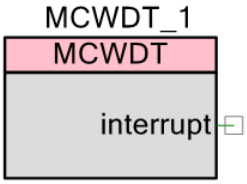
Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

Multi-Counter Watchdog (MCWDT_PDL) 1.10



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 Clk_LF

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 Peripheral Driver Library (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 – Counter 0
- C1 – Counter 1
- C2 – Counter 2
- Clk_LF – Low Frequency Clock
- 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 Counter 0 interrupt in the MCWDT_PDL Component. The interrupt occurs every 2 seconds; the ISR toggles a 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:

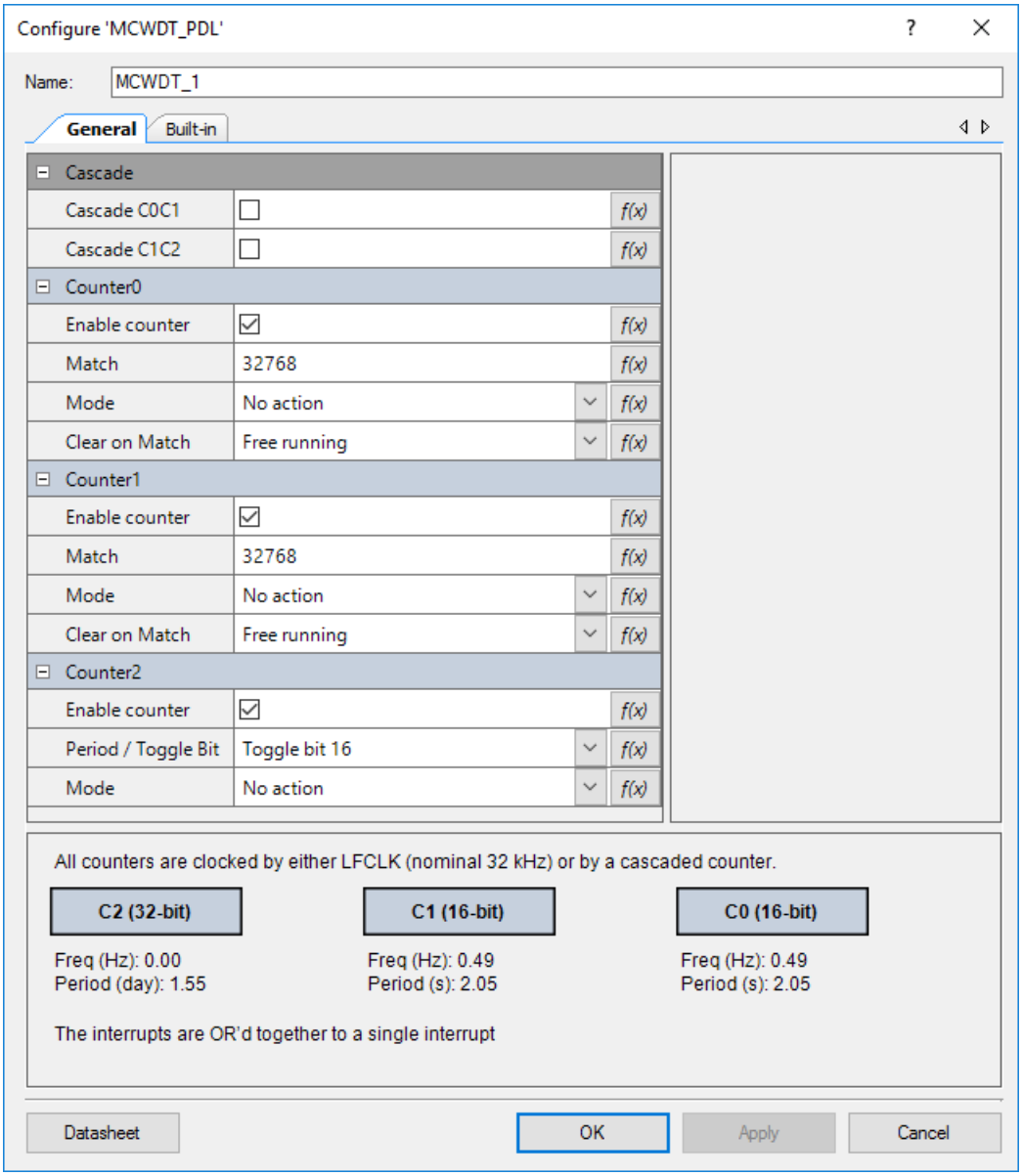
```
#include "project.h"
/* Interrupt handler */
void MCWDT_1_Interrupt(void)
{
    uint32 mcwdtIsrMask;
    mcwdtIsrMask = Cy_MCWDT_GetInterruptStatus(MCWDT_1_HW);

    /* Confirm the Counter0 is the interrupt source. */
    if(0u != (CY_MCWDT_CTR0 & mcwdtIsrMask))
    {
        Cy_MCWDT_ClearInterrupt(MCWDT_1_HW, CY_MCWDT_CTR0);
        /* Toggle Pin_1. */
        Cy_GPIO_Inv(Pin_1_0_PORT, Pin_1_NUM);

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

int main(void)
{
    /* Configure the interrupts. */
    Cy_SysInt_Init(&SysInt_1_cfg, MCWDT_1_Interrupt);
    NVIC_EnableIRQ(srss_interrupt_mcwdt_0_IRQn);
    __enable_irq();  /* Enable global interrupts. */

    /* Enable the interrupts for MCWDT Counter 0 only. */
    Cy_MCWDT_SetInterruptMask(MCWDT_1_HW, CY_MCWDT_CTR0);

    /* Start MCWDT Counter 0. */
    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. 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 Counter 1 is clocked by Clk_LF or from Counter 0 cascade. Range: true / false. Default: false.
Cascade C1C2	Controls whether Counter 2 is clocked by Clk_LF or from Counter 1 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

The Application Programming Interface (API) is provided by 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.

Note Instance name "CY_MCWDT" is not recommended and causes a build error because the MCWDT_PDL Component uses PDL API elements that have a "CY_MCWDT" prefix.

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.

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.

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 three periodic interrupts. Interrupts from all three counters of the MCWDT block are mapped as a single interrupt to the CPU. 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 counter interrupt outputs are combined into a single interrupt request for each MCWDT. There is local masking before combining, so each 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_mcwdt_0_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 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.

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.

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.

Refer to **PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6)** for information on MISRA compliance and deviations of the files generated by PSoC Creator.

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.

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

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.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.10.b	Minor datasheet edits.	
1.10.a	Added additional reset in the Cy_MCWDT_ResetCounters() function to prevent the case when the Counter 1 is not reset when the counters are cascaded.	Defect fix.
1.10	New Component version: • Fixed MISRA rule violations: 12.6, 10.1. • Added MCWDT_GetCountCascaded() function. • Added parameter checks for the MCWDT driver module from the PDL.	Component maintenance.
1.0.b	Updated datasheet.	Updated example code and MISRA information.
1.0.a	Updated datasheet.	Updated example code and added information about triggering an interrupt. Removed errata item 256194.
1.0	Initial Version	