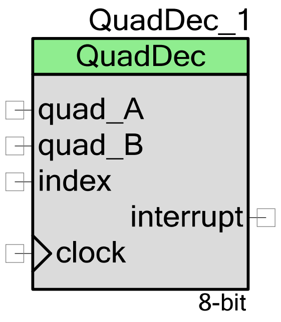
Quadrature Decoder (QuadDec)

2.0



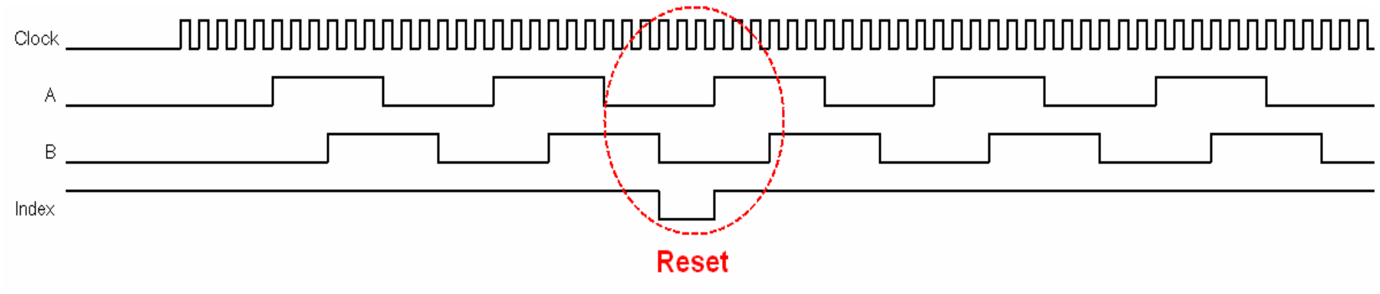
Features

- Adjustable counter size: 8, 16, or 32 bits
- Counter resolution of 1x, 2x, or 4x the frequency of the A and B inputs, for more accurate determination of position or speed
- Optional index input to determine absolute position
- Optional glitch filtering to reduce the impact of system-generated noise on the inputs

General Description

The Quadrature Decoder (QuadDec) Component gives you the ability to count transitions on a pair of digital signals. The signals are typically provided by a speed/position feedback system mounted on a motor or trackball.

The signals, typically called A and B, are positioned 90 degrees out of phase, which results in a Gray code output. A Gray code is a sequence where only one bit changes on each count. This is essential to avoid glitches. It also allows detection of direction and relative position. A third optional signal, named Index, is used as a reference to establish an absolute position once per rotation.



When to Use a Quadrature Decoder

A quadrature decoder is used to decode the output of a quadrature encoder. A quadrature encoder senses the current position, velocity, and direction of an object (for example, mouse, trackball, robotic axles, and others).

A quadrature decoder can also be used for precision measurement of speed, acceleration, and position of a motor's rotor and with rotary knobs to determine user input.

Input/Output Connections

This section describes the various input and output connections for the Quadrature Decoder Component. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

quad_A – Input

The “A” input of the Quadrature Decoder.

quad_B – Input

The “B” input of the Quadrature Decoder.

index – Input *

This input detects a reference position for the Quadrature Decoder. When using an index input, if inputs A, B, and index are all zero, the counter is also reset to zero. Additional logic is typically added to gate the index pulse. Index gating allows the counter to only be reset during one of many possible rotations. An example is a linear actuator that only resets the counter when the far limit of travel has been reached. This limit is signaled by a mechanical limit switch whose output is connected to the Index pulse.

This input displays by default, but it can be hidden by deselecting the **Use index input** parameter.

clock – Input

Clock signal for sampling and glitch filtering the inputs. If you are using glitch filtering, the filtered outputs will not change until three successive samples of the input have the same value. For effective glitch filtering, the sample clock period should be greater than the maximum time during which glitching is expected to take place. A counter can be incremented or decremented at a resolution of 1x, 2x, or 4x the frequency of the A and B inputs.

The clock input frequency should be greater than or equal to 10x the maximum A or B input frequency.

interrupt – Output

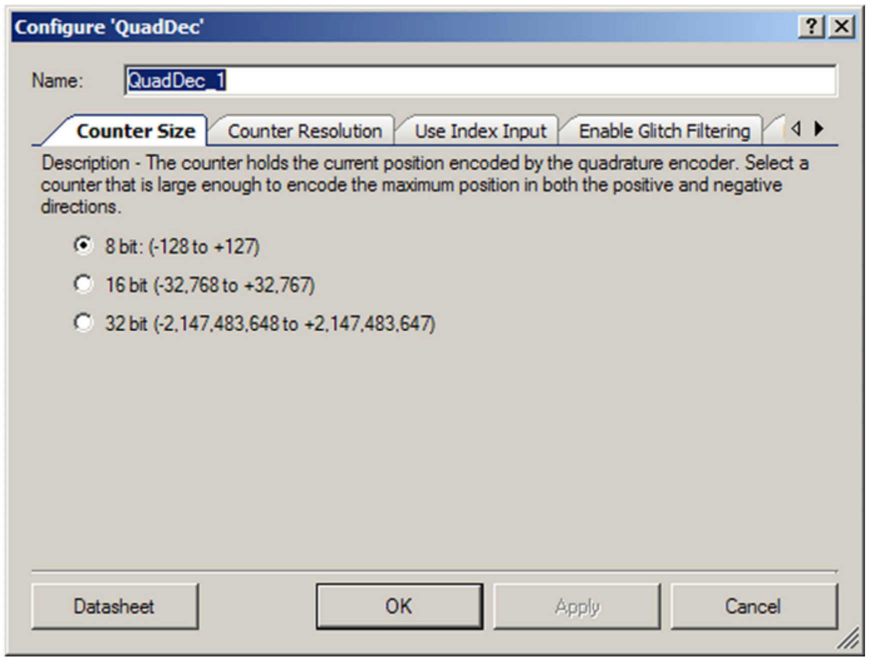
Interrupt on one or more of the following events:

- Counter overflow and underflow
- Counter reset due to index input (if index is used)
- Invalid state transition on the A and B inputs

Component Parameters

Drag a Quadrature Decoder component onto your design and double-click it to open the **Configure** dialog. The dialog contains multiple tabs with categorized parameters.

Counter Size Tab



Name: QuadDec_1

Description - The counter holds the current position encoded by the quadrature encoder. Select a counter that is large enough to encode the maximum position in both the positive and negative directions.

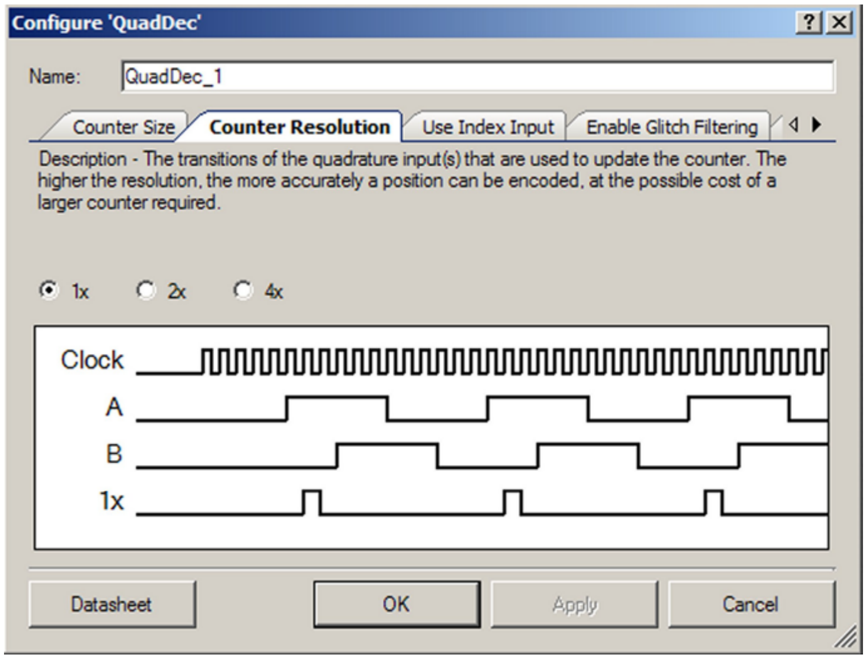
[✓ 8 bit: (-128 to +127)]
16 bit (-32,768 to +32,767)
32 bit (-2,147,483,648 to +2,147,483,647)

This tab is used to define the counter size, in bits. The counter holds the current position encoded by a quadrature encoder.

Select a counter that is large enough to encode the maximum position in both the positive and negative directions. The setting can be: **8 bit**, **16 bit**, or **32 bit**.

The 32-bit counter implements the lower 16 bits in the hardware counter and the upper 16 bits in software to reduce hardware resource use. For this target, an additional ISR is used. To work properly with the 32-bit counter, interrupts must be enabled. You can add ISR code to source files as needed; see the Interrupt Component datasheet for more details.

Counter Resolution Tab


Configure 'QuadDec'

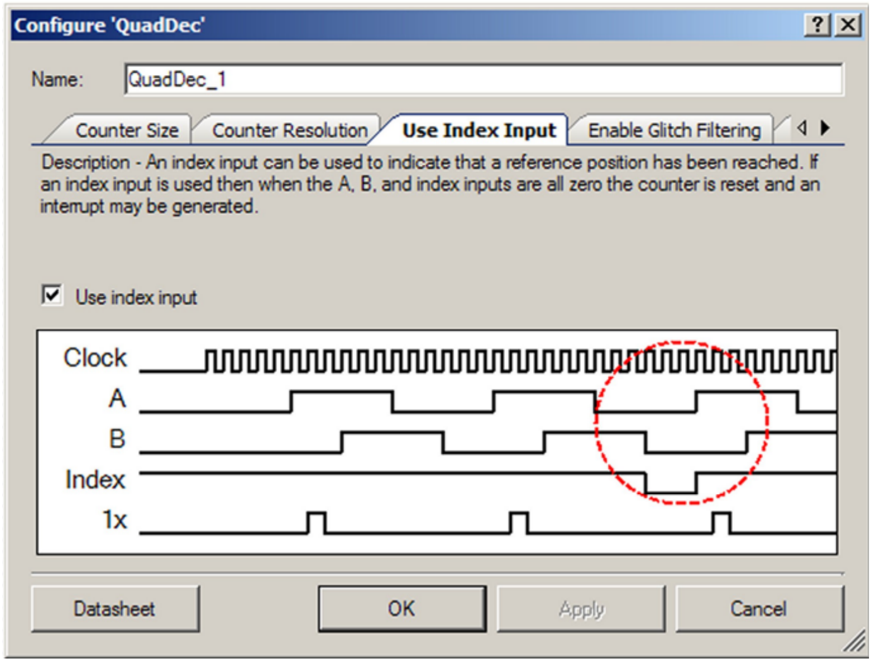
Name: QuadDec_1

Description - The transitions of the quadrature input(s) that are used to update the counter. The higher the resolution, the more accurately a position can be encoded, at the possible cost of a larger counter required.

[✓ 1x] 2x 4x

This tab contains the number of counts recorded in one period of the A and B inputs. It shows the transitions of the input signals that are used to update the counter. As the resolution gets higher, the position can be resolved more accurately, at the possible cost of a larger counter. The setting can be **1x**, **2x**, or **4x**.

Use Index Input Tab


Configure 'QuadDec'

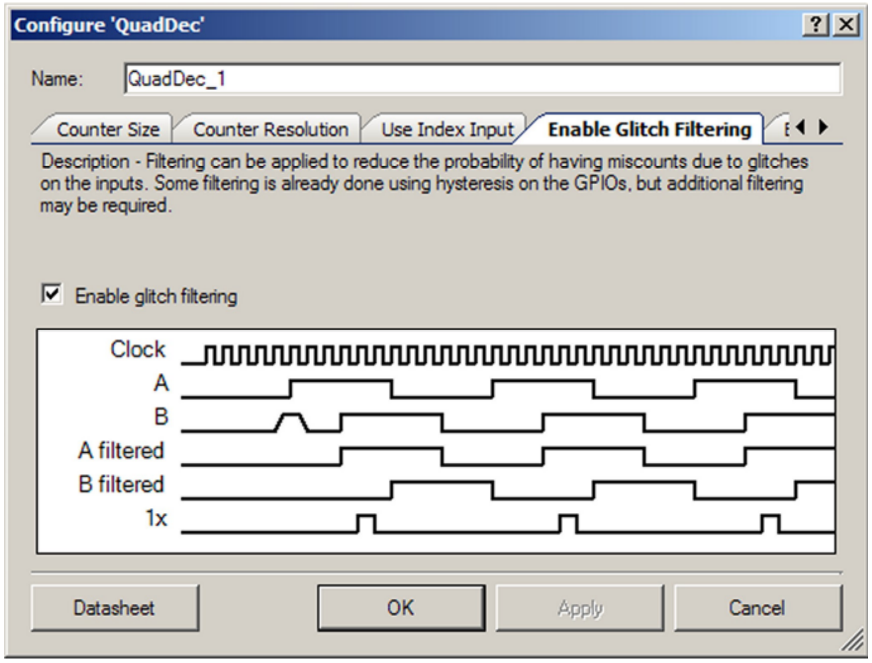
Name: QuadDec_1

Description - An index input can be used to indicate that a reference position has been reached. If an index input is used then when the A, B, and index inputs are all zero the counter is reset and an interrupt may be generated.

[✓] Use index input

This tab contains a field to enable or disable the index input. An index input can be used to indicate that a reference position has been reached. If an index input is used, then when the A, B, and index inputs are all zero, the counter is reset and an interrupt can be generated. Index input is enabled by default.

Enable Glitch Filtering Tab



Name: QuadDec_1

Description - Filtering can be applied to reduce the probability of having miscounts due to glitches on the inputs. Some filtering is already done using hysteresis on the GPIOs, but additional filtering may be required.

[✓] Enable glitch filtering

This tab contains a field to enable or disable digital glitch filtering. Filtering can be applied to reduce the probability of miscounts because of glitches on the inputs. Some filtering is already done using hysteresis on the GPIOs, but additional filtering could be required.

If enabled, filtering is applied to all inputs. The filtered outputs do not change until three successive samples of the input have the same value. For effective filtering, the period of the sample clock should be greater than the maximum time during which glitching is expected to occur. Glitch filtering is enabled by default.

Clock Selection

A clock source for clocking the Quadrature Decoder component must be connected. It clocks the status register and generates interrupts.

Placement

The Quadrature Decoder component is placed in the UDB array and all placement information is provided to the API through the *cyfitter.h* file.

Resources

1x Resolution without Glitch Filtering

Resources	Resource Type					API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Interrupts	Flash	RAM	
8 bits	1	6	2	1	0	566	7	2
8 bits *	1	6	2	1	0	566	7	3
16 bits	2	6	2	1	0	652	9	2
16 bits *	2	6	2	1	0	652	9	3
32 bits	2	6	2	1	1	906	14	2
32 bits *	2	9	2	1	1	906	14	3

* Using Index Input

1x Resolution with Glitch Filtering

Resources	Resource Type					API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Interrupts	Flash	RAM	
8 bits	1	7	2	1	0	566	7	2
8 bits *	1	9	2	1	0	566	7	3
16 bits	2	7	2	1	0	652	9	2
16 bits *	2	9	2	1	0	652	9	3
32 bits	2	7	2	1	1	906	14	2
32 bits *	2	9	2	1	1	906	14	3

* Using Index Input

2x Resolution without Glitch Filtering

Resources	Resource Type					API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Interrupts	Flash	RAM	
8 bits	1	6	2	1	0	566	7	2
8 bits *	1	7	2	1	0	566	7	3
16 bits	2	6	2	1	0	652	9	2
16 bits *	2	7	2	1	0	652	9	3
32 bits	2	6	2	1	1	906	14	2
32 bits *	2	7	2	1	1	906	14	3

* Using Index Input

2x Resolution with Glitch Filtering

Resources	Resource Type					API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Interrupts	Flash	RAM	
8 bits	1	8	2	1	0	566	7	2
8 bits *	1	9	2	1	0	566	7	3
16 bits	2	8	2	1	0	652	9	2
16 bits *	2	9	2	1	0	652	9	3
32 bits	2	8	2	1	1	906	14	2
32 bits *	2	9	2	1	1	906	14	3

* Using Index Input

4x Resolution without Glitch Filtering

Resources	Resource Type					API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Interrupts	Flash	RAM	
8 bits	1	7	2	1	0	566	7	2
8 bits *	1	7	2	1	0	566	7	3
16 bits	2	7	2	1	0	652	9	2
16 bits *	2	7	2	1	0	652	9	3
32 bits	2	7	2	1	1	906	14	2
32 bits *	2	7	2	1	1	906	14	3

* Using Index Input

4x Resolution with Glitch Filtering

Resources	Resource Type					API Memory (Bytes)		Pins (per External I/O)
	Datapath Cells	PLDs	Status Cells	Control/ Count7 Cells	Interrupts	Flash	RAM	
8 bits	1	8	2	1	0	566	7	2
8 bits *	1	9	2	1	0	566	7	3
16 bits	2	8	2	1	0	652	9	2
16 bits *	2	9	2	1	0	652	9	3
32 bits	2	8	2	1	1	906	14	2
32 bits *	2	9	2	1	1	906	14	3

* Using Index Input

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 “QuadDec_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 “QuadDec.”

Function	Description
QuadDec_Start()	Initializes UDBs and other relevant hardware
QuadDec_Stop()	Turns off UDBs and other relevant hardware
QuadDec_GetCounter()	Reports the current value of the counter
QuadDec_SetCounter()	Sets the current value of the counter
QuadDec_GetEvents()	Reports the current status of events
QuadDec_SetInterruptMask()	Enables or disables interrupts due to the events
QuadDec_GetInterruptMask()	Reports the current interrupt mask settings
QuadDec_Sleep()	Prepares the component to go to sleep
QuadDec_Wakeup()	Prepares the component to wake up
QuadDec_Init()	Initializes or restores default configuration provided with the customizer
QuadDec_Enable()	Enables the Quadrature Decoder
QuadDec_SaveConfig()	Saves the current user configuration
QuadDec_RestoreConfig()	Restores the user configuration

Global Variables

Function	Description
QuadDec_initVar	QuadDec_initVar indicates whether the Quadrature Decoder has been initialized. The variable is initialized to 0 and set to 1 the first time QuadDec_Start() is called. This allows the component to restart without reinitialization after the first call to the QuadDec_Start() routine. If reinitialization of the component is required, then the QuadDec_Init() function can be called before the QuadDec_Start() or QuadDec_Enable() function.
QuadDec_count32SoftPart	High 16 bits of 32-bit counter value is stored in this variable.
QuadDec_swStatus	Status register value is stored in this variable.

void QuadDec_Start(void)

Description:	Initializes UDBs and other relevant hardware. Resets counter to 0, and enables or disables all relevant interrupts. Starts monitoring the inputs and counting.
Parameters:	None
Return Value:	None
Side Effects:	None

void QuadDec_Stop(void)

Description:	Turns off UDBs and other relevant hardware.
Parameters:	None
Return Value:	None
Side Effects:	None

int8/16/32 QuadDec_GetCounter(void)

Description:	Reports the current value of the counter.
Parameters:	None
Return Value:	int8/16/32: Counter value. Return type is signed depending on the counter size setting. A positive value indicates clockwise movement (B before A).
Side Effects:	None

void QuadDec_SetCounter(int8/16/32 value)

Description:	Sets the current value of the counter.
Parameters:	int8/16/32 value: The new value. Parameter type is signed depending on the counter size setting.
Return Value:	None
Side Effects:	None

uint8 QuadDec_GetEvents(void)

Description: Reports the current status of events.

Parameters: None

Return Value: The events, as bits in an unsigned 8-bit value:

Bit	Description
QuadDec_COUNTER_OVERFLOW	Counter overflow
QuadDec_COUNTER_UNDERFLOW	Counter underflow
QuadDec_COUNTER_RESET	Counter reset due to index, if index input is used
QuadDec_INVALID_IN	Invalid A, B inputs state transition

Side Effects: None

void QuadDec_SetInterruptMask(uint8 mask)

Description: Enables or disables interrupts caused by the events. For the 32-bit counter, the overflow, underflow, and reset interrupts cannot be disabled; these bits are ignored.

Parameters: uint8 mask: Enable or disable bits in an 8-bit value, where 1 enables the interrupt:

Bit	Description
QuadDec_COUNTER_OVERFLOW	Enable interrupt caused by counter overflow
QuadDec_COUNTER_UNDERFLOW	Enable interrupt caused by counter underflow
QuadDec_COUNTER_RESET	Enable interrupt caused by counter reset
QuadDec_INVALID_IN	Enable interrupt caused by invalid input state transition

Return Value: None

Side Effects: None

uint8 QuadDec_GetInterruptMask(void)

Description: Reports the current interrupt mask settings.

Parameters: None

Return Value: Enable or disable bits in an 8-bit value, where 1 enables the interrupt.

For the 32-bit counter, the overflow, underflow, and reset enable bits are always set.

Bit	Description
QuadDec_COUNTER_OVERFLOW	Interrupt caused by counter overflow
QuadDec_COUNTER_UNDERFLOW	Interrupt caused by counter underflow
QuadDec_COUNTER_RESET	Interrupt caused by counter reset
QuadDec_INVALID_IN	Interrupt caused by invalid A, B inputs state transition

Side Effects: None

void QuadDec_Sleep(void)

Description: This is the preferred routine to prepare the component for sleep. The QuadDec_Sleep() routine saves the current component state. Then it calls the QuadDec_Stop() function and calls QuadDec_SaveConfig() to save the hardware configuration.

Call the QuadDec_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: None

void QuadDec_Wakeup(void)

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

Parameters: None

Return Value: None

Side Effects: Calling the QuadDec_Wakeup() function without first calling the QuadDec_Sleep() or QuadDec_SaveConfig() function may produce unexpected behavior.

void QuadDec_Init(void)

Description:	Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call QuadDec_Init() because the QuadDec_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 QuadDec_Enable(void)

Description:	Activates the hardware and begins component operation. It is not necessary to call QuadDec_Enable() because the QuadDec_Start() routine calls this function, which is the preferred method to begin component operation.
Parameters:	None
Return Value:	None
Side Effects:	None

void QuadDec_SaveConfig(void)

Description:	This function saves the component configuration and nonretention registers. This function 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 QuadDec_Sleep() function.
Parameters:	None
Return Value:	None
Side Effects:	None

void QuadDec_RestoreConfig(void)

Description:	This function restores the component configuration and nonretention registers. This function also restores the component parameter values to what they were before calling the QuadDec_Sleep() function.
Parameters:	None
Return Value:	None
Side Effects:	Calling this function without first calling the QuadDec_Sleep() or QuadDec_SaveConfig() function may produce unexpected behavior.

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.

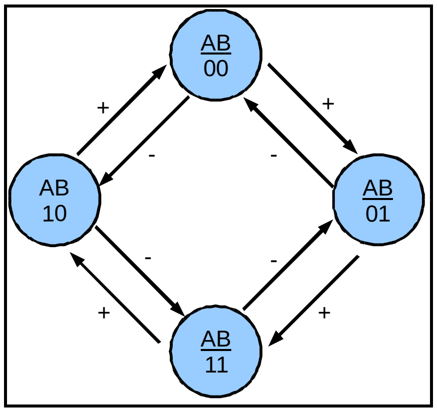
Functional Description

Default Configuration

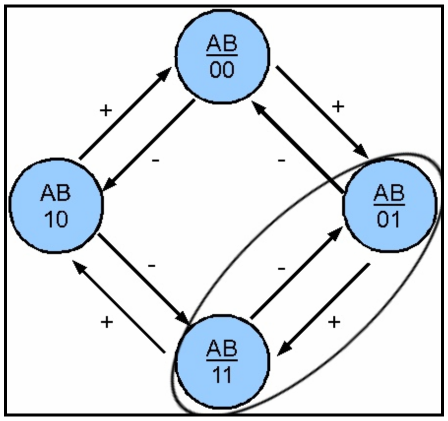
The default configuration for the Quadrature Decoder is an 8-bit up and down counter with 1x resolution, enabled index input, and enabled glitch filtering.

State Transition

Quadrature phase signals are typically decoded with a state machine and an up/down counter. A conventional decoder has four states, corresponding to all possible values of the A and B inputs. The state transition diagram is shown below (same-state transitions are not depicted). State transitions marked with a “+” and “–” indicate increment and decrement operations on the quadrature phase counter.



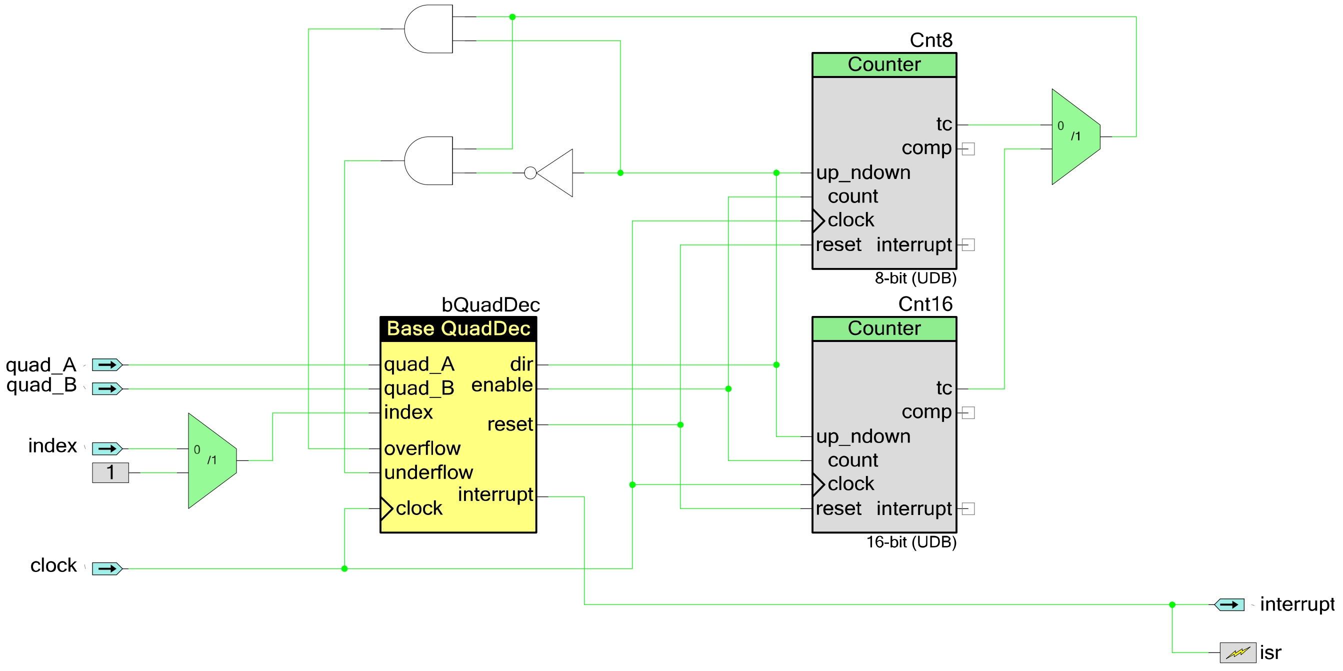
For each full cycle of the quadrature phase signal, the quadrature phase counter changes by four counts. Lower-resolution counters can also be used by implementing up/down operations on only a subset of the state transitions. A quarter-resolution decoder is shown below.



All inputs are sampled using a clock signal derived internally within the device.

Block Diagram and Configuration

The Quadrature Decoder is only available as a UDB configuration of blocks. The APIs are described earlier in this document and the registers are described in the next section to define the overall implementation of the component.



Registers

Status

Bits	7	6	5	4	3	2	1	0
Value	reserved				invalid in	reset	underflow	overflow

The status register is read-only. It contains the various status bits defined for the Quadrature Decoder. The value of this register is available with the QuadDec_GetEvents() function. The interrupt output signal is generated from an ORing of the masked bit fields within the status register.

You can set the mask using the QuadDec_SetInterruptMask() function. After you receive an interrupt you can retrieve the interrupt source by reading the status register with the QuadDec_GetEvents() function. The status register is transparent, so the QuadDec_GetEvents() function does not clear the bits of the status register. All operations on the status register must use the following defines for the bit fields, because these bit fields may be moved within the status register at build time.

There are several bit field masks defined for the status registers. Any of these bit fields may be included as an interrupt source. All bit fields are configured as sticky bits in the status register. Defines are available in the generated header (.h) file as follows:

- **QuadDec_COUNTER_OVERFLOW** – Defined as the bit mask of the Status register bit "counter overflow."
- **QuadDec_COUNTER_UNDERFLOW** – Defined as the bit mask of the Status register bit "Counter underflow."
- **QuadDec_RESET** – Defined as the bit-mask of the Status register bit "reset due index."
- **QuadDec_INVALID_IN** – Defined as the bit- mask of the Status register bit "invalid state transition on the A and B inputs."

DC and AC Electrical Characteristics

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	Config 1 [1]			34	MHz
		Config 2 [2]			29	MHz
		Config 3 [3]			16	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		$1/f_{CLOCK}$
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_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

[1] Config 1 options:

CounterResolution:	1
CounterSize:	8
UsingGlitchFiltering:	false
UsingIndexInput:	true

[2] Config 2 options:

CounterResolution:	2
CounterSize:	16
UsingGlitchFiltering:	true
UsingIndexInput :	true

[3] Config 3 options:

CounterResolution:	4
CounterSize:	32
UsingGlitchFiltering:	true
UsingIndexInput:	true

[4] $t_{CY\_clock} = 1/f_{CLOCK}$ - Cycle time of one clock period

[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 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.

Parameter	Description	Config.	Min	Typ	Max	Units
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_{IH}	Input high time	1,2	t_{CY_clock}			ns
t_{IL}	Input low time	1,2	t_{CY_clock}			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
f_{AB}	Component A and B Frequency	N/A			$f_{CLOCK}/10$	MHz
t_{IND}	Index signal width	Without glitch filtering	$2 \times t_{CY\_clock} + 5$			ns
		With glitch filtering	$3 \times t_{CY\_clock} + 5$			ns
t_{RD}	Index input low to reset time	Without glitch filtering		2		t_{CY_clock}
		With glitch filtering		5		t_{CY_clock}
t_{GL}	Time during which glitching is expected to occur	With glitch filtering			3	t_{CY_clock}
t_{CD}	Delay time, rising edge of clock to count valid	N/A		2		t_{CY_clock}
t_{E}	Encoder pulse width (low or high)	N/A	4			t_{CY_clock}
t_{ES}	Encoder state period	N/A	2			t_{CY_clock}
t_{ELP}	Encoder period width	N/A	10			t_{CY_clock}

Timing Characteristics "Maximum with All Routing"

Parameter	Description	Config.	Min	Typ	Max [1]	Units
f_{CLOCK}	Component clock frequency	Config 1 [2]			17	MHz
		Config 2 [3]			14	MHz
		Config 3 [4]			8	MHz
t_{CLOCKH}	Input clock high time [5]	N/A		0.5		$1/f_{CLOCK}$
t_{CLOCKL}	Input clock low time [5]	N/A		0.5		$1/f_{CLOCK}$
Inputs						
t_{PD_ps}	Input path delay, pin to sync [6]	1			STA [7]	ns
t_{PD_ps}	Input path delay, pin to sync [8]	2			8.5	ns
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_si}	Sync output to input path delay (route)	1,2,3,4			STA [7]	ns
t_{I_clk}	Alignment of clockX and clock	1,2,3,4	0		1	t_{CY_clock}

[1] Maximum for "All Routing" is calculated by <nominal>/2 rounded to the nearest integer. This value provides a basis for you to not have to worry about meeting timing if the component is running at or below this component frequency.

[2] Config 1 options:

CounterResolution:	1
CounterSize:	8
UsingGlitchFiltering:	false
UsingIndexInput:	true

[3] Config 2 options:

CounterResolution:	2
CounterSize:	16
UsingGlitchFiltering:	true
UsingIndexInput :	true

[4] Config 3 options:

CounterResolution:	4
CounterSize:	32
UsingGlitchFiltering:	true
UsingIndexInput:	true

[5] $t_{CY\_clock} = 1/f_{CLOCK}$ - Cycle time of one clock period

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

[7] 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.

[8] 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.	Min	Typ	Max [1]	Units
t_{IH}	Input high time	1,2	t_{CY_clock}			ns
t_{IL}	Input low time	1,2	t_{CY_clock}			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
f_{AB}	Component A and B frequency	N/A			$f_{CLOCK}/10$	MHz
t_{IND}	Index signal width	Without glitch filtering	$2 \times t_{CY\_clock} + 5$			ns
		With glitch filtering	$3 \times t_{CY\_clock} + 5$			ns
t_{RD}	Index input low to reset time	Without glitch filtering		2		t_{CY_clock}
		With glitch filtering		5		t_{CY_clock}
t_{GL}	Time during which glitching is expected to occur	With glitch filtering			3	t_{CY_clock}
t_{CD}	Delay time, rising edge of clock to count valid	N/A		2		t_{CY_clock}
t_E	Encoder pulse width (low or high)	N/A	4			t_{CY_clock}
t_{ES}	Encoder state period	N/A	2			t_{CY_clock}
t_{ELP}	Encoder period width	N/A	10			t_{CY_clock}

Figure 1. Timing Diagram Without Using Glitch Filtering

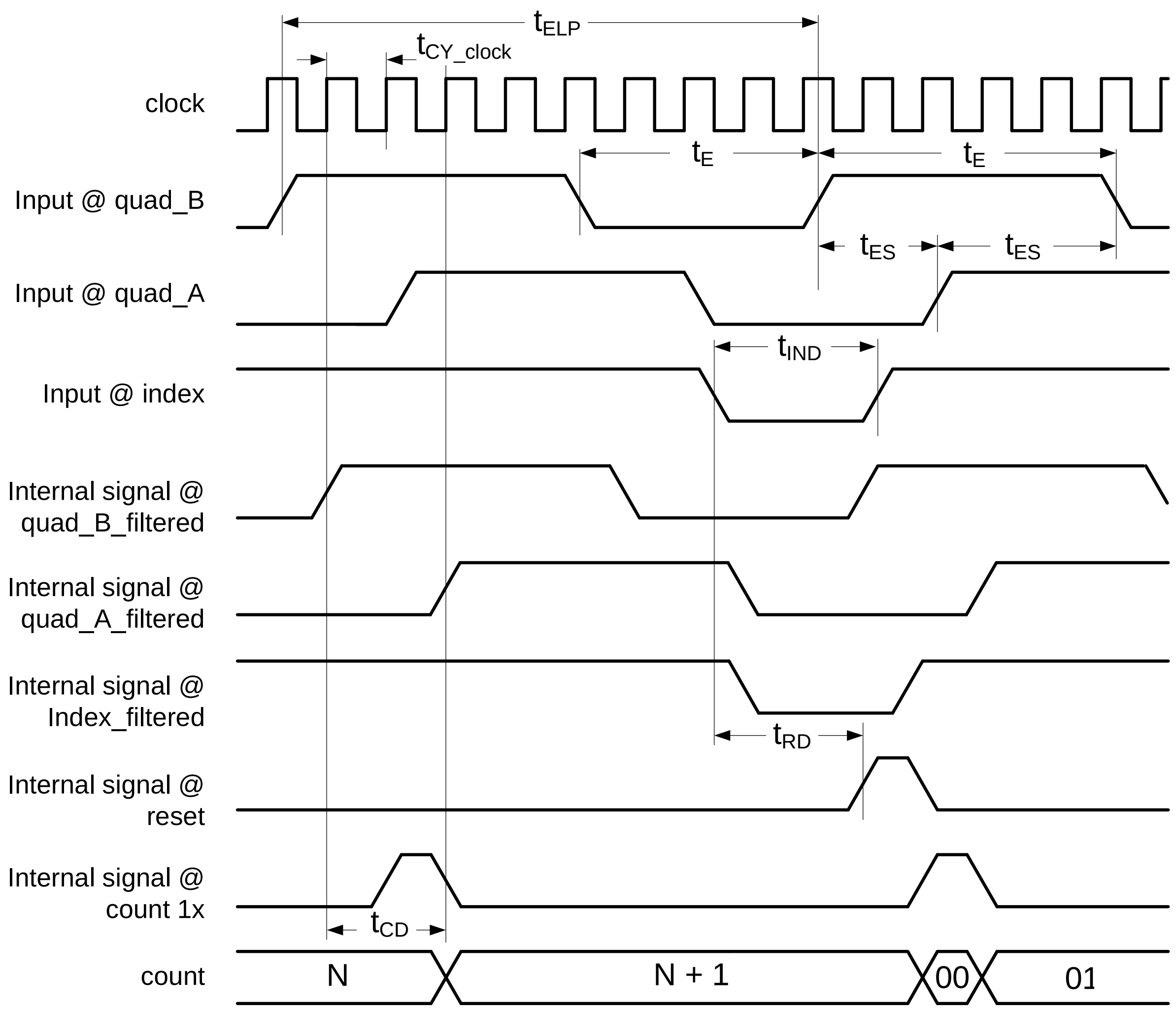
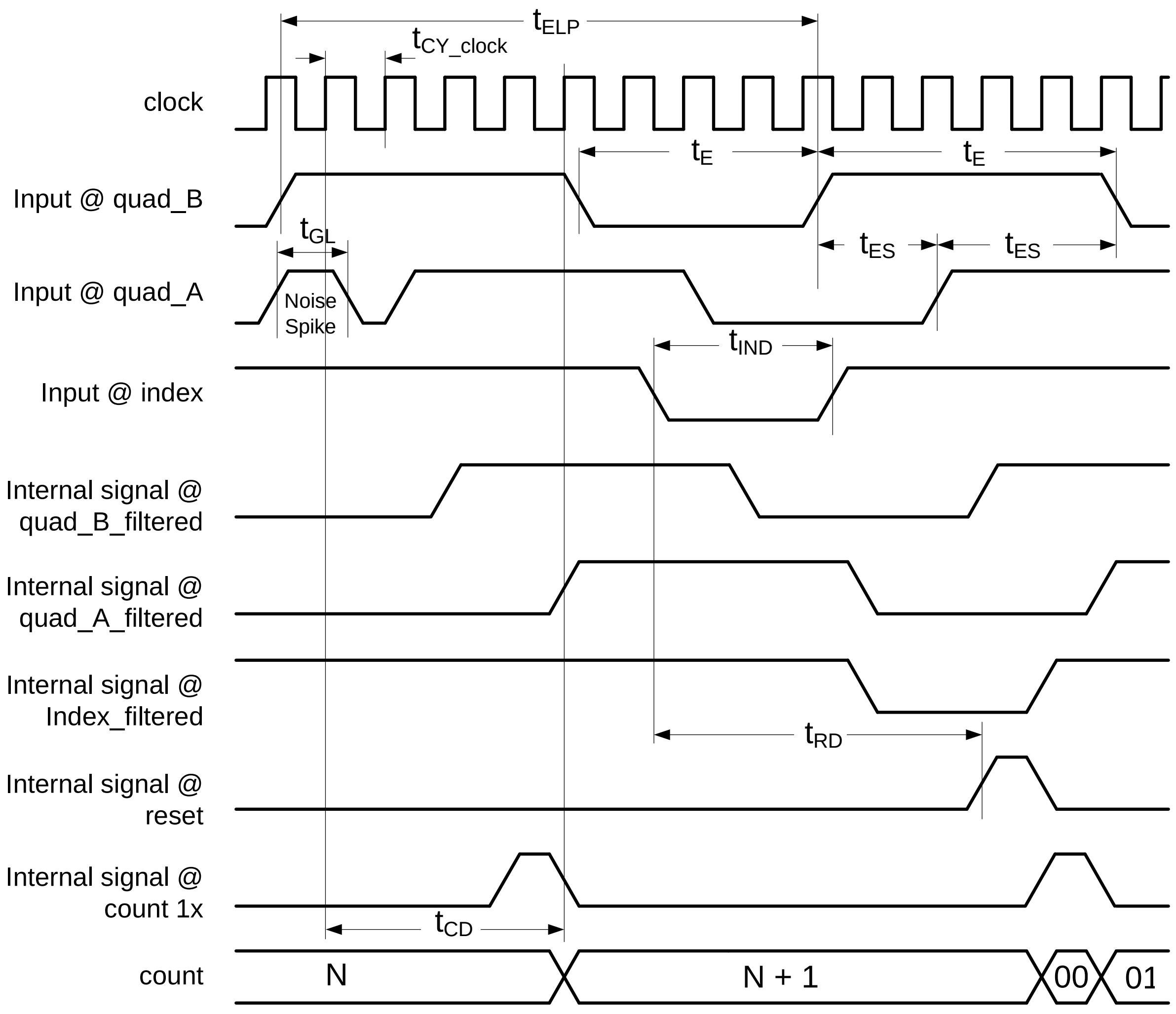


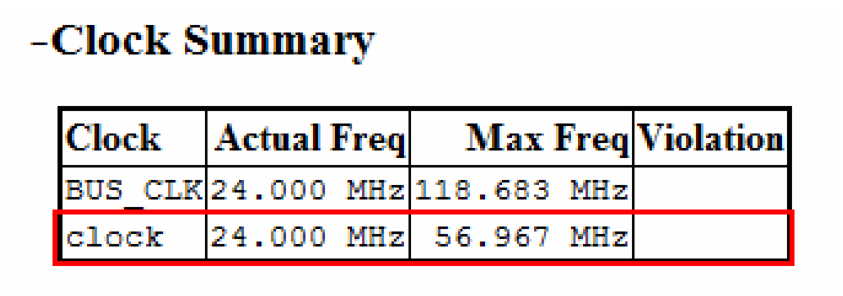
Figure 2. Timing Diagram Using Glitch Filtering



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* file:



-Clock Summary

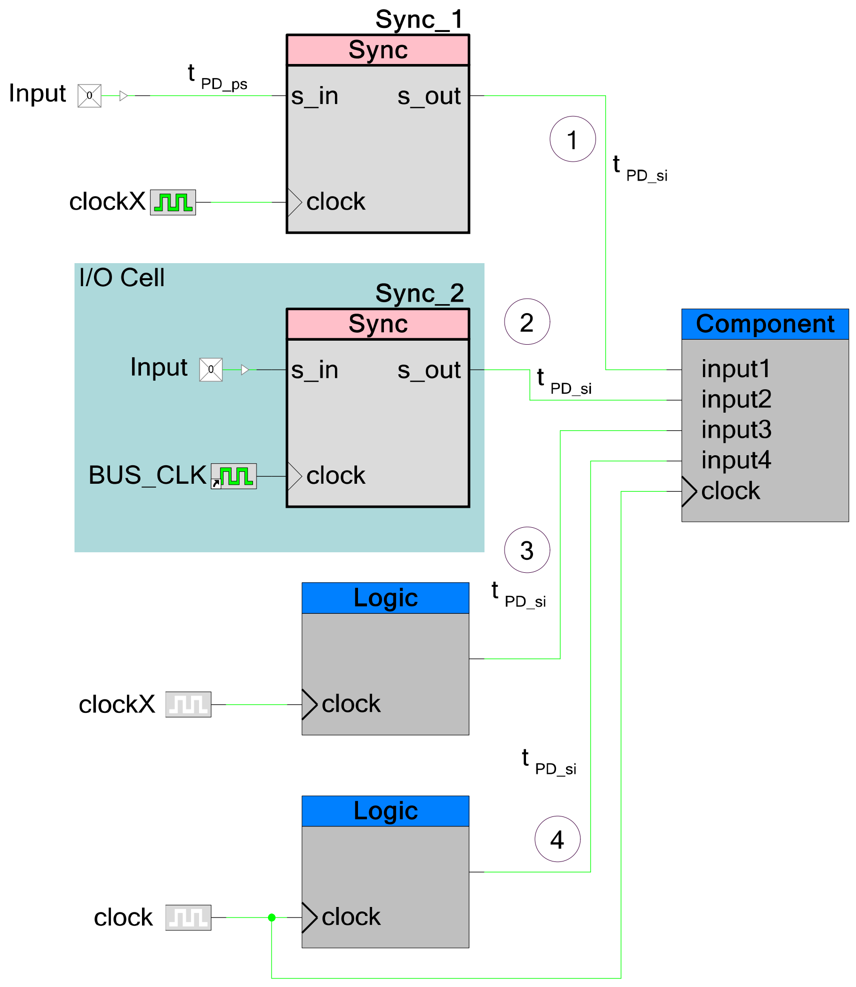
Clock	Actual Freq	Max Freq	Violation
BUS_CLK	24.000 MHz	118.683 MHz	
clock	24.000 MHz	56.967 MHz	

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 the four possible configurations shown in Figure 3.

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 3. Input Configurations for Component Timing Specifications



Configuration	Component Clock	Synchronizer Clock (Frequency)	Figures
1	master_clock	master_clock	Figure 8
1	clock	master_clock	Figure 6
1	clock	clockX = clock [1]	Figure 4
1	clock	clockX > clock	Figure 5
1	clock	clockX < clock	Figure 7
2	master_clock	master_clock	Figure 8
2	clock	master_clock	Figure 6
3	master_clock	master_clock	Figure 13

[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 11
3	clock	clockX = clock [1]	Figure 9
3	clock	clockX > clock	Figure 10
3	clock	clockX < clock	Figure 12
4	master_clock	master_clock	Figure 13
4	clock	clock	Figure 9

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 4, Figure 5, Figure 7, and Figure 8.

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). This produces the characterization parameters shown in Figure 5 and Figure 8.

Figure 4. Input Configuration 1 and 2; Sync Clock Freq.= Component Clock Frequency (Edge alignment of clock and clockX is not guaranteed)

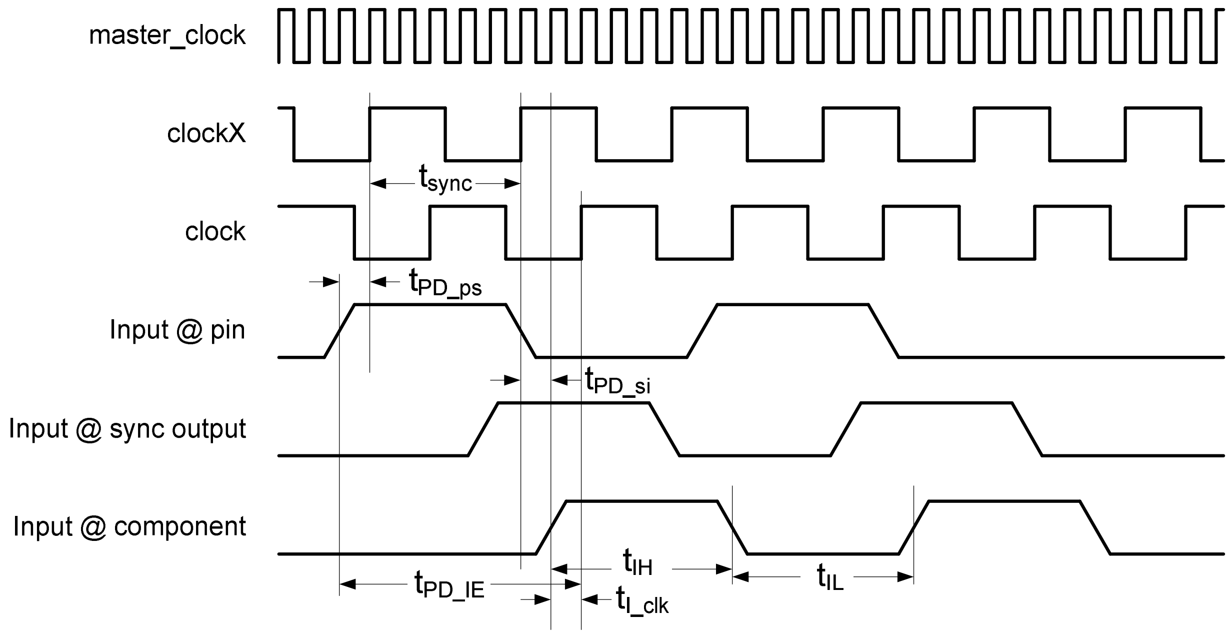


Figure 5. Input Configuration 1 and 2; Sync. Clock Freq. > Component Clock Frequency

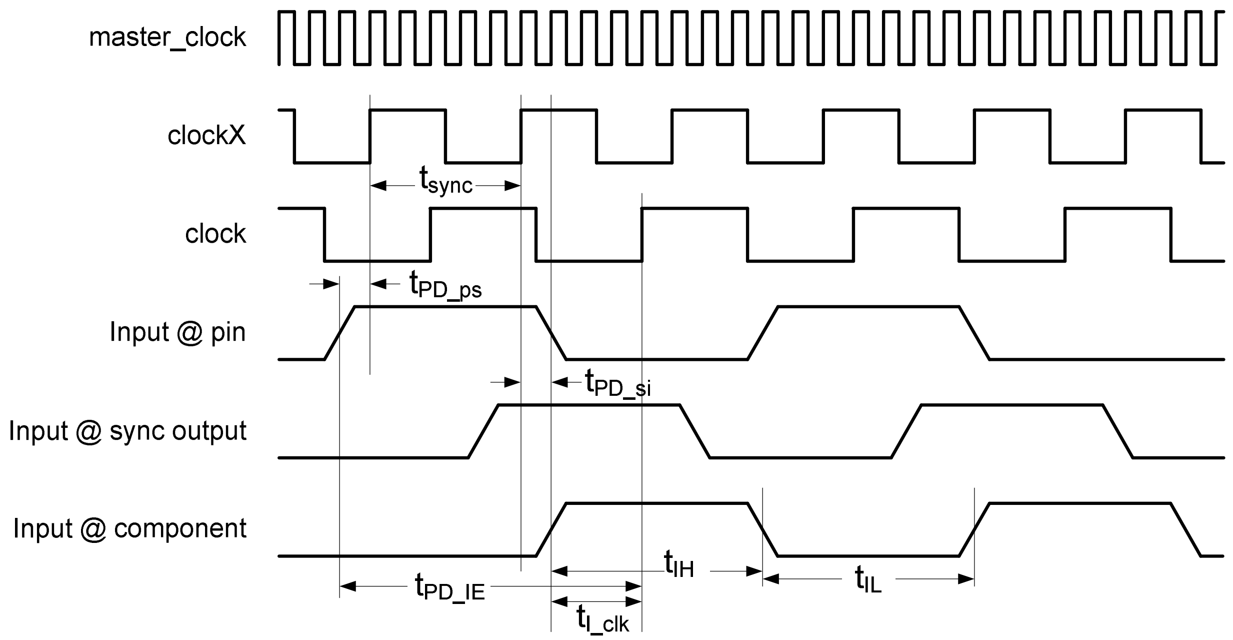


Figure 6. Input Configuration 1 and 2; [Sync. Clock Freq. == master_clock] > Component Clock Frequency

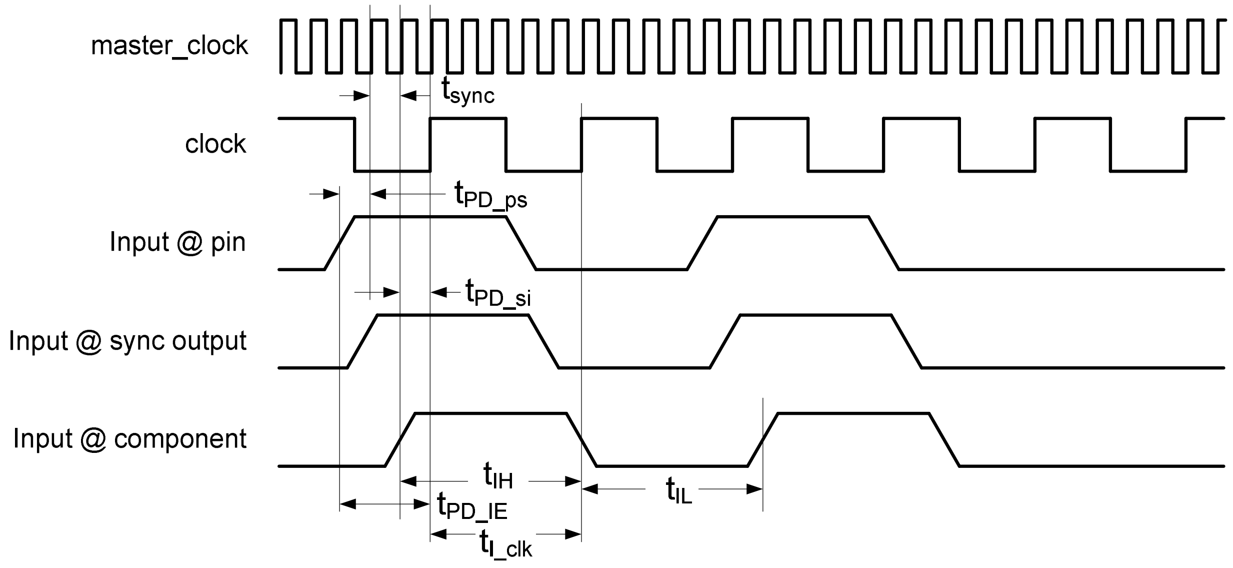


Figure 7. Input Configuration 1; Sync. Clock Freq. < Component Clock Frequency

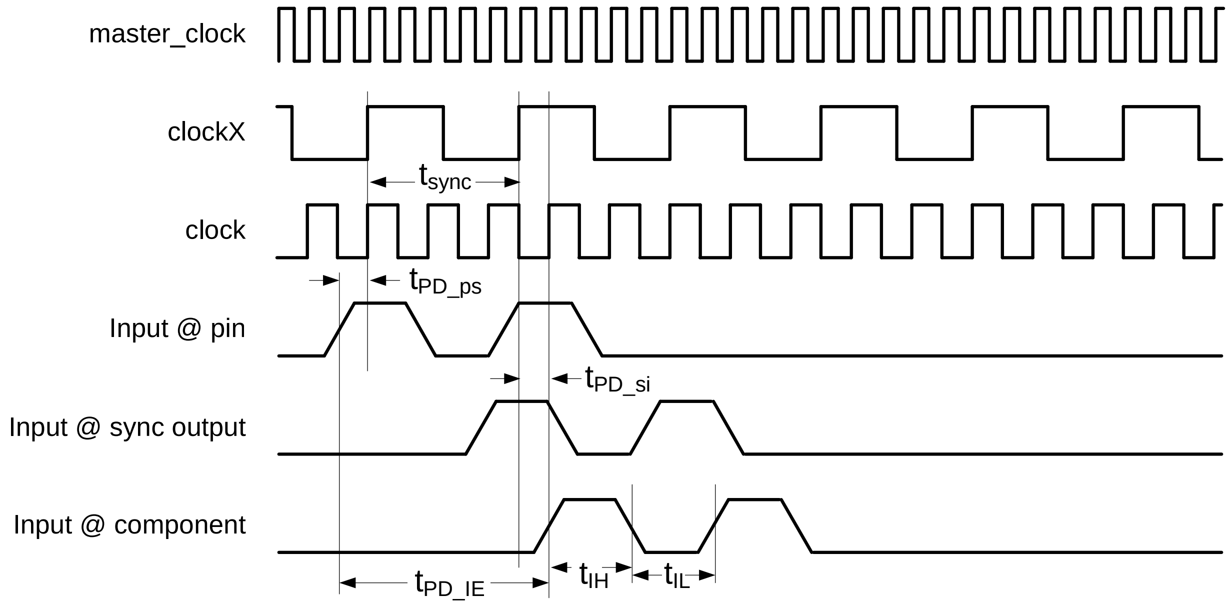
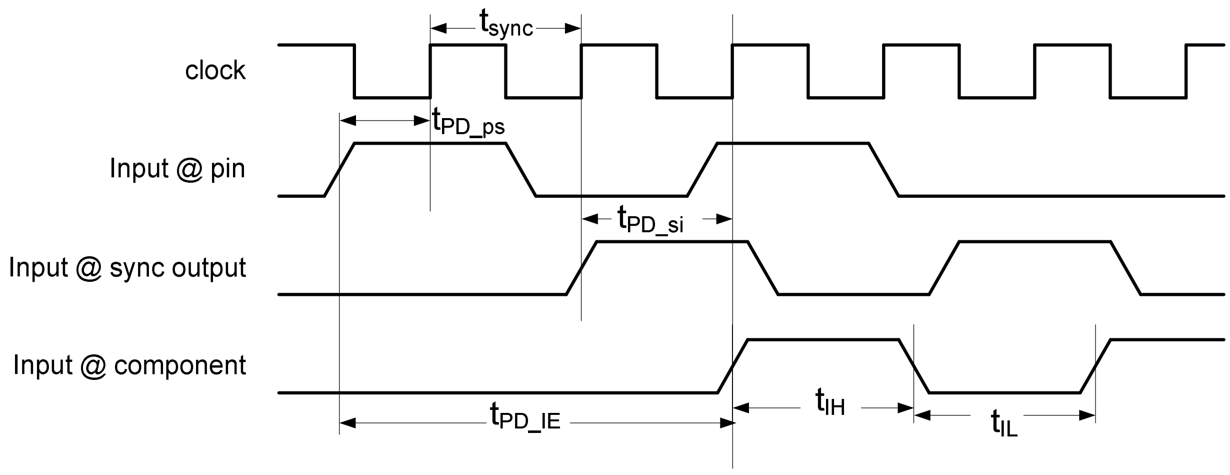


Figure 8. Input Configuration 1 and 2; Sync. Clock = Component Clock = master_clock



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, which produces the characterization parameters shown in Figure 9, Figure 10, and Figure 12.

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 13.

Figure 9. Input Configuration 3 only; Sync. Clock Freq. = Component Clock Frequency (Edge alignment of clock and clockX is not guaranteed)

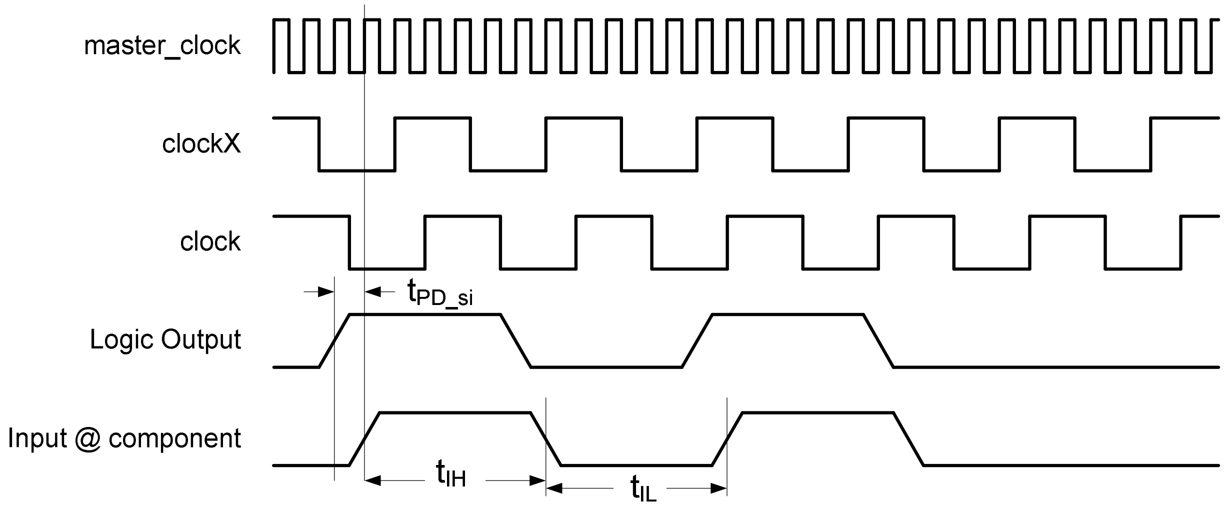
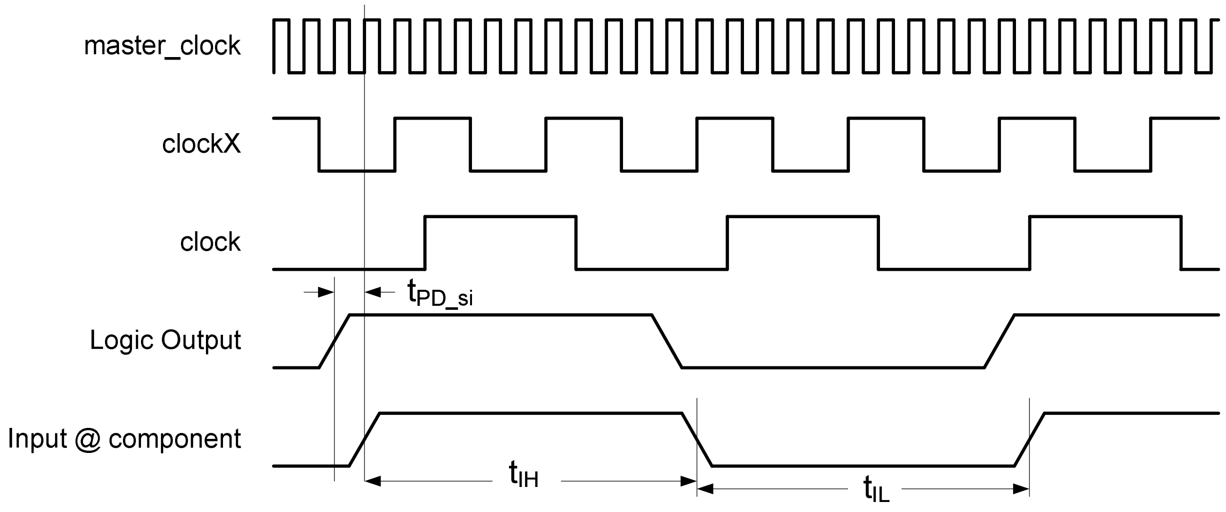


Figure 9 represents the extent to which Static Timing Analysis understands the clocks. All clocks in the digital clock domain are synchronous to master_clock. However, it is possible that two clocks with the same frequency are not rising-edge aligned. Therefore, the static timing analysis tool does not 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 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 10. Input Configuration 3; Sync. Clock Freq. > Component Clock Frequency



In much the same way as shown in Figure 9, 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 11. Input Configuration 3; Synchronizer Clock Frequency = master_clock > Component Clock Frequency

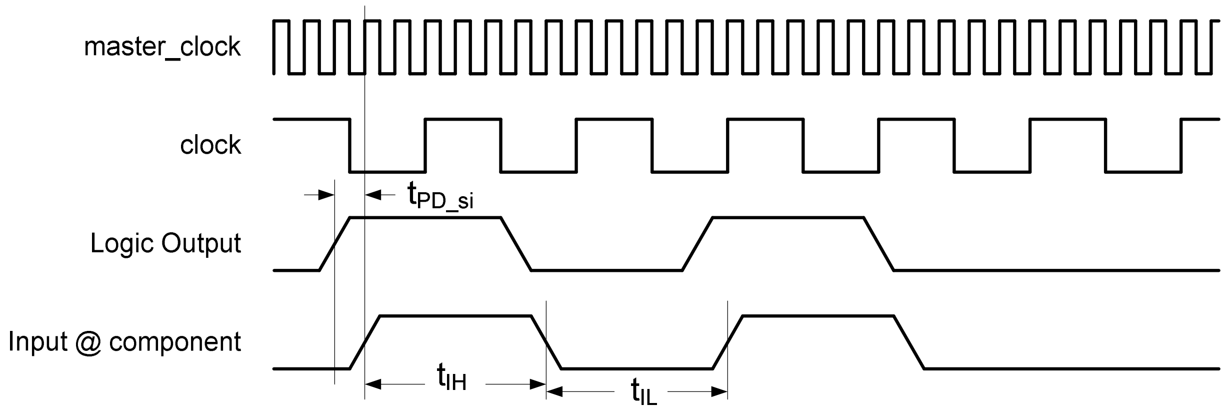
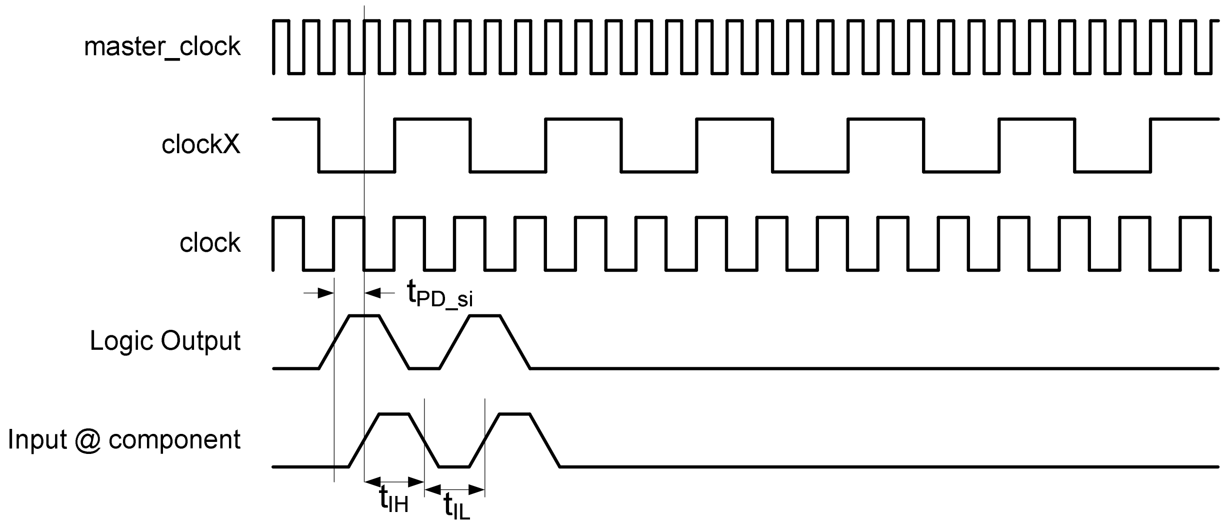
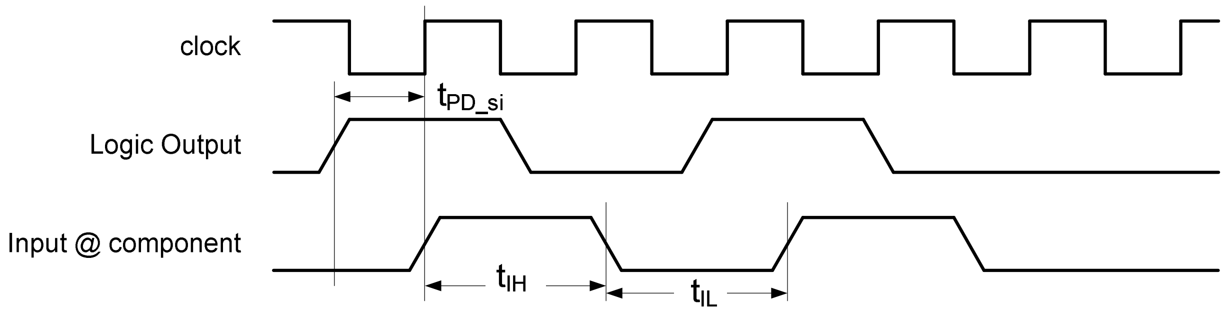


Figure 12. Input Configuration 3; Synchronizer Clock Frequency < Component Clock Frequency



In much the same way as shown in Figure 9, 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.

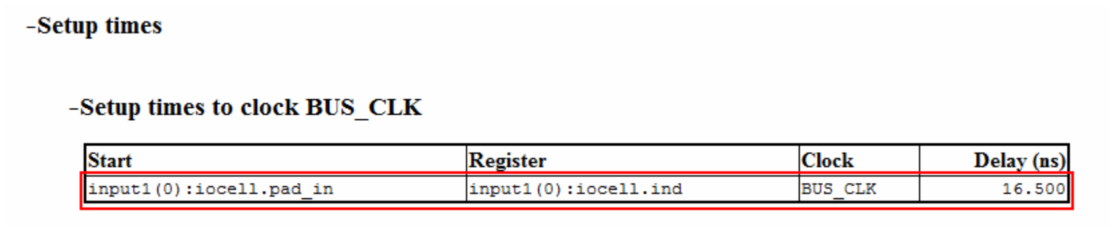
Figure 13. Input Configuration 4 only; Synchronizer Clock = Component Clock



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}. It is crucial 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.

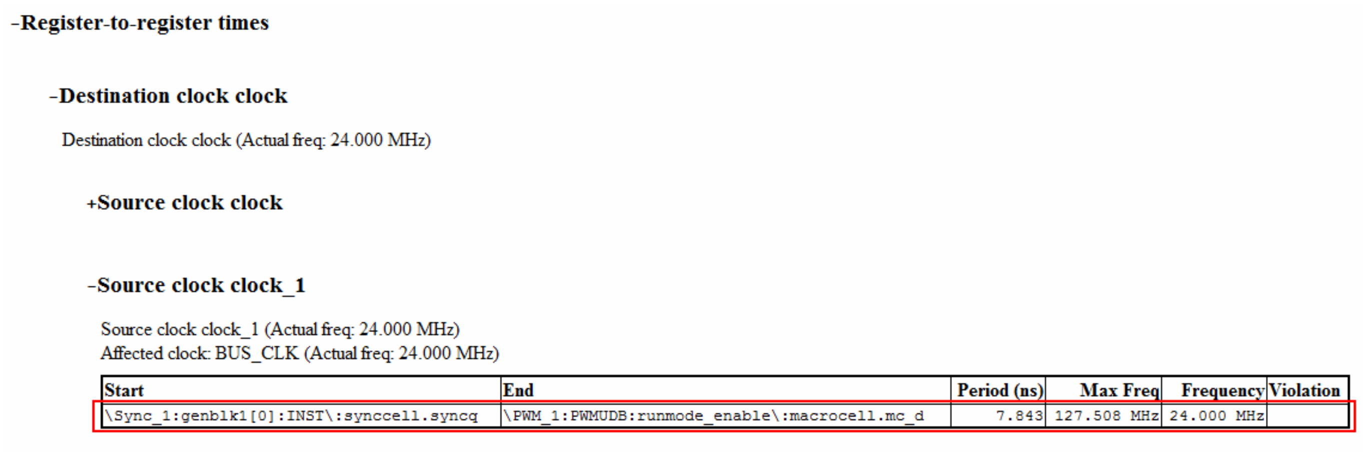


-**Setup times**

-**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 will 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 given previously (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	Updated block diagram of Quadrature Decoder in the Block Diagram and Configuration section of the datasheet.	For use with the latest version of the Counter component.
	Updated internal Counter component to version 2.0 on Quadrature Decoder Component schematic.	For use with the latest version of the Counter component.
	Removed obsolete defines.	
1.50.a	Added characterization data to datasheet	
	Minor datasheet edits and updates	
1.50	Changed QuadDec_Start() API: removed write to Control Register.	Beta5 STA-Based Optimization.
	Added QuadDec_Sleep()/ QuadDec_Wakeup() APIs.	Added APIs to support the low power modes.
	Added QuadDec_Init() API.	Added to provide an API to initialize/restore the component without starting it.
1.20	Updated the Configure dialog. Removed the *QuadDec_INT.c* file after compilation if the counter size is less than 32. Removed the checking condition in the *QuadDec_INT.c* file for counter size = 32 bit.	