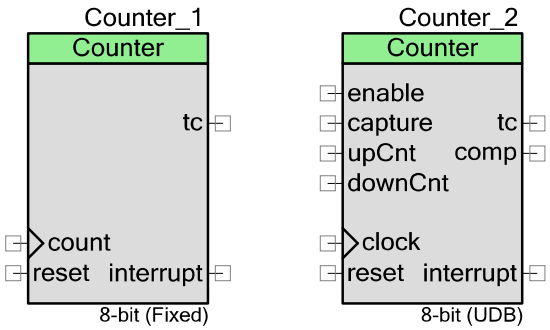
Counter
2.0

Features



- Fixed function (FF) and universal digital block (UDB) implementations
- 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 it offers advanced features such as capture, compare output, and count direction control.

This component can be implemented using FF blocks or UDBs. A UDB implementation typically has more features than an FF implementation. If your design is simple enough, consider using FF and conserve UBD 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
No. 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
Capture mode	None	Rising edge, falling edge, either edge, or software controlled
Capture FIFO	No (One capture register)	Yes (Up to four captures)

Feature	FF	UDB
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/or 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 like 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/ count	N	Under the fixed function implementation, the clock input defines the signal to count. The counter is decremented on every rising edge of the clock input. Under the UDB implementation, depending on the 4 modes the inputs clock and count may be hidden or shown • Up Counter /Down Counter: The count input defines the signal to count and is sampled with respect to the clock input. At every rising edge of the count input, the Counter counts up or down depending on the **Clock Mode** parameter. The counter counts synchronously to the clock input • Clock + upCnt & dwnCnt: The upCnt and dwnCnt terminals signify the count direction, whether the counter has to count up or count down or NOP with respect to the clock input. The count input is invisible in this mode Clock with Direction: The clock input defines the signal to count. The direction input defines the direction of the counter to count up or down. The counter is synchronous to the clock input and operates on its rising edge. At the maximum, the count input can be half as much as the frequency of the clock input.
upCnt	Y	Increment signal to the counter. When **Clock Mode** is set to "Clock + upCnt & dwnCnt" this input is used in conjunction with the dwnCnt input and the clock input to provide the ability for the Counter to be used as an encoder. A rising edge on this input increments the count value.
dwnCnt	Y	Decrement signal to the counter. When **Clock Mode** of the counter is set to "Clock + upCnt & dwnCnt" this input is used in conjunction with the upCnt input and the clock input to provide the ability for the counter to be used as an encoder. A rising edge on this input decrements the count value.
up_ndown	Y	Defines the counting direction of the counter. This input is only available if **Clock Mode** is set to enable direction control ("Count + Direction Inputs"). 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. • For the "Up Counter" configuration, the starting value is zero. • For "Down Counter", "Count Input and Direction" and "Clock With UpCnt & DwnCnt" configurations, the starting value is set to the current period register value The reset input is sampled on the count/clock input. **Note** For PSoC 3 ES2 silicon, the Terminal Count pin for the fixed function counter is held HIGH in Reset. A schematic fix for this is provided under "Reset in Fixed Function Block" in the Functional Description section of this datasheet. For PSoC 3 ES3 or later silicon, the Terminal Count pin for the fixed function counter is held LOW in Reset. The reset input is used to reset the control register in PSoC3 ES3 and PSOC5 ES1 UDB counter implementation, and further to implement One-Shot Run Mode feature

Input	May Be Hidden	Description
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. The capture input is sampled on the clock input.

Output	May Be Hidden	Description
tc	N	Terminal count is a synchronous output which 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. When implemented as a down counter, terminal count value is zero. When implemented as an up counter, this value is the max count (period). If the Counter is disabled when the count is at terminal count, the output will stay high until the Counter is re-enabled.
comp	Y	The compare output indicates the counter value compared to the compare value based on the configuration in the **Compare Mode** parameter. The output goes high one clock cycle of the clock input, after the compare event has taken place.
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 a "Capture".

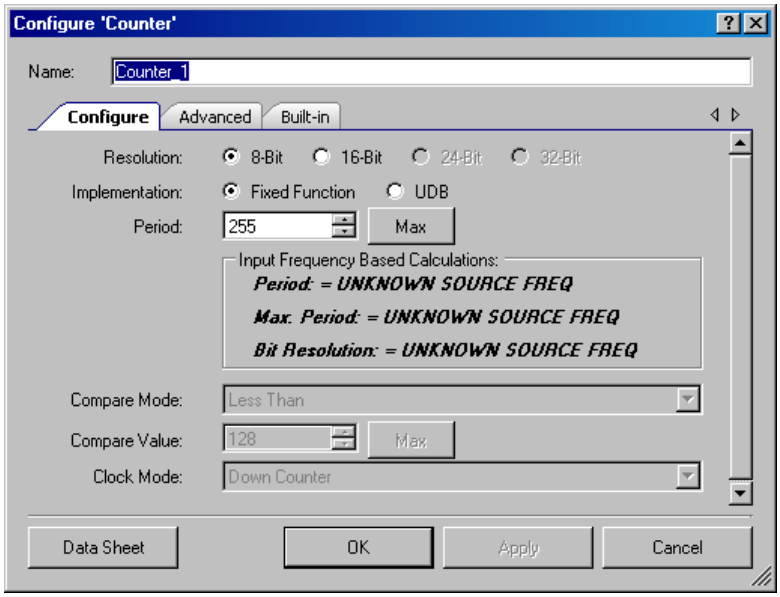
Parameters and Setup

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

Hardware vs. 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

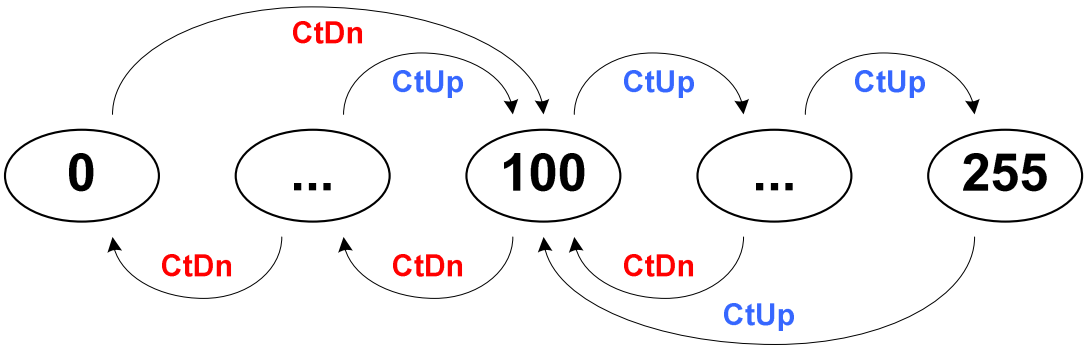
The **Implementation** parameter allows you to choose between a fixed function block implementation and a UDB implementation of the Counter. If FF is selected, then UDB functions become disabled.

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 can be changed at any time by the software 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 + Direction," the counter is set to the period at start and any time the counter overflows or underflows. In these clock modes, the period value should not be 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 8 bit counter, 0x7F). The following figure shows **Clock Mode** set to "Count Input + Direction."



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. It may be updated 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 To" – 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 To" – The counter value is greater than or equal to the compare value
- "Software Controlled" – The compare mode can be set during runtime with the Counter_SetCompareMode() API call to any one of the five compare modes listed above.

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 selected to define the operation of the compare output.

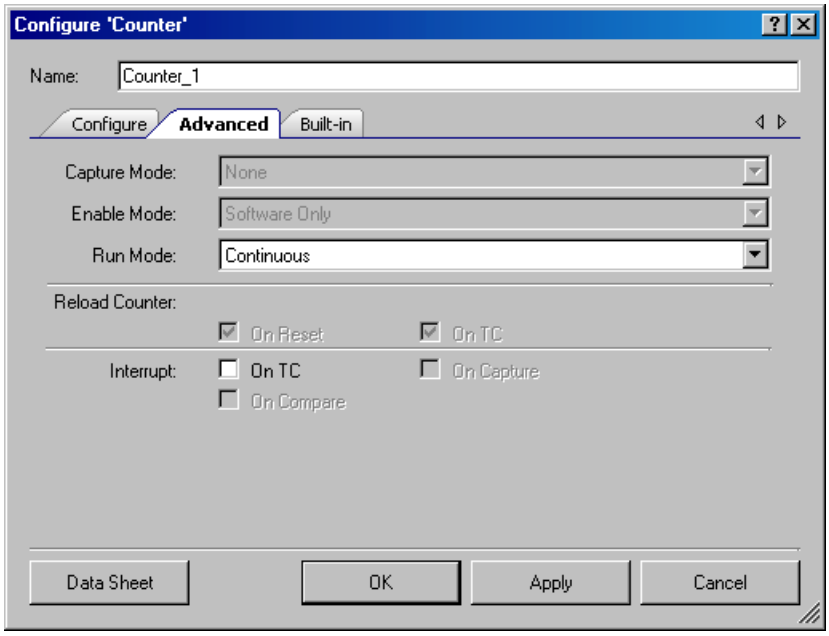
This value can be any unsigned integer from 0 to (2^Resolution-1), but it must be less than or equal to the period value.

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 + Direction" – Counter is a bi-directional counter counting up while the up_ndown input is high on each rising edge of the input clock and counting down while up_ndown is low on each rising edge of the input clock.
- "Clock with UpCnt & DwnCnt" – Counter is a bi-directional counter incrementing the counter for each rising edge on the upCnt input with respect to the clock input and decrementing the counter for each rising edge of the dwnCnt input with respect to the clock input. In this mode the clock input freq must be at least 2x the freq of UpCnt and DwnCnt inputs.
- "Up Counter" – Counter is an up counter only. It is configured to increment 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 is configured to decrement 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 and 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" – For Software Controlled mode, set the mode at runtime by setting the capture mode bits in the control register Counter_CTRL_CAPMODE_MASK with the enumerated capture mode types 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" – The Counter is enabled based on the enable bit of the control register only
- "Hardware" – The Counter is enabled based on the enable input only,
- "Software and Hardware" – 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 (for an up counter this is reloaded to a value of "Zero," for a down counter this is reloaded to the max counts or period value).

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

Interrupt

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

- On TC –This parameter is always active; it is set to "false" by default.
- On Capture – This parameter is set to "false" by default. It is always shown, but it is only active when the **Implementation** parameter is set to "UDB."
- On Compare – This parameter is set to "false" by default. It is always shown, but it is only active when the **Implementation** parameter is set to "UDB."

Clock Selection

For the Counter component,

- When **Clock Mode** parameter is set to "Up Counter" or "Down Counter," the count input can be any signal of which the rising edges are to be counter. The clock input to the component is used to sample the count input and shall be at least twice the frequency of that of the count input.
- When the **Clock Mode** parameter is set to "Count input with Direction'" the clock input can be any signal of which the rising edges are to be counter. The counter shall 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 shall count on a rising edge on the upCnt signal and count down on the rising edge on the dwnCnt signal.

See the Clock component datasheet and the appropriate device datasheet for more details on PSoC 3 or PSoC 5 clocking system.

Fixed Function Components

When configured to utilize the FF block in the device, the Counter component will have 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, then 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, etc.

For 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 specifications section in this datasheet. The count input shall 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.

Resources

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
8 Bits UDB Basic Up or Down Counter	1	6	1	1	0	240	4	0
8 Bits UDB Counter with direction*	1	10	1	1	0	240	4	0
8 Bits FF Counter	0	0	0	0	0	352	5	0
16 Bits UDB Basic Up or Down Counter	2	6	1	1	0	298	5	0
16 Bits UDB Counter with direction*	2	14	1	1	0	452	6	0
16 Bits FF Counter	0	0	0	0	0	372	6	0
24 Bits Basic Counter	3	6	1	1	0	448	8	0
32 Bits Basic Up or Down Counter	4	6	1	1	0	448	8	0

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.

Function	Description
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 per 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. 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: 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: Enables or disables the sources of the interrupt output.

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

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

uint8 Counter_ReadControlRegister(void)

Description: Returns the current state of the control register. This function 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, refer to the Control Register section of this datasheet.

Side Effects: None

void Counter_WriteControlRegister(uint8 control)

Description: Sets the bit-field of the control register. This function 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, refer to the Control Register section of this datasheet.

Return Value: None

Side Effects: None

void Counter_WriteCounter (uint8/16/32 count)

Description: Writes a new value directly into the counter register.

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 undesired 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: Forces a capture, and then returns the capture value.

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: 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: 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 counter is reloaded.

uint8/16/32 Counter_ReadPeriod (void)

Description: 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: Writes the compare register. This function is available only for 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: Reads the compare register. This function 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: Sets the compare mode. This function is available only for UDB implementation and when software compare mode is selected.

Parameters: (uint8) Enumerated compare mode. Refer also to 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: Sets the capture mode. This function is available only for UDB implementation and when Capture Mode parameter is set to Software Controlled.

Parameters: (uint8) Enumerated capture mode. Refer also to 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: Clears the capture FIFO. This function is available only for UDB implementation. Refer to the "UDB FIFOs" section 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 the FF implementation, all registers are retained across low power modes. For the 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 will also re-enable 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," then this function has no effect on the operation of the Counter.

void Counter_SaveConfig(void)

Description: This function saves the component configuration and non-retention registers. This function will also save 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 non-retention registers. This function will also restore the component parameter values to what they were prior to 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 APIs require two conditional compile definitions to handle the multiple configurations it must support. It is required that the API 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 should never use these parameters directly but should use the two define list below.

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 as 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 bit's maximum data width.

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

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

Functional Description

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 per rising edge on the count input with respect to the clock input.
- If set to "Clock Input + 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 have occurred and 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 Mode	Overflow occurs when	Underflow occurs when
Clock Input + 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 may be monitored and reloaded. Two outputs, tc and comp, are available for monitoring the current value of the counter register and may be configured as reload events. Refer to the Input/Output Connections section for more details.

The counter register is reloaded from a 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 Counter register is equal to 0	contents of period register as soon as Counter register is equal to 0
Up Counter	One clock input cycle after Counter register equals the period register	Counter is reset to 0 as soon as counter register equals period register
Clock Input + Direction	One clock input cycle after Counter register rolls over to 0	None – counter rolls over
Clock with UpCnt & DwnCnt	One clock input cycle after 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 (e.g., "Less Than Or Equal," "Greater Than," etc.). 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 may 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 may 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 for communicating 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

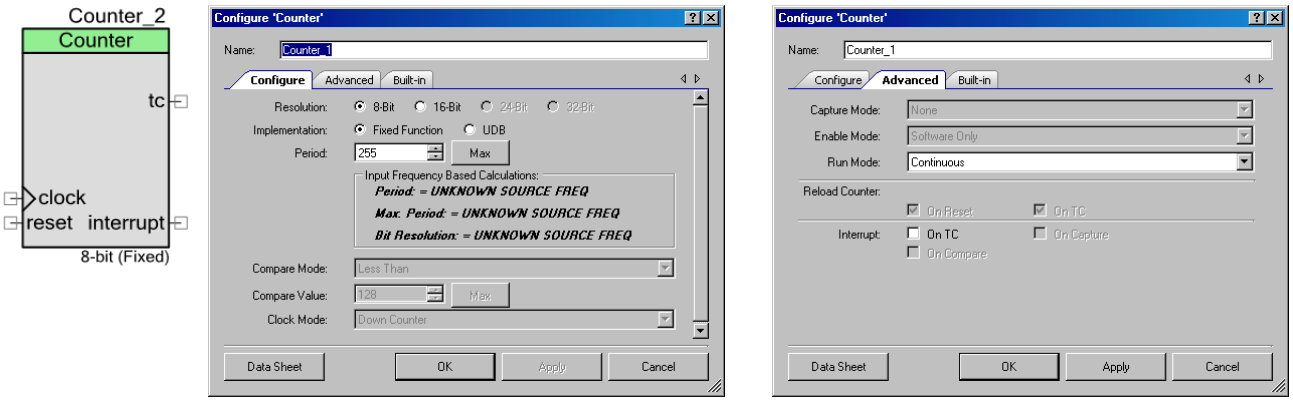
There are two registers: status and control. Refer to the Registers section.

Configurations

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

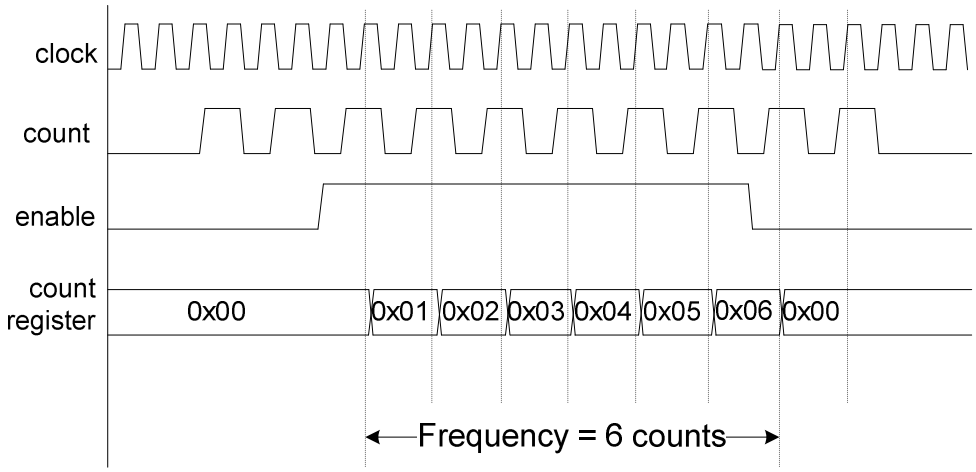
Default Configuration

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 count input. The following diagram shows the default component symbol and Configure dialog tabs.



The following figure shows the timing diagram for the default configuration.

Figure 1. Default Configuration Waveform

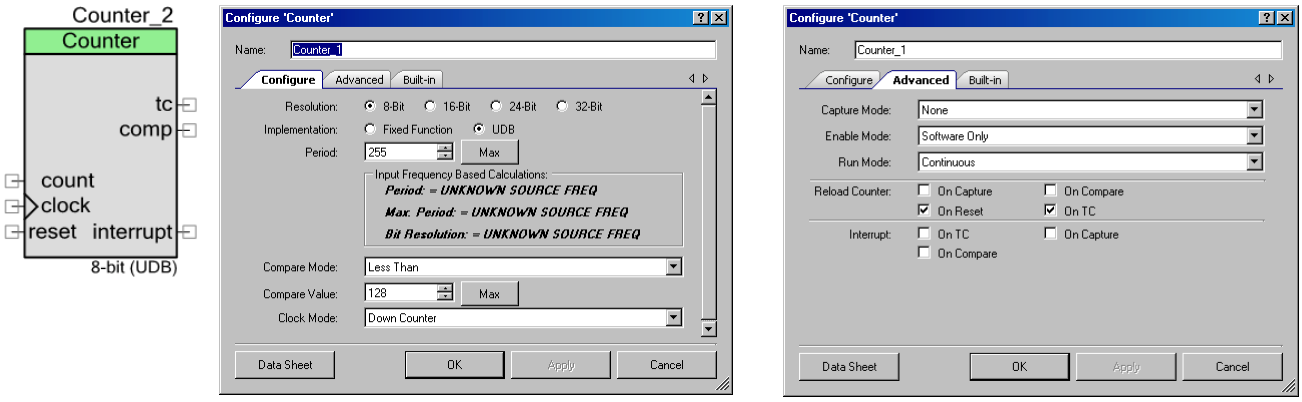


Event Counter Configuration

There are limitations on what signal may 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 may 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.

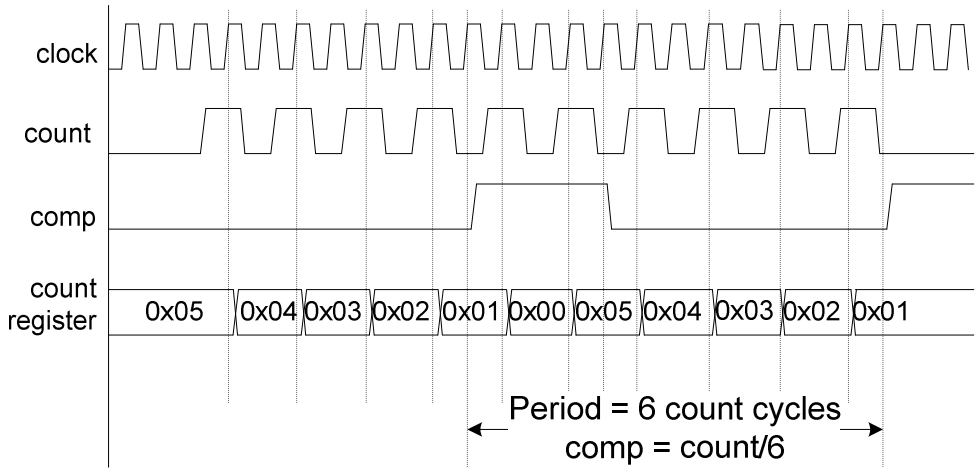
Clock Divider Configuration

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% duty cycle clock whose frequency is 1/256 the frequency of the input clock.



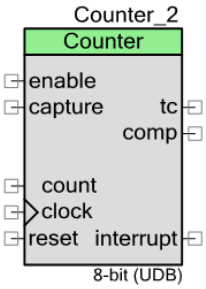
The following figure is an example waveform where the period is 6, the compare value is 2 and the **Compare Mode** parameter is set to "Less Than."

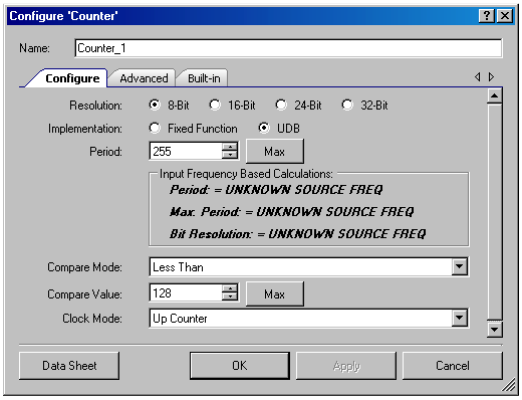
Figure 2. Clock Divider Configuration Example Waveform



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, then the frequency of a signal on the count input can be determined. The math may be simplified if the **Clock Mode** parameter is set to "Up Counter" instead of "Down Counter."





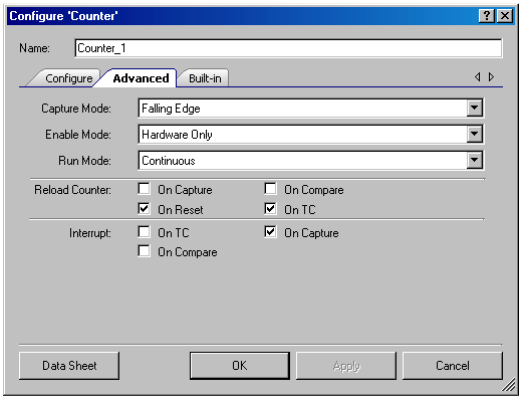
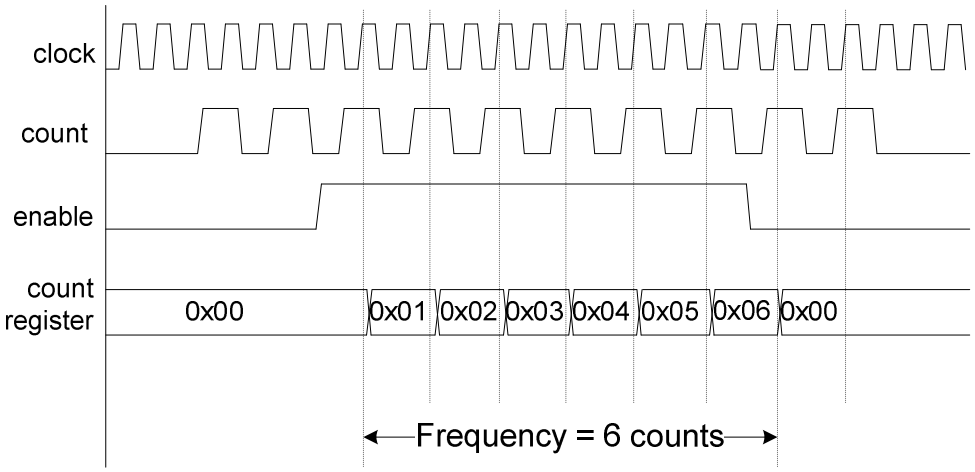


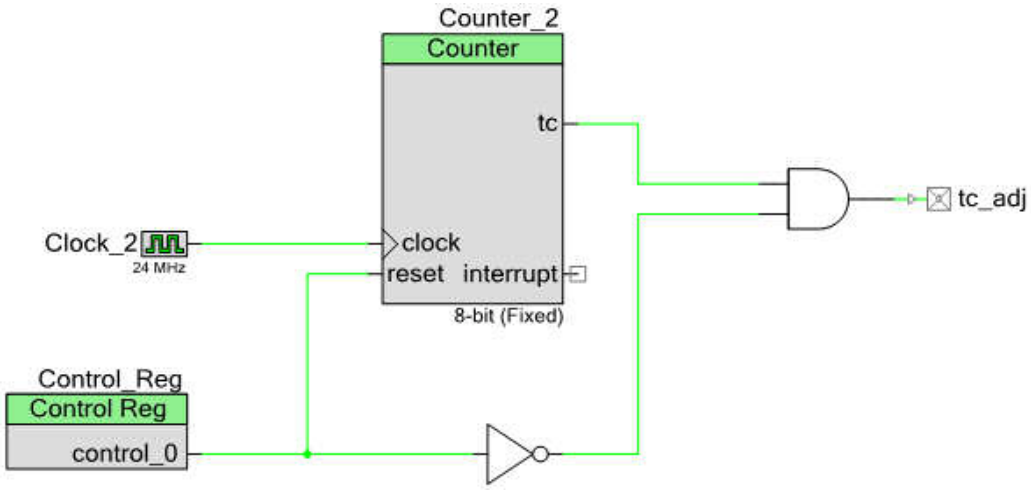
Figure 3. Frequency Counter Configuration Example Waveform



Reset in Fixed Function Block

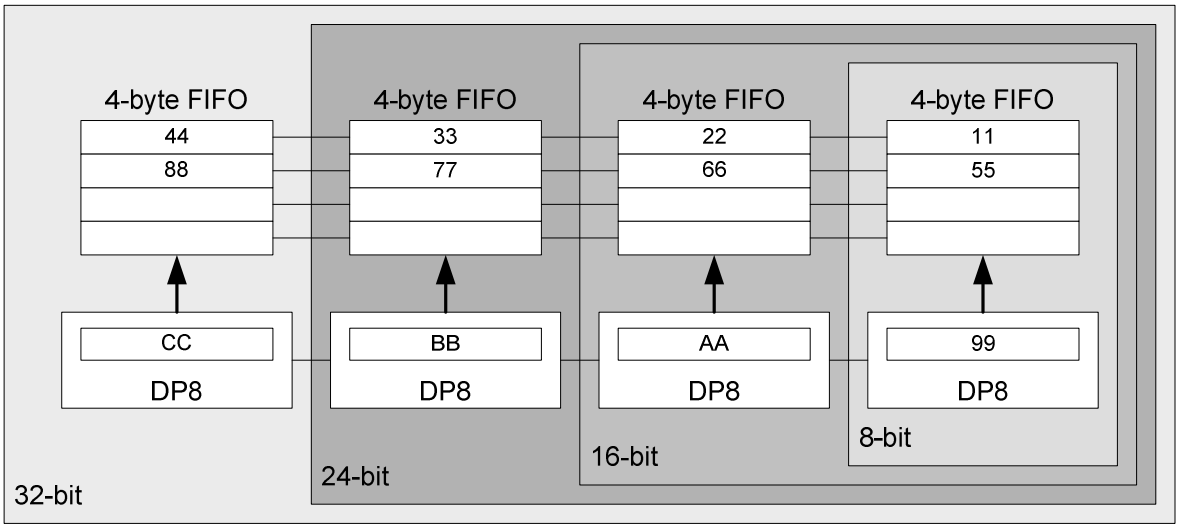
On PSoC 3 ES2 silicon, the FF implementation of the Counter differs from the UDB implementation in that the tc output goes high during reset. In the UDB implementation, tc goes low. The following schematic shows a FF implementation which drives tc low while reset is active, thus giving the same functionality as the UDB implementation of the same component.

Figure 4. Terminal Count Adjust Circuit for PSoC 3 ES2 Silicon



UDB FIFOs

Each UDB datapath contains two 8-bit FIFO registers: F0 and F1 (refer to 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 for the addressing of all 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 which 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 as 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 is registered at the input clock edge of the counter giving all sticky bits the timing resolution of the counter. For the UDB implementation, the status register is clocked from the bus clock. For the FF implementation, the

status register is clocked from the timer input clock. All non-sticky 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

Bit Name	#define in header file	Description
Cmp	Counter_STATUS_CMP	This bit will go to 1 when the compare output is high.
Zero	Counter_STATUS_ZERO	This bit will go to 1 when the counter value is equal to zero.
Overflow	Counter_STATUS_OVERFLOW	This bit will go to 1 when the counter value is equal to the period value.
Underflow	Counter_STATUS_UNDERFLOW	This bit will go high when the counter value is equal to zero.
Capture	Counter_STATUS_CAPTURE	This bit will go to 1 whenever a valid capture event has been triggered. This does not include software capture.
FIFO Full	Counter_STATUS_FIFOFULL	This bit will go to 1 when the UDB FIFO reaches the full state defined as 4 entries.
FIFO Not Empty	Counter_STATUS_FIFONEMP	This bit will go 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, as 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 call and read with the Counter_ReadControlRegister() function. All operations on the control register must use the following defines for the bit-fields as 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	Enable	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD

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 CompareMode parameter. • Counter__B_COUNTER__CM_LESSTHAN • Counter__B_COUNTER__CM_LESSTHANOREQUAL • Counter__B_COUNTER__CM_EQUAL • Counter__B_COUNTER__CM_GREATERTHAN • Counter__B_COUNTER__CM_GREATERTHANOREQUAL
Capture Mode	Counter_CTRL_CAPMODE_MASK	The capture mode control bits are a 2-bit field used to define the expected capture input operation. This bit-field is configured at initialization with the capture mode defined in the Capture Mode parameter. • Counter__B_COUNTER__CPTM_NONE • Counter__B_COUNTER__CPTM_RISINGEDGE • Counter__B_COUNTER__CPTM_FALLINGEDGE • Counter__B_COUNTER__CPTM_EITHEREDGE
Enable	Counter_CTRL_ENABLE	Enables counting under software control. This bit is valid only if the **Enable Mode** parameter is set to "Software Only" or "Hardware and Software."

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 call.

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 UDB FIFO 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 call 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.

DC and AC Electrical Characteristics (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	PSoC 3	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 (UDB Implementation)

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

Timing Characteristics "Maximum with Nominal Routing"

Parameter	Description	Config. [1]	Min	Typ	Max	Units
f_{clock}	Component Clock Frequency [2]	Config 1			45	MHz
		Config 2			40	MHz
		Config 3			35	MHz
		Config 4			30	MHz
		Config 5			25	MHz
t_{clockH}	Input Clock High Time [3]	N/A		0.5		$1/f_{clock}$
t_{clockL}	Input Clock Low Time [3]	N/A		0.5		$1/f_{clock}$
Inputs						
$t_{PD\ ps}$	Input path delay, pin to sync [4]	1			STA [5]	ns
$t_{PD\ ps}$	Input path delay, pin to sync [6]	2			8.5	ns
t_{PD_si}	Sync output to Input Path Delay (route)	1,2,3,4			STA [5]	ns
$t_{I\ clk}$	Alignment of clockX and clock	1,2,3,4	0		1	$t_{CY\ clock}$

[1] Configurations:
- Config 1:
  - Resolution: 8 bits
  - Implementation: Basic Up/Down Counter
- Config 2:
  - Resolution: 8 bits
  - Implementation: Counter with Direction
- Config 3:
  - Resolution: 16 bits
  - Implementation: Up/Down Counter or with direction (includes Clock with Direction and Clock with UpCnt & DwnCnt)
- Config 4:
  - Resolution: 24 bits
  - Implementation: Up/Down Counter or with direction (includes Clock with Direction and Clock with UpCnt & DwnCnt)
- Config 5:
  - Resolution: 32 bits
  - Implementation: Up/Down Counter or with direction (includes Clock with Direction and Clock with UpCnt & DwnCnt)

[2] If Time Division Multiplex Implementation is selected, than Component Clock Frequency must be 4 times greater than the data rate.

[3] $t_{CY_clock} = 1/f_{clock}$ - Cycle time of one clock period.

[4] t_{PD_ps} is found in the Static Timing Results as described later. The number listed here is a nominal value based on STA analysis on many inputs.

[5] t_{PD_ps} and t_{PD_si} are route path delays. Because routing is dynamic, these values can change and will directly affect the maximum component clock and sync clock frequencies. The values must be found in the Static Timing Analysis results.

[6] t_{PD_ps} in configuration 2 is a fixed value defined per pin of the device. The number listed here is a nominal value of all of the pins available on the device.

Parameter	Description	Config. [1]	Min	Typ	Max	Units
t_{PD_IE}	Input Path Delay to Component Clock (Edge Sensitive Input)	1,2	t_{PD_ps} + t_{sync} + t_{PD_si}		t_{PD_ps} + t_{sync} + t_{PD_si} + $t_{I\ clk}$	ns
t_{PD_IE}	Input Path Delay to Component Clock (Edge Sensitive Input)	3,4	t_{sync} + t_{PD_si}		t_{sync} + t_{PD_si} + $t_{I\ clk}$	ns
t_{IH}	Input High Time	1,2,3,4	$t_{CY\ clock}$ [7]			ns
t_{IL}	Input Low Time	1,2,3,4	$t_{CY\ clock}$ [7]			ns

[7] t_{CY_clock} = 4* [1/f_{clock}] if Time Division Multiplex Implementation is selected.

Timing Characteristics “Maximum with All Routing”

Parameter	Description	Config. [1]	Min	Typ	Max [2]	Units
f_{clock}	Component Clock Frequency [3]	Config 1			22	MHz
		Config 2			20	MHz
		Config 3			17	MHz
		Config 4			15	MHz
		Config 5			12	MHz
t_{clockH}	Input Clock High Time [4]	N/A		0.5		1/f_{clock}
t_{clockL}	Input Clock Low Time [4]	N/A		0.5		1/f_{clock}

[1] Configurations:
- Config 1:
  - Resolution: 8 bits
  - Implementation: Basic Up/Down Counter
- Config 2:
  - Resolution: 8 bits
  - Implementation: Counter with Direction
- Config 3:
  - Resolution: 16 bits
  - Implementation: Up/Down Counter or with direction (includes Clock with Direction and Clock with UpCnt & DwnCnt)
- Config 4:
  - Resolution: 24 bits
  - Implementation: Up/Down Counter or with direction (includes Clock with Direction and Clock with UpCnt & DwnCnt)
- Config 5:
  - Resolution: 32 bits
  - Implementation: Up/Down Counter or with direction (includes Clock with Direction and Clock with UpCnt & DwnCnt)

[2] Maximum for “All Routing” is calculated by <nominal>/2 rounded to the nearest integer tested against empirical values obtained using the set of characterization unit tests. This value provides a basis for the user to not have to worry about meeting timing if they are running at or below this component frequency.

[3] If Time Division Multiplex Implementation is selected, then Component Clock Frequency must be 4 times greater than the data rate.

[4] t_{CY_clock} = 1/f_{clock} - Cycle time of one clock period.

Parameter	Description	Config. [1]	Min	Typ	Max [2]	Units
Inputs						
t_{PD_ps}	Input path delay, pin to sync [5]	1			STA [6]	ns
t_{PD_ps}	Input path delay, pin to sync [7]	2			8.5	ns
t_{PD_si}	Sync output to Input Path Delay (route)	1,2,3,4			STA [6]	ns
t_{I_clk}	Alignment of clockX and clock	1,2,3,4	0		1	t_{CY_clock}
t_{PD_IE}	Input Path Delay to Component Clock (Edge Sensitive Input)	1,2	$t_{PD_ps} + t_{sync} + t_{PD_si}$		$t_{PD_ps} + t_{sync} + t_{PD_si} + t_{I_clk}$	ns
t_{PD_IE}	Input Path Delay to Component Clock (Edge Sensitive Input)	3,4	$t_{sync} + t_{PD_si}$		$t_{sync} + t_{PD_si} + t_{I_clk}$	ns
t_{IH}	Input High Time	1,2,3,4	t_{CY_clock} [8]			ns
t_{IL}	Input Low Time	1,2,3,4	t_{CY_clock} [8]			ns

[5] t_{PD_ps} is found in the Static Timing Results as described later. The number listed here is a nominal value based on STA analysis on many inputs.

[6] t_{PD_ps} and t_{PD_si} are route path delays. Because routing is dynamic, these values can change and will directly affect the maximum component clock and sync clock frequencies. The values must be found in the Static Timing Analysis results.

[7] t_{PD_ps} in configuration 2 is a fixed value defined per pin of the device. The number listed here is a nominal value of all of the pins available on the device.

[8] $t_{CY_clock} = 4^* [1/f_{clock}]$ if Time Division Multiplex Implementation is selected.

How to Use STA Results for Characteristics Data

Nominal route maximums are gathered through multiple test passes with Static Timing Analysis (STA). You can calculate the maximums for your designs using the STA results using the following methods:

f_{clock} Maximum Component Clock Frequency appears in Timing results in the clock summary as the named external clock. The graphic below shows an example of the clock limitations from the *_timing.html*:

-Clock Summary

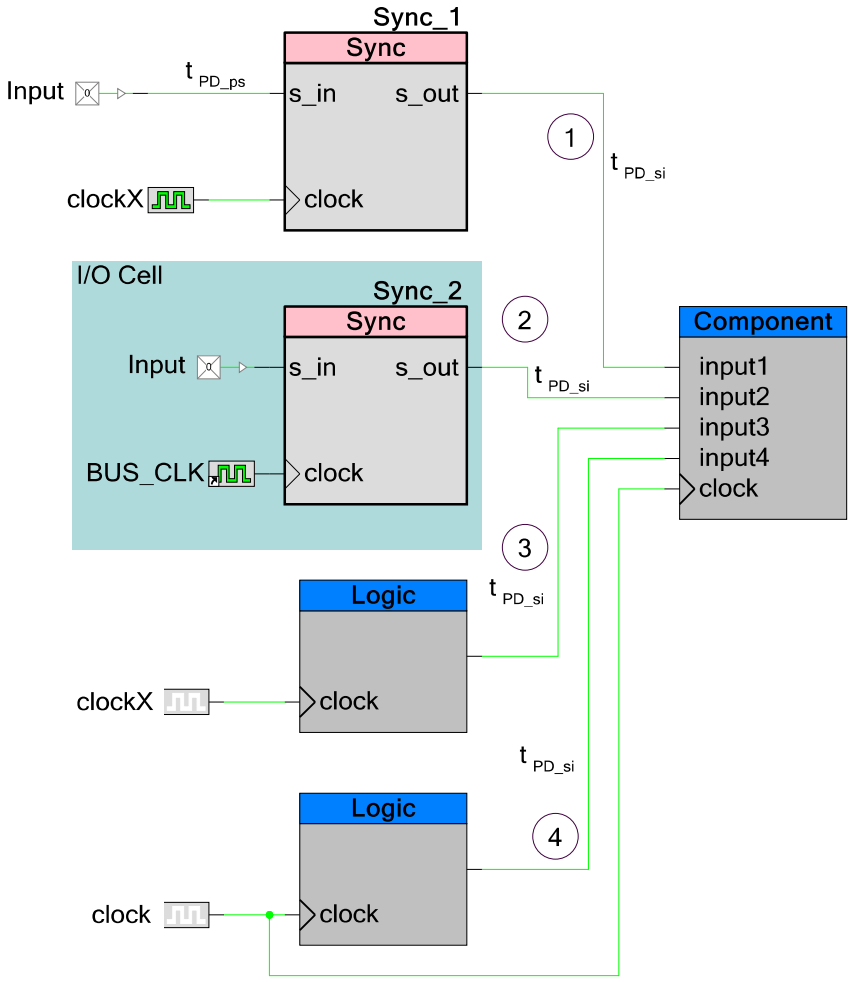
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

Input Path Delay and Pulse Width

When characterizing the functionality of inputs, all inputs, no matter how you have configured them, look like one of four possible configurations, as shown in Figure 5.

All inputs must be synchronized. The synchronization mechanism depends on the source of the input to the component. To fully interpret how your system will work you must understand which input configuration you have set up for each input and the clock configuration of your system. This section describes how to use the Static Timing Analysis (STA) results to determine the characteristics of your system.

Figure 5. Input Configurations for Component Timing Specifications



Configuration	Component Clock	Synchronizer Clock (Frequency)	Figures
1	master_clock	master_clock	Figure 10
1	clock	master_clock	Figure 8
1	clock	clockX = clock [1]	Figure 6
1	clock	clockX > clock	Figure 7
1	clock	clockX < clock	Figure 9
2	master_clock	master_clock	Figure 10
2	clock	master_clock	Figure 8
3	master_clock	master_clock	Figure 15
3	clock	master_clock	Figure 13
3	clock	clockX = clock [1]	Figure 11
3	clock	clockX > clock	Figure 12
3	clock	clockX < clock	Figure 14
4	master_clock	master_clock	Figure 15
4	clock	clock	Figure 11

[1] Clock frequencies are equal but alignment of rising edges is not guaranteed.

1. The input is driven by a device pin and synchronized internally with a “sync” component. This component is clocked using a different internal clock than the clock the component uses (all internal clocks are derived from master_clock).

   When characterizing inputs configured in this way clockX may be faster, equal to, or slower than the component clock. It may also be equal to master_clock, which produces the characterization parameters shown in Figure 6, Figure 7, Figure 9, and Figure 10.

2. The input is driven by a device pin and synchronized at the pin using master_clock.

   When characterizing inputs configured in this way, master_clock is faster than or equal to the component clock (it is never slower than). This produces the characterization parameters shown in Figure 7 and Figure 10.

Figure 6: Input Configuration 1 and 2; Sync Clock Freq.= Component Clock Freq. (Edge alignment of clock and clockX is not guaranteed)

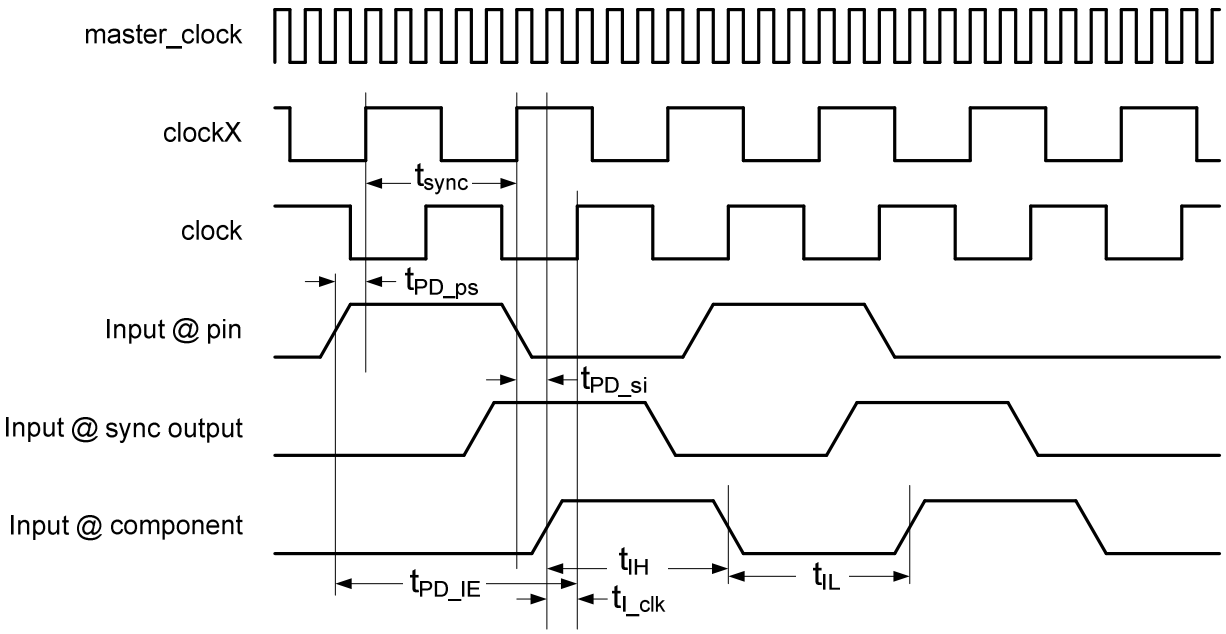


Figure 7: Input Configuration 1 and 2; Sync. Clock Freq. > Component Clock Freq.

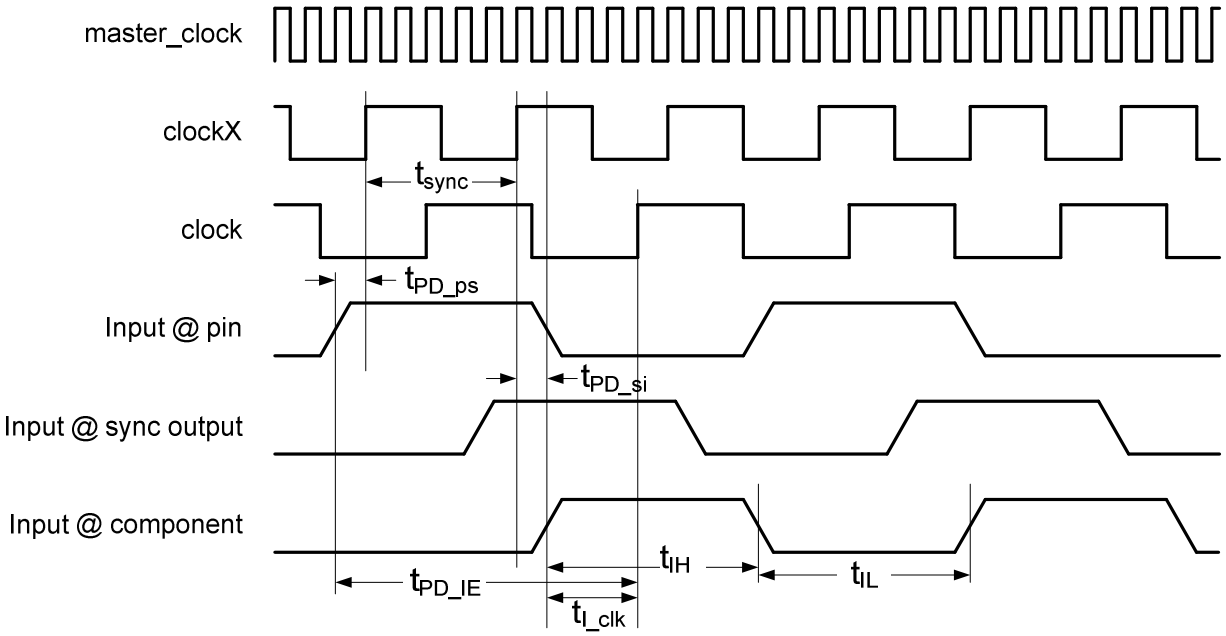


Figure 8: Input Configuration 1 and 2; [Sync. Clock Freq. == master_clock] > Component Clock Freq.

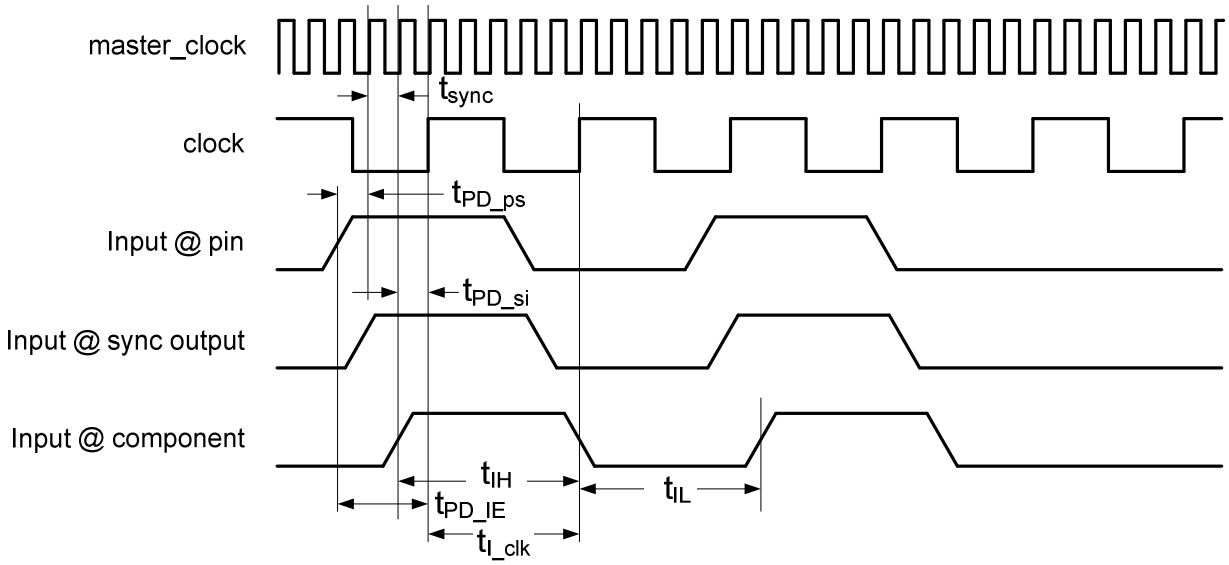


Figure 9: Input Configuration 1; Sync. Clock Freq. < Component Clock Freq.

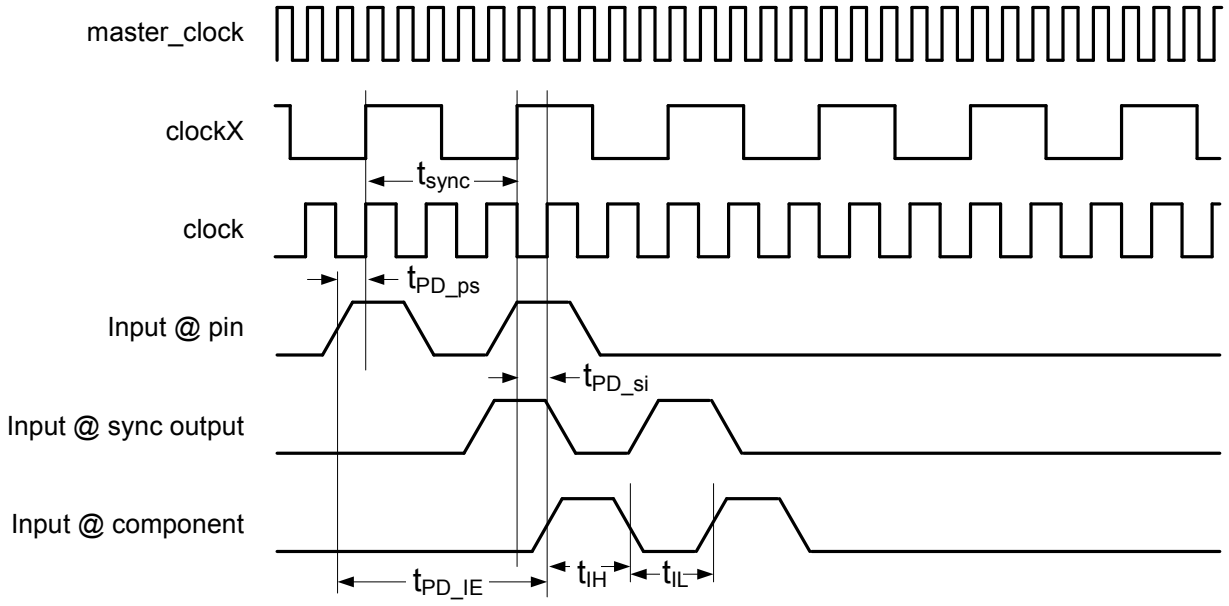
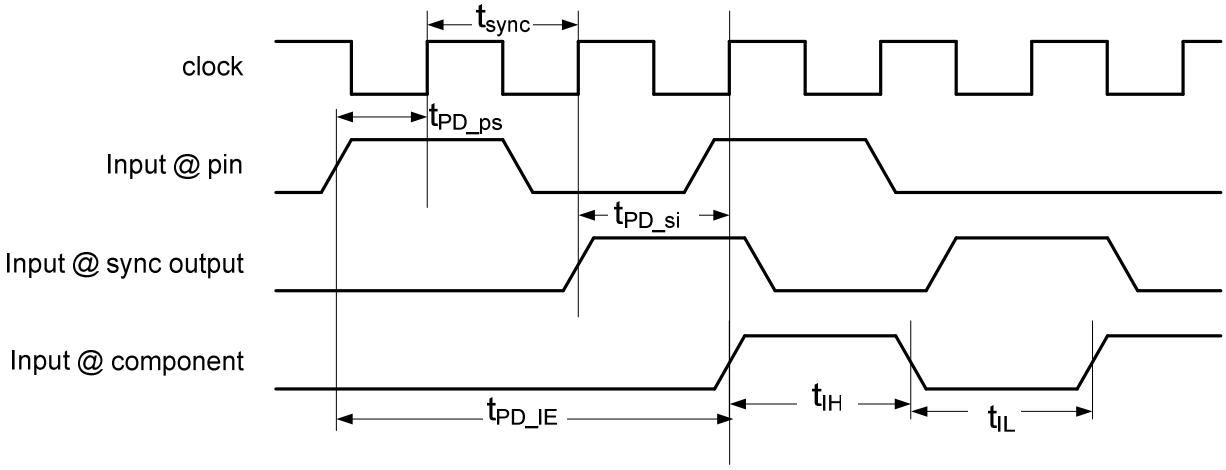


Figure 10: Input Configuration 1 and 2; Sync. Clock = Component Clock = master_clock

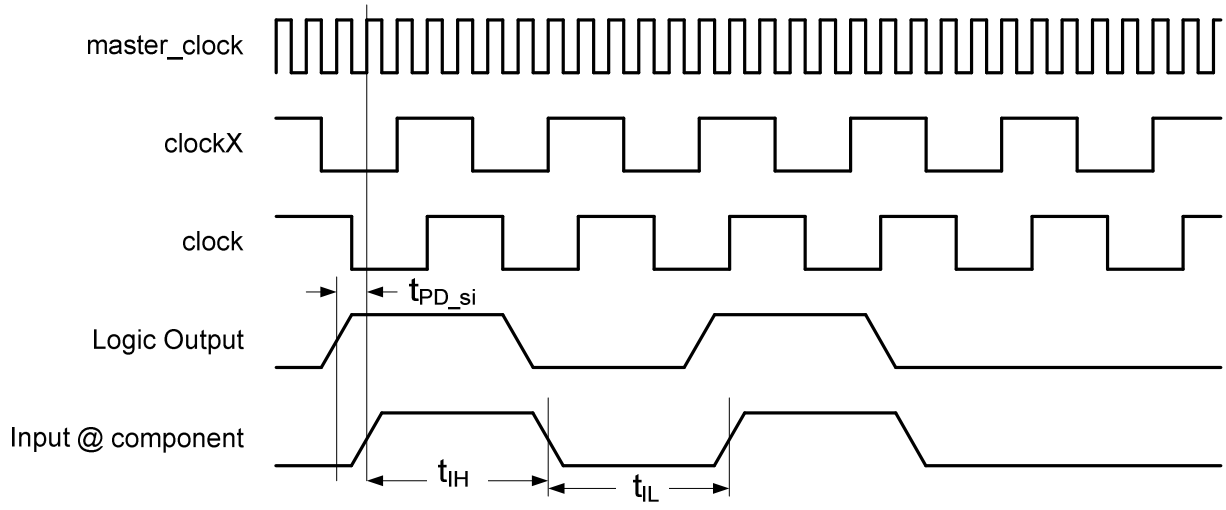


3. The input is driven by logic internal to the PSoC, which is synchronous based on a clock other than the clock the component uses (all internal clocks are derived from master_clock).

   When characterizing inputs configured in this way, the synchronizer clock is faster than, less than, or equal to the component clock, which produces the characterization parameters shown in Figure 11, Figure 12, and Figure 14.
4. The input is driven by logic internal to the PSoC, which is synchronous based on the same clock the component uses.

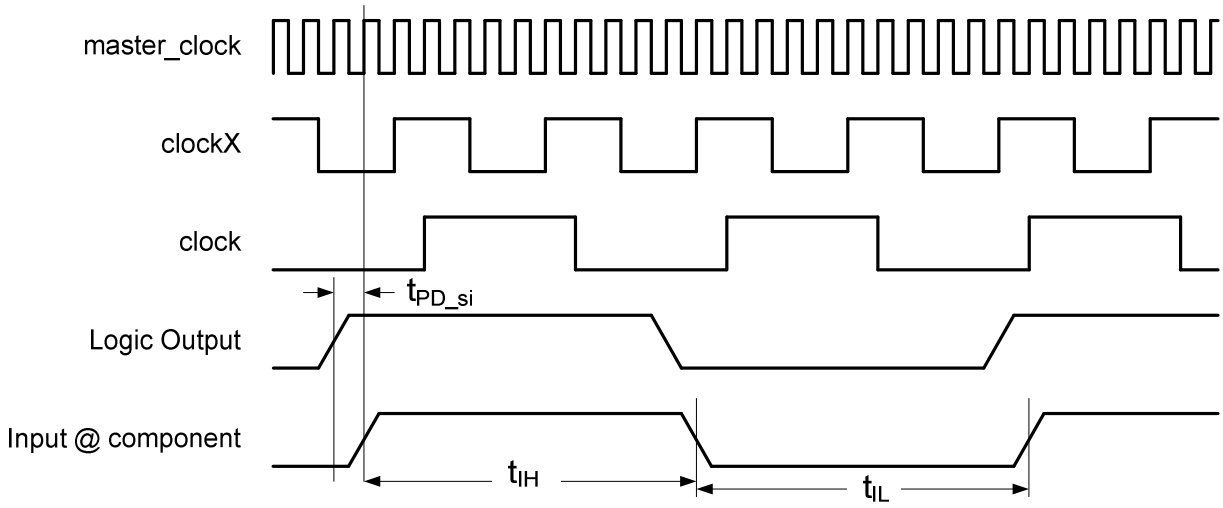
When characterizing inputs configured in this way, the synchronizer clock will be equal to the component clock, which will produce the characterization parameters as shown in Figure 15.

Figure 11: Input Configuration 3 only; Sync. Clock Freq. = Component Clock Freq. (Edge alignment of clock and clockX is not guaranteed)



This figure represents the understanding that Static Timing Analysis holds on the clocks. All clocks in the digital clock domain are synchronous to master_clock. However, it is possible that two clocks with the same frequency are not rising-edge-aligned. Therefore, the static timing analysis tool does not know which edge the clocks are synchronous to and must assume the minimum of 1 master_clock cycle. This means that t_{PD_si} now has a limiting effect on master_clock of the system. Master_clock setup time violations appear if this path delay is too long. You must change the synchronization clocks of your system or run master_clock at a slower frequency.

Figure 12: Input Configuration 3; Sync. Clock Freq. > Component Clock Freq.



In much the same way as shown in Figure 11, all clocks are derived from master_clock. STA indicates the t_{PD_si} limitations on master_clock for one master_clock cycle in this configuration. Master_clock setup time violations appear if this path delay is too long. You must change the synchronization clocks of your system or run the master_clock at a slower frequency.

Figure 13: Input Configuration 3; Synchronizer Clock Frequency = master_clock > Component Clock Frequency

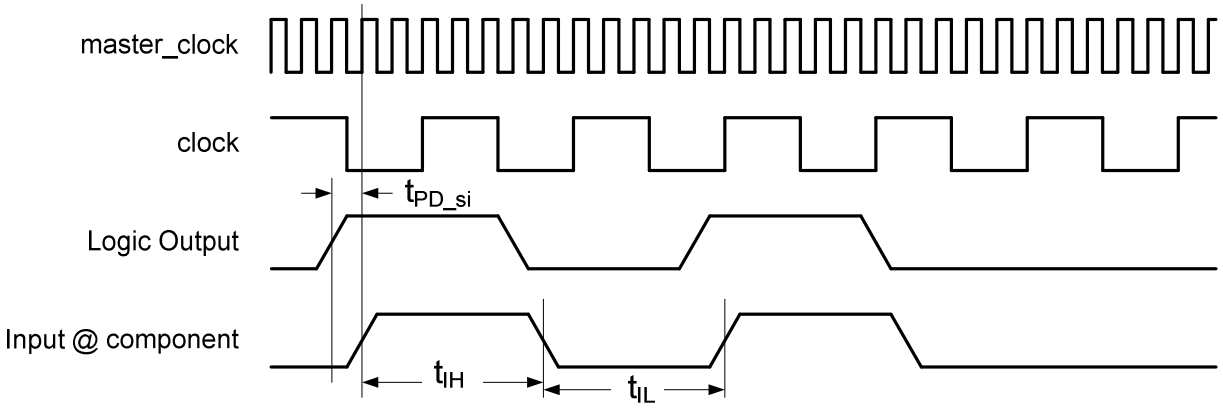
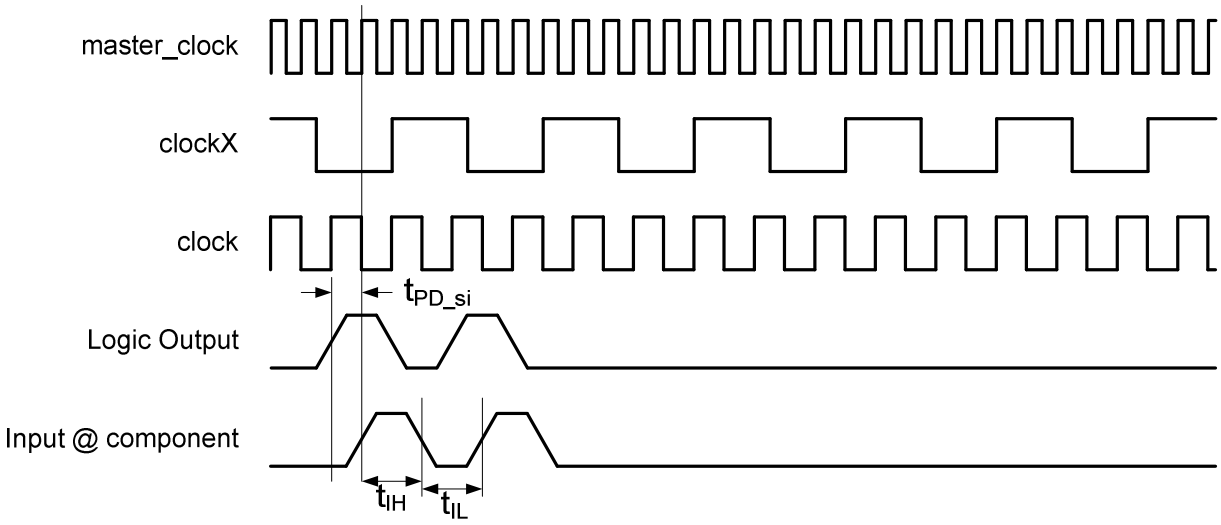
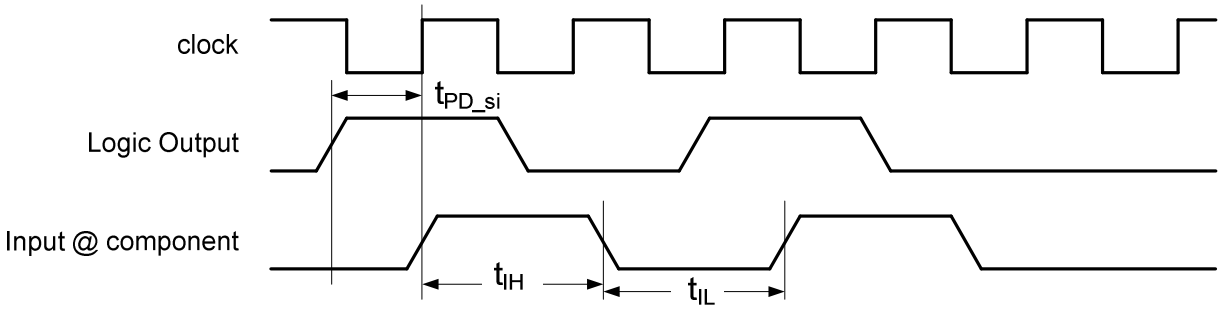


Figure 14: Input Configuration 3; Synchronizer Clock Frequency < Component Clock Frequency



In much the same way as shown in Figure 11, all clocks are derived from master_clock. STA indicates the t_{PD_si} limitations on master_clock for one master_clock cycle in this configuration. master_clock setup time violations appear if this path delay is too long. You must change the synchronization clocks of your system or run master_clock at a slower frequency.

Figure 15: Input Configuration 4 only; Synchronizer Clock = Component Clock



In all previous figures in this section, the most critical parameters to use when understanding your implementation are f_{clock} and t_{PD_IE}. t_{PD_IE} is defined by t_{PD_ps} and t_{sync} (for configurations 1 and 2 only), t_{PD_si}, and t_{I_Clk}. Of critical importance is the fact that t_{PD_si} defines the maximum component clock frequency. t_{I_Clk} does not come from the STA results but is used to represent when t_{PD_IE} is registered. This is the margin left over after the route between the synchronizer and the component clock.

t_{PD_ps} and t_{PD_si} are included in the STA results.

To find t_{PD_ps}, look at the input setup times defined in the *_timing.html* file. The fan-out of this input may be more than 1 so you will need to evaluate the maximum of these paths.

-Setup times

-Setup times to clock BUS_CLK

Start	Register	Clock	Delay (ns)
input1(0):iocell.pad_in	input1(0):iocell.ind	BUS_CLK	16.500

t_{PD_si} is defined in the Register-to-register times. You will need to know the name of the net to use the *_timing.html* file. The fan-out of this path may be more than 1 so you will need to evaluate the maximum of these paths.

-Register-to-register times

-Destination clock clock

Destination clock clock (Actual freq: 24.000 MHz)

+Source clock clock

-Source clock clock_1

Source clock clock_1 (Actual freq: 24.000 MHz)
Affected clock: BUS_CLK (Actual freq: 24.000 MHz)

Start	End	Period (ns)	Max Freq	Frequency	Violation
\Sync_1:genblk1[0]:INST\:synccell.syncq	\PWM_1:PWMUDB:runmode_enable\:macrocell.mc_d	7.843	127.508 MHz	24.000 MHz	

Output Path Delays

When characterizing the path delays of outputs, you must consider where the output is going in order to know where you can find the data in the STA results. For this component, all outputs are synchronized to the component clock. Outputs fall into one of two categories. The output goes either to another component inside the device, or to a pin to the outside of the device. In the first case, you must look at the Register-to-register times shown for the Logic-to-input descriptions above (the source clock is the component clock). For the second case, you can look at the Clock-to-Output times in the *_timing.html* STA results.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.0	Redesigned as a synchronous counter with over-sampling in all modes.	The architecture of the device and the tool have 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.

Version	Description of Changes	Reason for Changes / Impact
	Synchronized inputs	All inputs are synchronized in the fixed function implementation, at the input of the block.
	Counter_GetInterruptSource() function was converted 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.
	Outputs are now Registered to the component clock	To avoid glitches on the outputs of the component it was required that all outputs are synchronized. This is done inside of the Datapath when possible, to avoid excess resource usage.
	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, 2008-2011. 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.