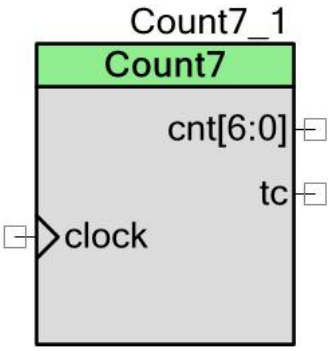
Down Counter 7-bit (Count7)

1.0



Features

- 7-bit read/write period register
- 7-bit count register that is read/write
- Automatic reload of the period to the count register on terminal count
- Routed load and enable signals

General Description

The Count7 Component is a 7-bit down counter with the count value available as hardware signals. This counter is implemented using a specific configuration of a universal digital block (UDB). To implement the counter, pieces of the control and status registers are used along with counter logic that is present in the UDB specifically for this function.

When to Use a Count7

There are multiple Components that implement counter functionality. The Count7 Component provides the most resource efficient implementation of a 7-bit counter that also exposes the count value directly as hardware signals. That makes the Count7 particularly useful for the hardware implementation of state machine functionality where a counter with a period of 128 or less is required.

Input/Output Connections

This section describes the various input and output connections for the Count7. 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.

en – Input*

Synchronous active high enable signal. Present if **Enabled** is selected for the **EnableSignal** parameter. If it is visible it requires a connection. A high signal state enables the decrement operation of the counter at the rising edge of the clock.

load – Input*

Synchronous active high load signal. Present if **Enabled** is selected for the **LoadSignal** parameter. If it is visible it requires a connection. A high signal state loads the period value into the counter if the counter is enabled. The load signal has no effect if the counter is disabled.

clock – Input

Component clock. The Component can support clock sources of up to 67 MHz.

reset – Input*

Asynchronous active high reset signal. Present if **Enabled** is selected for the **ResetSignal** parameter. If it is visible it requires a connection. A high signal state clears the counter to a value of zero immediately. When the reset signal is returned to a low state and if the Component is enabled, on the next positive edge of the clock the counter will be loaded with the period value.

cnt[6:0] – Output

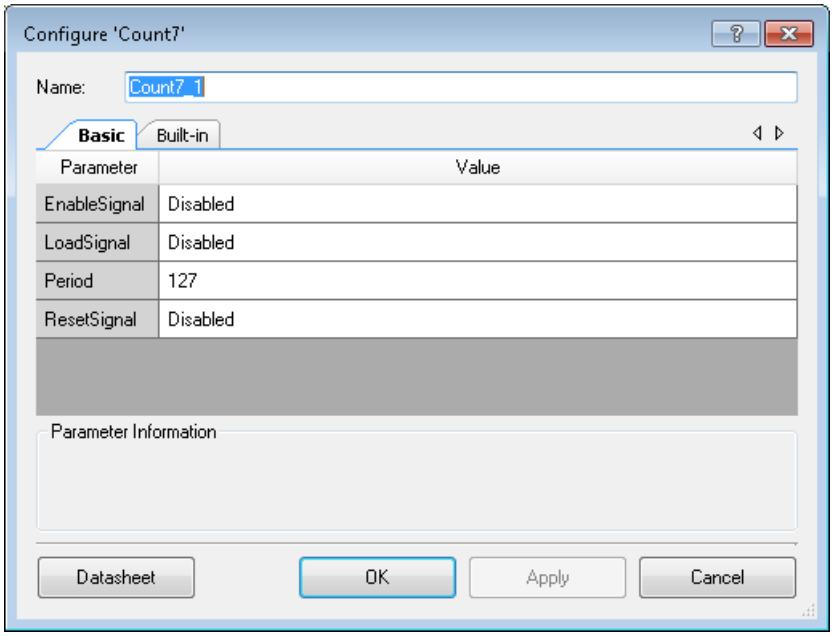
The 7-bit counter value.

tc – Output

The terminal count output is high when the counter is equal to zero. If the counter is enabled this output is high for a single cycle and the counter is reloaded with the period value during the next cycle. If the Component is stopped with the period counter equal to zero, then the terminal count signal remains high until the period is no longer zero. This output is registered by the clock signal which results in the terminal count transitioning with one clock cycle of latency from the counter value.

Component Parameters

Drag a Count7 onto your design and double click it to open the Configure dialog.



The Count7 provides the following parameters:

Basic Options

EnableSignal

Enables the connection of a hardware enable signal.

LoadSignal

Enables the connection of a hardware load signal.

Period

Defines the initial period register value. For a period of N clocks, the period value should be set to the value of N-1. The counter will count from N-1 down to 0 which results in an N clock cycle period. A period register value of 0 is not supported and will result in the terminal count output held at a constant high state.

ResetSignal

Enables the connection of an asynchronous reset signal.

Clock Selection

There is no internal clock in this Component. You must attach a clock source. The Component can operate with input clock frequencies up to 67 MHz.

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 discuss each function in more detail.

By default, PSoC Creator assigns the instance name “Count7_1” to the first instance of a Component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “Count7.”

Functions

Function	Description
Count7_Start()	Performs all of the required initialization for the Component and enables the counter.
Count7_Init()	Initializes or restores the Component according to the customizer settings.
Count7_Enable()	Enables the software enable of the counter.
Count7_Stop()	Disables the software enable of the counter.
Count7_WriteCounter()	This function writes the counter directly. The counter should be disabled before calling this function.
Count7_ReadCounter()	This function reads the counter value.
Count7_WritePeriod()	This function writes the period register.
Count7_ReadPeriod()	This function reads the period register.
Count7_Sleep()	This is the preferred API to prepare the Component for low power mode operation.
Count7_Wakeup()	This is the preferred API to restore the Component to the state when Count7_Sleep() was called.
Count7_SaveConfig()	This function saves the value of the Component’s count register prior to entering the low power mode.
Count7_RestoreConfig()	This function restores the value of Component’s count register which was previously stored.

void Count7_Start(void)

Description: Performs all of the required initialization for the Component and enables the counter. The first time the routine is executed, the period is set as configured in the customizer. When called to restart the counter following a Count7_Stop() call, the current period value is retained.

void Count7_Init(void)

Description: Initializes or restores the Component according to the customizer settings. It is not necessary to call Count7_Init() because the Count7_Start() API calls this function and is the preferred method to begin Component operation.

void Count7_Enable(void)

Description: Enables the software enable of the counter. The counter is controlled by a software enable and an optional hardware enable. It is not necessary to call Count7_Enable() because the Count7_Start() API calls this function, which is the preferred method to begin Component operation.

void Count7_Stop(void)

Description: Disables the software enable of the counter. This API halts the counter. Therefore, when you start it again (Count7_Start() call), it will continue counting from the last counter value.

void Count7_WriteCounter(uint8 count)

Description: This function writes the counter directly. The counter should be disabled before calling this function.

Parameters: uint8 count: Value to be written to counter.

The allowed values are in the range from 0 to 127.

uint8 Count7_ReadCounter(void)

Description: This function reads the counter value.

Return Value: uint8: Current counter value.

void Count7_WritePeriod(uint8 period)

Description: This function writes the period register. The actual period is one greater than the value in the period register since the counting sequence starts with the period register value and counts down to 0 inclusive. The period of the counter output does not change until the counter is reloaded following the terminal count value of 0 or due to a hardware load signal.

Parameters: uint8 period: Period value to be written.

The allowed values are in the range from 0 to 127.

uint8 Count7_ReadPeriod(void)

Description: This function reads the period register.

Return Value: uint8: Current period value.

void Count7_Sleep(void)

Description: This is the preferred API to prepare the Component for low power mode operation. The Count7_Sleep() API saves the current Component state using Count7_SaveConfig() and disables the counter.

void Count7_Wakeup(void)

Description: This is the preferred API to restore the Component to the state when Count7_Sleep() was called. The Count7_Wakeup() function calls the Count7_RestoreConfig() function to restore the configuration.

void Count7_SaveConfig(void)

Description: This function saves the current counter value prior to entering low power mode. This function is called by the Count7_Sleep() function.

void Count7_RestoreConfig(void)

Description: This function restores the counter value, which was previously stored. This function is called by the Count7_Wakeup() function.

Global Variables

Variable	Description
Count7_initVar	Indicates whether the Count7 has been initialized. The variable is initialized to 0 and set to 1 the first time Count7_Start() is called. This allows the Component to restart without reinitialization after the first call to the Count7_Start() routine. If reinitialization of the Component is required, then the Count7_Init() function can be called before the Count7_Start() or Count7_Enable() function.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined: project deviations – deviations that are applicable for all PSoC Creator Components and specific deviations – deviations that are applicable only for this Component. This section provides information on Component specific deviations. Non PSoC 6 project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment. For PSoC 6, refer to PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) for information on MISRA compliance and deviations for files generated by PSoC Creator.

The Count7 Component does not have any specific deviations.

Sample Firmware Source Code

PSoC Creator provides numerous code examples that include schematics and example code in the Find Code Example 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 Code Example " topic in the PSoC Creator Help for more information.

API Memory Usage

The Component memory usage varies significantly, depending on the compiler, device, number of APIs used and Component configuration. The following table provides the memory usage for all APIs available in the given Component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

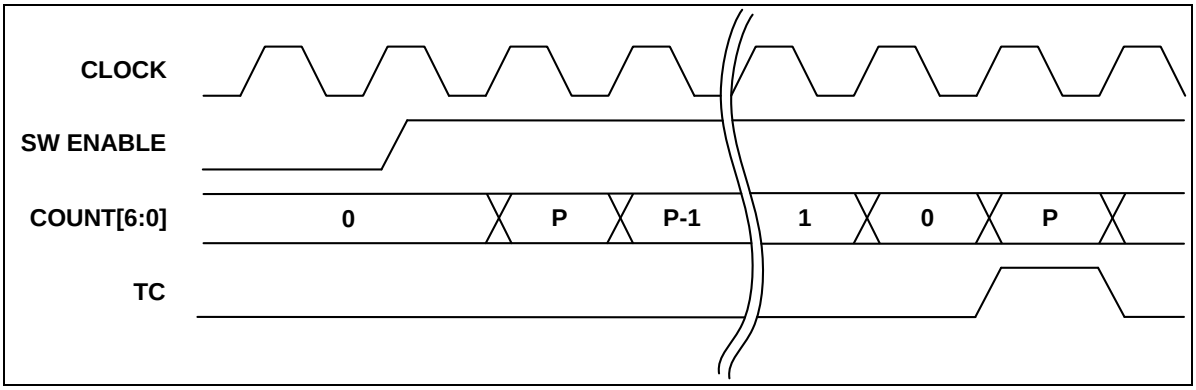
Configuration	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)		PSoC 6 (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default	141	3	256	3	264	3	264	3

Functional Description

Each UDB has a datapath, control register, status register and 2 PLDs. An additional resource that is available is a Count7. A Count7 is 7-bit down counter that borrows some of the resources from the other parts of the UDB. Specifically the Count7 uses the control register, it uses the mask register of the status register and if a hardware load or hardware enable is used, then the inputs that would be used by the status register are consumed. If neither the hardware load nor enable are used, then a status register can still be used in the UDB, but the interrupt capability (statusi) is not available.

Default configuration

By default, the Count7 Component has all optional hardware inputs, which are load, en and reset, not present on the symbol. This is the common use case. In this configuration, the counter will begin counting once Count7_Start() is called and will cycle through the counter values with a single cycle terminal count pulse once per period. In the following diagram, the SW ENABLE is the internal software enable signal (set by Count7_Start()) and the P value is the configured period value:



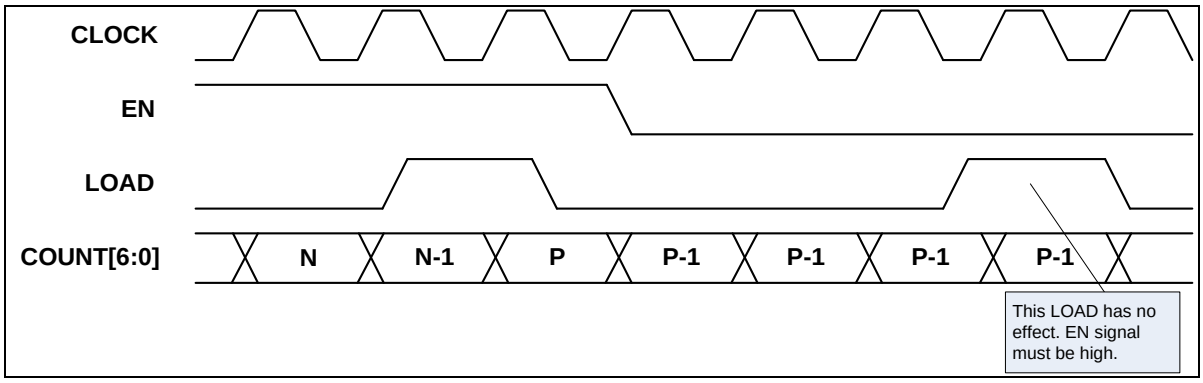
Since the counter counts from the period value down to 0, the overall period is one greater than the value in the period register. Note that the terminal count signal is driven high for the clock cycle following the cycle when the count value is zero.

Using optional inputs

The optional hardware enable signal (en, if enabled in GUI) and the software enable signal (set by the Count7_Start() function) are used as conjunction (AND logic) in the Component.

Optionally a hardware load signal can be used to reload the period value.

Note The load signal is ignored if the counter is not enabled.



The Component may also optionally use the asynchronous reset input. If the reset signal is in the high state, then the value of the counter will always be zero. In this case, the state of the other control signals does not matter.

Registers

The Count7 Component uses the Count and Period registers of the Status & Control block. These registers should not be accessed directly. The provided API functions should be used instead.

Resources

The Count7 Component is placed throughout the UDB array. The Component utilizes only one Count7 Cell. No other resources are utilized by the Component.

Component Debug Window

The Count7 Component supports the PSoC Creator Component debug window. The following registers are displayed in the Count7 Component debug window.

COUNT

This register contains the current value of the COUNT register.

PERIOD

This register contains the current value of the PERIOD register.

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.

Note Final characterization data for PSoC 6 devices is not available at this time. Once the data is available, the Component datasheet will be updated on the Cypress web site.

DC Specifications

Parameter	Description	Min	Typ[1]	Max	Units
I_{DD}	Block current consumption	-	2	-	µA/MHz[2]

AC Specifications

Parameter	Description	Min	Typ	Max	Units
F_{clk}	Clock Frequency	-	-	67[3]	MHz

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

2 Current consumption is given per MHz of input Component clock.

3 The Count7 Component is capable of operating with a clock frequency of up to 67 MHz. However, for PSoC 4 the maximum clock frequency is 48 MHz.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.0.b	Datasheet update	Updated Functional Description section. Updated Application Programming Interface section with the Counter7_Stop() API description. Updated API Memory Usage section with PSoC 6 device data. Updated MISRA section.
	Added PSoC 6 device support	
1.0.a	Minor datasheet update	
1.0	Version 1.0 is the first release of the Count7 Component	

© Cypress Semiconductor Corporation, 2013-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.