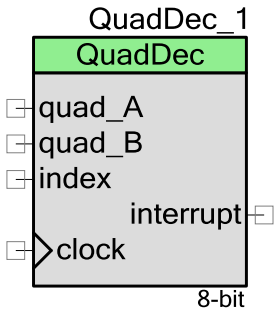
Quadrature Decoder (QuadDec)

2.10

Features

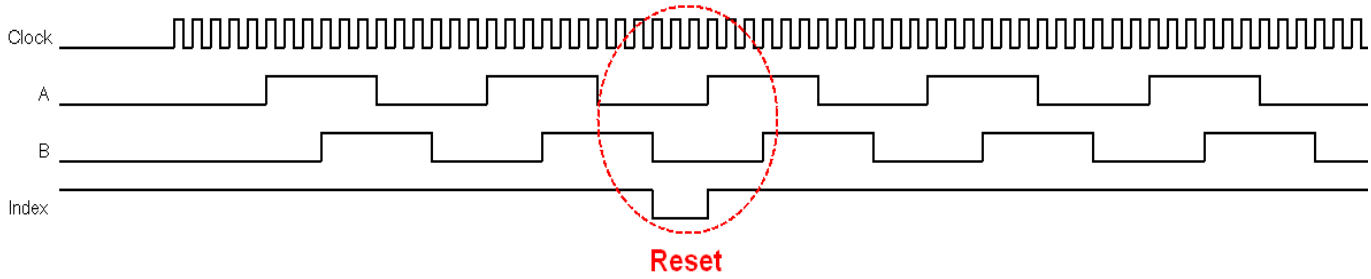


- Adjustable counter size: 8, 16, or 32 bits
- Counter resolution of 1x, 2x, or 4x the frequency of the A and B inputs, for more accurate determination of position or speed
- Optional index input to determine absolute position
- Optional glitch filtering to reduce the impact of system-generated noise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

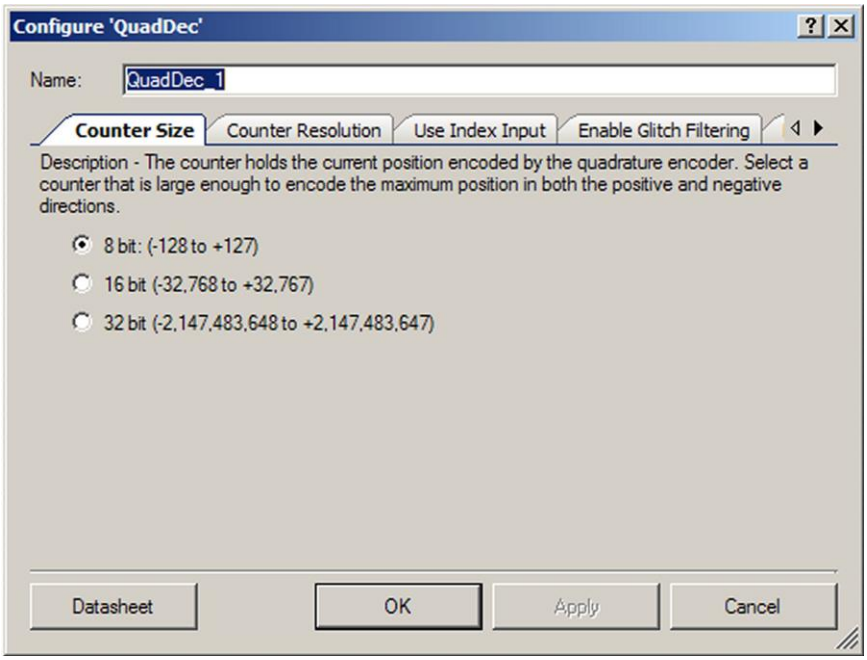
Interrupt on one or more of the following events:

- 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

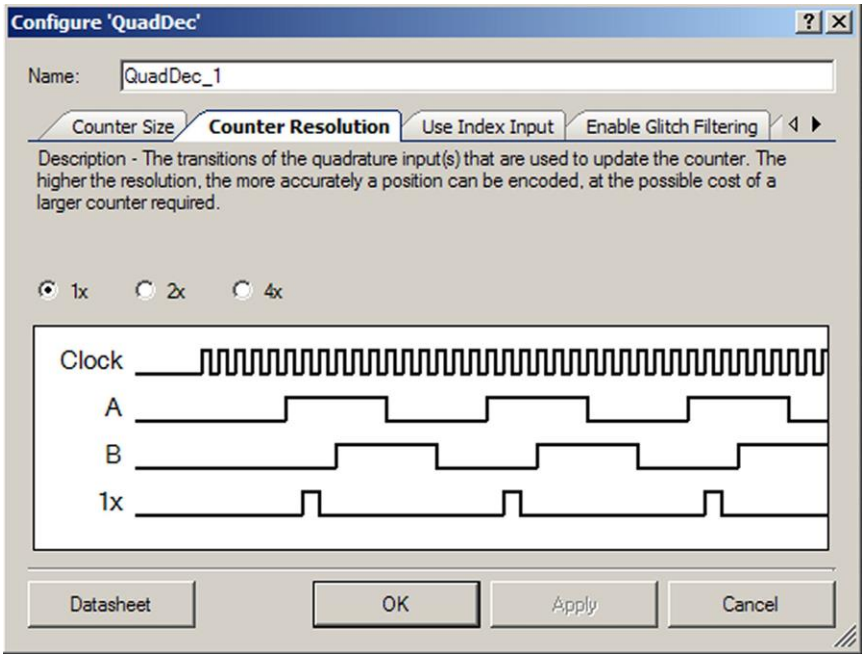


This tab is used to define the counter size, in bits. The counter holds the current position encoded by a quadrature encoder.

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**.

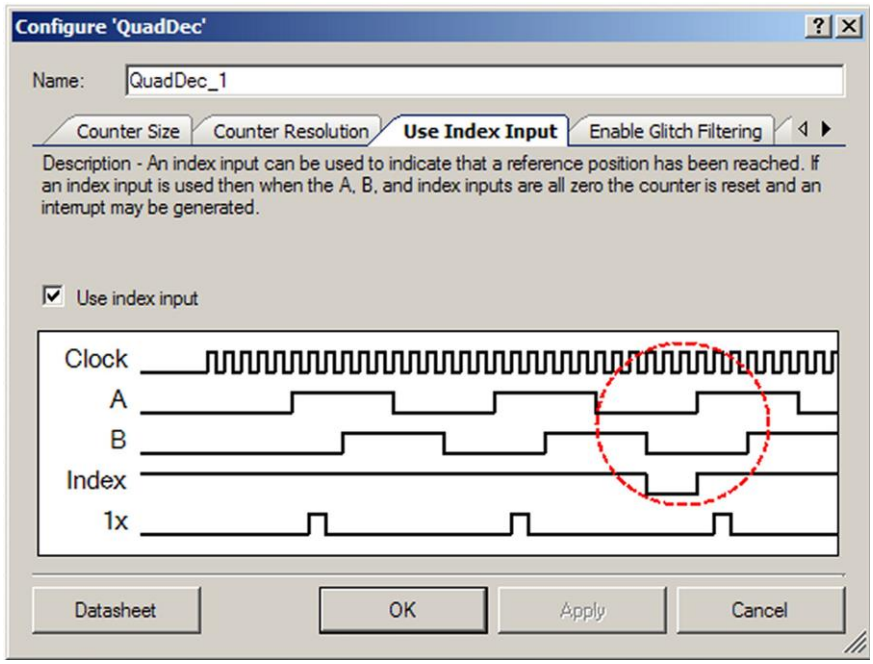
The 32-bit counter implements the lower 16 bits in the hardware counter and the upper 16 bits in software to reduce hardware resource use. For this target, an additional ISR is used. To work properly with the 32-bit counter, interrupts must be enabled. You can add ISR code to source files as needed;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

Counter Resolution Tab



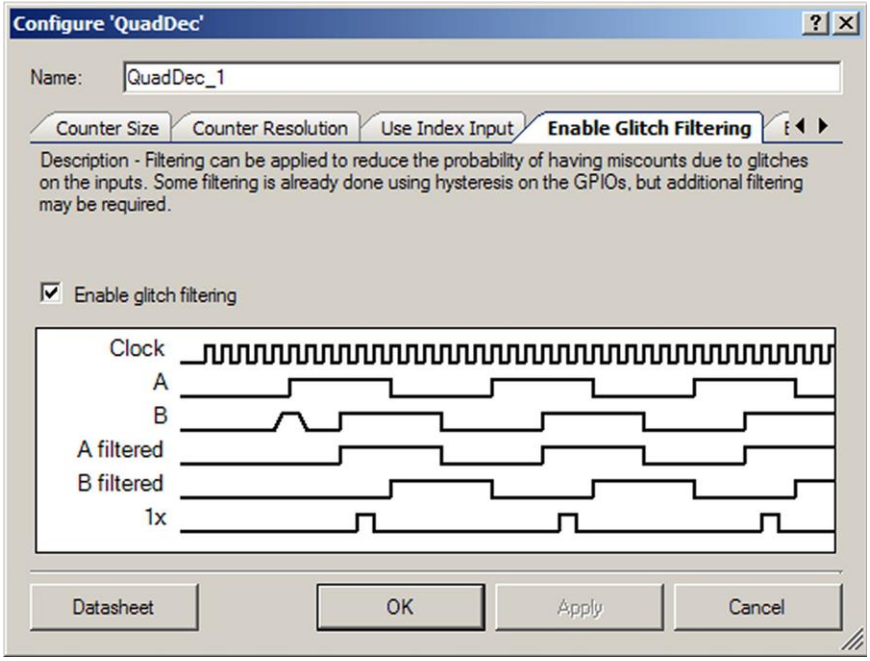
This tab contains the number of counts recorded in one period of the A and B inputs. It shows the transitions of the input signals that are used to update the counter. As the resolution gets higher, the position can be resolved more accurately, at the possible cost of a larger counter. The setting can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

Use Index Input Tab



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



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

There is no internal clock in this component. You must attach a clock source. This component operates from a single clock connected to the component.

Depending on the frequency of BUS_CLK, the current component implementation may cause Setup time violations when the target device is a PSoC 5. It is recommended to limit the BUS_CLK frequency to 36 MHz for PSoC 5 projects.

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.

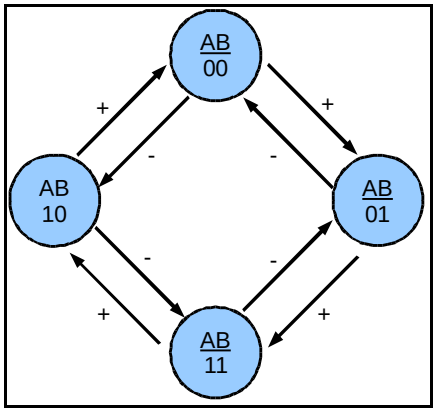
Functional Description

Default Configuration

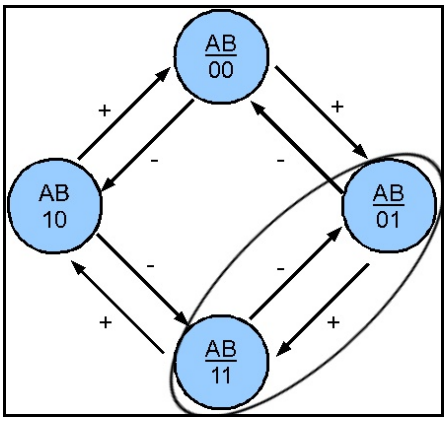
The default configuration for the Quadrature Decoder is an 8-bit up and down counter with 1x resolution, enabled index input, and enabled glitch filtering.

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.



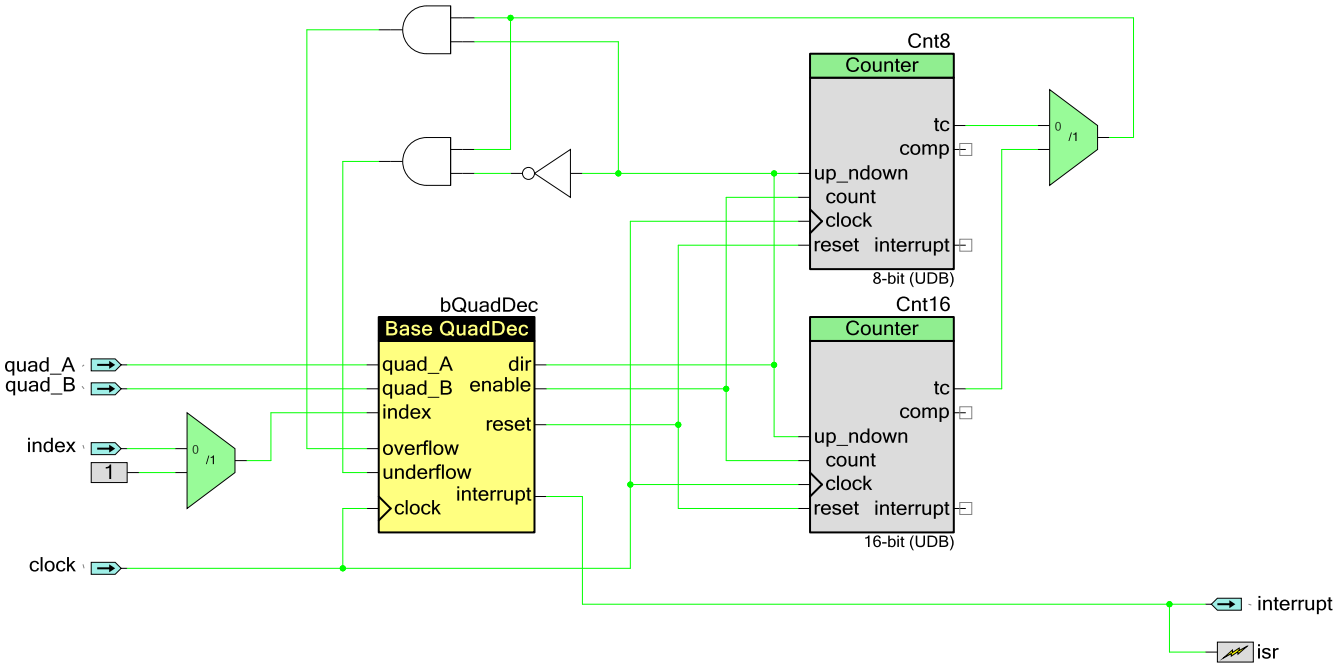
For each full cycle of the quadrature phase signal, the quadrature phase counter changes by four counts. Lower-resolution counters can also be used by implementing up/down operations on only a subset of the state transitions. A quarter-resolution decoder is shown below.



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.”

Resources

The Quadrature Decoder component is placed throughout the UDB array. The component utilizes the following resources.

Configuration	Resource Type					
	Datapath Cells	Macrocells	Status Cells	Control Cells	DMA Channels	Interrupts
8-bit, resolution 1x, no glitch filtering, use index	1	22	2	1	–	–
16-bit, resolution 2x, glitch filtering, use index	2	31	2	1	–	–
32-bit, resolution 4x, glitch filtering, use index	2	32	2	1	–	1

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.
The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration	PSoC 3 (Keil_PK51)		PSoC 5 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
8-bit, resolution 1x, no glitch filtering, use index	391	7	656	14	588	10
16-bit; resolution 2x, glitch filtering, use index	437	8	662	18	594	14
32-bit; resolution 4x, glitch filtering, use index	701	13	824	26	756	22

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

DC Characteristics

Parameter	Description	Min	Typ[1]	Max	Units
I_{DD}	Component current consumption				
	8-bit, resolution 1x, no glitch filtering, use index	–	15	–	µA/MHz
	16-bit, resolution 2x, glitch filtering, use index	–	20	–	µA/MHz
	32-bit, resolution 4x, glitch filtering, use index	–	26	–	µA/MHz

AC Characteristics

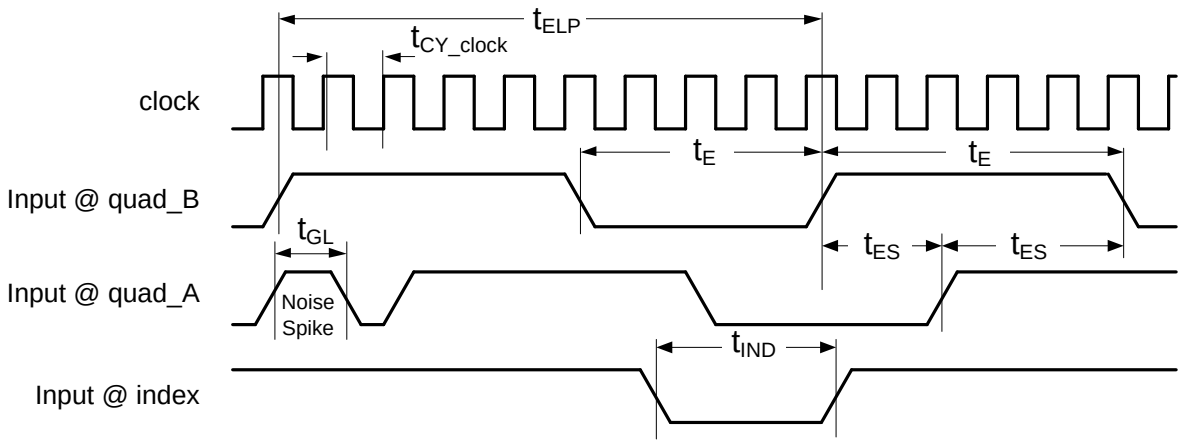
Parameter	Description		Min	Typ	Max[2]	Units
f_{CLOCK}	Component clock frequency					
	8-bit, resolution 1x, no glitch filtering, use index		–	–	33	MHz
	16-bit, resolution 2x, glitch filtering, use index		–	–	29	MHz
	32-bit, resolution 4x, glitch filtering, use index		–	–	28	MHz
f_{AB}	Component A and B Frequency		–	–	f_{CLOCK}/10	MHz
t_{IND}	Index signal width	no glitch filtering	2	–	–	t_{CY_clock}[3]
		glitch filtering	3			
t_{GL}	Time during which glitching is expected to occur		–	–	3	t_{CY_clock}
t_E	Encoder pulse width (low or high)		4	–	–	t_{CY_clock}
t_{ES}	Encoder state period		2	–	–	t_{CY_clock}
t_{ELP}	Encoder period width		10	–	–	t_{CY_clock}

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

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

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

Figure 1. Timing Diagram



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.a	Minor datasheet edit.	
2.10	Added PSoC 5LP device support.	
	Added all Quadrature Decoder APIs with CYREENTRANT keyword when they included in .cyre file.	Not all APIs are truly reentrant. Comments in the component API source files indicate which functions are candidates. This change is required to eliminate compiler warnings for functions that are not reentrant used in a safe way: protected from concurrent calls by flags or Critical Sections.
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.	

© Cypress Semiconductor Corporation, 2012-2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks and of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.