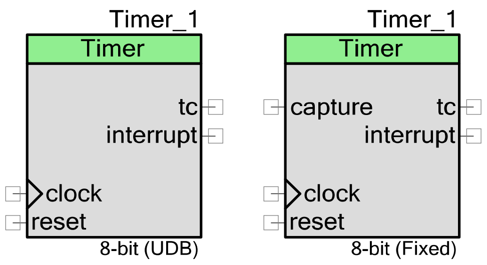
Timer
2.10

Features



- Fixed-function (FF) and universal digital block (UDB) implementations
- 8-, 16-, 24-, or 32-bit timer
- Optional capture input
- Enable, trigger, and reset inputs, for synchronizing with other components
- Continuous or one shot run modes

General Description

The Timer component provides a method to measure intervals. It can implement a basic timer function and offers advanced features such as capture with capture counter and interrupt/DMA generation.

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 save 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
Number of bits	8 or 16	8, 16, 24, or 32
Run Mode	Continuous or one shot	Continuous, one shot, or one shot halt on interrupt
Count Mode	Down only	Down only
Enable input	Yes (hardware or software enable)	Yes (hardware or software enable)
Capture input	Yes	Yes
Capture mode	Rising edge only	Rising edge, falling edge, either edge, or software controlled
Capture FIFO	No (one capture register)	Yes (up to four captures)
Trigger input	No	Yes

Feature	FF	UDB
Trigger mode	None	Rising edge, falling edge, either edge, or software controlled
Reset input	Yes	Yes
Terminal count output	Yes	Yes
Interrupt output	Yes	Yes
Interrupt conditions	TC, capture	TC, capture, and FIFO full
Capture output	No	Yes
Period register	Yes	Yes
Period reload	Yes (always reload on reset or TC)	Yes (always reload on reset or TC)
Clock input	Limited to digital clocks in the clock system	Any signal

When to Use a Timer

The default use of the Timer is to generate a periodic event or interrupt signal. However, there are other potential uses:

- Create a clock divider by driving a clock into the clock input and using the terminal count output as the divided clock output.
- Measure the length of time between hardware events by driving a clock into the clock input and driving the test signal to the enable or capture input.

Note A Counter component is better used in situations focused on counting 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.

A Timer is typically used to record the number of clock cycles between events. An example of this is measuring the number of clocks between two rising edges as might be generated by a tachometer sensor. A more complex use is to measure the period and duty cycle of a PWM input. For PWM measurement, the Timer component is configured to start on a rising edge, capture the next falling edge, and then capture and stop on the next rising edge. An interrupt on the final capture signals the CPU that all of the captured values are ready in the FIFO.

Input/Output Connections

This section describes the various input and output connections for the Timer. 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.

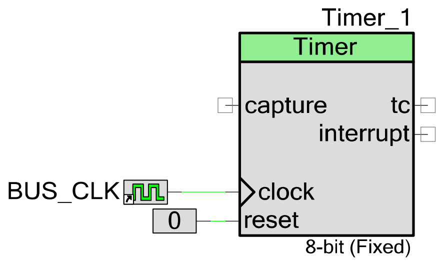
Input	May Be Hidden	Description
clock	N	The clock input defines the operating frequency of the Timer component. That is, the timer period counter value is decremented on the rising edge of this input while the Timer component is enabled.
reset	N	This input is a synchronous reset. It requires at least one rising edge of the clock to implement the resets of the counter value and the capture counter. It resets the period counter to the period value. It also resets the capture counter. **Note** For PSoC 3 ES2 silicon, the Terminal Count pin for the fixed-function Timer is held high during reset. You can find a schematic fix for this in Reset in Fixed-Function Block in the Functional Description section of this datasheet. For PSoC 3 Production or later silicon, the Terminal Count pin for the fixed-function Timer is held low in Reset
enable	Y	This input is the Timer hardware enable. This connection enables the period counter to decrement on each rising edge of the clock. If this input is low the outputs are still active but the Timer component does not change states. This input is visible when the **Enable Mode** parameter is set to **Hardware**.
capture	Y	The capture input captures the current count value to a capture register or FIFO. The 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. No values are captured if the Timer is disabled.
trigger	Y	The trigger input enables the timer to start and stop counting based on configurable hardware events. The input is visible if the **Trigger Mode** parameter is set to any mode other than **None**. It causes the Timer to delay counting until the appropriate edge is detected. The trigger edge is not captured nor does it generate an interrupt.

Output	May Be Hidden	Description
tc	N	Terminal count is a synchronous output that indicates that the count value equals zero. The output is synchronous to the clock input of the Timer. The signal goes high one clock cycle after the count value becomes zero and stays high while the count value equals zero. If the Timer is disabled when the count is at zero, the output will remain high until the Timer is re-enabled.
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: Terminal Count, Capture, or FIFO full. Once an interrupt is triggered, the interrupt output remains asserted until the status register is read.

Output	May Be Hidden	Description
capture_out	Y	The capture_out output is an indicator of when a hardware capture has been triggered. The output pin is available for UDB implementation only. This output is synchronized to the clock input of the Timer.

Schematic Macro Information

The default Timer in the Component Catalog is a schematic macro using a Timer component with default settings. It is connected to bus clock and a Logic Low component.



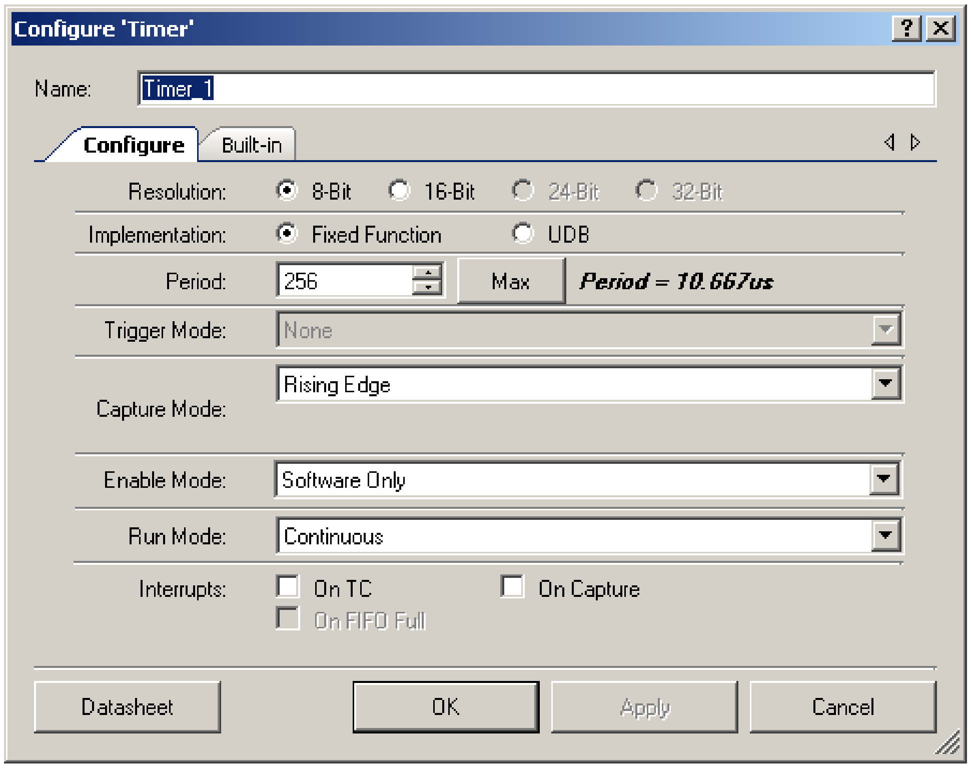
Component Parameters

Drag a Timer 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 values. These may be modified at any time with the APIs provided. Most parameters described in the next sections are hardware options. The software options are noted as such.

Configure Tab



Name: Timer_1

Resolution: [✓ 8-Bit] 16-Bit 24-Bit 32-Bit

Implementation: [✓ Fixed Function] UDB

Period: 256 *Period = 10.667us*

Trigger Mode: None

Capture Mode: Rising Edge

Enable Mode: Software Only

Run Mode: Continuous

Interrupts: ☐ On TC ☐ On Capture ☐ On FIFO Full

Resolution

The **Resolution** parameter defines the bit-width resolution of the Timer. 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 Timer. If FF is selected, UDB functions are disabled.

Period (Software Option)

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

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

Trigger Mode (Software Option)

The **Trigger Mode** parameter configures the implementation of the trigger input. This parameter is only active when **Implementation** is set to **UDB**.

Trigger Mode can be set to any of the following values:

- **None** (default) – No trigger implemented and the trigger input pin is hidden
- **Rising Edge** – Trigger (enable) counting on the first rising edge of the trigger input
- **Falling Edge** – Trigger (enable) counting on the first falling edge of the trigger input
- **Either Edge** – Trigger (enable) counting on the first edge (rising or falling) of the trigger input
- **Software Controlled –** The trigger mode can be set during run time, to one of the four trigger modes listed above, using the Timer_SetTriggerMode() API call.

Capture Mode (Software Option)

The Capture Mode section contains three parameters: **Capture Mode Value**, **Enable Capture Counter**, and **Capture Count**.

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 (for fixed-function implementation, only **None** and **Rising Edge** are available):

- **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** – The capture mode can be set during run time, to one of the four capture modes listed above, using the Timer_SetCaptureMode() API call.

Enable Capture Counter (Software Option)

The **Enable Capture Counter** parameter allows you to define how many capture events happen before the counter is actually captured. For example, it may be necessary to capture every third event, in which case you should set the capture counter to a value of 3. This parameter is only available for a UDB implementation.

Capture Count (Software Option)

The **Capture Count** parameter sets the initial number of capture events that occur before the counter is actually captured. It can be set to 2-127. The capture count value may be modified, at runtime by calling the API function Timer_SetCaptureCount().This parameter is only available for a UDB implementation.

Enable Mode

The **Enable Mode** parameter configures the enable implementation of the Timer. 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 Timer is enabled based on the enable bit of the control register only.
- **Hardware** – The Timer is enabled based on the enable input only.
- **Software and Hardware** – The Timer is enabled if both hardware and software enables are true.

Run Mode

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

- **Continuous** – The Timer runs continuously while it is enabled.
- **One Shot** – The Timer starts counting and stops counting when zero is reached. After it is reset, it begins another cycle. On stop, for a UDB Timer, it reloads period into the count register; for a fixed-function Timer the count register remains at terminal count.
- **One Shot (Halt on Interrupt)** – The Timer starts counting and stops counting when zero is reached or an interrupt occurs. After it is reset, it begins another cycle. On stop, for a UDB Timer, it reloads period into the count register; for a fixed-function Timer the count register remains at terminal count.

  Note In order to be sure that One Shot mode does not start prematurely, you should use a **Trigger Mode** to control the start time, or use some form of software enable mode (**Software Only** or **Software and Hardware**).

Interrupt (Software Option)

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. The software can reconfigure this mode at any time; this parameter defines an initial configuration.

- **On TC** –This parameter is always active; it is cleared by default.
- **On Capture (1-4) –** Allows you to interrupt on a given number of captures; it is cleared by default.
- **On FIFO Full –** Allows you to interrupt when the capture FIFO is full; it is cleared by default.

Clock Selection

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 use the FF block in the device, the Timer component has the following restrictions:

- The clock 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, and so on.

For UDB-based Components

You can connect any digital signal from any source to the clock input. The frequency of that signal is limited to the frequency range defined in the DC and AC Electrical Characteristics (UDB Implementation) section of this datasheet.

Placement

PSoC Creator places the Timer 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 within 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 Timer [1]	1	6	1	1	0	257	5	-
8 bits FF Timer [2]	0	0	0	0	0	234	2	-
16-Bits UDB Timer [1]	2	6	1	1	0	295	6	-
16-Bits FF Timer [2]	0	0	0	0	0	248	2	
24-Bits UDB Timer [1]	3	6	1	1	0	287	8	-
32-Bits UDB Timer [1]	4	6	1	1	0	287	8	-
8 bit UDB Timer One Shot [3]	1	8	1	1	0	257	5	-
16 bit UDB Timer One Shot [3]	2	8	1	1	0	295	6	-

[1] The UDB Timer with corresponding resolution is configured for Software Only Enable mode, Rising Edge Trigger mode, Continuous Run mode and Interrupt on TC with no Capture mode.

[2] The FF Timer with corresponding resolution is configured for Software Only Enable mode, Rising Edge Capture mode, Continuous Run mode and Interrupt on TC.

[3] The UDB Timer with corresponding resolution is configured for Software Only Enable mode, Rising Edge Trigger mode, One Shot mode Interrupt on TC with no Capture mode.

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 “Timer_1” to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “Timer”.

Function	Description
Timer_Start()	Sets the initVar variable, calls the Timer_Init() function, and then calls the Enable function.
Timer_Stop()	Disables the Timer;
Timer_SetInterruptMode()	Enables or disables the sources of the interrupt output.
Timer_ReadStatusRegister()	Returns the current state of the status register.
Timer_ReadControlRegister()	Returns the current state of the control register.
Timer_WriteControlRegister()	Sets the bit-field of the control register.
Timer_WriteCounter()	Writes a new value directly into the counter register.
Timer_ReadCounter()	Forces a capture, and then returns the capture value.
Timer_WritePeriod()	Writes the period register.
Timer_ReadPeriod()	Reads the period register.
Timer_ReadCapture()	Returns the contents of the capture register or the output of the FIFO.
Timer_SetCaptureMode()	Sets the hardware or software conditions under which a capture will occur.
Timer_SetCaptureCount()	Sets the number of capture events to count before capturing the counter register to the FIFO.
Timer_ReadCaptureCount()	Reports the current setting of the number of capture events.
Timer_SoftwareCapture()	Forces a capture of the period counter to the capture FIFO
Timer_SetTriggerMode()	Sets the hardware or software conditions under which a trigger will occur.
Timer_EnableTrigger()	Enables the trigger mode of the timer.
Timer_DisableTrigger()	Disables the trigger mode of the timer.
Timer_SetInterruptCount()	Sets the number of captures to count before an interrupt is triggered.
Timer_ClearFIFO()	Clears the capture FIFO.
Timer_Sleep()	Stops the Timer and saves its current configuration.

Function	Description
Timer_Wakeup()	Restores the Timer configuration, and re-enables the Timer.
Timer_Init()	Initializes or restores the Timer per the Configure dialog settings.
Timer_Enable()	Enables the Timer.
Timer_SaveConfig()	Saves the current configuration of the Timer.
Timer_RestoreConfig()	Restores the configuration of the Timer.

Global Variables

Variable	Description
Timer_initVar	Indicates whether the Timer has been initialized. The variable is initialized to 0 and set to 1 the first time Timer_Start() is called. This allows the component to restart without reinitialization after the first call to the Timer_Start() routine. If reinitialization of the component is required, then the Timer_Init() function can be called before the Timer_Start() or Timer_Enable() function.

void Timer_Start(void)

Description: This is the preferred method to begin component operation. Timer_Start() sets the initVar variable, calls the Timer_Init() function, and then calls the Timer_Enable() function.

Parameters: None

Return Value: None

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

void Timer_Stop(void)

Description: Disables the Timer 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 Timer_SetInterruptMode(uint8 interruptMode)

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 Timer_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:

[7:4]: Unused (0)

[3]: FIFO not empty status

[2]: FIFO full status

[1]: Registered Capture value

[0]: Terminal count

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 Timer_ReadControlRegister(void)

Description: Returns the current state of the control register. This API is available only if the control register is not removed.

Parameters: None

Return Value: uint8: Control register bit field. The control register bits are:

[7]: Timer Enable

[6:5]: Capture Mode select

[4]: Trigger Enable

[3:2]: Trigger Mode select

[1:0]: Interrupt count

For bit definitions, refer to the Control Register section of this datasheet.

Side Effects: None

void Timer_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]: Timer Enable

[6:5]: Capture Mode select

[4]: Trigger Enable

[3:2]: Trigger Mode select

[1:0]: Interrupt count

For bit definitions, refer to the Control Register section of this datasheet.

Return Value: None

Side Effects: None

void Timer_WriteCounter(uint8/16/32 counter)

Description: Writes a new value directly into the counter register. This function is available only for UDB implementation.

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

Return Value: None

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

uint8/16/32 Timer_ReadCounter(void)

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

Parameters: None

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

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

void Timer_WritePeriod(uint8/16/32 period)

Description: Writes the period register.

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

Return Value: None

Side Effects: The period of the Timer does not change until counter is reloaded from the period register.

uint8/16/32 Timer_ReadPeriod(void)

Description: Reads the period register.

Parameters: None

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

Side Effects: None

uint8/16/32 Timer_ReadCapture(void)

Description: Returns the contents of the capture register or the output of the FIFO This function is available only for UDB implementation.

Parameters: None

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

Side Effects: None

void Timer_SetCaptureMode(uint8 captureMode)

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

Parameters: uint8: Enumerated capture mode. Refer also to the Control Register section:

```
Timer__B_TIMER__CM_NONE
Timer__B_TIMER__CM_RISINGEDGE
Timer__B_TIMER__CM_FALLINGEDGE
Timer__B_TIMER__CM_EITHEREDGE
Timer__B_TIMER__CM_SOFTWARE
```

Return Value: None

Side Effects: None

void Timer_SetCaptureCount(uint8 captureCount)

Description: Sets the number of capture events to count before a capture is performed. This function is available only for UDB implementation and when the **Enable Capture Counter** parameter is selected in the Configure dialog.

Parameters: uint8 captureCount: The desired number of capture events to count before capturing the counter value to the capture FIFO. A value from 2 to 127 is valid.

Return Value: None

Side Effects: None

uint8 Timer_ReadCaptureCount(void)

Description:	Reads the current value setting for the captureCount parameter as set in the Timer_SetCaptureCount() function. This function is only available for UDB implementation and when the **Enable Capture Counter** parameter is selected on the Configure dialog.
Parameters:	None
Return Value:	uint8: Current capture count
Side Effects:	None

void Timer_SoftwareCapture(void)

Description:	Forces a software capture of the current counter value to the FIFO. This function is available only for UDB implementation.
Parameters:	None
Return Value:	None:
Side Effects:	None

void Timer_SetTriggerMode(uint8 triggerMode)

Description: Sets the trigger mode. This function is available only for UDB implementation and when Trigger Mode parameter is set to Software Controlled.

Parameters: uint8: Enumerated capture mode. Refer also to the Control Register section.

```
Timer__B_TIMER__TM_NONE
Timer__B_TIMER__TM_RISINGEDGE
Timer__B_TIMER__TM_FALLINGEDGE
Timer__B_TIMER__TM_EITHEREDGE
Timer__B_TIMER__TM_SOFTWARE
```

Return Value: None

Side Effects: None

void Timer_EnableTrigger(void)

Description:	Enables the trigger. This function is available only when **Trigger Mode** is set to **Software Controlled**.
Parameters:	None
Return Value:	None
Side Effects:	None

void Timer_DisableTrigger(void)

Description:	Disables the trigger. This function is available only when **Trigger Mode** is set to **Software Controlled**.
Parameters:	None
Return Value:	None
Side Effects:	None

void Timer_SetInterruptCount(uint8 interruptCount)

Description:	Sets the number of captures to count before an interrupt is generated for the InterruptOnCapture source. This function is available only when InterruptOnCaptureCount is enabled.
Parameters:	uint8 interruptCount: The number of capture events to count before the interrupt on capture is generated. A value from 0 to 3 is valid.
Return Value:	None
Side Effects:	None

void Timer_ClearFIFO(void)

Description:	Clears the capture FIFO. This function is available only for UDB implementation. Refer to UDB FIFOs in the Functional Description section of this datasheet.
Parameters:	None
Return Value:	None
Side Effects:	None

void Timer_Sleep(void)

Description:	This is the preferred routine to prepare the component for sleep. Timer_Sleep() saves the current component state. Then it calls the Timer_Stop() function and calls Timer_SaveConfig() to save the hardware configuration. Call the Timer_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 a FF implementation, all registers are retained across low-power modes. For a UDB implementation, the control register and counter value register are saved and restored. Additionally when calling Timer_Sleep(), the enable state is stored in case you call Timer_Sleep() without calling Timer_Stop().

void Timer_Wakeup(void)

Description: This is the preferred routine to restore the component to the state when Timer_Sleep() was called. The Timer_Wakeup() function calls the Timer_RestoreConfig() function to restore the configuration. If the component was enabled before the Timer_Sleep() function was called, the Timer_Wakeup() function will also re-enable the component.

Parameters: None

Return Value: None

Side Effects: Calling the Timer_Wakeup() function without first calling the Timer_Sleep() or Timer_SaveConfig() function may produce unexpected behavior.

void Timer_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call Timer_Init() because the Timer_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 Timer_Enable(void)

Description: Activates the hardware and begins component operation. It is not necessary to call Timer_Enable() because the Timer_Start() routine calls this function, which is the preferred method to begin component operation. This function enables the Timer 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 Timer..

void Timer_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 Timer_Sleep() function.

Parameters: None

Return Value: None

Side Effects: None

void Timer_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 Timer_Sleep() function.
Parameters:	None
Return Value:	None
Side Effects:	Calling this function without first calling the Timer_Sleep() or Timer_SaveConfig() function may produce unexpected behavior.

Conditional Compilation Information

The Timer component API files require two conditional compile definitions to handle the multiple configurations it must support. The API files conditionally must compile on the **Resolution** chosen and the **Implementation** chosen from either the fixed-function block or the UDB blocks. 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 below.

Timer_DataWidth

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

Timer_UsingFixedFunction

The Using Fixed Function define is used mostly in the header file to make the correct register assignments. This is needed because the registers provided in the FF block are different than those used when the Timer 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.

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

As described previously, the Timer component can be configured for multiple uses. This section describes those configurations in more detail.

General Operation

On each rising edge of the clock input, the Timer component always counts down. It reloads the counter register from the period register on the next clock edge after the counter reaches a value of zero.

The timer remains disabled until enabled by hardware or software, depending on the configuration setting. You cannot use the component until Timer_Start() is called because this function sets the registers for the defined configuration.

Timer Outputs

The counter register can be monitored and reloaded. The tc output is available to monitor the current value of the counter register; it is high while the counter is zero.

Timer 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 Timer component to be synchronized to other components. The Timer component counts only when enabled and not held in reset. Counting can also be initiated on a trigger input event. It can be reset or enabled by either hardware or firmware. All triggering is hardware.

Timer Interrupt

An interrupt output is available to communicate event occurrences to the CPU or to other components. You can set the interrupt to be active on a combination of one or more events. You should design the interrupt handler carefully so that you can determine the source of the interrupt and whether it is edge- or level-sensitive, and clear the source of the interrupt.

Timer Registers

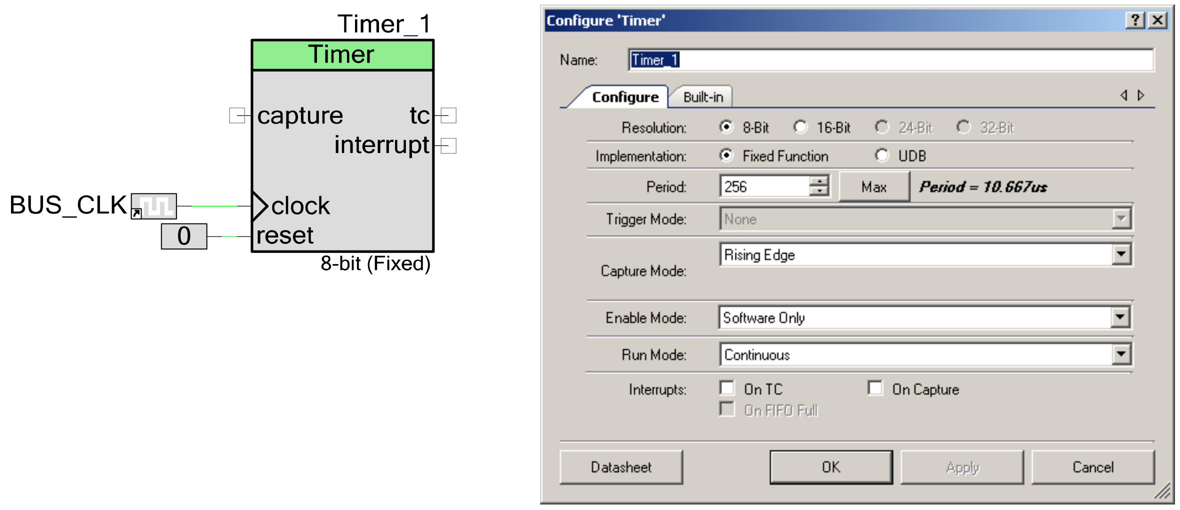
There are two registers: status and control. Refer to the Registers section.

Configurations

Default Configuration

When you drag a Timer component onto a PSoC Creator schematic, the default configuration is an 8-bit, FF timer that decrements the counter register on a rising edge at the clock input. Figure 1 shows the default schematic macro and Configure dialog settings.

Figure 1. Default Timer Configuration



Name: Timer_1

Resolution: [✓ 8-Bit] 16-Bit 24-Bit 32-Bit

Implementation: [✓ Fixed Function] UDB

Period: 256 Max *Period = 10.667us*

Trigger Mode: None

Capture Mode: Rising Edge

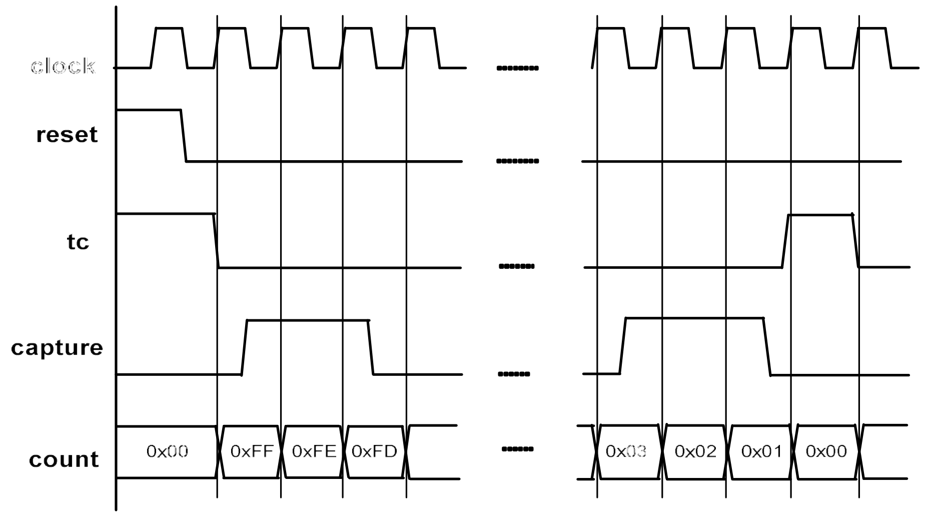
Enable Mode: Software Only

Run Mode: Continuous

Interrupts: ☐ On TC ☐ On Capture ☐ On FIFO Full

q shows the timing diagram for the default configuration.

Figure 2. Default Timer Implementation Example Waveform



Terminal count indicates in real time whether the counter value is at the terminal count (zero). The period is programmable to be any value from 1 to (2 ^ Resolution) – 1.

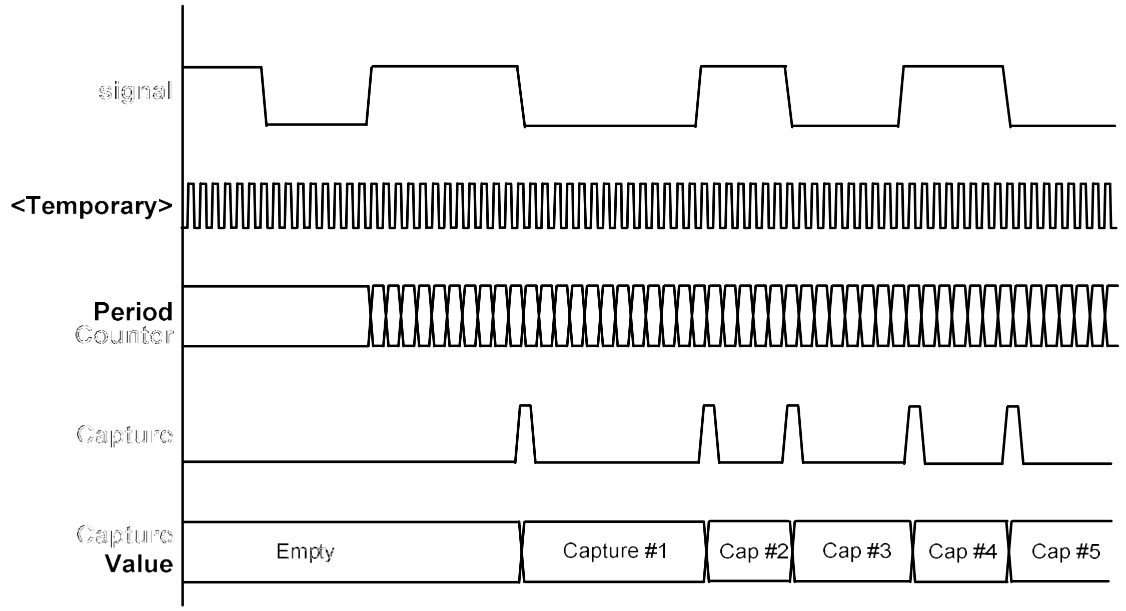
By default, the capture functionality is configured to capture on every rising edge of the capture input. Because the default configuration is to use the fixed-function block, the only option for capture mode is rising edge. Other modes are available when the implementation is changed to UDB.

The tc output becomes a periodic event signal. For a periodic interrupt to the CPU, you should connect an interrupt component to the tc or interrupt output. If you use the interrupt output, you must configure the interrupt source.

High/Low Time Measurement Configuration

To measure the high and low times, or pulse width, of a signal, connect the signal to both the trigger and capture inputs. Configure the **Trigger Mode** as **Rising Edge** and the **Capture Mode** as **Either Edge** or **Falling Edge**. The Timer will start counting on the first rising edge and will be captured at the next falling edge. This gives you the pulse width of the signal. Also, subsequent captures of either edge will give you sequential high and low times of the signal. Subsequent falling edge captures will give you further pulse widths

Figure 3. High/Low and Pulse Width Time Calculations



High Time #1 = (Period – Capture #1) × Clock Frequency

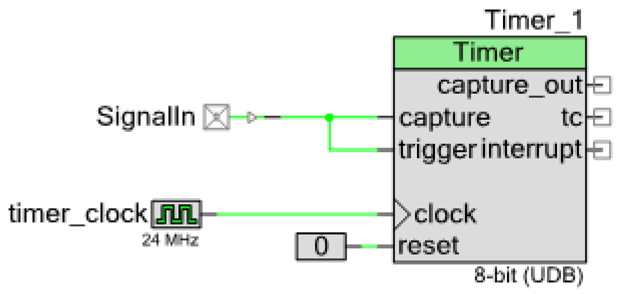
Low Time #1 = (Capture #1 – Capture #2) × Clock Frequency

High Time #2 = (Capture #2 – Capture #3) × Clock Frequency

And so on.

With the schematic implementation shown in Figure 4, setting the **Trigger Mode** to **Falling Edge** first measures the low time and continues with alternating edge types until the timer is enabled or reset:

Figure 4. Timer Schematic



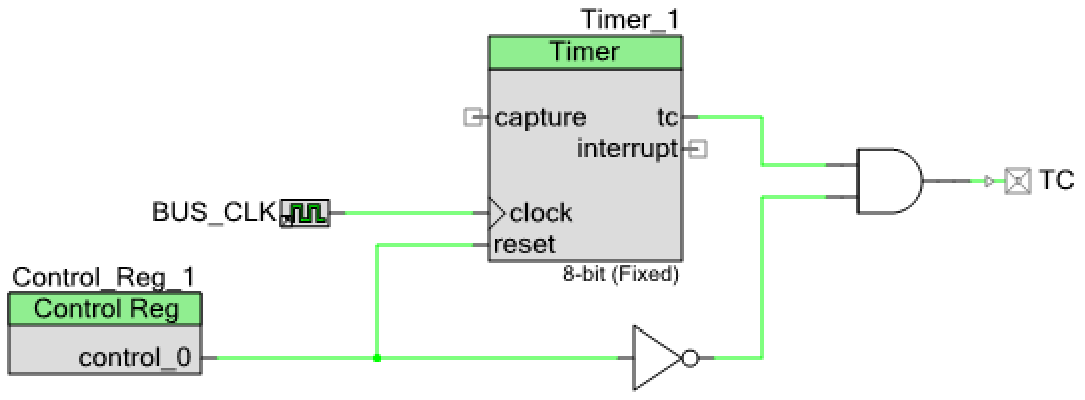
Fixed-Function Configuration

When configured to use the fixed-function block for the Timer implementation, the Timer component is limited in placement options to one of the fixed-function blocks on the chip. You should consider your resource needs when choosing to implement the Timer component in the fixed-function block because an 8-bit timer placed there wastes half of the block.

Reset in Fixed-Function Block

On PSoC3 ES2 silicon, the fixed-function implementation of the Timer differs from the UDB implementation in that the TC during reset goes high, whereas in the UDB implementation the TC goes low. Figure 5 shows a fixed-function Timer implementation that drives the TC low while the reset input is active, giving the same functionality as the UDB implementation of the same component.

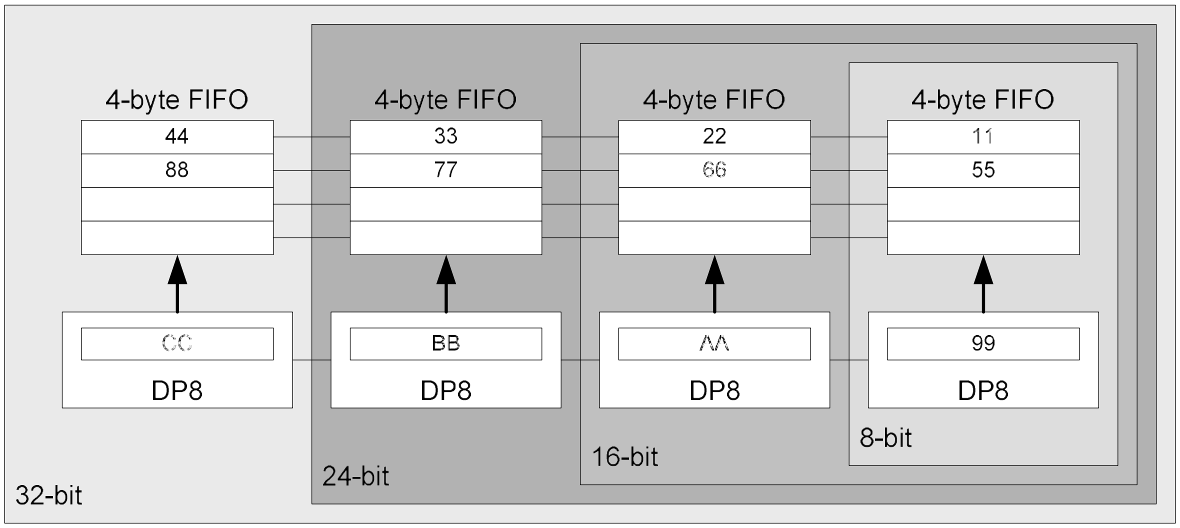
Figure 5. Fixed-Function Timer Implementation



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 Timer 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 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 Timer. Use the Timer_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 is registered at the input clock edge of the Timer, which gives all sticky bits the timing resolution of the Timer. For a UDB implementation, the status register is clocked from the bus clock. For a FF implementation, the status register is clocked from the input clock. All nonsticky bits are transparent and read directly from the inputs to the status register.

Timer_Status (UDB Implementation)

Bits	7	6	5	4	3	2	1	0
Name	RSVD	RSVD	RSVD	RSVD	FIFO Not Empty	FIFO Full	Capture	TC
Sticky	N/A	N/A	N/A	N/A	FALSE	FALSE	TRUE	TRUE

Timer_Status (Fixed Function Implementation)

Bits	7	6	5	4	3	2	1	0
Name	TC	Capture	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD
Sticky	TRUE	TRUE	N/A	N/A	N/A	N/A	N/A	N/A

Bit Name	#define in header file	Description
TC	Timer_STATUS_TC	This bit goes to 1 when the counter value is equal to zero.
Capture	Timer_STATUS_CAPTURE	This bit goes to 1 whenever a valid capture event is triggered. This does not include software capture.
FIFO Full	Timer_STATUS_FIFOFULL	This bit goes to 1 when the UDB FIFO reaches the full state defined as four entries.
FIFO Not Empty	Timer_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 Timer_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 Timer 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.

Timer_Mode (UDB Implementation)

Bits	7	6	5	4	3	2	1	0
Name	RSVD	RSVD	RSVD	RSVD	RSVD	FIFO Full	Catpure	TC

Timer_Mode (Fixed-Function Implementation)

Bits	7	6	5	4	3	2	1	0
Name	RSVD	RSVD	RSVD	RSVD	TC	Capture	RSVD	RSVD

Bit Name	#define in header file	Enables Interrupt Output On
TC	Timer_STATUS_TC_INT_MASK	Counter register equals 0
Capture	Timer_STATUS_CAPTURE_INT_MASK	Capture
FIFO Full	Timer_STATUS_FIFOFULL_INT_MASK	UDB FIFO full

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.

Timer_Control (UDB Implementation)

Bits	7	6	5	4	3	2	1	0
Name	Enable	Capture Mode [1:0]		Trigger Enable	Trigger Mode [1:0]		Interrupt Count [1:0]	

Timer_Control1 (Fixed-Function Implementation)

Bits	7	6	5	4	3	2	1	0
Name	Enable	RSVD	RSVD	RSVD	RSVD	ONESHOT	RSVD	EN

ONESHOT: The Timer stops when it reaches a stop condition defined by TMR_CFG bits. It can be restarted by asserting TIMER RESET or disabling and re-enabling block

EN: Enables the Timer.

Timer_Control2 (Fixed Function Implementation)

Bits	7	6	5	4	3	2	1	0
Name	Enable	RSVD	RSVD	RSVD	RSVD	RSVD	FTC	IRQ_SEL

FTC: First Terminal Count (FTC). Setting this bit forces a single pulse on the TC pin when first enabled.

IRQ_SEL: Irq selection. (0 = raw interrupts; 1 = status register interrupts)

Timer_Control3 (Fixed Function Implementation)

Bits	7	6	5	4	3	2	1	0
ame	Enable	RSVD	RSVD	RSVD	ROD	COD	TMR_CFG[1:0]	

ROD: Reset On Disable (ROD). Resets the internal state of the output logic.

COD: Clear On Disable (COD). Clears or gates outputs to zero.

TMR_CFG[1:0]: Timer configuration

The Timer_Control3 Register exists only for PSoC 3 Production silicon.

Bit Name	#define in header file	Description / Enumerated Type
Interrupt Count	Timer_CTRL_INTCNT_MASK	The interrupt count bits define the number of capture events to count before an interrupt is fired.
Trigger Mode	Timer_CTRL_TRIG_MODE_MASK	The trigger mode control bits define the expected trigger input functionality. This bit field is configured at initialization with the trigger mode defined in the TriggerMode parameter. ▪ Timer__B_TIMER__TM_NONE ▪ Timer__B_TIMER__TM_RISINGEDGE ▪ Timer__B_TIMER__TM_FALLINGEDGE ▪ Timer__B_TIMER__TM_EITHEREDGE ▪ Timer__B_TIMER__TM_SOFTWARE
Trigger Enable	Timer_CTRL_TRIG_EN	The Trigger Enable bit allows for software control of when to prepare for a trigger event.

Bit Name	#define in header file	Description / Enumerated Type
Capture Mode	Timer_CTRL_CAP_MODE_MASK	The capture mode control bits are a two-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. ▪ Timer__B_TIMER__CM_NONE ▪ Timer__B_TIMER__CM_RISINGEDGE ▪ Timer__B_TIMER__CM_FALLINGEDGE ▪ Timer__B_TIMER__CM_EITHEREDGE ▪ Timer__B_TIMER__CM_SOFTWARE
Enable	Timer_CTRL_ENABLE	Enables counting under software control. This bit is valid only if the **Enable Mode** parameter is set to **Software Only** or **Software and Hardware**.

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

The counter register contains the current counter value. This register is decremented in response to the rising edge of all clock inputs. This register may be read at any time with the Timer_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 the UDB FIFOs section for details.

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

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

Component Debug Window

The Timer component supports the PSoC Creator component debug window. The following registers are displayed in the debug window. Some registers are available in the UDB implementation (indicated by *) and some registers are only available in the fixed-function Implementation (indicated by **). All other registers are available for either configuration.

Register:	Timer_1_CONTROL
Name:	Control Register
Description:	Refer to the Timer_Control register description earlier in this datasheet for bit-field definitions.

Register:	Timer_1_CONTROL2 **
Name:	Fixed-Function Control Register #2
Description:	The fixed-function Timer block has a second configuration register. Refer to the Technical Reference Manual for bit field definitions.

Register:	Timer_1_STATUS_MASK *
Name:	Status Register Interrupt Mask Configuration
Description:	Allows you to enable any status bit as an interrupt source at the interrupt output pin of the component. Refer to the Timer_Status register description earlier in this datasheet for one-to-one correlation of bit-field definitions.

Register:	Timer_1_STATUS_AUX_CTRL *
Name:	Auxiliary Control Register for the Status Register
Description:	Allows you to enable the interrupt output of the internal status register through the bit field INT_EN. Refer to the Technical Reference Manual for bit-field definitions.

Register:	Timer_1_STATUS_MASK *
Name:	Status Register Interrupt Mask Configuration
Description:	Enables interrupt mask bits with a one-to-one correspondence to the status register (Timer_Status) bit-field descriptions described earlier.

Register:	Timer_1_PERIOD
Name:	Timer Period Register
Description:	Defines the period value reloaded into the period counter at the beginning of each cycle of the Timer.

Register:	Timer_1_COUNTER
Name:	Timer Counter Register
Description:	Indicates the current counter value (in clock cycles from **Period** down to zero) of the current timer period cycle.

Register:	Timer_1_GLOBAL_ENABLE **
Name:	Fixed Function Timer Global Enable Register
Description:	Enables the Fixed-Function Timer for operation. Refer to the Technical Reference Manual for bit-field definitions.

DC and AC Electrical Characteristics (FF Implementation)

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

Timer DC Specifications

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

Timer AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Operating frequency		DC	–	67	MHz
	Capture pulse width (internal)		15	–	–	ns
	Capture pulse width (external)		30	–	–	ns
	Timer resolution		15	–	–	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.	Min	Typ	Max	Units
f_{CLOCK}	Component clock frequency	8-bit UDB Timer	–	–	40	MHz
		16-bit UDB Timer	–	–	38	MHz
		24-bit UDB Timer	–	–	33	MHz
		32-bit UDB Timer	–	–	27	MHz
t_{clockH}	Input clock high time [1]	N/A	–	0.5	–	t_{CY_clock}
t_{clockL}	Input clock low time [1]	N/A	–	0.5	–	t_{CY_clock}
Inputs						
t_{PD_ps}	Input path delay, pin to sync [2]	1	–	–	STA [3]	ns
t_{PD_ps}	Input path delay, pin to sync	2	–	–	8.5	ns
t_{PD_si}	Sync output to input path delay (route) [2]	1,2,3,4	–	–	STA [3]	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}	–	–	ns
t_{IL}	Input low time	1,2,3,4	t_{CY_clock}	–	–	ns

[1] $t_{CY\_clock} = 1/f_{CLOCK}$. This is the cycle time of one clock period.

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

[3] 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.

Timing Characteristics “Maximum with All Routing”

Parameter	Description	Config.	Min	Typ	Max [1]	Units
f_{CLOCK}	Component clock frequency	8-bit UDB Timer	–	–	20	MHz
		16-bit UDB Timer	–	–	15	MHz
		24-bit UDB Timer	–	–	20	MHz
		32-bit UDB Timer	–	–	15	MHz
t_{clockH}	Input clock high time [2]	N/A	–	0.5	–	$1/f_{clock}$
t_{clockL}	Input clock low time	N/A	–	0.5	–	$1/f_{clock}$
Inputs						
t_{PD_ps}	Input path delay, pin to sync [3]	1	–	–	STA	ns
t_{PD_ps}	Input path delay, pin to sync [4]	2	–	–	8.5	ns
t_{PD_si}	Sync output to input path delay (route) [3]	1,2,3,4	–	–	STA [3]	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}	–	–	ns
t_{IL}	Input low time	1,2,3,4	t_{CY_clock}	–	–	ns

[1] 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.

[2] $t_{CY\_clock} = 1/f_{CLOCK}$. This is the cycle time of one clock period.

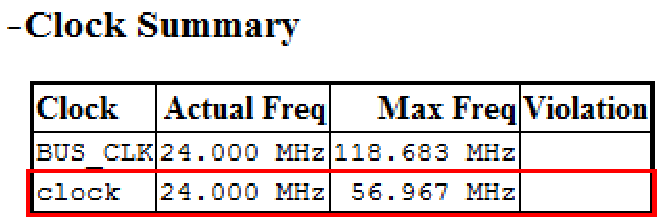
[3] t_{PD_ps} and t_{PD_si} are route path delays. Because routing is dynamic, these values can change and directly affect the maximum component clock and sync clock frequencies. The values must be found in the Static Timing Analysis results.

[4] 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.

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 with 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

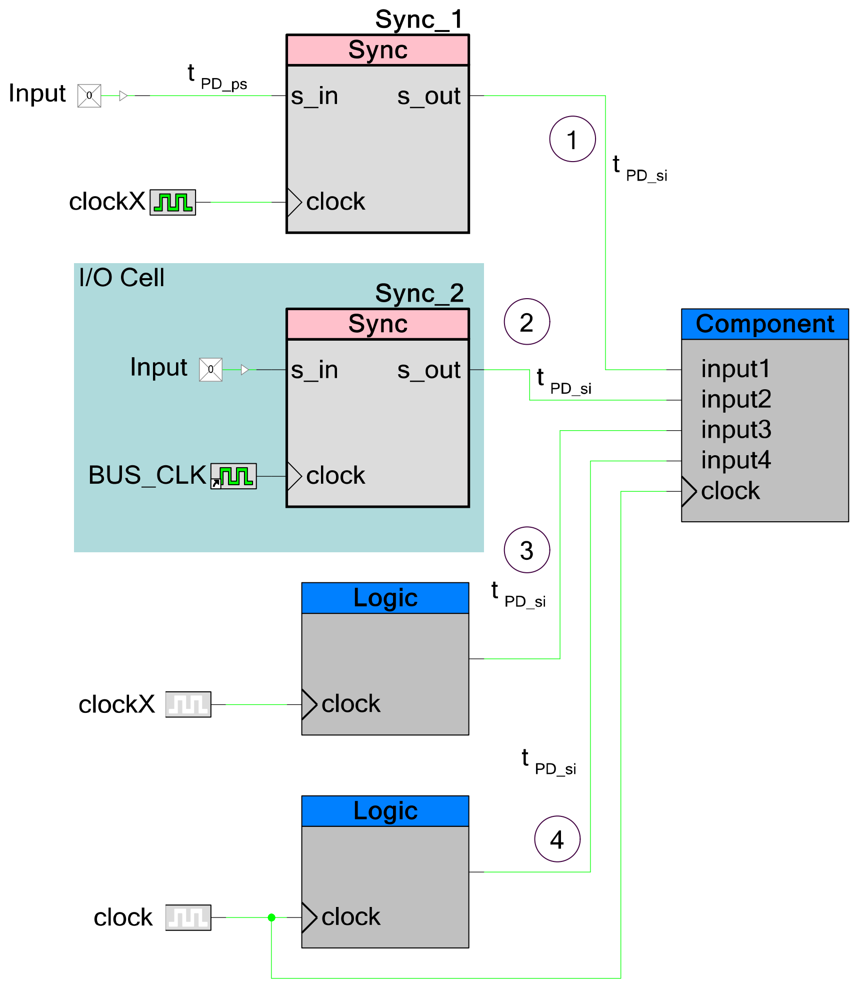
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

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 6.

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 6. Input Configurations for Component Timing Specifications



Configuration	Component Clock	Synchronizer Clock (Frequency)	Figures
1	master_clock	master_clock	Figure 11
1	clock	master_clock	Figure 9
1	clock	clockX = clock [1]	Figure 7
1	clock	clockX > clock	Figure 8
1	clock	clockX < clock	Figure 10
2	master_clock	master_clock	Figure 11
2	clock	master_clock	Figure 9
3	master_clock	master_clock	Figure 16

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

Configuration	Component Clock	Synchronizer Clock (Frequency)	Figures
3	clock	master_clock	Figure 14
3	clock	clockX = clock [1]	Figure 12
3	clock	clockX > clock	Figure 13
3	clock	clockX < clock	Figure 15
4	master_clock	master_clock	Figure 16
4	clock	clock	Figure 12

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 than, equal to, or slower than the component clock. It may also be equal to master_clock. This produces the characterization parameters shown in Figure 7, Figure 8, Figure 10, and Figure 11.

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 8 and Figure 11.

Figure 7. Input Configuration 1 and 2; Sync Clock Frequency = Component Clock Frequency (Edge alignment of clock and clockX is not guaranteed)

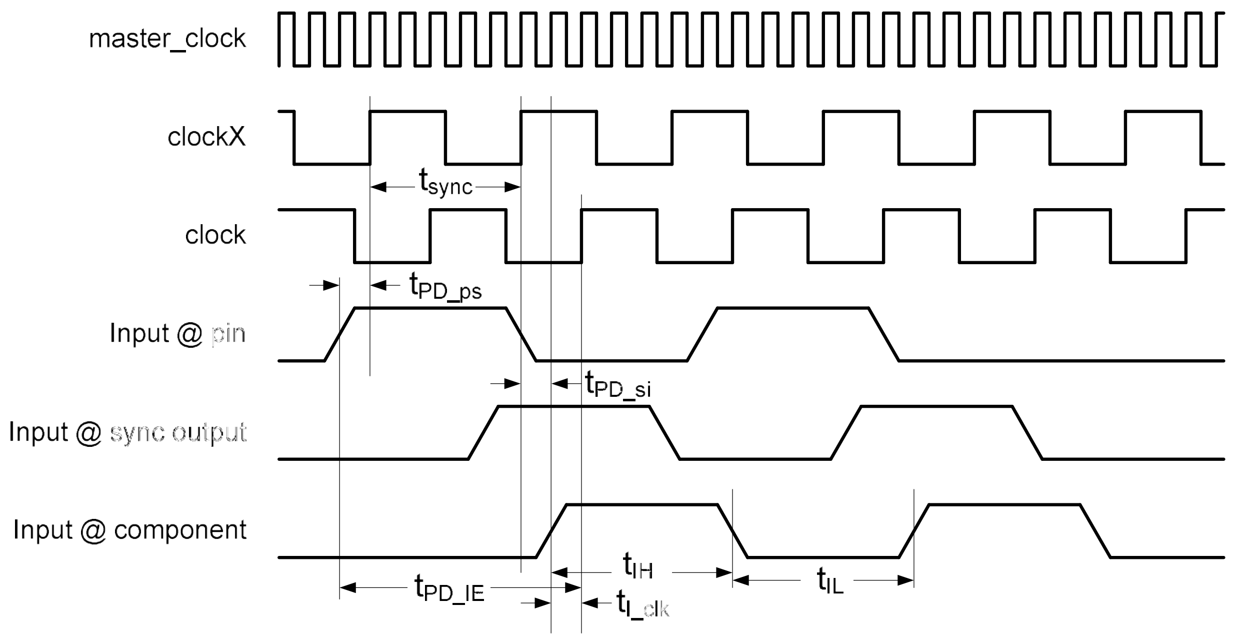


Figure 8. Input Configuration 1 and 2; Sync. Clock Frequency > Component Clock Frequency

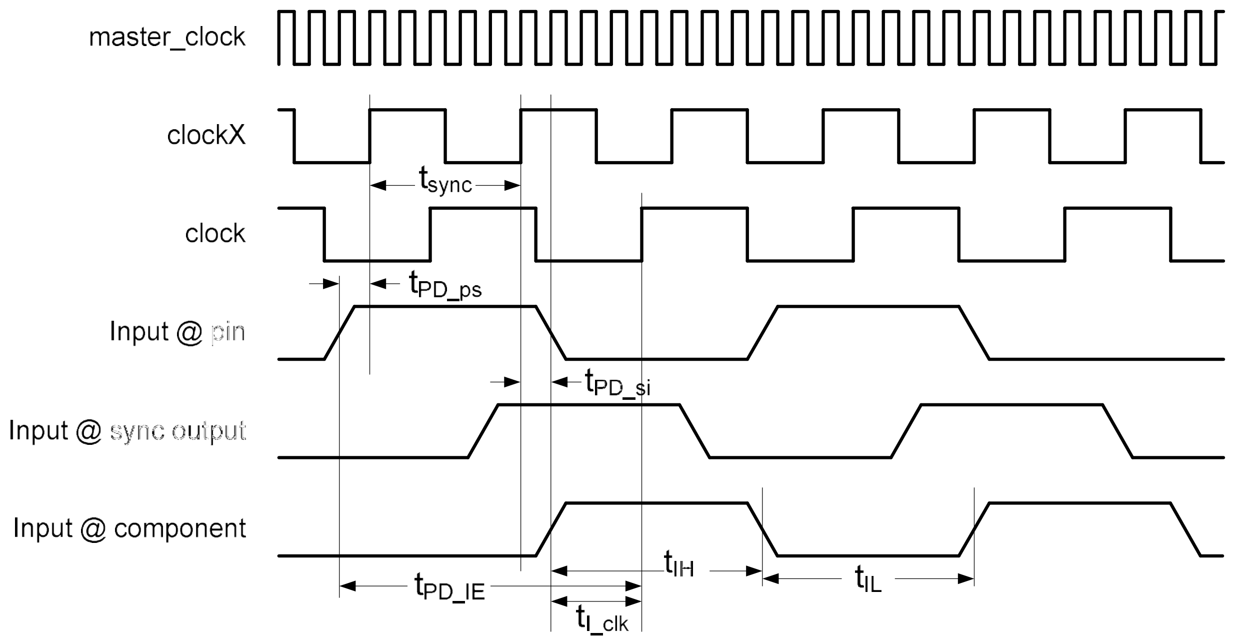


Figure 9. Input Configuration 1 and 2; [Sync. Clock Frequency == master_clock] > Component Clock Frequency

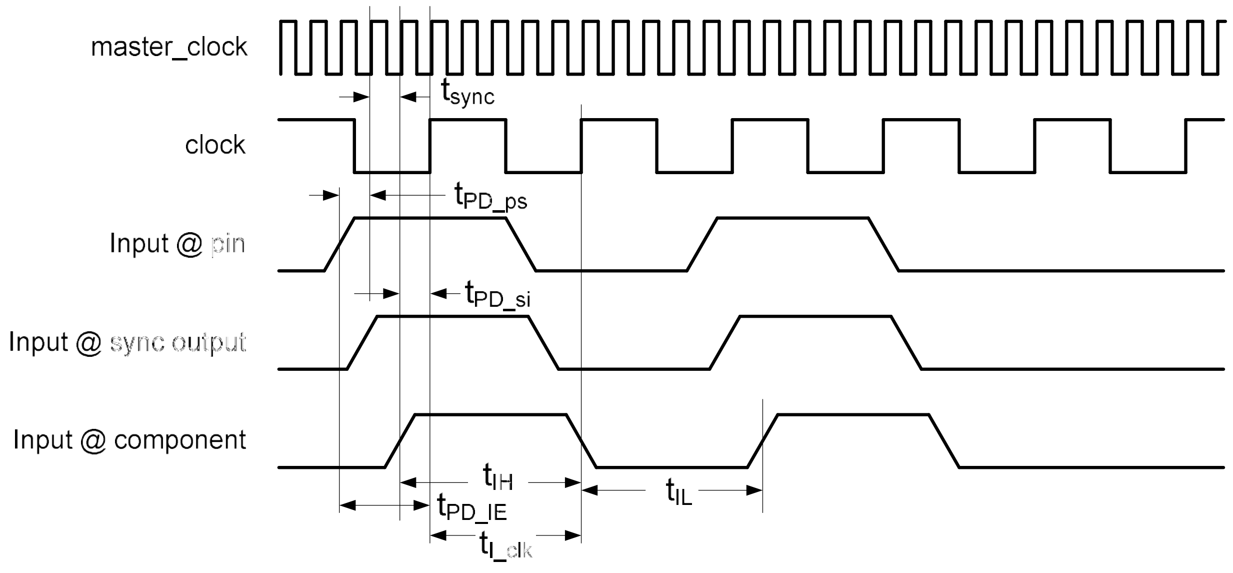


Figure 10. Input Configuration 1; Sync. Clock Frequency < Component Clock Frequency

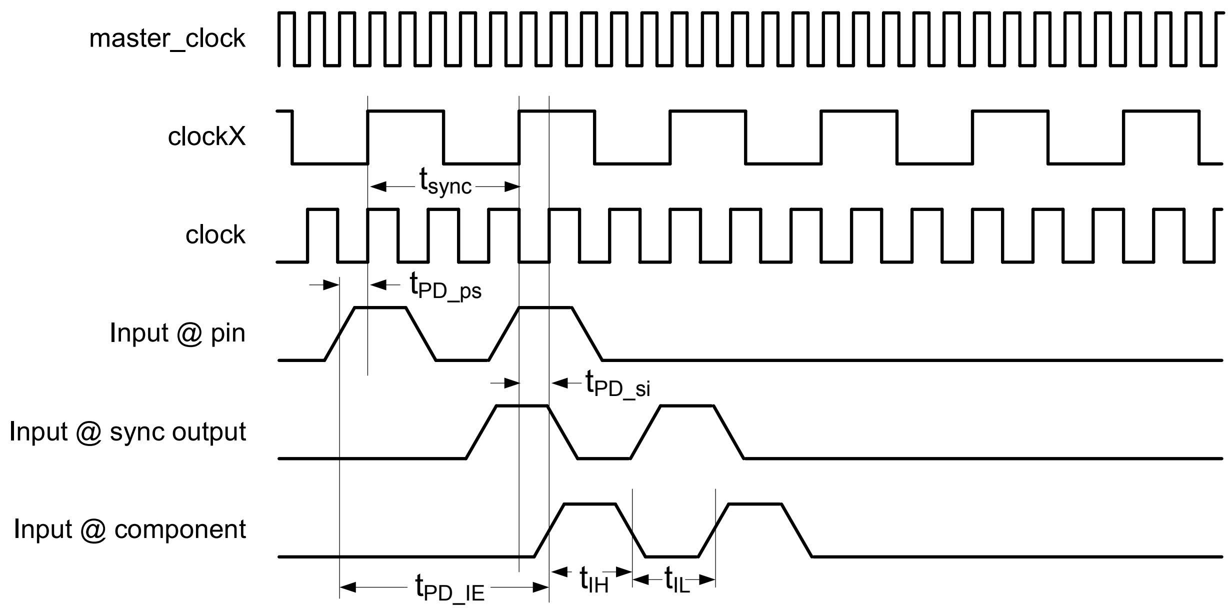
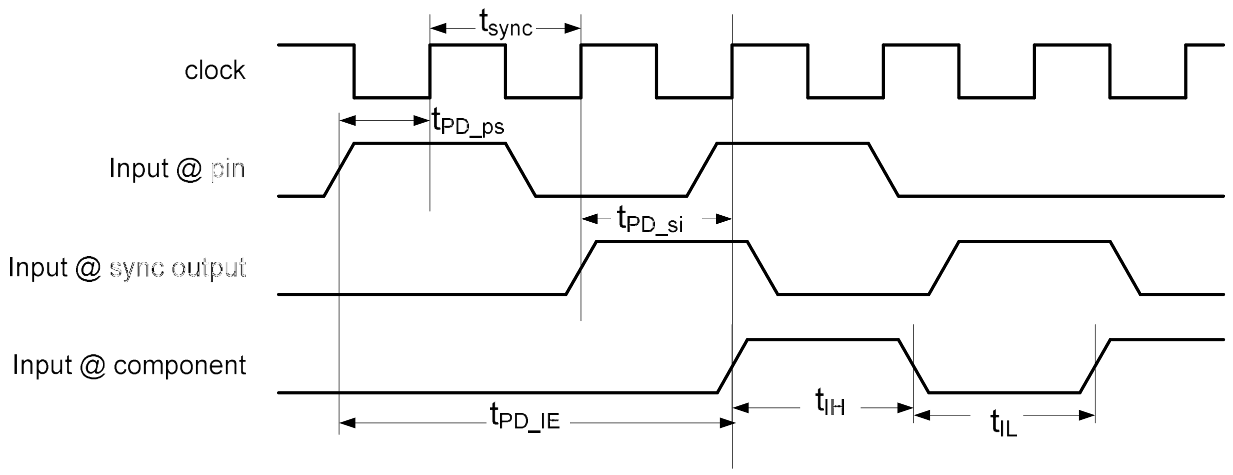


Figure 11. Input Configuration 1 and 2; Sync. Clock = Component Clock = master_clock



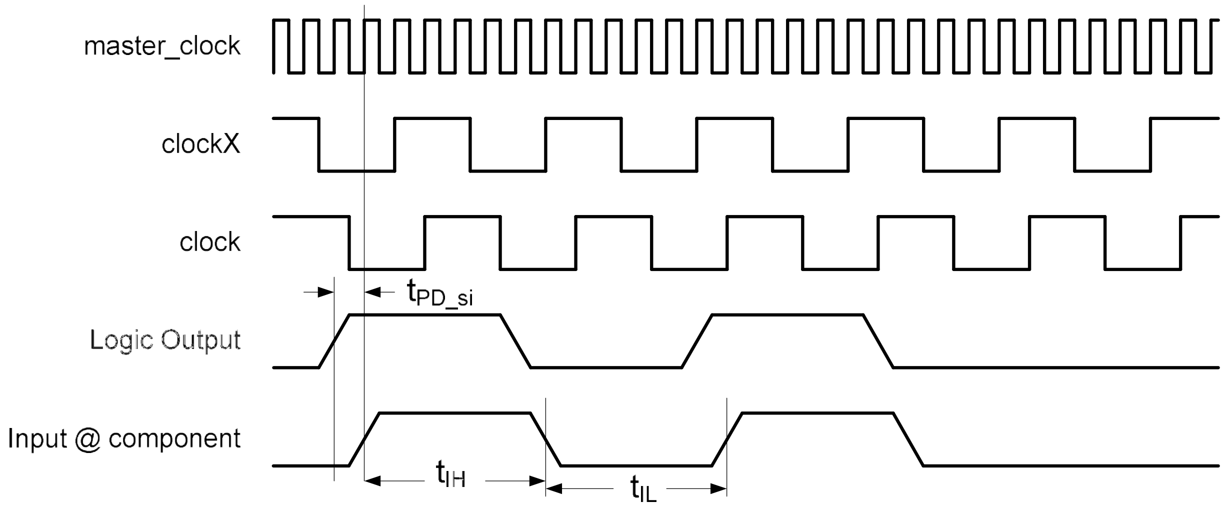
3. The input is driven by logic internal to the PSoC, which is synchronous based on a clock other than the clock the component uses (all internal clocks are derived from master_clock).

   When characterizing inputs configured in this way, the synchronizer clock is faster than, slower than, or equal to the component clock. This produces the characterization parameters shown in Figure 12, Figure 13, and Figure 15.

4. The input is driven by logic internal to the PSoC, which is synchronous based on the same clock the component uses.

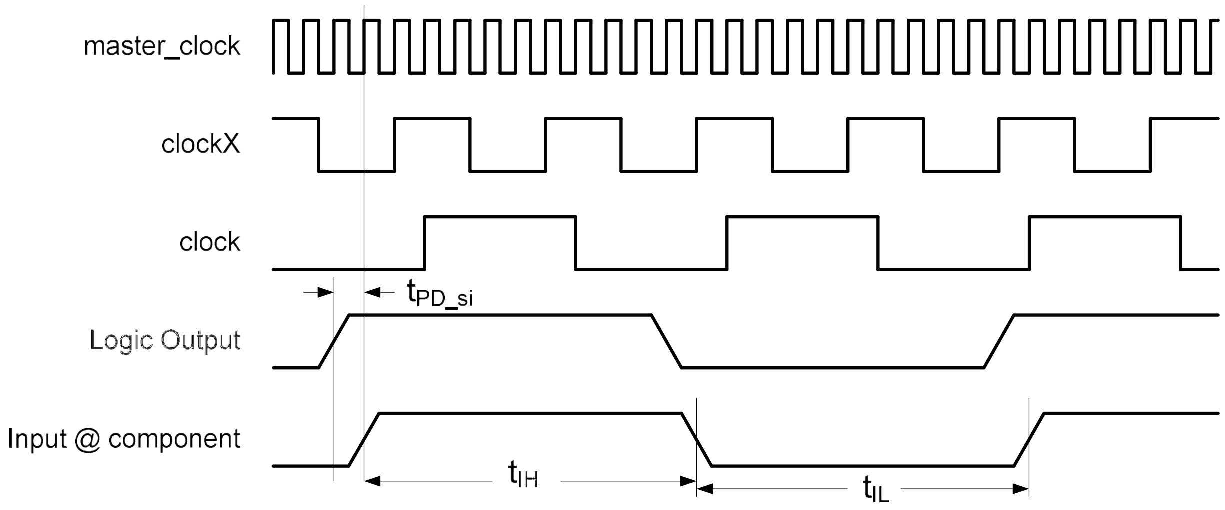
   When characterizing inputs configured in this way, the synchronizer clock is equal to the component clock. This produces the characterization parameters as shown in Figure 16.

Figure 12. Input Configuration 3 only; Sync. Clock Frequency = Component Clock Frequency (Edge alignment of clock and clockX is not guaranteed)



This figure represents the information that Static Timing Analysis has about 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 one master_clock cycle. This means that t_{PD_si} now has a limiting effect on the system master_clock. 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 13. Input Configuration 3; Sync. Clock Frequency > Component Clock Frequency



In much the same way as shown in Figure 12, 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 14. Input Configuration 3; Synchronizer Clock Frequency = master_clock > Component Clock Frequency

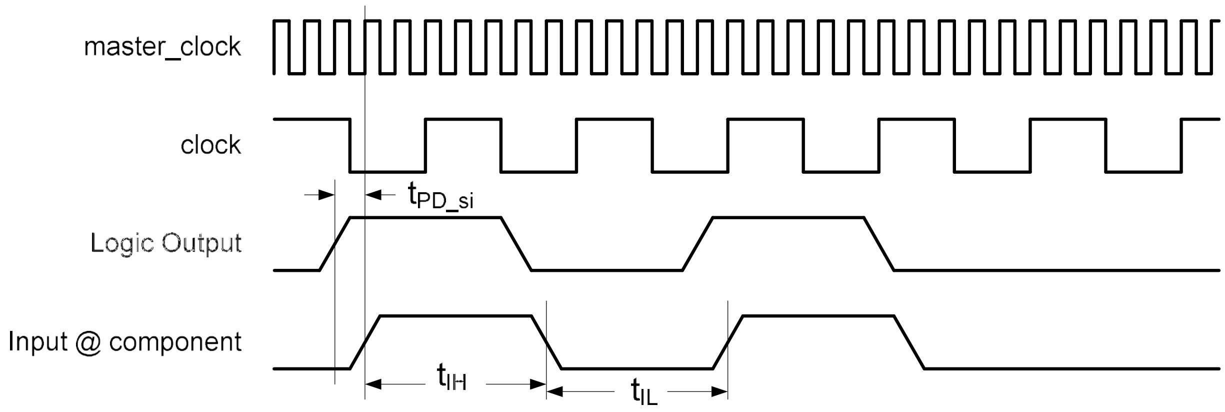
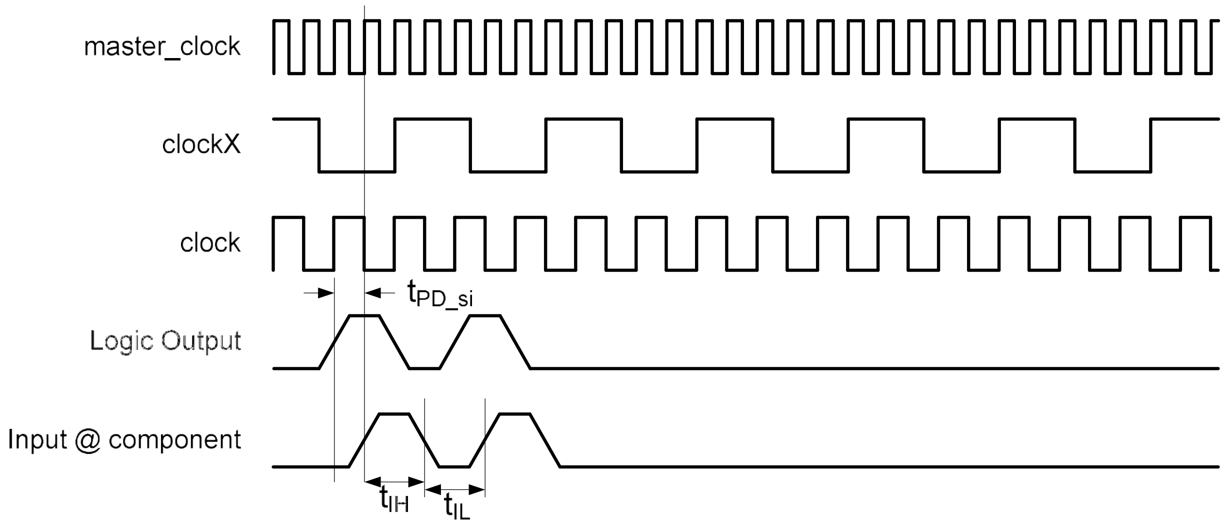
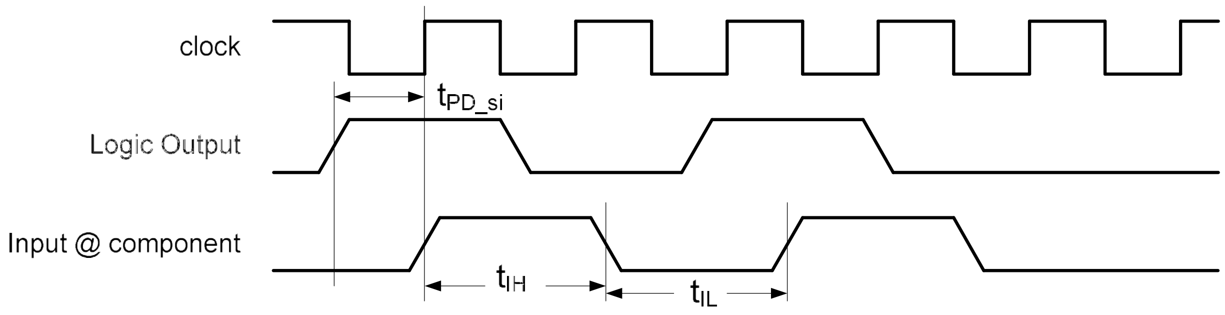


Figure 15. Input Configuration 3; Synchronizer Clock Frequency < Component Clock Frequency



In much the same way as shown in Figure 12, 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.

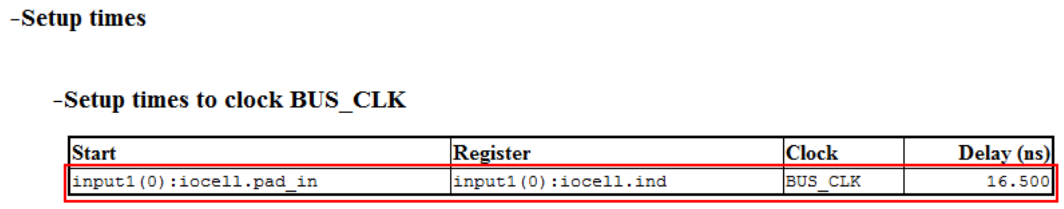
Figure 16. Input Configuration 4 only; Synchronizer Clock = Component Clock



In all previous figures in this section, the most critical parameters to use to understand 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}. It is critical to note 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 fanout of this input may be more than 1 so you will need to evaluate the maximum of these paths.

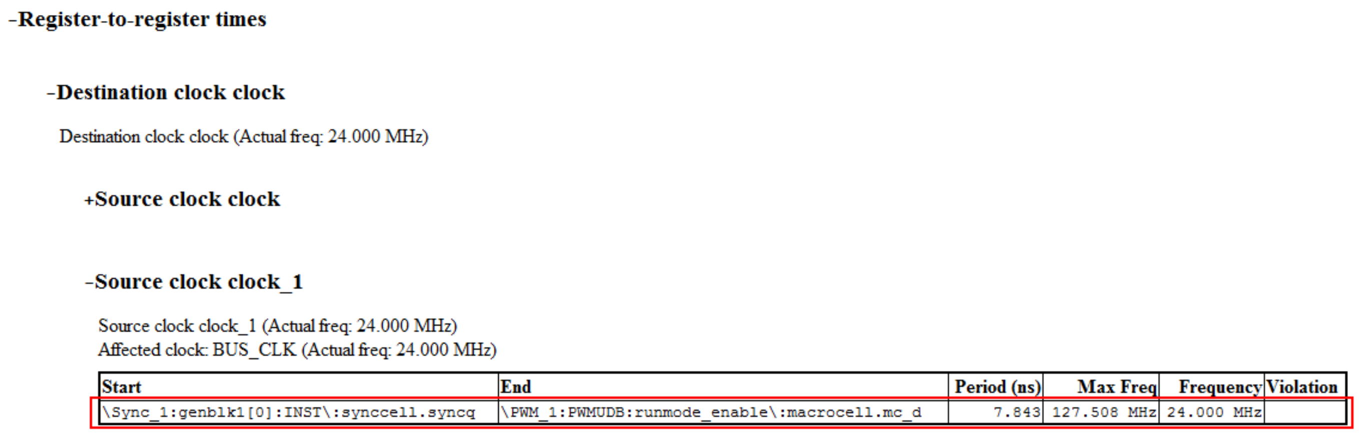


-**Setup times**

-**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 need to know the name of the net to use the *_timing.html* file. The fanout 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 just shown (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
2.10	Verilog update and customizer related updates	To fix a minor issue with Trigger logic and GUI related issues
	"Interrupt on Capture" is disabled when Capture Mode is set to None	"Interrupt on Capture" check box option was available even when Capture Mode is set to "None" and should not be made available
2.0	Synchronized inputs	All inputs are synchronized in the fixed-function implementation, at the input of the block.
	Timer_GetInterruptSource() function was converted to a Macro	The Timer_GetInterruptSource() function is exactly the same implementation as the Timer_ReadStatusRegister() function. To save code space this was converted to a macro substitution of the Timer_ReadStatusRegister() function.
	Outputs are now registered to the component clock	To avoid glitches on the outputs of the component it is required that all outputs be synchronized. This is done inside of the datapath when possible, to avoid excess resource use.
	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.
	Incorrect masking rectified while setting capture mode using SetCaptureMode() API.	Masking used for setting capture mode has erroneous value.
	Added characterization data to datasheet	
	Minor datasheet edits and updates	