Using the watchdog timer in XMC7000 family MCUs

About this document

Scope and purpose

This application note describes how to handle the watchdog timer in XMC7000 family MCUs. It introduces the functions of the basic watchdog timer and multi-counter watchdog timer and the necessary configurations to generate faults, interrupts, and resets.

Associated part family

XMC7000 family of XMC™ industrial microcontrollers

Intended audience

This document is intended for anyone using XMC7000 family MCUs.

Table of contents

1 Introduction

This application note describes the watchdog timer (WDT) for the XMC7000 family MCUs. A WDT detects an unexpected firmware execution path by generating warning interrupts, faults, or resets. It allows the system to recover from an unsafe execution of an application program.

The WDT includes different counters that are used to observe a predetermined period and monitors the normal operation of the application software by periodically clearing the timer. When the WDT reaches the predetermined period, it detects the condition as an abnormality and generates a reset or an interrupt, or a fault event.

XMC7000 supports two types of WDTs:

- A Basic WDT
- A Multi-counter WDT (MCWDT).

Both WDTs support window mode which allows defining an upper and lower time limit within which the watchdog timer must be served. The Basic WDT is activated by hardware after reset release. Its operation mode is set by the application software during the initial setting. It counts in Active, Sleep, DeepSleep, and Hibernate power modes. The MCWDT is activated and configured by the application software. It counts in Active, Sleep, and DeepSleep power modes.

This document is applicable for XMC7100 series and XMC7200 series devices. Figure 1 shows the block diagram of the WDT. It includes both sub-structures, the Basic WDT, and the MCWDT. See the "Watchdog Timer" chapter of the Architecture Technical Reference Manual (TRM) for more details.

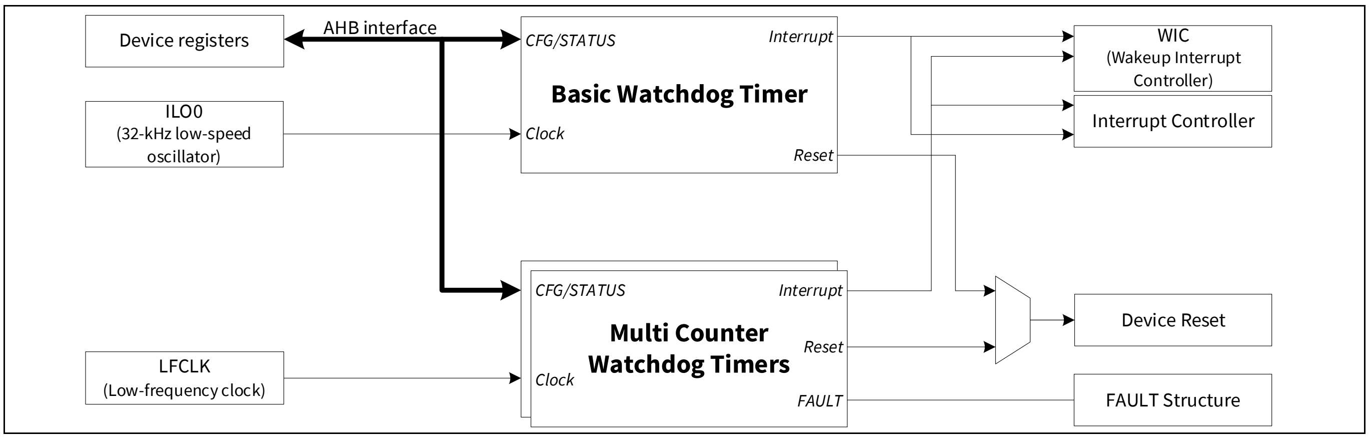


Figure 1 WDT block diagram

2 Basic WDT

Figure 2 shows the block diagram of the Basic WDT. It supports one 32-bit free-running counter that counts up with the ILO0 clock if the ENABLE [31] bit is set to ‘1’ in the WDT_CTL register.

Operation during Hibernate mode is possible because the WDT logic and ILO0 are supplied by the external high-voltage supply (V_{DDD}). A WDT reset restores the chip to Active mode. By default, the Basic WDT is enabled, UPPER_ACTION is configured as reset, UPPER_LIMIT is set with the value 0x8000, and all protectable registers are locked. UPPER_ACTION and UPPER_LIMIT are configuration registers that are used to define the behavior of the Basic WDT in case it is not serviced in time and if a reset should be executed.

WDT configuration registers are in a protection region separate from the register that is used to service it. Protection regions are handled by the Peripheral Protection Unit (PPU). See the CPU Subsystem (CPUSS) chapter in the Architecture TRM for more information.

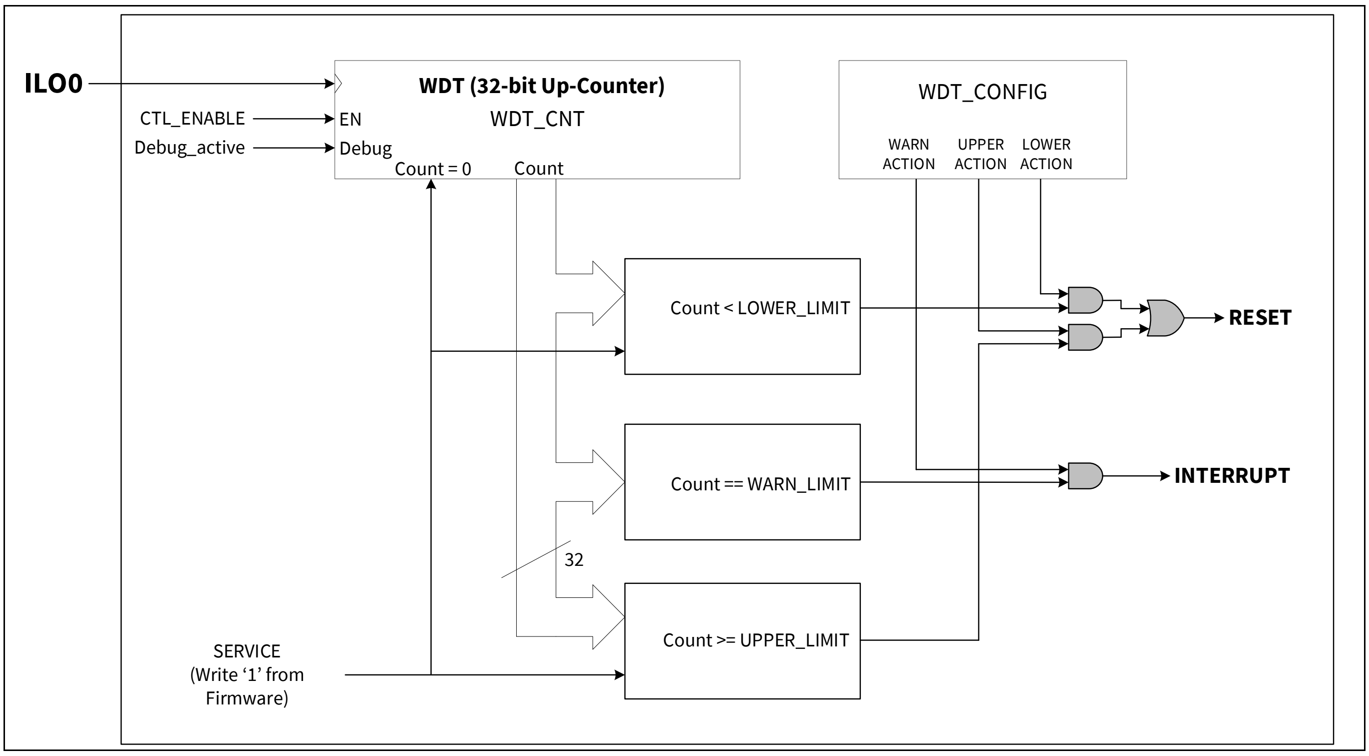


Figure 2 Basic WDT block diagram

Depending on the configuration in the WDT_CONFIG register, an interrupt or a reset event can be generated when the counter reaches related counter limits. Three threshold limits can be used for the following actions:

- LOWER-LIMIT: If the LOWER_ACTION[0] bit is set to ‘1’ in the WDT_CONFIG register, a reset is issued when the watchdog routine is serviced before the WDT reaches the LOWER_LIMIT value.
- UPPER-LIMIT: If the UPPER_ACTION[4] bit is set to ‘1’ in the WDT_CONFIG register, a reset is issued when the WDT reaches the UPPER_LIMIT value before the WDT is serviced.
- WARN-LIMIT: If the WARN_ACTION[8] bit is set to ‘1’ in the WDT_CONFIG register, an interrupt is issued when the WDT reaches the WARN_LIMIT value.

UPPER-LIMIT and LOWER-LIMIT in combination are used to build the window mode for the Basic WDT.

Depending on the Basic WDT mode defined by the ACTION bits in the WDT_CONFIG register, servicing of the watchdog counter must be handled differently. In window mode, the firmware must ensure adequate watchdog servicing timing to fulfill the window timing conditions. If the LOWER_ACTION bit is not set, the Basic WDT can be serviced any time before the UPPER_LIMIT value is reached.

2.1 Source clock

The source clock that can be selected for the Basic WDT is fixed to the ILO0 clock: 32.768 kHz.

2.2 WDT timer counter

The Basic WDT count width is 32 bits. Therefore, the timer period that can be set is between 30.518 µs and 131,072 s. These values are calculated with the typical ILO0 timing. Tolerances must also be considered. See the device datasheet for details.

2.3 Register protection

Changing the register values that are used to configure the Basic WDT requires an UNLOCK sequence of the WDT_LOCK[1:0] bits located in the LOCK register. The following write access sequence to the WDT_LOCK bit field must be performed for unlocking CNT, CTL, LOWER_LIMIT, UPPER_LIMIT, WARN_LIMIT, CONFIG, and SERVICE registers:

- WDT_LOCK = 1
- WDT_LOCK = 2

To regain the lock for the Basic WDT registers, one single access to the LOCK register is required:

- WDT_LOCK = 3

Check the lock status by reading the WDT_LOCK register. If the read value is not equal to ‘0’, it indicates that Basic WDT registers are locked.

After a transition from DeepSleep or Hibernate mode to Active mode, all Basic WDT registers are locked.

2.4 Warning interrupt

The Basic WDT supports a WARN limit that can be used to define a dedicated timing to generate an interrupt. It can be used for different purposes such as follows:

- **Pre-warning event:** The WARN_LIMIT value is defined as lower than the UPPER_LIMIT value. It is enabled if the WARN_ACTION[8] bit in the CONFIG register is set to ‘1’. Note that you should use adequate limits to execute the WARN interrupt in time.
- **Wakeup event:** The Basic WDT can be used as a simple wakeup timer by setting the warning interrupt for the desired wakeup time period. The watchdog counter can send interrupt requests to the wakeup interrupt controller (WIC) in Sleep and DeepSleep power modes. In addition, the Basic WDT is capable of waking up the device from Hibernate power mode. This can be used with or without the normal watchdog reset behavior.

  The configuration of wakeup from Hibernate mode is done in the PWR_HIBERNATE register. See the “System Resources Registers” chapter in the Architecture Technical Reference Manual (TRM) for more details.

  The Basic WDT can be serviced automatically by setting the AUTO_SERVICE[12] bit to ‘1’ in the CONFIG register. Setting up automatic servicing of the Basic WDT creates a periodic interrupt if the Basic WDT counter is not used as a watchdog timer with the timeout reset function. This means that the LOWER_ACTION[0] and UPPER_ACTION[1] bits are set to ‘0’ in the CONFIG register. Servicing the Basic WDT counter in the corresponding interrupt service routine (ISR) is not required. The Basic WDT counter is serviced by the hardware.

Figure 3 illustrates an example for a 500-milliseconds periodic wakeup timing with auto servicing activated. The calculation is done using the following equation:

$$WARN\_LIMIT = 32768\,Hz \times 500\,ms = 16384 = 0x00004000$$

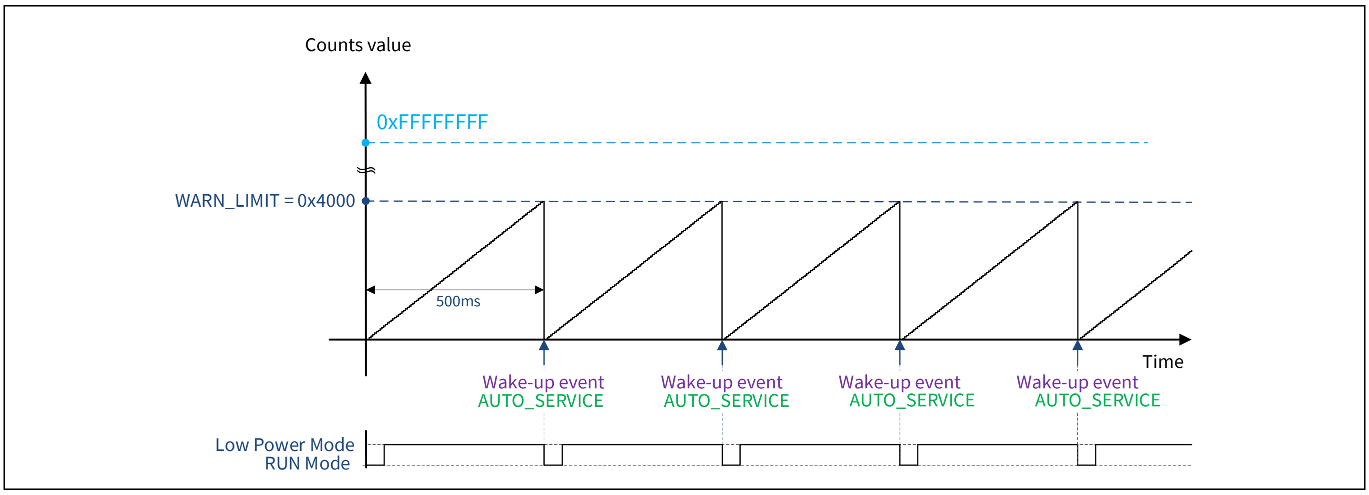


Figure 3 **Periodic wakeup with Basic WDT**

2.5 Timeout mode

The legacy mode of the Basic WDT is the standard watchdog behavior with a timeout condition for resetting the MCU. It uses the UPPER_LIMIT register for generating a reset if the Basic WDT is not serviced in time. Set the UPPER_ACTION[4] bit in the CONFIG register to '1' to trigger a reset when the watchdog counter matches with the UPPER_LIMIT value.

The WARN_LIMIT register can be used as a pre-warning event to indicate an incorrect watchdog counter service timing. Set the WARN_ACTION[8] bit in the CONFIG register to '1' to enable the warn interrupt.

Figure 4 shows an example for the Basic WDT which demonstrates how to define the upper limit timeout period of 1 second and 875-milliseconds pre-warning interrupt timing. Corresponding register values are calculated as follows:

$$UPPER\_LIMIT = 32768\,Hz \times 1\,sec = 32768 = 0x00008000$$
$$WARN\_LIMIT = 32768\,Hz \times 875\,ms = 28672 = 0x00007000$$

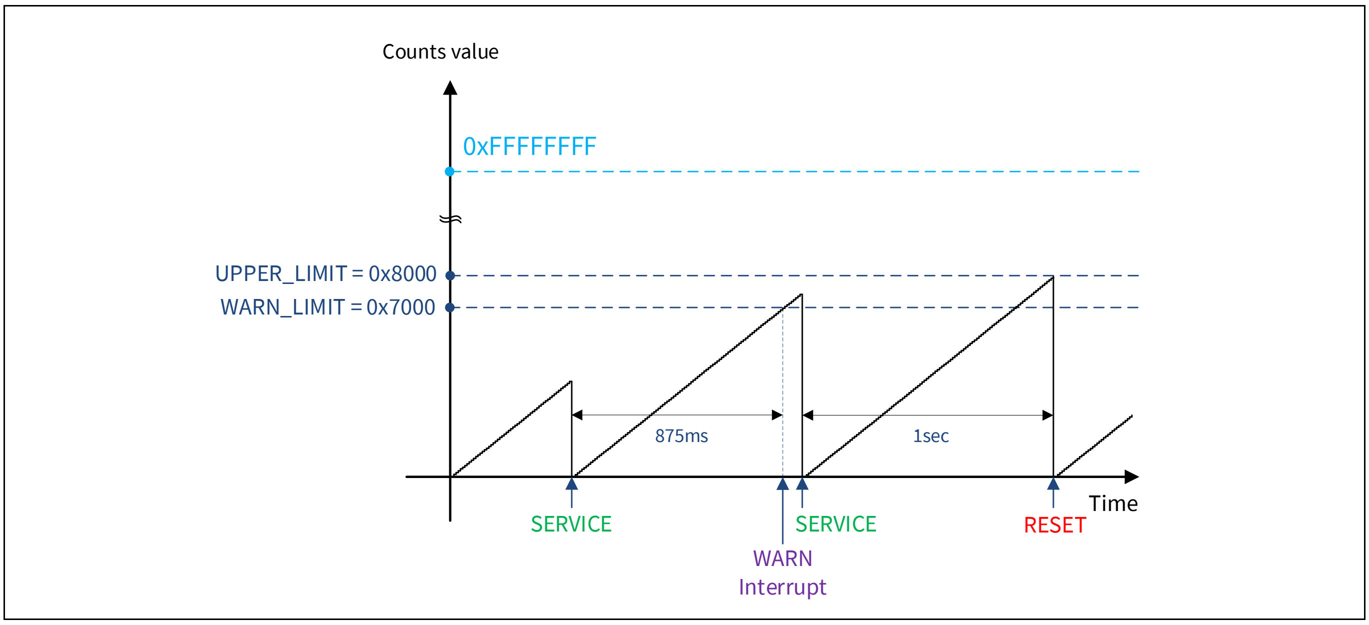


Figure 4 **Basic WDT with timeout and pre-warning**

The example shows the following three scenarios:

- Service the Basic WDT counter before it reaches the WARN_LIMIT.
- Service the Basic WDT counter within the pre-warning ISR.
- If the Basic WDT counter is not serviced in time, a RESET is issued after 1 second.

2.6 Window mode

XMC7000 MCUs support the option to define a lower counter threshold that allows a WDT window mode. WDT window mode supports the observation of two counter limits: a lower limit and an upper limit. If the watchdog is serviced before the counter has reached the configured lower limit value in the LOWER_LIMIT register, a reset is issued. If the watchdog is not serviced before the upper limit of the Basic WDT counter is reached, a reset is issued. The two limits define the window timing within which the Basic watchdog timer must be serviced. To enable this function, the LOWER_ACTION[0] bit in the CONFIG register must be set to ‘1’, and an adequate lower limit period must be defined in the LOWER_LIMIT register.

The following example calculates the LOWER_LIMIT of 150 ms:

$$LOWER\_LIMIT = 32.768\,kHz \times 150\,ms = 4915 = 0x00000CCC$$

2.7 Basic WDT settings

This section describes how to configure the WDT based on a use case using the ModusToolbox™ CAT1 Peripheral Driver Library provided by Infineon. The code snippets in this application note are part of the ModusToolbox™ CAT1 Peripheral Driver Library.

ModusToolbox™ CAT1 Peripheral Driver Library has a configuration part and a driver part. The configuration part configures the parameter values for the desired operation.

Figure 5 shows an example flow to configure the Basic WDT.

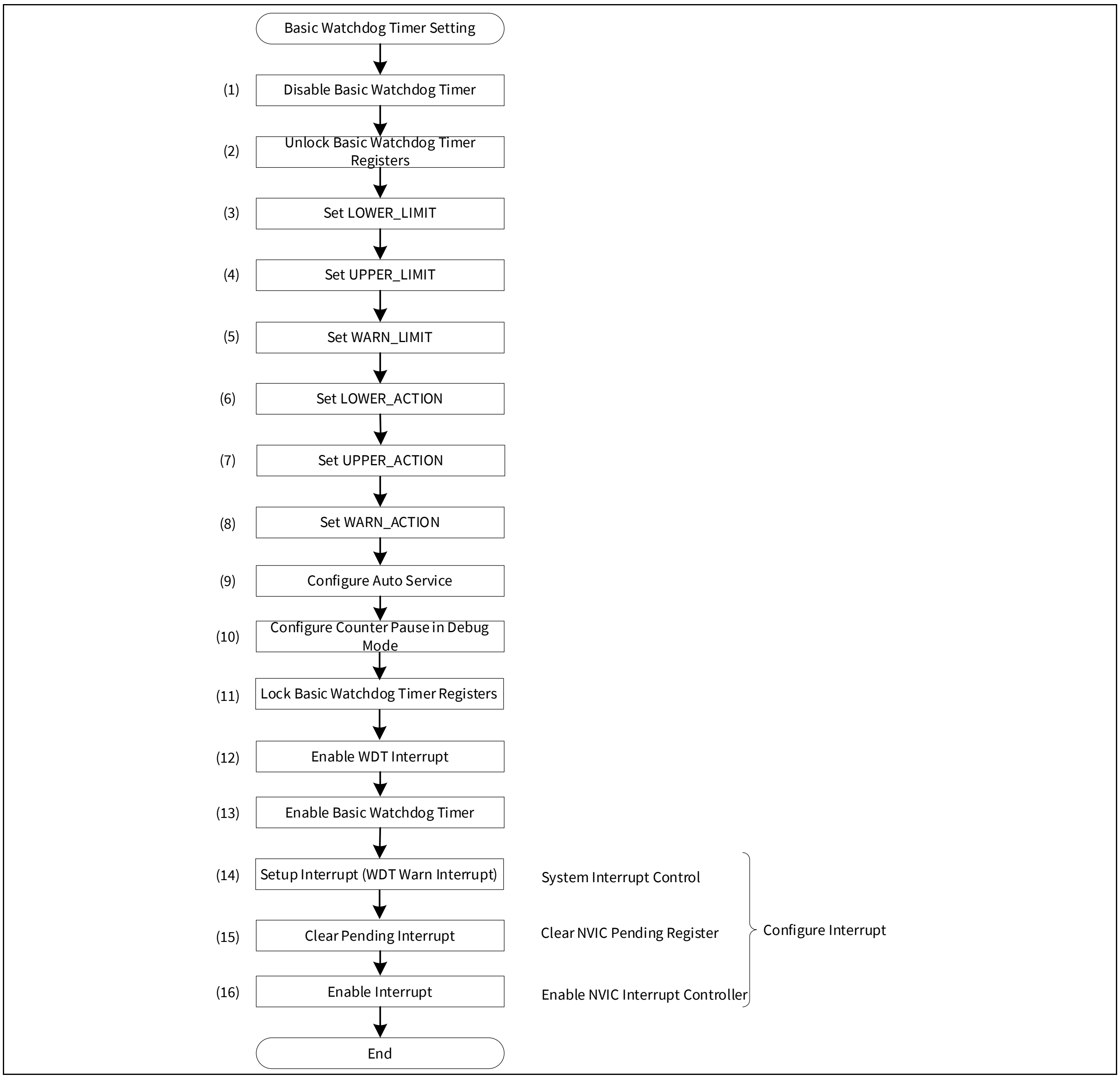

Basic Watchdog Timer Setting
(1) Disable Basic Watchdog Timer
(2) Unlock Basic Watchdog Timer Registers
(3) Set LOWER_LIMIT
(4) Set UPPER_LIMIT
(5) Set WARN_LIMIT
(6) Set LOWER_ACTION
(7) Set UPPER_ACTION
(8) Set WARN_ACTION
(9) Configure Auto Service
(10) Configure Counter Pause in Debug Mode
(11) Lock Basic Watchdog Timer Registers
(12) Enable WDT Interrupt
(13) Enable Basic Watchdog Timer
(14) Setup Interrupt (WDT Warn Interrupt)
(15) Clear Pending Interrupt
(16) Enable Interrupt
End
System Interrupt Control
Clear NVIC Pending Register
Enable NVIC Interrupt Controller
Configure Interrupt

Figure 5 Example flow to configure Basic WDT

2.7.1 Use case

This section explains an example of the Basic WDT using the following use case. The Basic WDT is cleared in the warn interrupt handler. A reset is triggered if the Basic WDT is not cleared between LOWER_LIMIT and UPPER_LIMIT.

Use case:

- LOWER_LIMIT: 125 ms
- UPPER_LIMIT: 1 second

- WARN_LIMIT: 875 ms
- Window mode: Used
- Warn interrupt: Used
- Auto service: Unused
- Debugger configuration: Enables the trigger input for WDT to pause the counter during debug mode

2.7.2 Configuring the Basic WDT

Table 1 lists the parameters of the configuration part in ModusToolbox™ CAT1 Peripheral Driver Library for Basic WDT.

Table 1 List of Basic WDT parameters

Function	Description	Value
`Cy_WDT_SetLowerLimit()`	Sets the lower limit (unsigned integer 32-bit)	4096ul
`Cy_WDT_SetUpperLimit()`	Sets the upper limit (unsigned integer 32-bit)	32768ul
`Cy_WDT_SetWarnLimit()`	Sets the warn limit (unsigned integer 32-bit)	28672ul

`Cy_WDT_SetLowerAction()`	Sets the lower action to “no action” or “reset”:
	CY_WDT_LOW_UPP_ACTION_NONE = 0ul
	CY_WDT_LOW_UPP_ACTION_RESET = 1ul
		CY_WDT_LOW_UPP_ACTION_RESET

`Cy_WDT_SetUpperAction()`	Sets the upper action to “no action” or “reset”:
	CY_WDT_LOW_UPP_ACTION_NONE = 0ul
	CY_WDT_LOW_UPP_ACTION_RESET = 1ul
		CY_WDT_LOW_UPP_ACTION_RESET

`Cy_WDT_SetWarnAction()`	Sets the warn action to “no action” or “interrupt”:
	CY_WDT_WARN_ACTION_NONE = 0ul
	CY_WDT_WARN_ACTION_INT = 1ul
		CY_WDT_WARN_ACTION_INT

`Cy_WDT_SetAutoService()`	Configures to automatically clear the Basic WDT when the count value reaches WARN_LIMIT:
	CY_WDT_DISABLE = 0ul
	CY_WDT_ENABLE = 1ul
		CY_WDT_DISABLE

`Cy_WDT_SetDebugRun()`	Sets the debugger configuration (required when using debugger)
	CY_WDT_DISABLE = 0ul
	CY_WDT_ENABLE = 1ul
		CY_WDT_ENABLE

Code Listing 1 shows an example program of the Basic WDT configuration part. For details of the interrupt initial setting procedure, see the “Interrupt Structure” section in AN234226 listed in Related documents.

Code Listing 1 Example of Basic WDT configuration using Peripheral driver library (PDL)

```
/* Match count */
#define WDT_LOWER_MATCH_COUNT                          125*32000    /* 125ms */
#define WDT_UPPER_MATCH_COUNT                          1000*32000   /* 1000ms */
#define WDT_WARN_MATCH_COUNT                           875*32000    /* 875ms */

/* WDT interrupt configuration structure */
cy_stc_sysint_t wdt_irq_cfg =
{
    .intrSrc = ((NvicMux3_IRQn << 16) | srss_interrupt_wdt_IRQn),
    .intrPriority = 2UL
};

void InitializeWDT()
{
     /* Unlock WDT */
    Cy_WDT_Unlock();

    Cy_WDT_SetLowerLimit(WDT_LOWER_MATCH_COUNT);
    Cy_WDT_SetUpperLimit(WDT_UPPER_MATCH_COUNT);
    Cy_WDT_SetWarnLimit(WDT_WARN_MATCH_COUNT);
    Cy_WDT_SetLowerAction(CY_WDT_LOW_UPPER_LIMIT_ACTION_NONE);
    Cy_WDT_SetUpperAction(CY_WDT_LOW_UPPER_LIMIT_ACTION_NONE);
    Cy_WDT_SetWarnAction(CY_WDT_WARN_ACTION_INT);
    Cy_WDT_SetAutoService(CY_WDT_DISABLE);
    Cy_WDT_SetDebugRun(CY_WDT_ENABLE);

    /* Lock WDT configuration */
    Cy_WDT_Lock();

    /* Clear match event interrupt */
    Cy_WDT_ClearInterrupt();

    /* Enable interrupt if periodic interrupt mode selected */
    Cy_SysInt_Init(&wdt_irq_cfg, WdtInterruptHandler);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    Cy_WDT_UnmaskInterrupt();

    /* Enable WDT */
    Cy_WDT_Enable();
```

```
}

void WdtInterruptHandler(void)
{
    /* Check if the interrupt is from WDT */
    if(SRSS_WDT_INTR & WDT_INTR_WDT_Msk)
    {
        /* Clear WDT Interrupt */
        Cy_WDT_ClearInterrupt();

        /* Invert the state of LED */
        cyhal_gpio_toggle(CYBSP_USER_LED);
    }
}

int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize the User LED */
    result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
            CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);
    /* LED initialization failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Check the reason for device restart */
    if (CYHAL_SYSTEM_RESET_WDT == (cyhal_system_get_reset_reason() &
            CYHAL_SYSTEM_RESET_WDT))
    {
        /* It's WDT reset event - blink LED twice */
        cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_ON);
        cyhal_system_delay_ms(100);
        cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_OFF);
        cyhal_system_delay_ms(200);
        cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_ON);
        cyhal_system_delay_ms(100);
        cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_OFF);
    }
    else
    {
        /* It's Power-On reset or XRES event - blink LED once */
        cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_ON);
        cyhal_system_delay_ms(100);
        cyhal_gpio_write(CYBSP_USER_LED, CYBSP_LED_STATE_OFF);;
        cyhal_system_delay_ms(100);
    }

    /* Clears the reset cause register */
    cyhal_system_clear_reset_reason();

    /* Initialize WDT */
    InitializeWDT();

    /* Enable global interrupt */
    __enable_irq();

    for(;;)
    {
        while(1)
        {
            /* Constant delay of 1000ms */
            Cy_SysLib_Delay(1000);

            /* Invert the state of LED */
            cyhal_gpio_toggle(CYBSP_USER_LED);
        }

    }
}
```

2.8 Clearing the Basic WDT

Clearing the Basic WDT is performed by setting the SERVICE[0] bit to ‘1’ in the SERVICE register. The firmware must consider reading this bit until it is ‘0’ before writing ‘1’ to this bit.

Servicing of the Basic WDT counter must be done regularly to ensure a stable software flow. Independent of the software concept used, the runtime calculation of software components is crucial to define the limits of the counter to be cleared. The window mode makes it even more complex because a minimum time period needs to be determined before which the software is not expected to service the Basic WDT. This minimum time period can be, for example, the minimum execution time of a low-priority main function.

Figure 6 shows an example of when the watchdog counter can be cleared within a system with different tasks. The calculation of each service moment must consider the following conditions:

1. In the window mode, do not service the watchdog before the counter reaches the LOWER_LIMIT.
2. Must service the watchdog counter before reaching the UPPER_LIMIT to avoid a reset event.

The following conditions are defined:

- UPPER_LIMIT = 0x8000: Upper reset threshold is 1 second
- LOWER_LIMIT = 0x1000: Minimum reset threshold is 125 ms
- Task 1 duration: 100 ms
- Task 2 duration: 300 ms
- Task 3 duration: 200 ms
- Task 4 duration: 150 ms
- Task 5 duration: 200 ms

There are different sequences assumed with different timings:

- Sequence 1: $t_{Task1} + t_{Task2} + t_{Task3} + t_{Task4}$ = 100 ms + 300 ms + 200 ms + 150 ms = 750 ms
- Sequence 2: $t_{Task1} + t_{Task4}$ = 100 ms + 150 ms = 250 ms
- Sequence 3: $t_{Task1} + t_{Task4} + t_{Task5}$ = 100 ms + 150 ms + 200 ms = 450 ms

In all cases, the following condition is met:

$t_{LOWER\_LIMIT} < t_{SEQUENCE} < t_{UPPER\_LIMIT}$

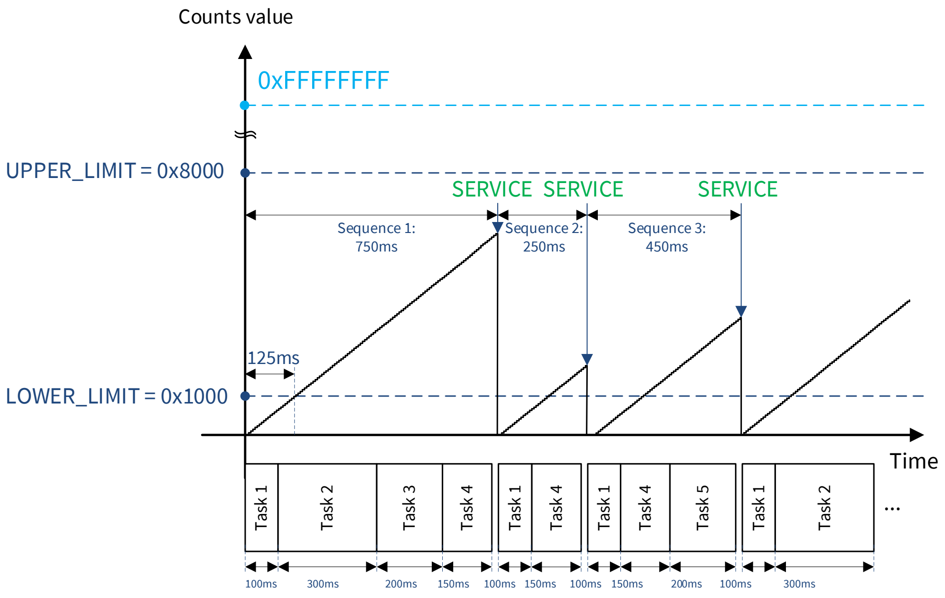


Figure 6 **Example of servicing the Basic WDT in window mode**

2.8.1 Use case

This section describes an example of clearing the Basic WDT using the use case discussed in 2.7.1 Use case.

2.8.2 Example flow to clear the basic WDT

Figure 7 shows an example flow to clear the Basic WDT.

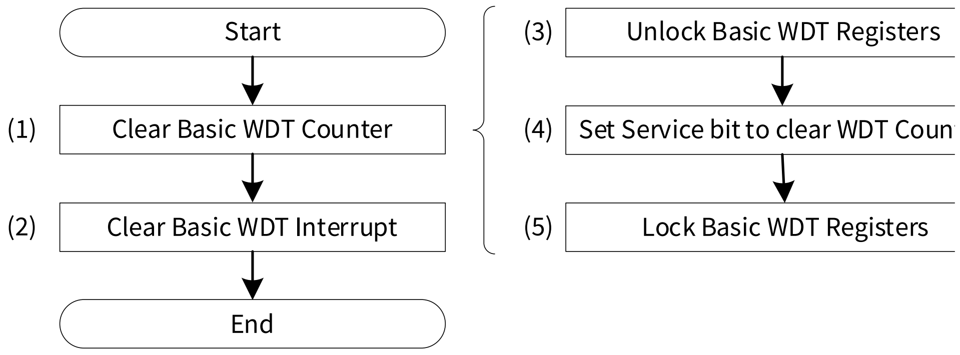

Start
(1) Clear Basic WDT Counter
(2) Clear Basic WDT Interrupt
End
(3) Unlock Basic WDT Registers
(4) Set Service bit to clear WDT Coun
(5) Lock Basic WDT Registers

Figure 7 **Example flow to clear the Basic WDT**

2.9 Reset cause indication for the Basic WDT

If the Basic WDT is not serviced or serviced too early, a system-wide reset is issued. The reset event is stored in the RESET_WDT[0] bit in the RES_CAUSE register.

Note: *The hardware clears this bit during power-on reset (POR). It cannot be distinguished whether a reset was caused by a LOWER_LIMIT or UPPER_LIMIT violation.*

2.10 Basic WDT registers

Table 2 Basic WDT registers

Name	Description
`WDT_CTL`	Watchdog Counter Control Register
`WDT_LOWER_LIMIT`	WDT Lower Limit Register
`WDT_UPPER_LIMIT`	WDT Upper Limit Register
`WDT_WARN_LIMIT`	WDT Warn Limit Register
`WDT_CONFIG`	WDT Configuration Register
`WDT_CNT`	WDT Count Register
`WDT_LOCK`	WDT Lock Register
`WDT_SERVICE`	WDT Service Register
`WDT_INTR`	WDT Interrupt Register
`WDT_INTR_SET`	WDT Interrupt Set Register
`WDT_INTR_MASK`	WDT Interrupt Mask Register
`WDT_INTR_MASKED`	WDT Interrupt Masked Register
`CLK_ILO0_CONFIG`	ILO0 configuration
`RES_CAUSE`	Reset cause observation register

3 Multi-counter WDT

The MCWDT includes three subcounters: Subcounters 0, 1, and 2.

Subcounter 0 and Subcounter 1 are 16-bit counters, which behave like the Basic WDT. Window mode and pre-warning interrupts are supported. If any window timing violation occurs, a FAULT or a reset after a FAULT can be generated if not handled within a timeout timing.

Subcounter 2 is a 32-bit counter, which can be configured to generate an interrupt when one of the pre-defined counter bits toggles. Both types of counters operate during Active, Sleep, and DeepSleep modes. They are not available during Hibernate mode.

Figure 8 illustrates the block diagram of the MCWDT with all three subcounters.

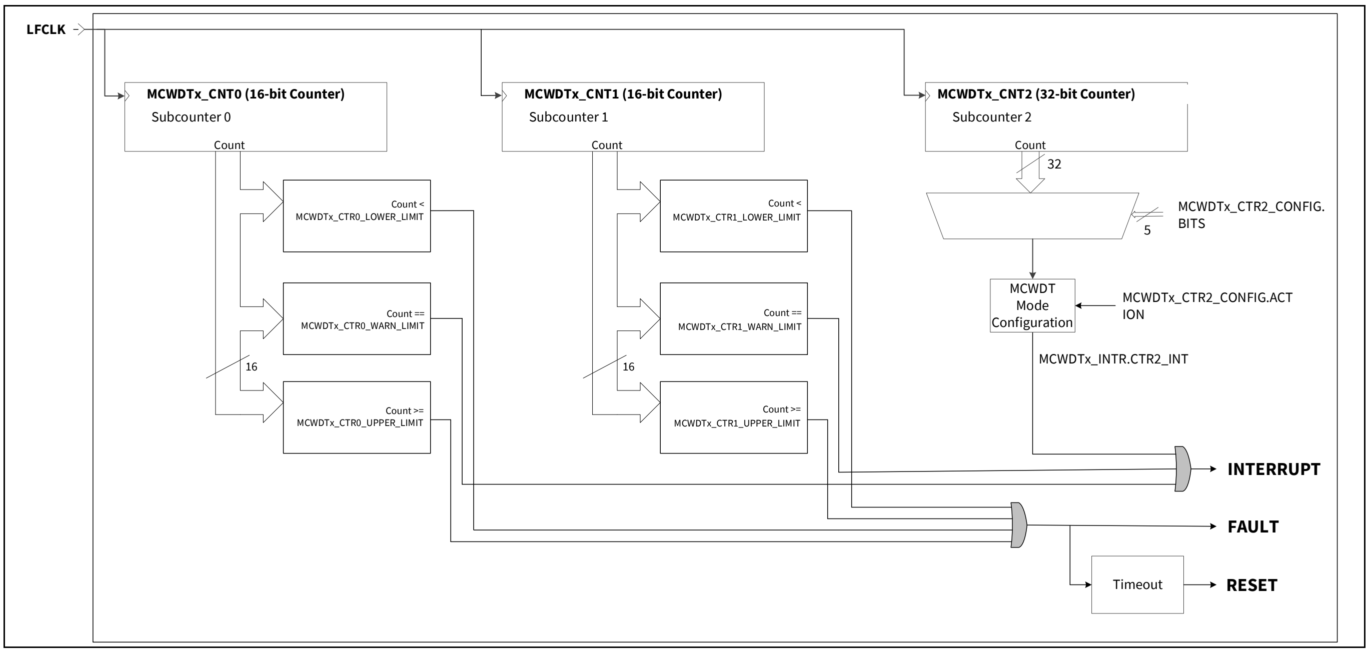


Figure 8 MCWDT block diagram

3.1 Source clock

The source clock that can be selected for MCWDT is LFCLK, which can be one of the following clock sources:

- ILO0/1: Internal low-speed oscillator (32.768 kHz nom.) with relatively poor accuracy
- WCO: Low-frequency watch crystal oscillator (32.768 kHz nom.)
- ECO: High-frequency crystal oscillator (8–33.34 MHz nom.)

3.2 Register protection in MCWDT

Changing the registers related to MCWDT requires an UNLOCK sequence of the MCWDT_LOCK[1:0] bits located in the LOCK register. The following access sequence must be performed for unlocking the following:

Subcounter 2: CTR2_CTL, CTR2_CONFIG, and CTR2_CNT registers

Subcounter 0 and Subcounter 1: CTL, LOWER_LIMIT, UPPER_LIMIT, WARN_LIMIT, CONFIG, SERVICE, and CNT registers

- MCWDT_LOCK = 1
- MCWDT_LOCK = 2

To protect the MCWDT registers, one single write access to the LOCK register is required:

- MCWDT_LOCK = 3

3.3 MCWDT interrupts

MCWDT supports different types of interrupts.

3.3.1 Pre-warning interrupt

Subcounter 0 and Subcounter 1 behave very similarly to the pre-warning interrupt of the Basic WDT. See 2.4 Warning interrupt for more details. The only difference is that the WARN_LIMIT is a 16-bit value that can generate an interrupt timing per the following equation:

$$t_{WARN\_IRQ} = \frac{WARN\_LIMIT}{f_{LFCLK}}$$

The interrupt can be used as a pre-warning event that indicates that the MCWDT counter must be serviced before a FAULT event is issued.

The interrupt is triggered to the related CPU when the WARN_ACTION[8] bit is set to '1' in the CONFIG register.

The MCWDT can be serviced automatically by the AUTO_SERVICE[12] bit in the CONFIG register. This allows the creation of a periodic interrupt if this counter is not needed as a watchdog.

3.3.2 MCWDT Subcounter 2 interrupt

Subcounter 2 interrupt behaves in a different way. A coarse-grained timing should be generated when a dedicated pre-defined counter bit is toggled. The interrupt timing is calculated with the following equation:

$$t_{IRQ} = 2^n \frac{1}{f_{LFCLK}}$$

Example:

LFCLK = ILO0 = 32.768 kHz

Toggle-Bit = Bit 12

$$t_{IRQ} = 2^{12} \frac{1}{32768} = 125\ ms$$

The toggle-bit is configured by BITS[20:16] in the CTR2_CONFIG register. The interrupt is triggered to the related CPU when the ACTION[0] bit is set to '1' in the CTR2_CONFIG register.

3.4 Timeout mode

This mode is related to Subcounter 0 and Subcounter 1 only, and is similar to that of the Basic WDT. See 2.5 Timeout mode for more details. The difference is that the UPPER_LIMIT is a 16-bit value; when the subcounter matches with the UPPER_LIMIT value, a FAULT is generated to be handled in the FAULT structures.

The UPPER_ACTION[1:0] bit field in the CONFIG register specifies how a FAULT is handled:

- No action is taken
- Generate only a FAULT to be handled by the FAULT structures
- Generate a FAULT and trigger a RESET if this FAULT is not handled in < 3 clock cycles

3.5 Window mode

This mode is related to Subcounter 0 and Subcounter 1 only, and is similar to that of the Basic WDT. See 2.6 Window mode for more details. The difference is that the LOWER_LIMIT is a 16-bit value, and if the subcounter is serviced before the counter reaches the LOWER_LIMIT value, a FAULT is generated to be handled in the FAULT structures.

The UPPER_ACTION[5:4] and LOWER_ACTION[1:0] bit fields in the CONFIG register specify how a FAULT is handled as follows:

- No action is taken
- Generate only a FAULT to be handled by the FAULT structures
- Generate a FAULT and trigger a RESET if this FAULT is not handled in < 3 clk cycles

In Figure 9, the window mode is shown when FAULT_THEN_RESET is selected as LOWER_ACTION and UPPER_ACTION. Four scenarios are possible while LOWER_ACTION, WARN_ACTION, and UPPER_ACTION are activated accordingly:

- Counter is serviced between LOWER_LIMIT and WARN_LIMIT: This is the regular behavior of the MCWDT. No WARN interrupt is issued and no RESET is done.
- Counter is serviced between WARN_LIMIT and UPPER_LIMIT: The service is done late; a WARN interrupt is issued but no RESET is done.
- Counter is not serviced at all: A WARN interrupt is issued but even then, the CTR0/1_SERVICE bit is not set. When the counter reaches the UPPER_LIMIT, a FAULT is issued. If the firmware does not handle this FAULT to bring the system back into a safe state, a RESET is issued after a fixed number of LFCLK cycles.
- Counter is serviced before the LOWER_LIMIT is reached: The counter is serviced too early; a FAULT is issued followed by a RESET in case the FAULT is not handled in time by the firmware.

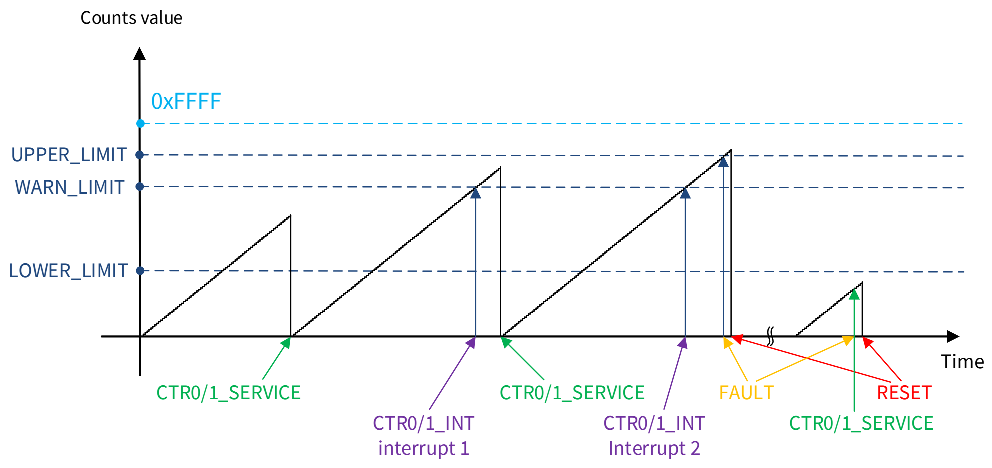


Figure 9 **Subcounter 0/1 operation in window mode with FAULT and RESET action**

3.6 Selecting the CPU

In a multi-CPU system, you should assign one MCWDT to a dedicated CPU to select the SLEEPDEEP for controlling the counter behavior in the respective CPU low-power mode. The counter pauses while the respective CPU is in a low-power mode if the SLEEPDEEP_PAUSE[30] bit is set to '1' in the CTR2_CONFIG register.

A single MCWDT is not intended to be used simultaneously by multiple CPUs because of the complexity involved in coordination.

CPU_SEL[1:0] bits in the CPU_SELECT register are defined in Table 3.

Table 3 MCWDT assignment to cores

CPU_SEL[1:0]	CPU
0	CM0+
1	CM7_0
2	CM7_1

3.7 MCWDT settings

Figure 10 illustrates an example flow to configure the MCWDT.

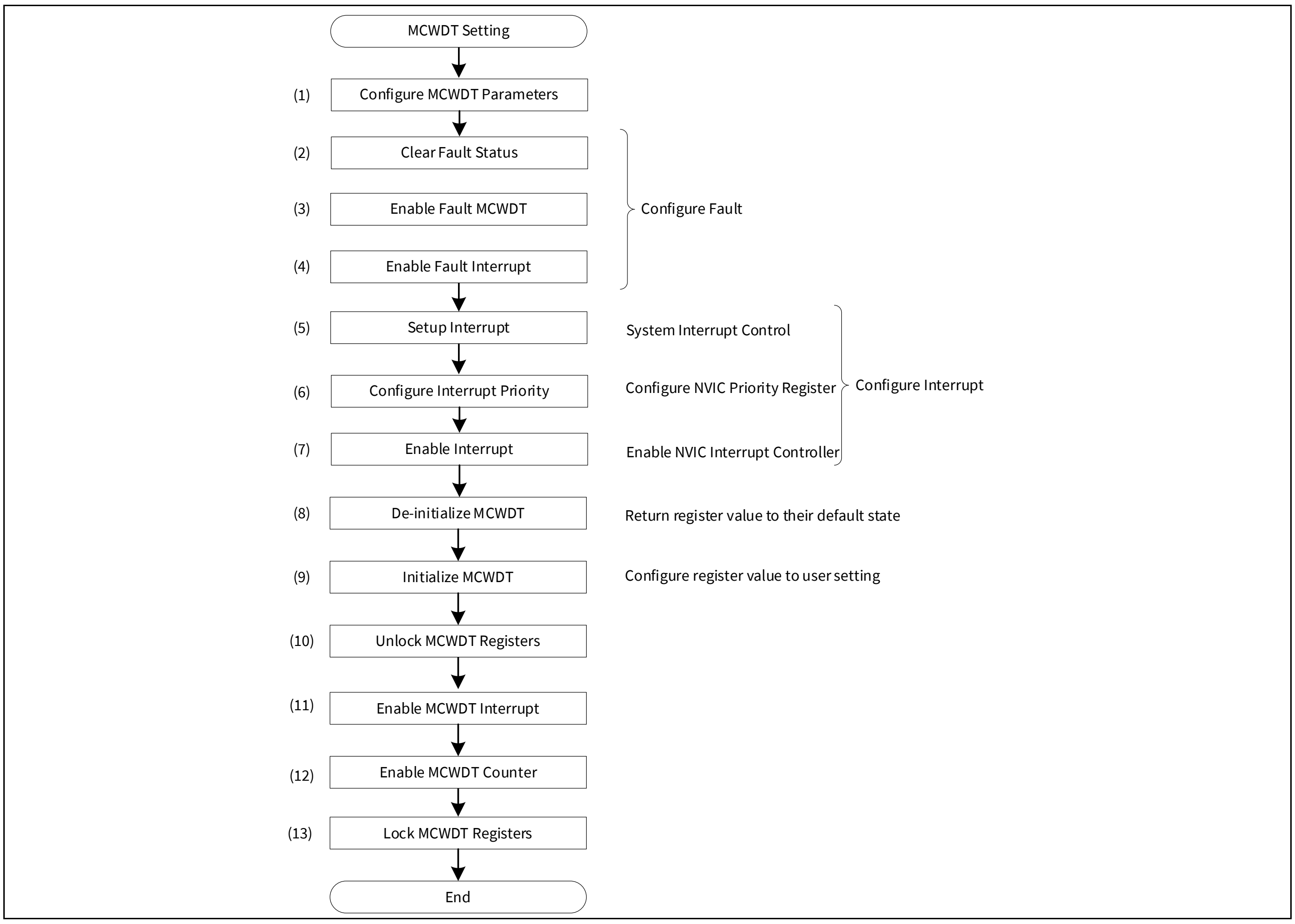

MCWDT Setting
(1) Configure MCWDT Parameters
(2) Clear Fault Status
(3) Enable Fault MCWDT
(4) Enable Fault Interrupt
Configure Fault
(5) Setup Interrupt
System Interrupt Control
(6) Configure Interrupt Priority
Configure NVIC Priority Register
Configure Interrupt
(7) Enable Interrupt
Enable NVIC Interrupt Controller
(8) De-initialize MCWDT
Return register value to their default state
(9) Initialize MCWDT
Configure register value to user setting
(10) Unlock MCWDT Registers
(11) Enable MCWDT Interrupt
(12) Enable MCWDT Counter
(13) Lock MCWDT Registers
End

Figure 10 Multi-counter WDT setting procedure

3.7.1 Use case

This section explains an example of the MCWDT using the following use case. The MCWDT is cleared in the main task loop. The fault interrupt is triggered if the MCWDT is not cleared within the UPPER_LIMIT.

Use case:

- MCWDT number: 0
- CPU: CM7_0
- Subcounter 0
  - LOWER_LIMIT: 0
  - UPPER_LIMIT: 0xFFFF
  - WARN_LIMIT: 1 second (32000, when clk_lf = 32 KHz)
  - LOWER Action: None
  - UPPER Action: None
  - WARN Action: Interrupt
  - Auto service: Enable
  - Deep Sleep Pause: Enable
  - Debug mode: Enable
- Subcounter 1: Same as Subcounter 0
- Subcounter 2:
  - Action: Interrupt
  - Auto service: Enable
  - Deep Sleep Pause: Enable
  - Debug mode: Enable

3.7.2 Configuring the MCWDT

You can configure the parameters in Peripherals tab of the Device Configurator as shown below; then ModusToolbox™ generates the corresponding code automatically.

Double-click *design.modus* to open the Device Configurator.

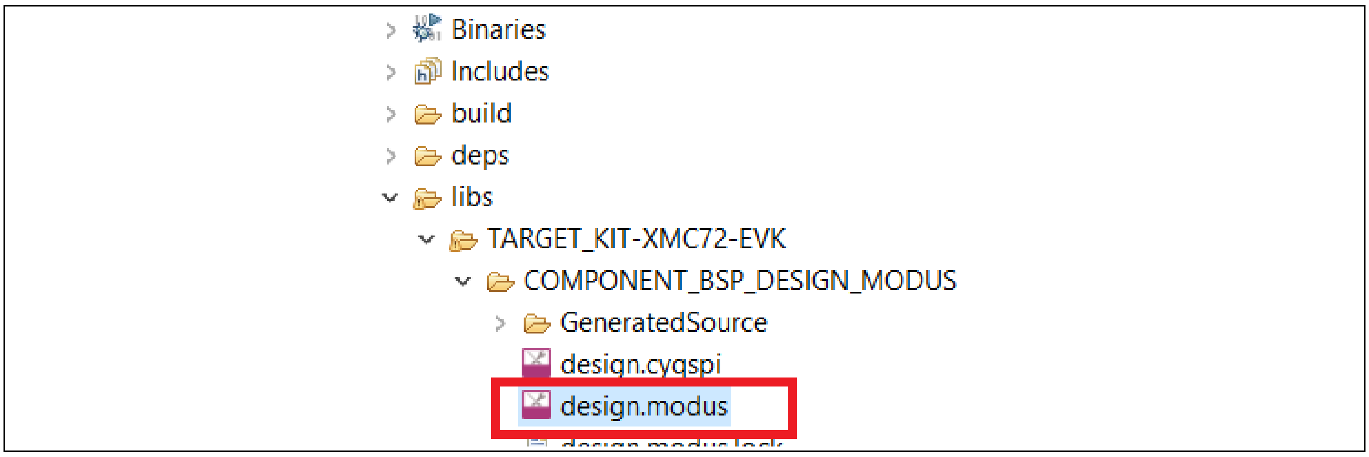
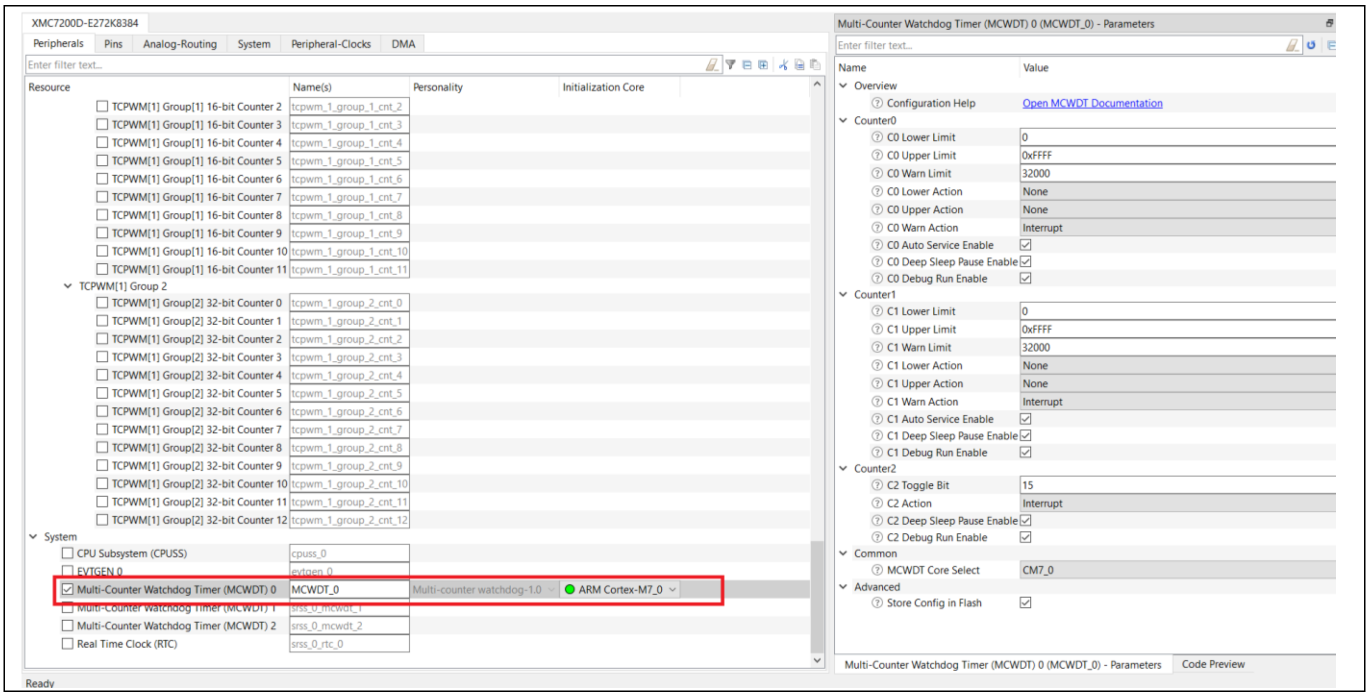


Figure 11 **Example figure**

Multi-counter WDT



XMC7200D-E272K8384

Peripherals | Pins | Analog-Routing | System | Peripheral-Clocks | DMA

Resource	Name(s)	Personality	Initialization Core
☐ TCPWM[1] Group[1] 16-bit Counter 2	tcpwm_1_group_1_cnt_2		
☐ TCPWM[1] Group[1] 16-bit Counter 3	tcpwm_1_group_1_cnt_3		
☐ TCPWM[1] Group[1] 16-bit Counter 4	tcpwm_1_group_1_cnt_4		
☐ TCPWM[1] Group[1] 16-bit Counter 5	tcpwm_1_group_1_cnt_5		
☐ TCPWM[1] Group[1] 16-bit Counter 6	tcpwm_1_group_1_cnt_6		
☐ TCPWM[1] Group[1] 16-bit Counter 7	tcpwm_1_group_1_cnt_7		
☐ TCPWM[1] Group[1] 16-bit Counter 8	tcpwm_1_group_1_cnt_8		
☐ TCPWM[1] Group[1] 16-bit Counter 9	tcpwm_1_group_1_cnt_9		
☐ TCPWM[1] Group[1] 16-bit Counter 10	tcpwm_1_group_1_cnt_10		
☐ TCPWM[1] Group[1] 16-bit Counter 11	tcpwm_1_group_1_cnt_11		
TCPWM[1] Group 2			
☐ TCPWM[1] Group[2] 32-bit Counter 0	tcpwm_1_group_2_cnt_0		
☐ TCPWM[1] Group[2] 32-bit Counter 1	tcpwm_1_group_2_cnt_1		
☐ TCPWM[1] Group[2] 32-bit Counter 2	tcpwm_1_group_2_cnt_2		
☐ TCPWM[1] Group[2] 32-bit Counter 3	tcpwm_1_group_2_cnt_3		
☐ TCPWM[1] Group[2] 32-bit Counter 4	tcpwm_1_group_2_cnt_4		
☐ TCPWM[1] Group[2] 32-bit Counter 5	tcpwm_1_group_2_cnt_5		
☐ TCPWM[1] Group[2] 32-bit Counter 6	tcpwm_1_group_2_cnt_6		
☐ TCPWM[1] Group[2] 32-bit Counter 7	tcpwm_1_group_2_cnt_7		
☐ TCPWM[1] Group[2] 32-bit Counter 8	tcpwm_1_group_2_cnt_8		
☐ TCPWM[1] Group[2] 32-bit Counter 9	tcpwm_1_group_2_cnt_9		
☐ TCPWM[1] Group[2] 32-bit Counter 10	tcpwm_1_group_2_cnt_10		
☐ TCPWM[1] Group[2] 32-bit Counter 11	tcpwm_1_group_2_cnt_11		
☐ TCPWM[1] Group[2] 32-bit Counter 12	tcpwm_1_group_2_cnt_12		
System			
☐ CPU Subsystem (CPUSS)	cpuss_0		
☐ EVTGEN 0	evtgen_0		
✓ Multi-Counter Watchdog Timer (MCWDT) 0	MCWDT_0	Multi-counter watchdog-1.0	● ARM Cortex-M7_0
☐ Multi-Counter Watchdog Timer (MCWDT) 1	srss_0_mcwdt_1		
☐ Multi-Counter Watchdog Timer (MCWDT) 2	srss_0_mcwdt_2		
☐ Real Time Clock (RTC)	srss_0_rtc_0		

Ready

Multi-Counter Watchdog Timer (MCWDT) 0 (MCWDT_0) - Parameters

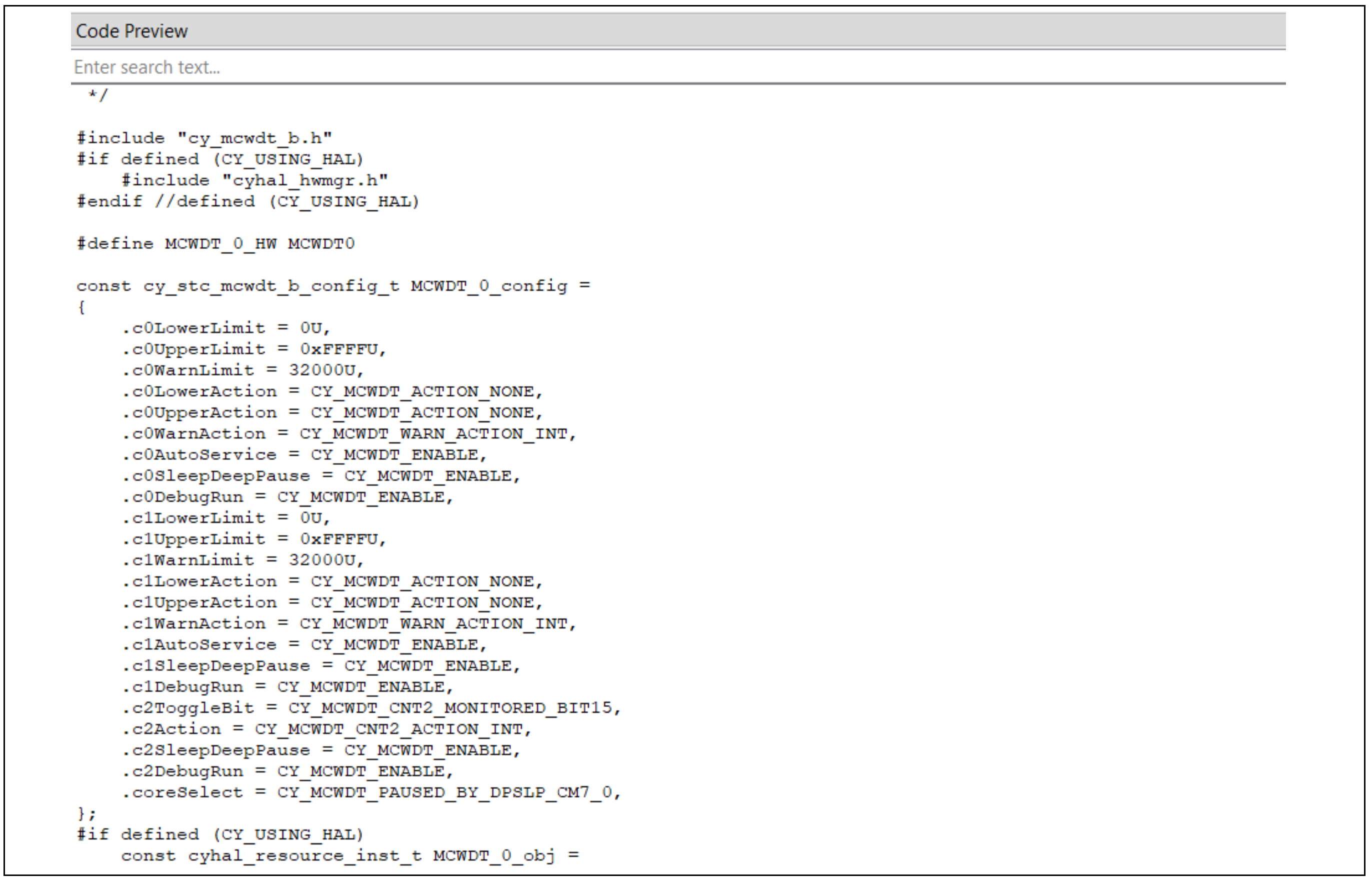
Name	Value
Overview	
Configuration Help	Open MCWDT Documentation
Counter0	
C0 Lower Limit	0
C0 Upper Limit	0xFFFF
C0 Warn Limit	32000
C0 Lower Action	None
C0 Upper Action	None
C0 Warn Action	Interrupt
C0 Auto Service Enable	✓
C0 Deep Sleep Pause Enable	✓
C0 Debug Run Enable	✓
Counter1	
C1 Lower Limit	0
C1 Upper Limit	0xFFFF
C1 Warn Limit	32000
C1 Lower Action	None
C1 Upper Action	None
C1 Warn Action	Interrupt
C1 Auto Service Enable	✓
C1 Deep Sleep Pause Enable	✓
C1 Debug Run Enable	✓
Counter2	
C2 Toggle Bit	15
C2 Action	Interrupt
C2 Deep Sleep Pause Enable	✓
C2 Debug Run Enable	✓
Common	
MCWDT Core Select	CM7_0
Advanced	
Store Config in Flash	✓

Multi-Counter Watchdog Timer (MCWDT) 0 (MCWDT_0) - Parameters | Code Preview

Figure 12 Example figure

Code preview:

This code is auto-generated after the configuration in the Device Configurator.



Code Preview

Enter search text...

```
 */

#include "cy_mcwdt_b.h"
#if defined (CY_USING_HAL)
    #include "cyhal_hwmgr.h"
#endif //defined (CY_USING_HAL)

#define MCWDT_0_HW MCWDT0

const cy_stc_mcwdt_b_config_t MCWDT_0_config =
{
    .c0LowerLimit = 0U,
    .c0UpperLimit = 0xFFFFU,
    .c0WarnLimit = 32000U,
    .c0LowerAction = CY_MCWDT_ACTION_NONE,
    .c0UpperAction = CY_MCWDT_ACTION_NONE,
    .c0WarnAction = CY_MCWDT_WARN_ACTION_INT,
    .c0AutoService = CY_MCWDT_ENABLE,
    .c0SleepDeepPause = CY_MCWDT_ENABLE,
    .c0DebugRun = CY_MCWDT_ENABLE,
    .c1LowerLimit = 0U,
    .c1UpperLimit = 0xFFFFU,
    .c1WarnLimit = 32000U,
    .c1LowerAction = CY_MCWDT_ACTION_NONE,
    .c1UpperAction = CY_MCWDT_ACTION_NONE,
    .c1WarnAction = CY_MCWDT_WARN_ACTION_INT,
    .c1AutoService = CY_MCWDT_ENABLE,
    .c1SleepDeepPause = CY_MCWDT_ENABLE,
    .c1DebugRun = CY_MCWDT_ENABLE,
    .c2ToggleBit = CY_MCWDT_CNT2_MONITORED_BIT15,
    .c2Action = CY_MCWDT_CNT2_ACTION_INT,
    .c2SleepDeepPause = CY_MCWDT_ENABLE,
    .c2DebugRun = CY_MCWDT_ENABLE,
    .coreSelect = CY_MCWDT_PAUSED_BY_DPSLP_CM7_0,
};
#if defined (CY_USING_HAL)
    const cyhal_resource_inst_t MCWDT_0_obj =
```

Figure 13 Code preview

Code Listing 2 shows an example program of the MCWDT configuration part. For details of the interrupt and initial fault setting procedure, see the “Interrupt and Fault Report Structure” section in AN234226 listed in Related documents.

Code Listing 2 Example program to configure MCWDT

```
cy_stc_sysint_t mcwdt_irq_cfg =
{
       .intrSrc = ((NvicMux3_IRQn << 16) | srss_interrupt_mcwdt_0_IRQn),
       .intrPriority = 2UL
};

int main(void)
{
    cy_rslt_t result;
    cy_en_mcwdt_status_t mcwdt_init_status = CY_MCWDT_SUCCESS;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;

    /* BSP initialization failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize retarget-io to use the debug UART port */
    result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                                 CY_RETARGET_IO_BAUDRATE);

    /* retarget-io initialization failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        handle_error();
    }

    /* Initialize the MCWDT_0 */
    mcwdt_init_status = Cy_MCWDT_Init(MCWDT_0_HW, &MCWDT_0_config);
```

Code Listing 2 Example program to configure MCWDT

```
    if(mcwdt_init_status!=CY_MCWDT_SUCCESS)
    {
        handle_error();
    }

    /* Sets up the interrupt handler */
    Cy_SysInt_Init(&mcwdt_irq_cfg, ISR_MCWDT_0);

    /* Enable the MCWDT interrupt in NVIC */
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Enable the MCWDT_0 counters */
    Cy_MCWDT_Unlock(MCWDT_0_HW);
    Cy_MCWDT_SetInterruptMask(MCWDT_0_HW, CY_MCWDT_CTR_Msk);
    Cy_MCWDT_Enable(MCWDT_0_HW, CY_MCWDT_CTR_Msk,
                    0u);
    Cy_MCWDT_Lock(MCWDT_0_HW);

    /* Print a message on UART */
    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");

    printf("*************** "
            "XMC7000 MCU: Multi-Counter Watchdog Timer Example "
            "*************** \r\n\n");

    printf("\r\nMCWDT initialization is complete. USE LED blinking \r\n");


    for(;;)
    {

    }
}
void ISR_MCWDT_0(void)
{
    uint32_t masked;
```

Code Listing 2 Example program to configure MCWDT

```
    masked = Cy_MCWDT_GetInterruptStatusMasked(MCWDT_0_HW);

    if(MCWDT_INTR_MASKED_CTR0_INT_Msk & masked)
    {
        Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);
    }
    if(MCWDT_INTR_MASKED_CTR1_INT_Msk & masked)
    {
        Cy_GPIO_Inv(CYBSP_USER_LED2_PORT, CYBSP_USER_LED2_PIN);
    }
    if(MCWDT_INTR_MASKED_CTR2_INT_Msk & masked)
    {
        Cy_GPIO_Inv(CYBSP_USER_LED3_PORT, CYBSP_USER_LED3_PIN);
    }

    Cy_MCWDT_ClearInterrupt(MCWDT_0_HW, masked);
}
```

3.8 Clearing the MCWDT

Clearing the MCWDT is performed by setting the CTR0_SERVICE[0] bit to '1' for Subcounter 0 and the CTR1_SERVICE[1] bit to '1' for Subcounter 1. Both bits are located in the SERVICE register. The firmware must consider reading the corresponding bit until it is '0' before it can be set to '1'.

- Servicing of the MCWDT counter must be done regularly to ensure a stable software flow. Independent of the software concept used, the runtime calculation of software components is crucial to define the limits of the counter to be cleared. The window mode makes it even more complex because a minimum time period needs to be determined before which the software is not expected to service the MCWDT. This minimum period can be, for example, the minimum execution time of a low-priority main function, and it is relevant to detect the abnormal situation such as the software continuously executing the MCWDT servicing routine without any other code being executed.
- The procedure is equal to the Basic WDT in the window mode. See Figure 7 which shows an example of when the watchdog counter can be cleared within a system with different tasks. The calculation of each service moment must consider that in window mode, the clearing is not done before the counter reaches the LOWER_LIMIT and must not reach the UPPER_LIMIT to avoid a FAULT and reset event.

3.8.1 Use case

This section describes an example of clearing the MCWDT using the use case discussed in 3.7.1 Use case.

3.8.2 Example flow to clear the MCWDT

Figure 14 shows an example flow to clear the MCWDT.

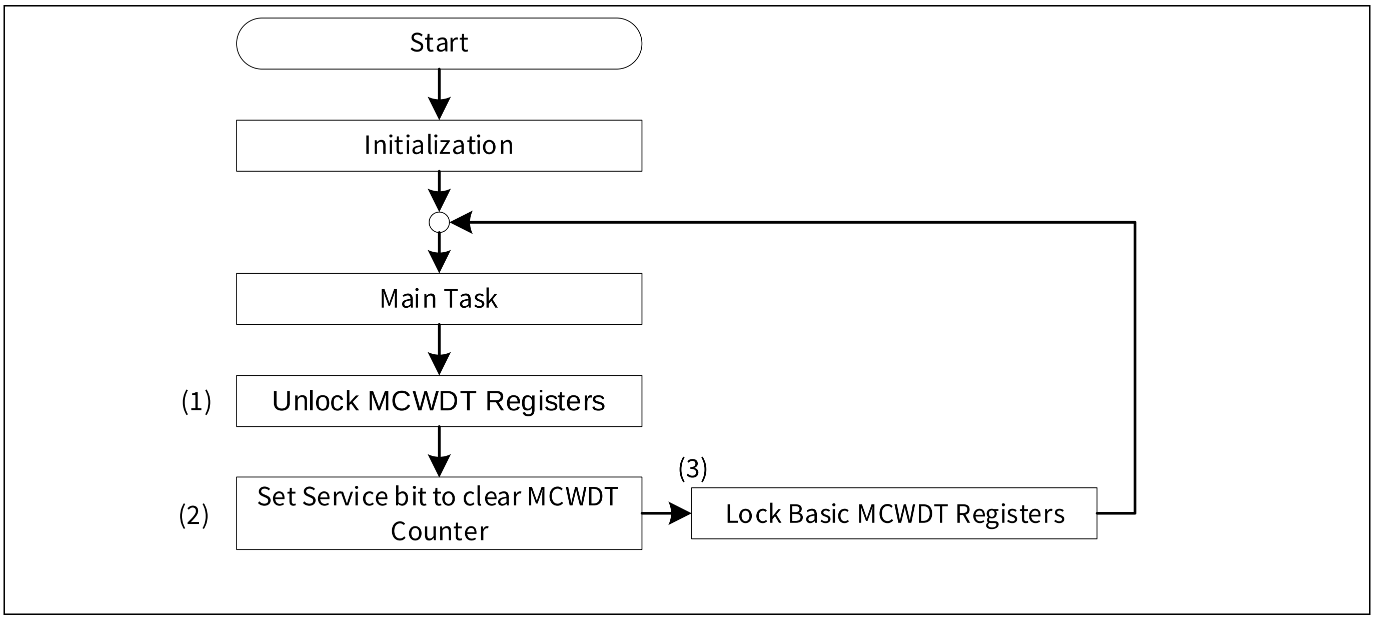

Start
Initialization
Main Task
(1)
Unlock MCWDT Registers
(2)
Set Service bit to clear MCWDT Counter
(3)
Lock Basic MCWDT Registers

Figure 14 Example flow to clear the MCWDT

3.8.3 Example program to clear the MCWDT

Code Listing 3 shows an example program to clear the MCWDT.

Code Listing 3 Example program to clear the MCWDT

```
int main(void)
{
for(;;)
{
Cy_MCWDT_Unlock(MCWDT_0_HW);
Cy_MCWDT_ResetCounters(MCWDT_0_HW, CY_MCWDT_CTR0, 0u);
Cy_MCWDT_Lock(MCWDT_0_HW);

}
}
```

3.10 MCWDT fault handling

The four faults are combined into a single fault report. This report includes the data of which fault is triggered, so the fault handler can record the correct fault cause. Different MCWDT instances have independent fault reports, so they can be handled by different processors.

The initialization of fault reporting is shown in Figure 10 and Code Listing 2. As an example, Fault structure 1 is used.

For details of the fault setting procedure, see the “Fault Report Structure” section in AN234226 listed in Related documents.

The fault is handled within a FAULT report handler. The MCWDT provides the following four FAULT sources:

- Lower limit Fault Subcounter 0
- Upper limit Fault Subcounter 0
- Lower limit Fault Subcounter 1
- Upper limit Fault Subcounter 1

The Fault status can be read from the related Fault structure.

3.10.1 Use case

This section describes an example of the MCWDT fault handling using the use case discussed in 3.7.1 Use case.

3.10.2 Example flow of MCWDT fault handler

Figure 15 shows an example flow of the MCWDT fault handler.

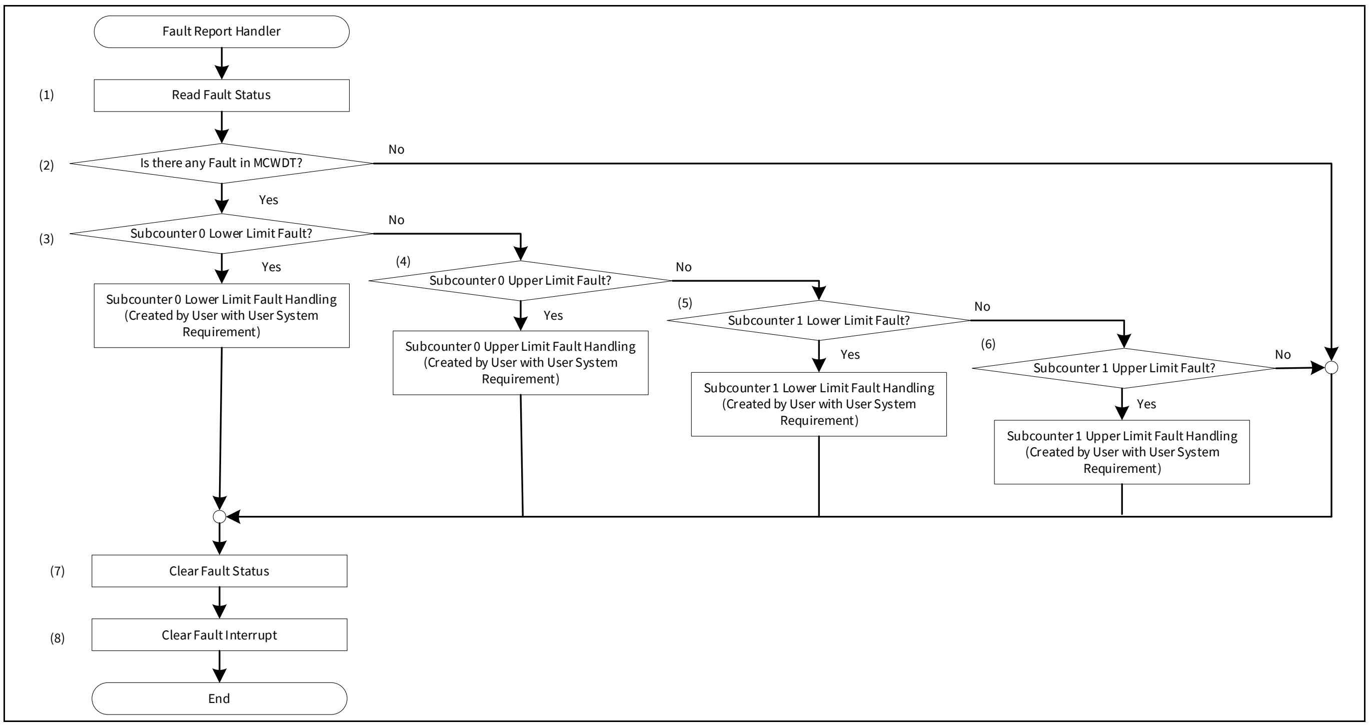

Fault Report Handler
(1)
Read Fault Status
(2)
Is there any Fault in MCWDT?
No
Yes
(3)
Subcounter 0 Lower Limit Fault?
No
Yes
Subcounter 0 Lower Limit Fault Handling (Created by User with User System Requirement)
(4)
Subcounter 0 Upper Limit Fault?
No
Yes
Subcounter 0 Upper Limit Fault Handling (Created by User with User System Requirement)
(5)
Subcounter 1 Lower Limit Fault?
No
Yes
Subcounter 1 Lower Limit Fault Handling (Created by User with User System Requirement)
(6)
Subcounter 1 Upper Limit Fault?
No
Yes
Subcounter 1 Upper Limit Fault Handling (Created by User with User System Requirement)
(7)
Clear Fault Status
(8)
Clear Fault Interrupt
End

Figure 15 Example flow of MCWDT fault handler

3.11 Reset cause indication for MCWDT

If the MCWDT counter is not serviced or serviced too early, a system reset can be issued after the FAULT is not handled in time. When the device comes out of reset, it is useful to know the cause of the reset. Reset causes are recorded in the RES_CAUSE register. Depending on the MCWDT instance used, the reset event is stored in the RESET_MCWDT0[5], RESET_MCWDT1[6], RESET_MCWDT2[7], and RESET_MCWDT3[8] bits in the RES_CAUSE register. The bits in the RES_CAUSE register are set on the occurrence of the corresponding reset and remain set until cleared by the user software or a power-on reset (POR).

3.12 MCWDT registers

Table 4 MCWDT registers

Name	Description
`MCWDTx_CTRy_CTL`	MCWDT Subcounter 0/1 Control Register
`MCWDTx_CTRy_LOWER_LIMIT`	MCWDT Subcounter 0/1 Lower Limit Register
`MCWDTx_CTRy_UPPER_LIMIT`	MCWDT Subcounter 0/1 Upper Limit Register
`MCWDTx_CTRy_WARN_LIMIT`	MCWDT Subcounter 0/1 Warn Limit Register
`MCWDTx_CTRy_CONFIG`	MCWDT Subcounter 0/1 Configuration Register
`MCWDTx_CTRy_CNTy`	MCWDT Subcounter 0/1 Count Register
`MCWDTx_CTR2_CTL`	MCWDT Subcounter 2 Control Register
`MCWDTx_CTR2_CONFIG`	MCWDT Subcounter 2 Configuration Register
`MCWDTx_CTR2_CNT`	MCWDT Subcounter 2 Count Register
`MCWDTx_LOCK`	MCWDT Lock Register
`MCWDTx_SERVICE`	MCWDT Service Register
`MCWDTx_INTR`	MCWDT Interrupt Register
`MCWDTx_INTR_SET`	MCWDT Interrupt Set Register
`MCWDTx_INTR_MASK`	MCWDT Interrupt Mask Register
`MCWDTx_INTR_MASKED`	MCWDT Interrupt Masked Register
`CLK_SELECT`	Clock selection register
`CLK_ILO0_CONFIG`	ILO0 configuration
`RES_CAUSE`	Reset cause observation register

4 Debug support

Both types of WDTs support different debug modes. The configuration is done with the DEBUG_TRIGGER_ENABLE[28] and DEBUG_RUN[31] bits, which are both located in the related CONFIG register for Basic WDT and MCWDT. The WDT reset request is blocked during debug modes, while debugging through MCWDT reset is possible using breakpoints during debug modes.

Table 5 Debug modes

DEBUG_RUN	DEBUG_TRIGGER_ENABLE	Description
0	0	Counter is stopped when a debugger is connected.
0	1	Counter is stopped only when a debugger is connected and the CPU is halted during a breakpoint.
1	X	Counter is running when debugger is connected. No reset is issued when the CPU is halted during a breakpoint but the counter is not stopped.

Note: In each case, no reset or FAULT is issued when the debugger is connected to the target system.

To pause at a breakpoint while debugging, configure the trigger matrix to connect the related 'CPU halted' signal to the trigger input for the related WDT. It takes up to two LFCLK cycles for the trigger signal to be processed. Triggers that are less than two LFCLK cycles may be missed. Synchronization errors can accumulate each time it is halted.

5 Definitions, acronyms, and abbreviations

Table 6 Definitions, acronyms, and abbreviations

Terms	Definitions
AHB	Advanced High-performance Bus
CPU	Central Processing Unit
CPUSS	CPU subsystem
ECO	External Crystal Oscillator
ILO0	32-kHz internal low-speed oscillator
IRQ	Interrupt request
ISR	Interrupt Service Routine
kHz	kilohertz
LFCLK	Low-frequency clock
MCWDT	Multi-Counter Watchdog Timer
ms, msec	milliseconds
POR	Power-on reset
PPU	Peripheral Protection Unit
sec	second
SW	Software
V_{DDD}	External high-voltage supply
WCO	Watch Crystal Oscillator
WDT	Watchdog Timer
WIC	Wakeup interrupt controller

6 Related documents

Contact Technical support to obtain XMC7000 family references documents.

Device datasheet

- XMC7100 series 32-bit Arm® Cortex®-M7 microcontroller datasheet
- XMC7200 series 32-bit Arm® Cortex®-M7 microcontroller datasheet

Device architecture TRM

- XMC7000 MCU family architecture technical reference manual

Application notes

- AN234226 – Interrupt usage in XMC7000 family

Revision history

Document revision	Date	Description of changes
**	2021-12-08	Initial release.
*A	2022-05-25	Updated the code listing.
*B	2023-04-10	Updated Related documents Deleted the redundant section 6 Other reference. Updated the broken links. Updated to new template.

Edition 2023-04-10

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-33887 Rev. *B