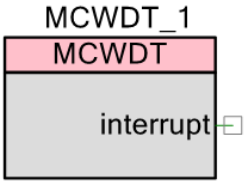
Multi-Counter Watchdog (MCWDT_PDL) 1.0



Features

- Configures up to three counters in a multi-counter watchdog (MCWDT) block
- Two 16-bit counters that can be free running, generate periodic interrupts, or generate a watchdog reset
- One 32-bit counter that can be free running or generate periodic interrupts
- Options to cascade counters
- All counters are clocked by LFCLK
- Peripheral Driver Library (PDL) Component (PDL Application Programming Interface (API) only)

General Description

The Multi-Counter Watchdog Timer (MCWDT_PDL) Component provides an interface to configure one or more MCWDT hardware blocks. Each block contains three counters, each of which can be configured for various system utility functions: free running counter, periodic interrupts, or watchdog reset.

The MCWDT_PDL Component is a graphical configuration entity built on top of the mcwdt driver available in the PDL. It allows schematic-based connections and hardware configuration as defined by the Component Configure dialog.

Note In addition to the MCWDTs, each device has a separate watchdog timer (WDT) that can also be used to generate a watchdog reset or periodic interrupts. For more information on the WDT, see the appropriate section of the PDL.

When to Use a MCWDT_PDL Component

A typical usage example is to use the WDT as a watchdog, and to use the MCWDT counters as free running counters or to generate periodic interrupts. Both have many applications in embedded systems:

- Measuring time between events
- Generating periodic events

- Synchronizing actions
- Real-time clocking
- Polling

A secondary use case is as a watchdog used for recovering from a CPU or firmware failure.

Definitions

- WDT – Watchdog timer, aka just “watchdog”.
- MCWDT – Multi-Counter Watchdog Timer
- C0 – Sub-Counter 0
- C1 – Sub-Counter 1
- C2 – Sub-Counter 2
- WRES – Watchdog Reset: A device-internal signal from the WDT or MCWDT that resets the device.
- ISR – Interrupt Service Routine, aka interrupt handler – code that is executed outside the normal flow of execution, in response to an interrupt signal to the CPU.

Quick Start

The following steps show how to configure the C0 counter interrupt in the MCWDT_PDL Component, which occurs every 2 secs and causes to toggle the GPIO pin:

1. Drag a MCWDT_PDL Component from the Component Catalog Cypress/System folder onto your empty schematic (the placed instance takes the name MCWDT_1).
2. Double-click to open the Configure dialog.
3. Select **Interrupt for Mode** parameter for Counter0.
4. Drag an Interrupt Component from the Component Catalog Cypress /System folder onto your schematic (the placed instance takes the name SysInt_1).
5. Connect the interrupt output from the MCWDT_1 Component to the SysInt_1 using the Wire tool (Hot Key: W).
6. Drag a Digital Output Pin from the Component Catalog Cypress/Ports and Pins folder onto your schematic (the placed instance takes the name Pin_1) and remove the check mark for HW connection.
7. Build the project in order to verify the correctness of your design.

8. In *main_cm4.c*, initialize the peripheral and start the application:

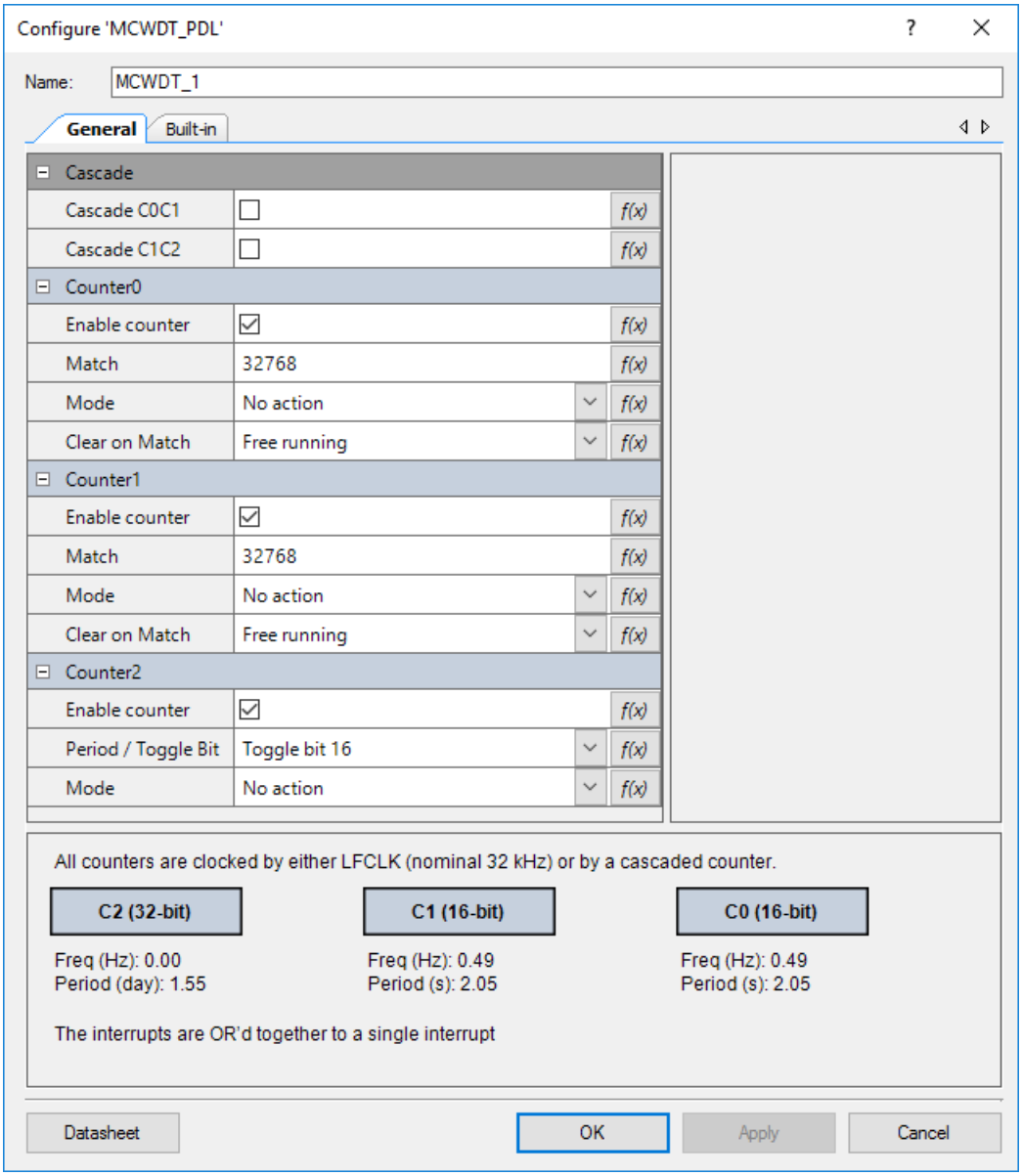
```
#include "project.h"
/* Interrupt prototype */
void MCWDT_1_Interrupt(void)
{
    uint32 mcwdtIsrMask;
    mcwdtIsrMask = Cy_MCWDT_GetInterruptStatus(MCWDT_1_HW);
    if(0u != (CY_MCWDT_CTR0 & mcwdtIsrMask))
    {
        Cy_MCWDT_ClearInterrupt(MCWDT_1_HW, CY_MCWDT_CTR0);
        /* Toggle Pin_1. */
        Cy_GPIO_Inv(Pin_1_0);

        /* Place your application code here. */
    }
}

int main(void)
{
    /* Configure the interrupts. */
    Cy_SysInt_Init(&SysInt_1_cfg, MCWDT_1_Interrupt);
    NVIC_EnableIRQ(srss_interrupt_mcwdt0_IRQn);
    __enable_irq();  /* Enable global interrupts. */

    /* Enable the interrupts for MCWDT counter C0 only. */
    Cy_MCWDT_SetInterruptMask(MCWDT_1_HW, CY_MCWDT_CTR0);

    /* Start MCWDT counter C0. */
    Cy_MCWDT_Init(MCWDT_1_HW, &MCWDT_1_config);
    Cy_MCWDT_Enable(MCWDT_1_HW, CY_MCWDT_CTR0,
MCWDT_1_TWO_LF_CLK_CYCLES_DELAY);

    for(;;)
    {
        /* Place your application code here. */
    }
}
```

9. Modify the void MCWDT_1_Interrupt(void) ISR handler with your code.
10. Build and program the device.

Component Parameters

Drag a MCWDT_PDL Component onto your design and double click it to open the Configure dialog. This dialog has the following tabs with different parameters.

General Tab



This tab has the following parameters:

Cascade pull-down menu

Parameter Name	Description
Cascade C0C1	Controls whether C1 is clocked by LFCLK or from C0 cascade. Range: true / false. Default: false.
Cascade C1C2	Controls whether C2 is clocked by LFCLK or from C1 cascade. Range: true / false. Default: false.

Counter0 pull-down menu

Parameter Name	Description
Enable counter	Enables or disables the counter. Range: true / false. Default: true.
Match (0 – 65535)	Counter match comparison value, for interrupt or watchdog timeout. The frequency is the interrupt frequency in interrupt mode. The period is the inverse of the frequency. Range: • 0 – 65535 for Clear on Match = Free running • 1 – 65535 for Clear on Match = Clear on match. Default: 32768
Mode	Counter 0 mode. Range: No action, Interrupt, Watchdog reset, Three interrupts then watchdog reset. Default: No action. **Note** Interrupt and reset modes have the frequency or period of the associated counter. In the No action mode, the counter has the output frequency and period, but they don't do anything and have no effect except of cascading.
Clear on Match	Controls whether the counter is free running, with a period of 65536 counts, or clear on match, where the period equals the match value + 1. Range: Free running, Clear on match. Default: Clear on match.

Counter1 pull-down menu

Parameter Name	Description
Enable counter	Enables or disables the counter. Range: true / false. Default: true.
Match (0 – 65535)	Counter match comparison value, for interrupt or watchdog timeout. The frequency is the interrupt frequency in interrupt mode. The period is the inverse of the frequency. Range: • 0 – 65535 for Clear on Match = Free running • 1 – 65535 for Clear on Match = Clear on match. Default: 32768
Mode	Counter 1 mode. Range: No action, Interrupt, Watchdog reset, Three interrupts then watchdog reset. Default: No action. **Note** Interrupt and reset modes have the frequency or period of the associated counter. In the No action mode, the counter has the output frequency and period, but they don't do anything and have no effect except of cascading.
Clear on Match	Controls whether the counter is free running, with a period of 65536 counts, or clear on match, where the period equals the match value + 1. Range: Free running, Clear on match. Default: Clear on match.

Counter2 pull-down menu

Parameter Name	Description
Enable counter	Enables or disables the counter. Range: true / false. Default: true.
Period / Toggle Bit	Counter 2 period select. The 32 values are calculated based on the counter clock frequency divided for each of the 32 bits in the counter. Range: 0 – 31. Default: 16.
Mode	Counter 2 mode. Range: No action, Interrupt. Default: No action. **Note** The interrupt mode has the frequency or period of the associated counter which is defined by Period / Toggle Bit. In the No action mode, the counter has the output frequency and period, but they don't do anything.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the Component using software.

By default, PSoC Creator assigns the instance name MCWDT_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.

Note Instance name "CY_MCWDT" is not recommended and causes a build error because the MCWDT_PDL Component uses APIs from the MCWDT PDL module which have "CY_MCWDT" prefix.

This Component uses the MCWDT driver module from the PDL. The driver is copied into the “pdl\drivers\peripheral\mcwdt\” directory of the application project after a successful build.

Refer to the PDL documentation for a detailed description of the complete API. To access this document, right-click on the Component symbol on the schematic and choose the “**Open PDL Documentation…**” option in the drop-down menu.

The Component generates the configuration structures and base address described in the Global Variables section. Pass the generated data structure and the base address to the associated MCWDT driver function in the application initialization code to configure the peripheral. Once the peripheral is initialized, the application code can perform run-time changes by referencing the provided base address in the driver API functions.

Global Variables

The MCWDT_PDL Component populates the following peripheral initialization data structure(s). The generated code is placed in C source and header files that are named after the instance of the Component (e.g. MCWDT_PDL_1.c). Each variable is also prefixed with the instance name of the Component.

const cy_stc_mcwdt_config_t MCWDT_1_config

The instance-specific configuration structure. This should be used in Init() function.

Data in RAM

The generated data may be placed in flash memory (const) or RAM. The former is the more memory-efficient choice if you do not wish to modify the configuration data at run-time. Under the Built-In tab of the Configure dialog set the parameter CONST_CONFIG to make your selection. The default option is to place the data in flash.

Interrupt Service Routine

An MCWDT can generate as many as 3 periodic interrupts. Each CPU has at least one MCWDT, which generates one combined interrupt. An ISR can handle the interrupt either as a periodic interrupt, or as an early indication of a firmware failure to clear the watchdog.

Three sub-counter interrupt outputs are combined into a single interrupt request for each MCWDT. There is local masking before combining, so each sub-counter interrupt may be individually blocked or passed to the combined interrupt output for that MCWDT.

The following sample code is recommended to initialize the interrupts:

```
Cy_SysInt_Init(&SysInt_1_cfg, MCWDT_1_Interrupt);
NVIC_EnableIRQ(srss_interrupt_mcwdt0_IRQn);
Cy_MCWDT_SetInterruptMask(MCWDT_1_HW, CY_MCWDT_CTR0|CY_MCWDT_CTR1|CY_MCWDT_CTR2);
Cy_MCWDT_Init(MCWDT_1_HW, &MCWDT_1_config);
Cy_MCWDT_Enable(MCWDT_1_HW, CY_MCWDT_CTR0|CY_MCWDT_CTR1|CY_MCWDT_CTR2,
MCWDT_1_TWO_LF_CLK_CYCLES_DELAY);
__enable_irq();
```

Code Examples and Application Notes

This section lists the projects that demonstrate the use of the Component.

Code Examples

PSoC Creator provides access to code examples in the 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 "Code Example" topic in the PSoC Creator Help for more information.

There are also numerous code examples that include schematics and example code available online at the Cypress Code Examples web page.

Application Notes

Cypress provides a number of application notes describing how PSoC can be integrated into your design. You can access the Cypress Application Notes search web page at www.cypress.com/appnotes.

Functional Description

DMA Support

The MCWDT_PDL Component supports Direct Memory Access (DMA) transfers. The Component may transfer to/from the following sources.

Name of DMA Source / Destination	Length	Direction	DMA Req Signal	DMA Req Type	Description
MCWDT_1_INSTANCE->MCWDT_MATCH	32	Source / Destination	N/A	N/A	Multi-Counter Watchdog Counter Match Register. 15:0 LSB bits provide the match value for sub-counter 0, 31:16 MSB bits provide the match value for sub-counter 1. It takes up to 2 LFCLK cycles for new match values to take effect.

N/A – The Component does not have any specific requirements.

Industry Standards

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
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The MCWDT_PDL Component has the following specific deviations:

MISRA-C: 2004 Rule	Rule Class (Required/Advisory)	Rule Description	Description of Deviation(s)
1.1	R	The keyword 'inline' has been used.	Deviated since INLINE functions are used to allow more efficient code.
3.1, 11.3	A	Cast between a pointer and an integral type.	The cast from unsigned int to pointer does not have any unintended effect, as it is a consequence of the definition of a structure based on hardware registers.
17.4	R	Array indexing shall be the only allowed form of pointer arithmetic	An array subscript operator is being used to subscript an expression which is not of array type. This is perfectly legitimate in the C language providing the pointer addresses an array element.

This Component uses firmware drivers from the MCWDT Peripheral Driver Library (PDL) module. Refer to the PDL documentation for information on their MISRA compliance and specific deviations.

Registers

See the Multi-Counter Watchdog Timer (MCWDT) Registers section in the chip Technical Reference Manual (TRM) for more information about the registers.

Resources

The MCWDT_PDL Component uses the Multi-counter watchdog timer block of the System Resources Sub-System (SRSS) resource.

DC and AC Electrical Characteristics

TBD

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.a	Updated datasheet.	Updated example code and added information about triggering an interrupt. Removed errata item 256194.
1.0	Initial Version	

© Cypress Semiconductor Corporation, 2016-2017. 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.

PRELIMINARY