | 460 | 9 |
| 16-bits Counter with direction | 317 | 6 | 412 | 5 | 460 | 9 |
| 24-bits UDB Up Counter | 307 | 8 | 436 | 5 | 472 | 13 |
| 32-bits UDB Up Counter | 306 | 8 | 412 | 5 | 460 | 13 |

# DC and AC Electrical Characteristics for PSoC 3 (FF Implementation)

The following values indicate expected performance and are based on initial characterization data.

## Counter DC Specifications

| Parameter | Description | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Block current consumption | 16-bit counter, at listed input clock frequency | – | – | – | µA |
| | 3 MHz | | – | 15 | – | µA |
| | 12 MHz | | – | 60 | -- | µA |
| | 48 MHz | | – | 260 | – | µA |
| | 67 MHz | | – | 350 | – | µA |

## Counter AC Specifications

| Parameter | Description | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Operating frequency | | DC | – | 67 | MHz |
| | Capture pulse | | 15 | – | – | ns |
| | Resolution | | 15 | – | – | ns |
| | Pulse width | | 15 | – | – | ns |
| | Pulse width (external) | | 30 | – | – | ns |
| | Enable pulse width | | 15 | – | – | ns |
| | Enable pulse width (external) | | 30 | – | – | ns |
| | Reset pulse width | | 15 | – | – | ns |
| | Reset pulse width (external) | | 30 | – | – | Ns |

# DC and AC Electrical Characteristics for PSoC 5 (FF Implementation)

The following values indicate expected performance and are based on initial characterization data.

## Counter DC Specifications

| Parameter | Description | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Block current consumption | 16-bit counter, at listed input clock frequency | – | – | – | µA |
| | 3 MHz | | – | 15 | – | µA |
| | 12 MHz | | – | 60 | -- | µA |
| | 48 MHz | | – | 260 | – | µA |
| | 67 MHz | | – | 350 | – | µA |

## Counter AC Specifications

| Parameter | Description | Conditions | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Operating frequency | | DC | – | 67.01 | MHz |
| | Capture pulse | | 13 | – | – | ns |
| | Resolution | | 13 | – | – | ns |
| | Pulse width | | 13 | – | – | ns |
| | Pulse width (external) | | 30 | – | – | ns |
| | Enable pulse width | | 13 | – | – | ns |
| | Enable pulse width (external) | | 30 | – | – | ns |
| | Reset pulse width | | 13 | – | – | ns |
| | Reset pulse width (external) | | 30 | – | – | ns |

# DC and AC Electrical Characteristics (UDB Implementation)

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.

### Table 1. DC Characteristics

| Parameter | Description | Min | Typ[1] | Max | Units |
|---|---|---|---|---|---|
| $I_{DD}$ | Component current consumption | | | | |
| | 8-bits UDB Up Counter | – | 10 | – | µA/MHz |
| | 8-bits UDB Counter with direction | – | 10 | – | µA/MHz |
| | 16-bits Up Counter | – | 16 | – | µA/MHz |
| | 16-bits Counter with direction | – | 15 | – | µA/MHz |
| | 24-bits UDB Up Counter | – | 31 | – | µA/MHz |
| | 32-bits UDB Up Counter | – | 32 | – | µA/MHz |

### Table 2. AC Characteristics

| Parameter | Description | Min | Typ | Max[2] | Units |
|---|---|---|---|---|---|
| $f_{CLOCK}$ | Component clock frequency | | | | |
| | 8-bits UDB Up Counter | – | – | 39 | MHz |
| | 8-bits UDB Counter with direction | – | – | 39 | MHz |
| | 16-bits Up Counter | – | – | 33 | MHz |
| | 16-bits Counter with direction | – | – | 33 | MHz |
| | 24-bits UDB Up Counter | – | – | 29 | MHz |
| | 32-bits UDB Up Counter | – | – | 26 | MHz |

1. Device IO and clock distribution current not included. The values are at 25 °C.

2 The values provide a maximum safe operating frequency of the component. The component may run at higher clock frequencies, at which point validation of the timing requirements with STA results is necessary.

# Component Changes

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

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 2.40 | Updated datasheet with memory usage for PSoC 4 | |
| 2.30 | Added MISRA Compliance section. | The component does not have any specific deviations. |
| 2.20 | Updated Counter_WriteCounter() API | Issue with Counter_WriteCounter() API for fixed-function counter. |
| | Updated Default Configuration waveform in the Counter datasheet. | Default configuration waveform shown in the counter datasheet is wrong. |
| | Updated Counter_Control register (for fixed function) description in the datasheet. | Fixed function Counter_Control register description was wrong. |
| | Updated screenshots of Counter configure window in the datasheet | Updates to remove frequency calculations field from Counter configure window. |
| 2.10 | Customizer related updates for UDB implementation. | The compare value is now allowed to take any values in the full range of resolution for modes that count both up and down. |
| | Updated Counter_RestoreConfig() API | To fix an issue with interrupt trigger after wakeup from low power mode. |
| | Added PSoC 5 DC and AC FF characteristics to datasheet | |
| 2.0.a | Updated Resource information in datasheet | |
| | Added additional configuration sections to datasheet | |
| 2.0 | Redesigned as a synchronous counter with oversampling in all modes. | The architecture of the device and the tool has proven that a synchronous design is the only viable solution. All modes are still supported but require a clock for oversampling. |
| | Removed “comp” terminal from fixed-function implementation | The implementation does not support a compare output. The pin is now correctly hidden. |
| | Synchronized inputs | All inputs are synchronized in fixed-function implementation, at the input of the block. |
| | Converted Counter_GetInterruptSource() function to a macro | The Counter_GetInterruptSource() function is exactly the same implementation as the Counter_ReadStatusRegister() function. To save code space, this was converted to a macro substitution of the Counter_ReadStatusRegister() function. |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| | Outputs are now registered to the component clock | To avoid glitches on the outputs of the component it was required that all outputs be synchronized. This is done inside of the datapath when possible, to avoid using too many resources. |
| | Implemented critical regions when writing to Aux Control registers. | CyEnterCriticalSection and CyExitCriticalSections functions are used when writing to Aux Control registers so that it is not modified by any other process thread. |
| | Added characterization data to datasheet | |
| | Minor datasheet edits and updates | |

© Cypress Semiconductor Corporation, 2013. 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 PARTICULAR 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.