Using the watchdog timer in TRAVEO™ T2G family MCUs

About this document

Scope and purpose

This application note describes how to handle the watchdog timer in the TRAVEO™ T2G 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 reset.

Intended audience

This document is intended for anyone using the TRAVEO™ T2G family MCUs.

Table of contents

1 Introduction

This application note describes the watchdog timer (WDT) for TRAVEO™ T2G 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.TRAVEO™ T2G supports two types of WDTs; a basic WDT and a multi-counter WDT (MCWDT). Both WDTs support window mode that 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 application software is responsible for activation of MCWDT and the configuration of its operation mode. It counts in Active, Sleep, and DeepSleep power modes. This document is applicable for CYT2 series, CYT3 series, CYT4 series, and CYT6 series devices. Figure 1 shows the block diagram of the WDT. It includes both sub structures, the basic WDT, and the MCWDT.

To understand the functionality described and terminology used in this application note, see the 'Watchdog Timer' chapter of the Architecture Technical Reference Manual (TRM).

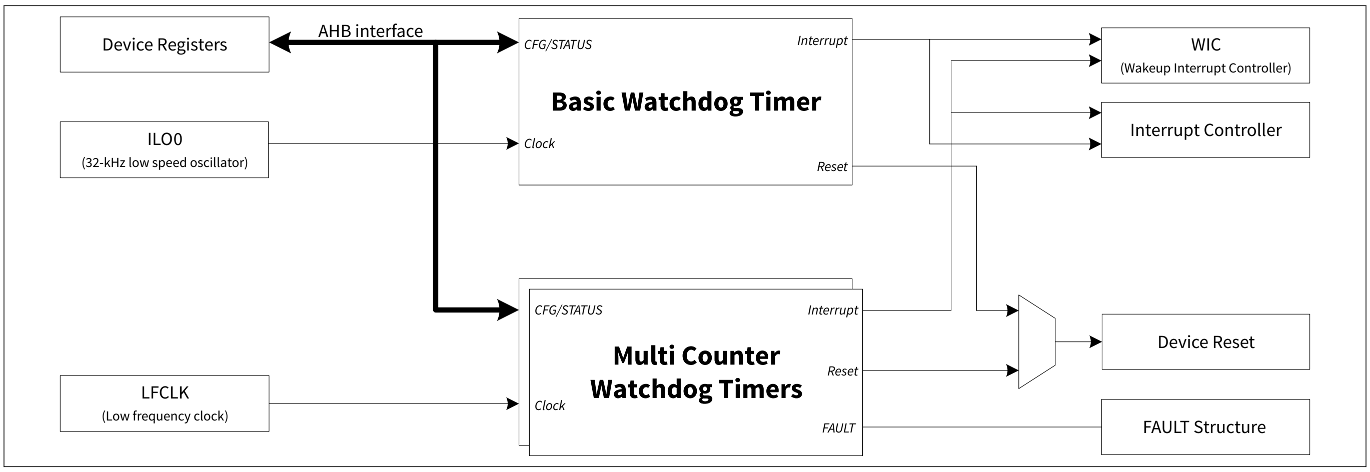

Figure 1 WDT block diagram

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). For more information, refer to the CPU Subsystem (CPUSS) chapter in the Architecture (TRM).

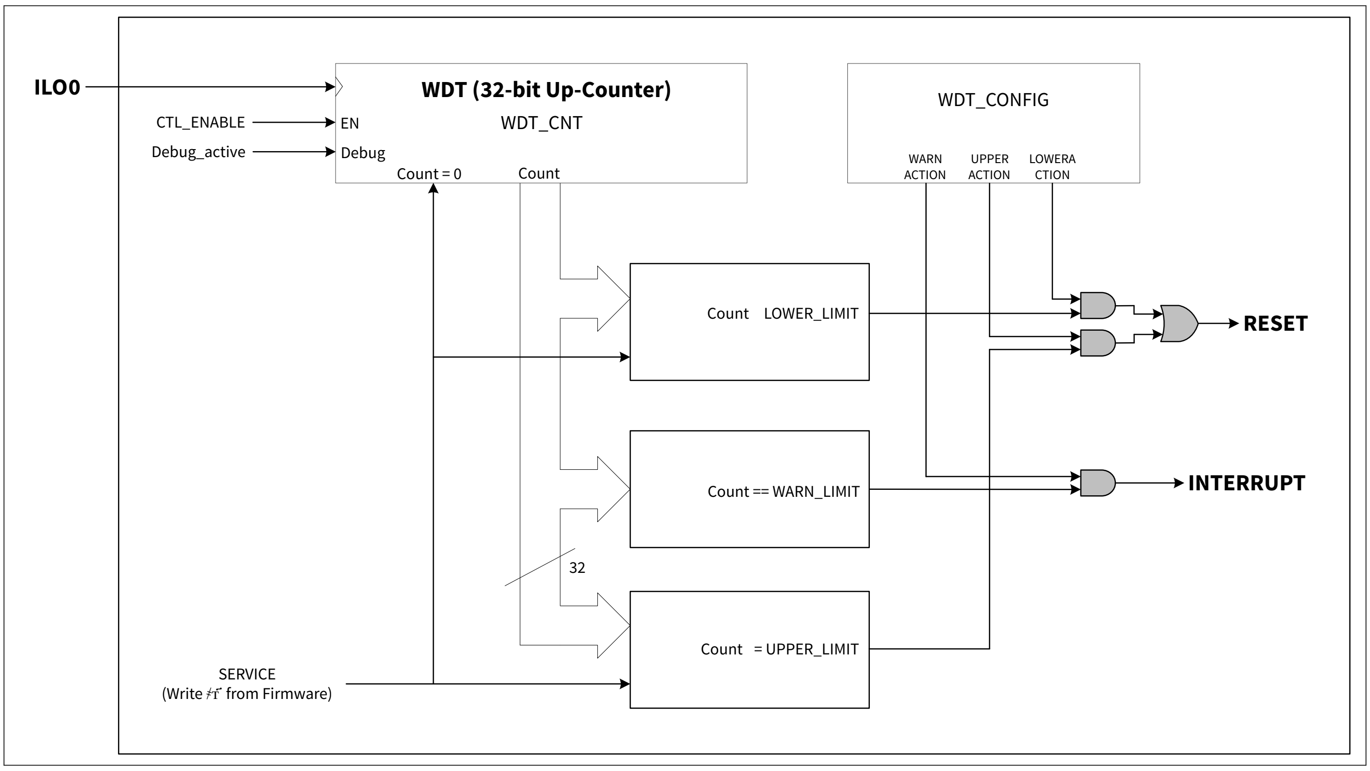

Figure 2 Basic WDT block diagram

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 anytime 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 LOCK register is required:

- WDT_LOCK = 3

Check the lock status by reading the WDT_LOCK register. If the read value is unequal 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
- **Wake-up 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 Systems 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 × 500 ms = 16384 = 0x00004000

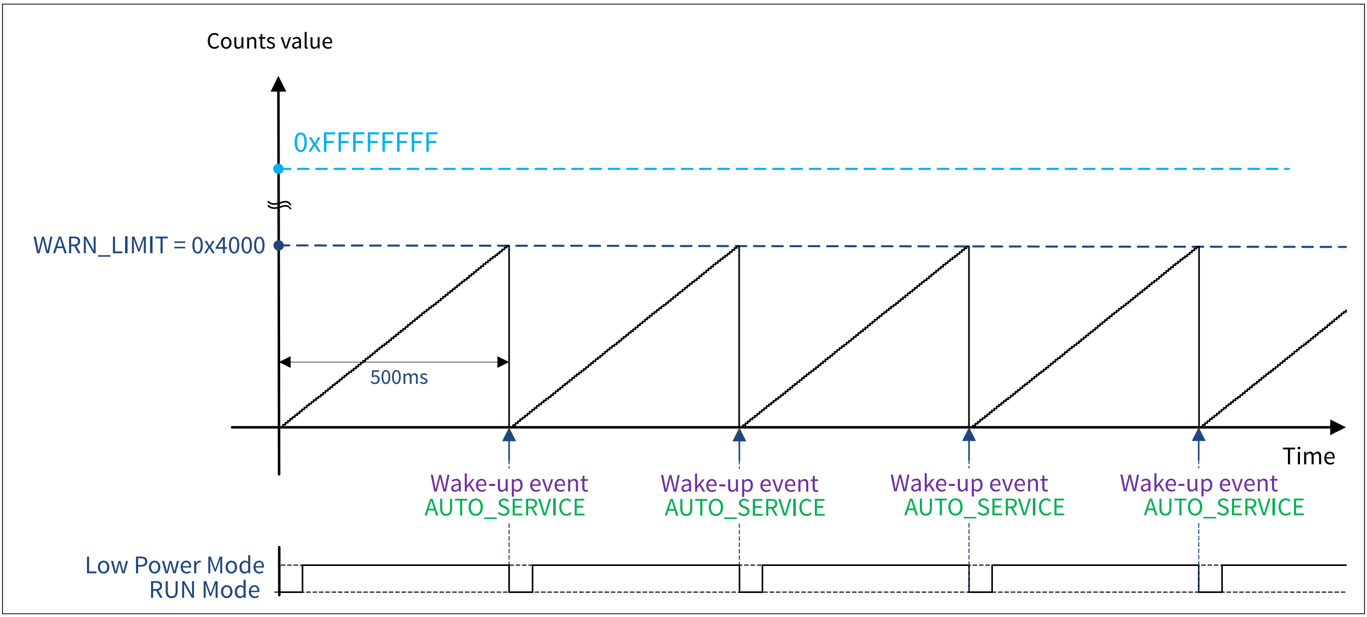

Figure 3 **Periodic wakeup with basic WDT**

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 × 1 sec = 32768 = 0x00008000

WARN_LIMIT = 32768 Hz × 875 ms = 28672 = 0x00007000

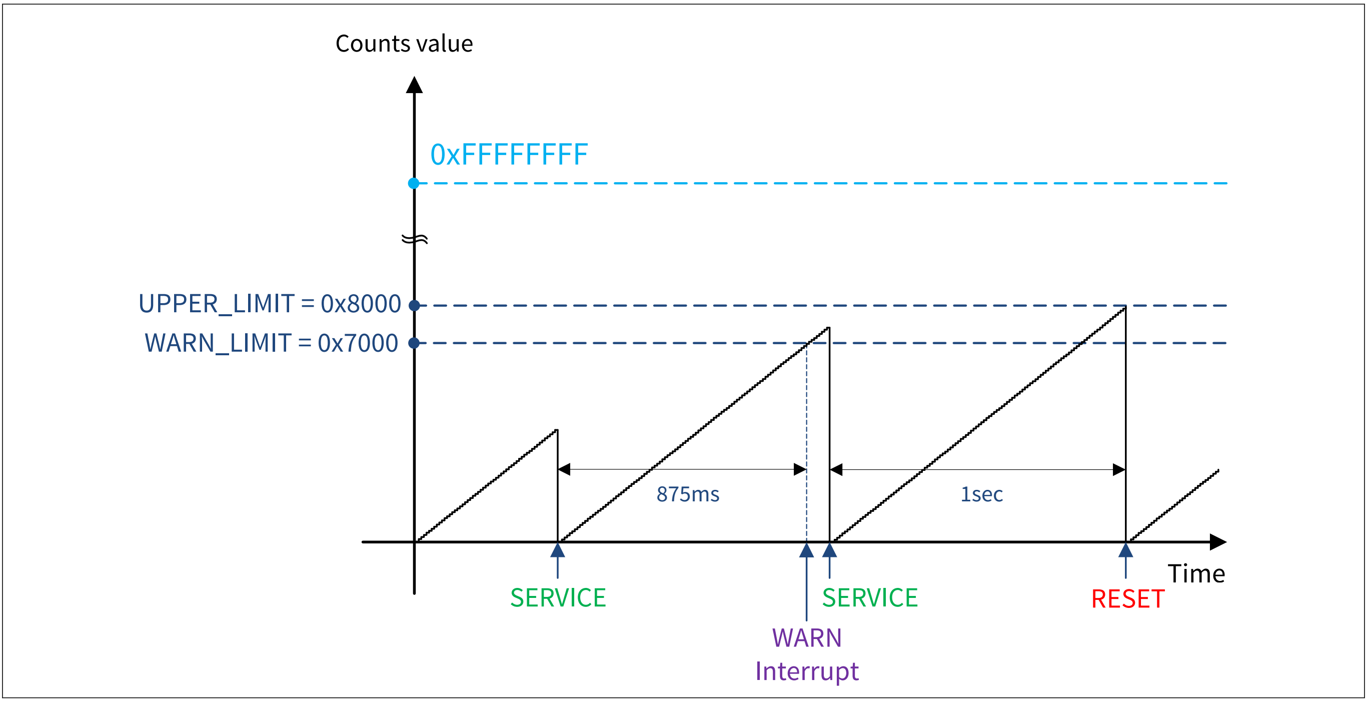

Figure 4 **Basic WDT with timeout and pre-Warning**

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 not serviced in time, a RESET is issued after 1 second

2.6 Window mode

TRAVEO™ T2G MCUs support the option to define a lower counter threshold that allows a WDT window mode. WDT window mode supports 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 LOWER_LIMIT register.

The following example calculates the LOWER_LIMIT of 150 ms:

LOWER_LIMIT = 32.768 kHz × 150 ms = 4915 = 0x00000CCC

2.7 Basic WDT settings

This section describes how to configure the WDT based on a use case using the sample driver library (SDL) provided by Infineon. The code snippets in this application note are part of the SDL. For more information, see Other references.

SDL has a configuration part and a driver part. The configuration part configures the parameter values for the desired operation. The driver part configures each register based on the parameter values in the configuration part.

You can configure the configuration part according to your system.

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

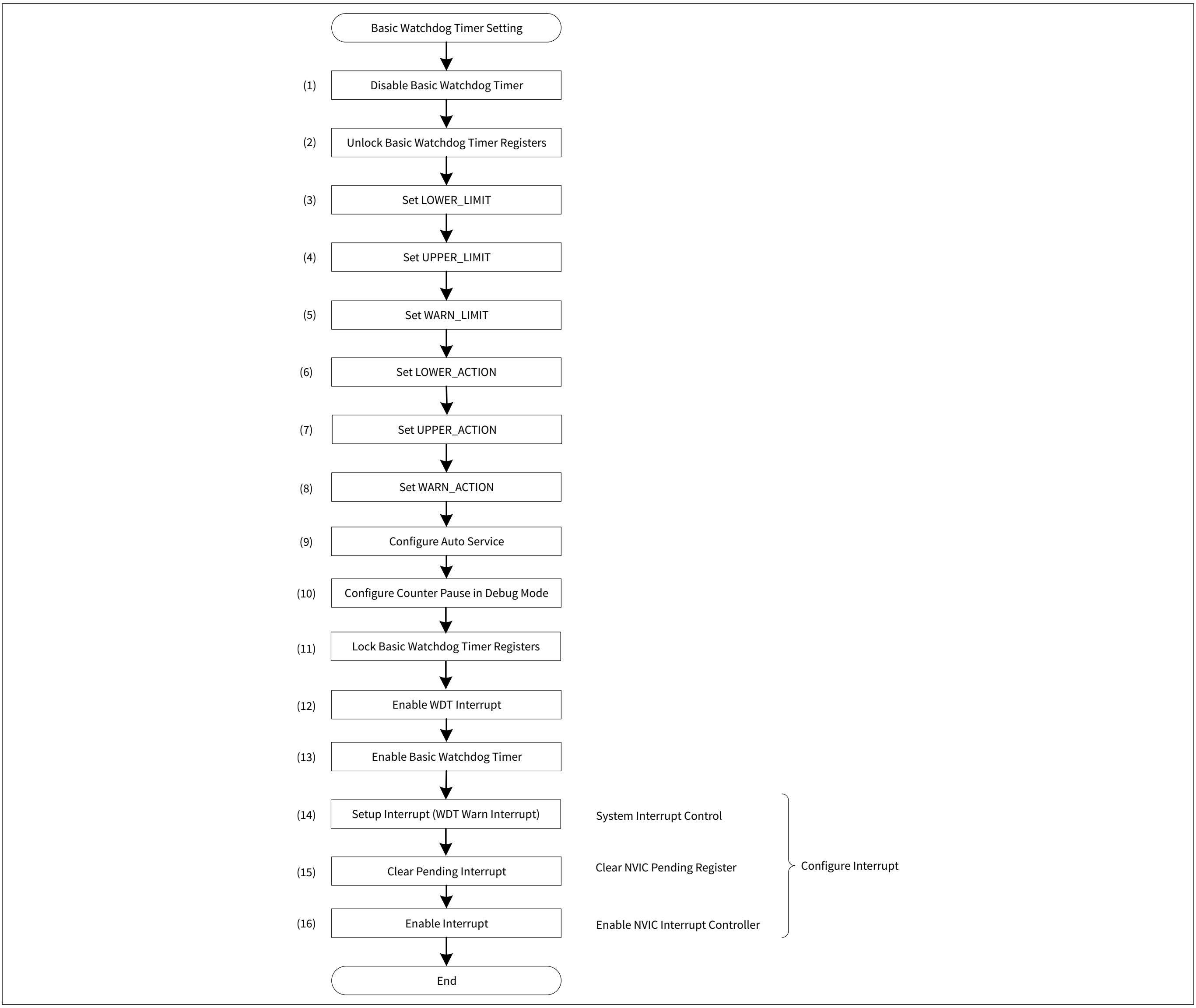
Basic Watchdog Timer Setting
(1) Disable Basic Watchdog Timer
(2) Unlock Basic Watchdog Timer Registers
(3) Set LOWER_LIMIT
(4) Set UPPER_LIMIT
(5) Set WARN_LIMIT
(6) Set LOWER_ACTION
(7) Set UPPER_ACTION
(8) Set WARN_ACTION
(9) Configure Auto Service
(10) Configure Counter Pause in Debug Mode
(11) Lock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

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 (IRQ number: 2)
- 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 SDL for basic WDT.

Table 1 List of Basic WDT Parameters

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

`Cy_WDT_SetLowerAction()`	Set 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()`	Set 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()`	Set 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()`	Configure 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()`	Set 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 AN219842 listed in Related documents.

Code Listing 1 Example of basic WDT configuration

```
cy_stc_sysint_irq_t stc_sysint_irq_cfg_wdt =
{
    .sysIntSrc = srss_interrupt_wdt_IRQn,
    .intIdx    = CPUIntIdx2_IRQn,
    .isEnabled = true,
};

int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

    /*-----------------------*/
    /* Configuration for WDT */
    /*-----------------------*/
    Cy_WDT_Disable();    /* (1) Disable Basic WDT */

    Cy_WDT_Unlock();    /* (2) Unlock Basic WDT registers */

    Cy_WDT_SetLowerLimit(4096ul);    /* (3) Set LOWER_LIMIT */

    Cy_WDT_SetUpperLimit(32768ul);    /* (4) Set UPPER_LIMIT */

    Cy_WDT_SetWarnLimit (28672ul);    /* (5) Set WARN_LIMIT */

    Cy_WDT_SetLowerAction(CY_WDT_LOW_UPP_ACTION_RESET);    /* (6) Set LOWER_ACTION */

    Cy_WDT_SetUpperAction(CY_WDT_LOW_UPP_ACTION_RESET);    /* (7) Set UPPER_ACTION */

    Cy_WDT_SetWarnAction (CY_WDT_WARN_ACTION_INT);    /* (8) Set WARN_ACTION */

    Cy_WDT_SetAutoService(CY_WDT_DISABLE);    /* (9) Disable Auto Service */

    Cy_WDT_SetDebugRun(CY_WDT_ENABLE);    /* (10) Enable counter pause in debug mode */

    Cy_WDT_Lock();    /* (11) Lock Basic WDT registers */

    Cy_WDT_MaskInterrupt();    /* (12) Enable Interrupt */

    Cy_WDT_Enable();    /* (13) Enable Basic WDT */

    /*----------------------------------*/
    /* Interrupt Configuration for WDT  */
    /*----------------------------------*/
    Cy_SysInt_InitIRQ(&stc_sysint_irq_cfg_wdt);   /*(14) Setup Interrupt (WDT Warn Interrupt)*/
    Cy_SysInt_SetSystemIrqVector(stc_sysint_irq_cfg_wdt.sysIntSrc, Wdt_Warn_IntrISR);

    NVIC_ClearPendingIRQ(stc_sysint_irq_cfg_wdt.intIdx);    /* (15) Clear Pending Interrupt */
    NVIC_EnableIRQ(stc_sysint_irq_cfg_wdt.intIdx);    /* (16) Enable Interrupt */
    for(;;);
}
```

2.7.3 Example program to configure basic WDT in driver part

Code Listing 2 to Code Listing 14show the example programs to configure the basic WDT in the driver part.

The following description will help you understand the register notation of the driver part of SDL:

- **WDT**->un***CTL***.stcField.ul***ENABLE*** is the WDT_CTL.ENABLE register mentioned in the Registers TRM. Other registers are also described in the same manner.

Code Listing 2 Example to disable basic WDT in driver part

```
void Cy_WDT_Disable(void)
{
    Cy_WDT_Unlock();
    /*  (1) Disable Basic WDT. WDT should be unlocked before being disabled.  */
    WDT->unCTL.stcField.u1ENABLE = 0ul;
    Cy_WDT_Lock();
}
```

Code Listing 3 Example to unlock basic WDT in driver part

```
void Cy_WDT_Unlock(void)
{
    uint32_t interruptState;
    interruptState = Cy_SysLib_EnterCriticalSection();

    /* The WDT lock is to be removed by two writes */
    /*  (2) Unlock Basic WDT registers when interrupts are disabled */
    WDT->unLOCK.stcField.u2WDT_LOCK = 1ul;
    WDT->unLOCK.stcField.u2WDT_LOCK = 2ul;

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 4 Example to set lower limit in driver part

```
__STATIC_INLINE void Cy_WDT_SetLowerLimit(uint32_t match)
{
    WDT->unLOWER_LIMIT.stcField.u32LOWER_LIMIT = match;   /*  (3) Set LOWER_LIMIT */
}
```

Code Listing 5 Example to set upper limit in driver part

```
__STATIC_INLINE void Cy_WDT_SetUpperLimit(uint32_t match)
{
    WDT->unUPPER_LIMIT.stcField.u32UPPER_LIMIT = match;   /*  (4) Set UPPER_LIMIT */
}
```

Code Listing 6 Example to set warn limit in driver part

```
__STATIC_INLINE void Cy_WDT_SetWarnLimit(uint32_t match)
{
    WDT->unWARN_LIMIT.stcField.u32WARN_LIMIT = match;   /*  (5) Set WARN_LIMIT */
}
```

Code Listing 7 Example to set lower action in driver part

```
   typedef enum
{
    CY_WDT_LOW_UPP_ACTION_NONE,
    CY_WDT_LOW_UPP_ACTION_RESET
} cy_en_wdt_lower_upper_action_t;

__STATIC_INLINE void Cy_WDT_SetLowerAction(cy_en_wdt_lower_upper_action_t action)
{
    WDT->unCONFIG.stcField.u1LOWER_ACTION = action;   /*  (6) Set LOWER_ACTION */
}
```

Code Listing 8 Example to set upper action in driver part

```
__STATIC_INLINE void Cy_WDT_SetUpperAction(cy_en_wdt_lower_upper_action_t action)
{
    WDT->unCONFIG.stcField.u1UPPER_ACTION = action;   /*  (7) Set UPPER_ACTION */
}
```

Code Listing 9 Example to set warn action in driver part

```
typedef enum
{
    CY_WDT_WARN_ACTION_NONE,
    CY_WDT_WARN_ACTION_INT
} cy_en_wdt_warn_action_t;

__STATIC_INLINE void Cy_WDT_SetWarnAction(cy_en_wdt_warn_action_t action)   /*  (8) Set
WARN_ACTION */
{
    WDT->unCONFIG.stcField.u1WARN_ACTION = action;
}
```

Code Listing 10 Example to configure auto service in driver part

```
typedef enum
{
    CY_WDT_DISABLE,
    CY_WDT_ENABLE
} cy_en_wdt_enable_t;

__STATIC_INLINE void Cy_WDT_SetAutoService(cy_en_wdt_enable_t enable)
{
    WDT->unCONFIG.stcField.u1AUTO_SERVICE = enable;   /*  (9) Configure Auto Service */
}
```

Code Listing 11 Example to set debugger configuration in driver part

```
__STATIC_INLINE void Cy_WDT_SetDebugRun(cy_en_wdt_enable_t enable)
{
    WDT->unCONFIG.stcField.u1DEBUG_RUN = enable;   /8  (10) Set Debugger Configuration */
}
```

Code Listing 12 Example to lock basic WDT in driver part

```
void Cy_WDT_Lock(void)
{
    uint32_t interruptState;
    interruptState = Cy_SysLib_EnterCriticalSection();

    WDT->unLOCK.stcField.u2WDT_LOCK = 3ul;   /*  (11) Lock Basic WDT registers during
interrupts disabled */

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 13 Example to enable WDT interrupt in driver part

```
__STATIC_INLINE void Cy_WDT_MaskInterrupt(void)
{
    WDT->unINTR_MASK.stcField.u1WDT = 1ul;   /*  (12) Enable WDT Interrupt- */
}
```

Code Listing 14 Example to enable basic WDT in driver part

```
void Cy_WDT_Enable(void)
{
    Cy_WDT_Unlock();
    WDT->unCTL.stcField.u1ENABLE = 1ul;   /*  (13) Enable Basic WDT during WDT unlocked */
    Cy_WDT_Lock();
}
```

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, run time 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 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:

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

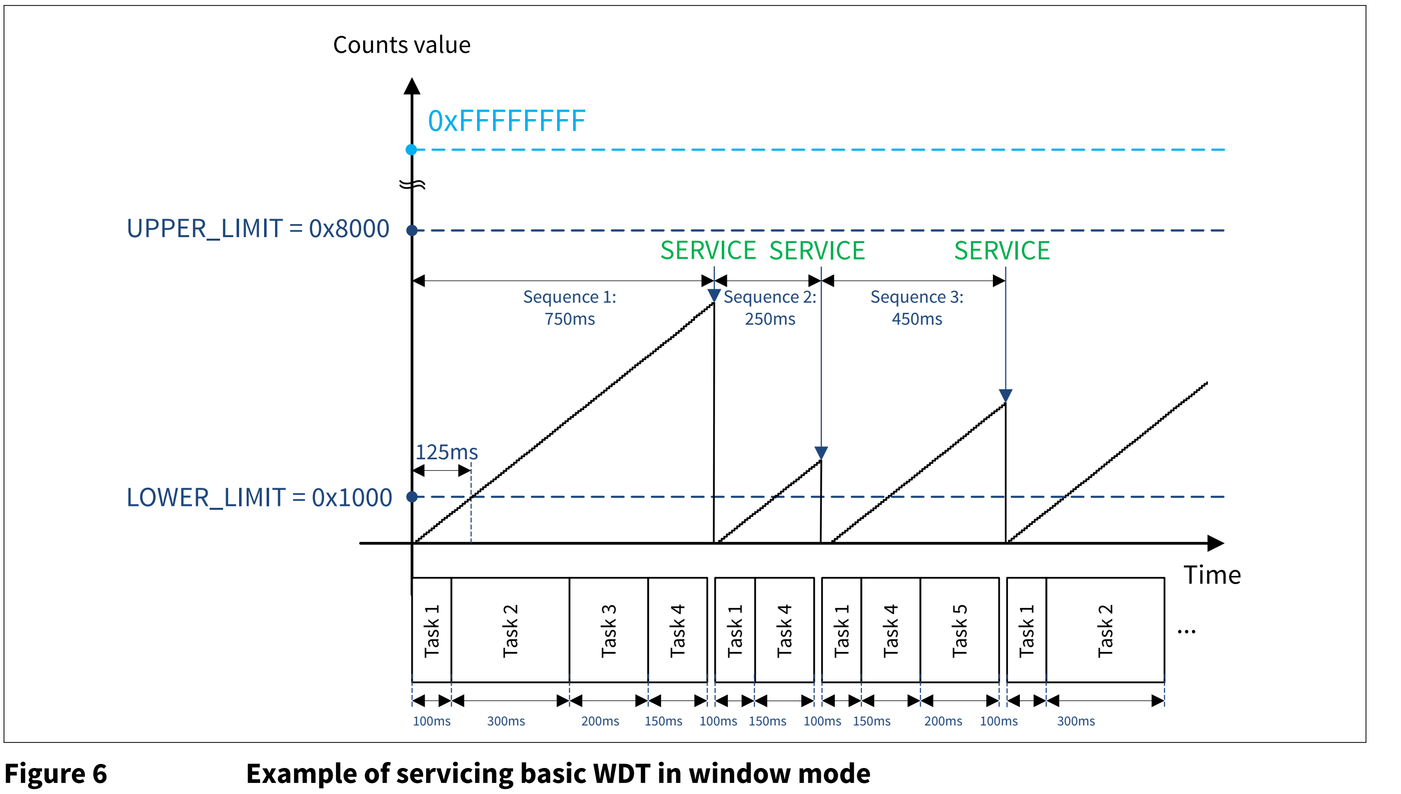

Figure 6 **Example of servicing basic WDT in window mode**

2.8.1 Use case

This section describes an example of clearing the basic WDT using the use case discussed in Chapter 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.

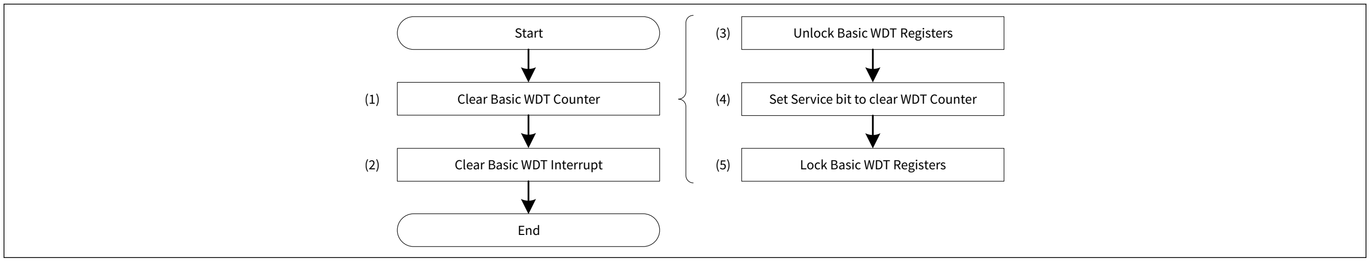
Start
(1) Clear Basic WDT Counter
(2) Clear Basic WDT Interrupt
End
(3) Unlock Basic WDT Registers
(4) Set Service bit to clear WDT Counter
(5) Lock Basic WDT Registers

Figure 7 **Example flow to clear basic WDT**

2.8.3 Example program to clear the basic WDT

Code Listing 15 shows an example program to clear the basic WDT.

Code Listing 15 Example program to clear basic WDT

```
void Wdt_Warn_IntrISR(void)
{
    Cy_WDT_ClearWatchdog();    /* (1) Clear Basic WDT Counter */
    Cy_WDT_ClearInterrupt();    /* (2) Clear Basic WDT Interrupt */
}
```

Code Listing 16 shows an example program to clear the basic WDT in the driver part.

Code Listing 16 Example program to clear basic WDT in driver part

```
void Cy_WDT_ClearWatchdog(void)
{
    Cy_WDT_Unlock();    /* (3) Unlock Basic WDT Registers */
    Cy_WDT_SetService();
    Cy_WDT_Lock();    /* (5) Unlock Basic WDT Registers */
}

__STATIC_INLINE void Cy_WDT_SetService()
{
    WDT->unSERVICE.stcField.u1SERVICE = 1ul;    /* (4) Set Service bit to clear Basic WDT
Counter */
}
```

Code Listing 17 shows an example program to clear the basic WDT interrupt in the driver part.

Code Listing 17 Example program to clear basic WDT interrupt in driver part

```
void Cy_WDT_ClearInterrupt(void)
{
    WDT->unINTR.stcField.u1WDT = 1ul;    /* (2) Clear Basic WDT Interrupt */

    /* Read the interrupt register to ensure that the initial clearing write has
    * been flushed out to the hardware.
    */
    (void) SRSS->unSRSS_INTR;
}
```

2.9 Reset cause indication for 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 that 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_SELECT	Clock selection register
CLK_ILO_CONFIG	ILO 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, that behave like the basic WDT. Window mode and prewarning 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, that 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.

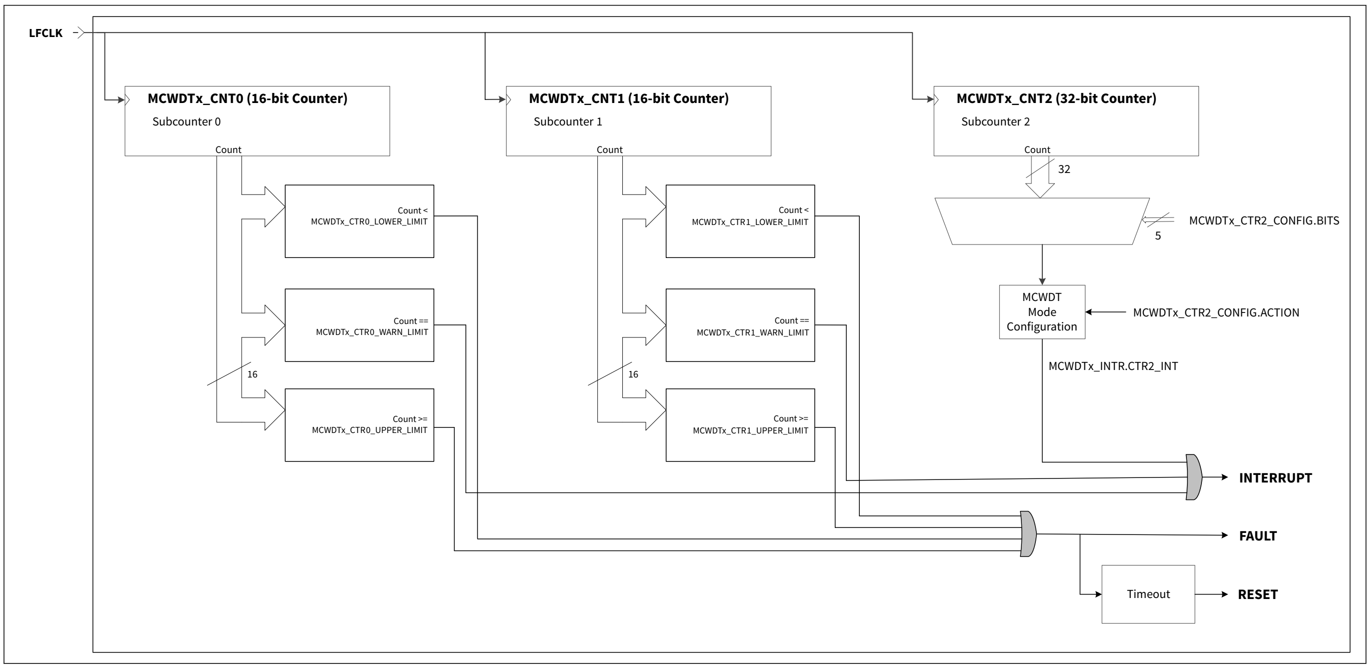
LFCLK
MCWDTx_CNT0 (16-bit Counter)
Subcounter 0
Count
MCWDTx_CNT1 (16-bit Counter)
Subcounter 1
Count
MCWDTx_CNT2 (32-bit Counter)
Subcounter 2
Count
32
Count < MCWDTx_CTR0_LOWER_LIMIT
Count == MCWDTx_CTR0_WARN_LIMIT
Count >= MCWDTx_CTR0_UPPER_LIMIT
16
Count < MCWDTx_CTR1_LOWER_LIMIT
Count == MCWDTx_CTR1_WARN_LIMIT
Count >= MCWDTx_CTR1_UPPER_LIMIT
16
MCWDTx_CTR2_CONFIG.BITS
5
MCWDT Mode Configuration
MCWDTx_CTR2_CONFIG.ACTION
MCWDTx_INTR.CTR2_INT
INTERRUPT
FAULT
Timeout
RESET

Figure 8 **Multi-counter WDT block diagram**

3.1 Source clock

The source clock that can be selected for MCWDT is LFCLK, that can be one of 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 (4–33.33 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 similar to the pre-warning interrupt of the basic WDT. See Chapter 2.4 Warning interrupt. The only difference is that the WARN_LIMIT is a 16-bit value that can generate an interrupt timing per the following equation:

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

Figure 9

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:

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

Figure 10

Example:

LFCLK = ILO0 = 32.768 kHz

Toggle-Bit = Bit 12

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

Figure 11

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 Chapter 2.5 Timeout mode. 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 Chapter 2.6 Window mode. 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 12, 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

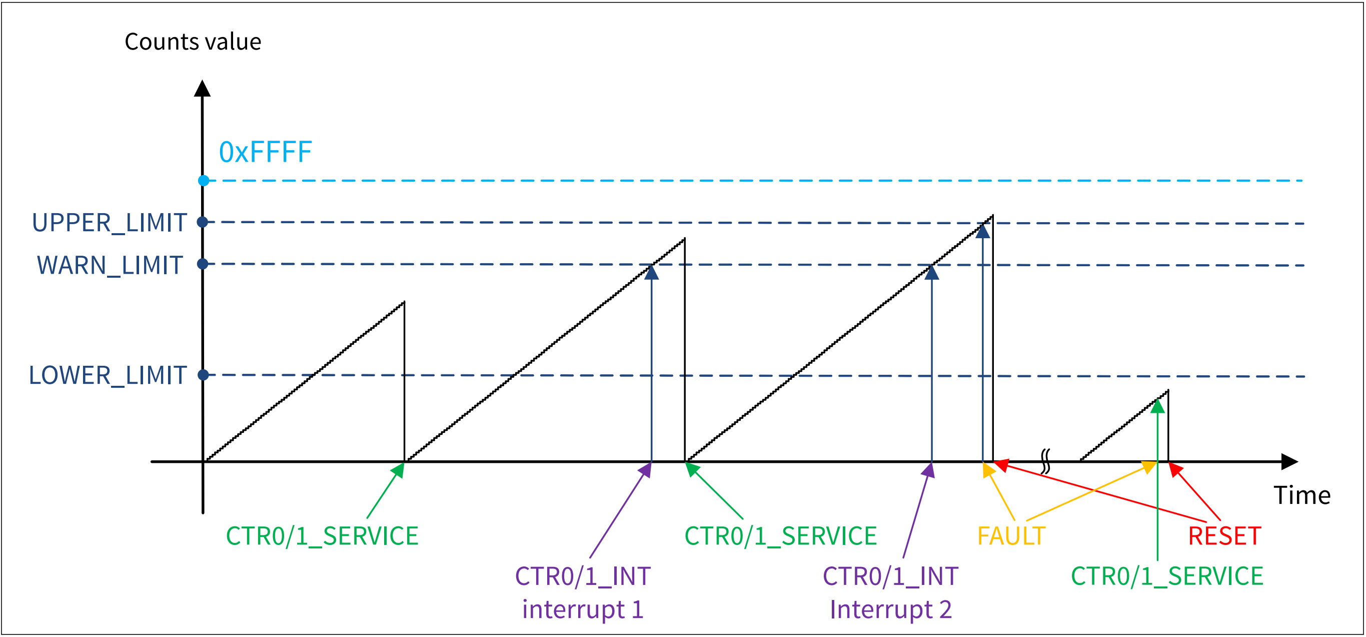

Figure 12 **Subcounter 0/1 operation in window mode with FAULT and RESET action**

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 CPUs

CPU_SEL[0:4]	CYT2 CPU	CYT3 CPU	CYT4 CPU	CYT6 CPU
0	CM0+	CM0+	CM0+	CM0+
1	CM4	CM7-0	CM7-0	CM7-0
2	-	-	CM7-1	CM7-1
3	-	-	-	CM7-2

***Note**: CYT6BJ has CM7_3 core. But CM7_3 is not connected to any MCWDT.*

3.7 MCWDT settings

Figure 13 illustrates an example flow to configure the MCWDT.

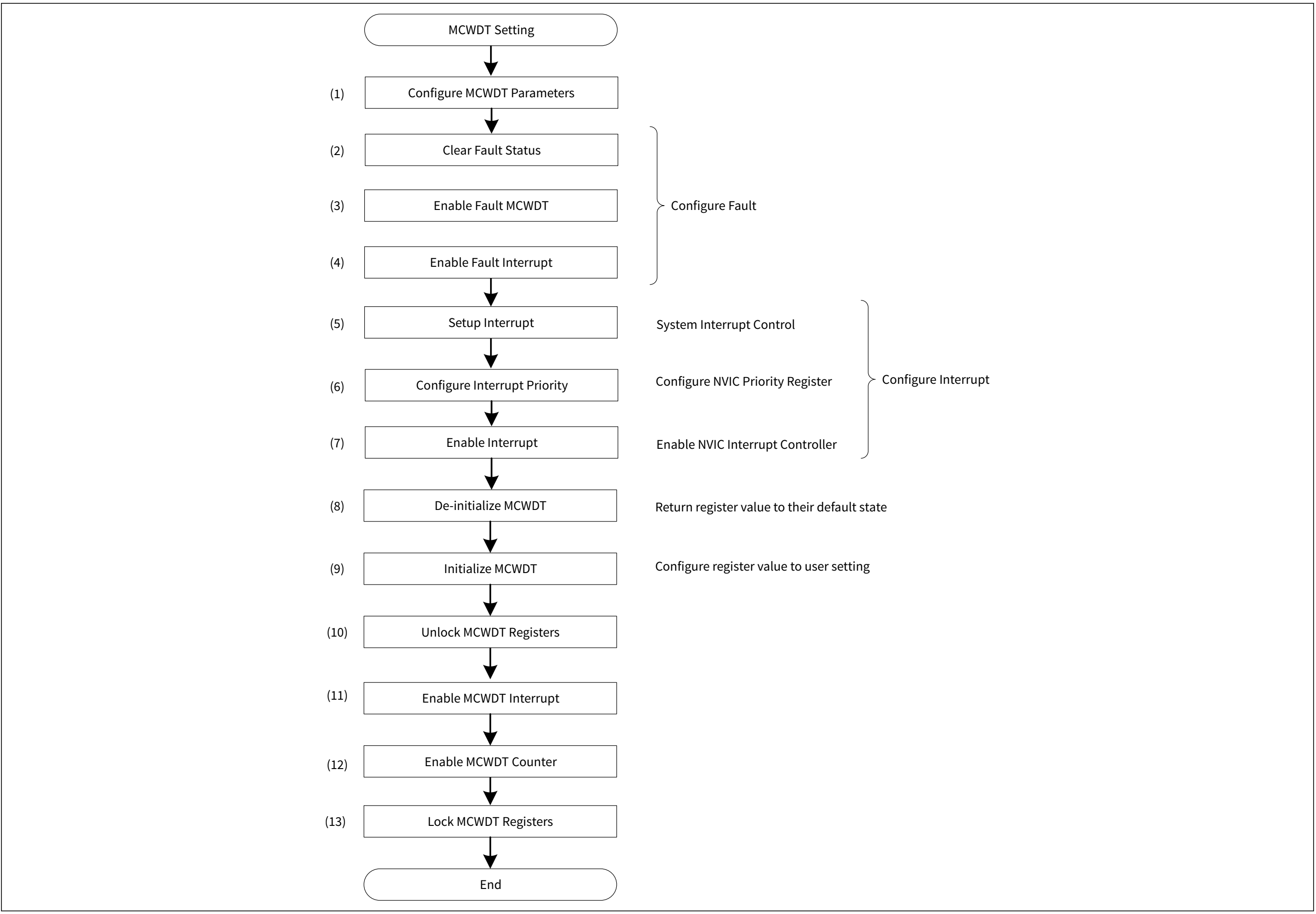
MCWDT Setting
(1) Configure MCWDT Parameters
(2) Clear Fault Status
(3) Enable Fault MCWDT
(4) Enable Fault Interrupt
Configure Fault
(5) Setup Interrupt
System Interrupt Control
(6) Configure Interrupt Priority
Configure NVIC Priority Register
(7) Enable Interrupt
Enable NVIC Interrupt Controller
Configure Interrupt
(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 13 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: 1
- CPU: CM4
- Subcounter 0
  - LOWER_LIMIT: Unused
  - UPPER_LIMIT: 1 second
  - WARN_LIMIT: Unused
  - Window mode: Unused
  - Upper limit action: Fault interrupt (IRQ number: 2)
  - Auto service: Unused
  - Debugger configuration: Enables the trigger input for MCWDT to pause the counter during debug mode
- Subcounter 1: Unused
- Subcounter 2: Unused
- Fault report: Fault structure 1

3.7.2 Configuring the MCWDT

Table 4 lists the parameters of the configuration part in SDL for MCWDT.

Table 4 List of MCWDT parameters

Parameters	Description	Value

`.coreSelect`	Select the CPU to be used for SleepDeepPause
	CY_MCWDT_PAUSED_BY_DPSLP_CM0 = 0ul
	CY_MCWDT_PAUSED_BY_DPSLP_CM4_CM7_0 = 1ul
	CY_MCWDT_PAUSED_BY_DPSLP_CM7_1 = 2ul
	CY_MCWDT_PAUSED_BY_NO_CORE = 3ul
		CY_MCWDT_PAUSED_BY_DPSLP_CM4_CM7_0

`.c0LowerLimit`	Set the Subcounter 0 lower limit (unsigned integer 32-bit)	0ul
`.c0UpperLimit`	Set the Subcounter 0 upper limit (unsigned integer 32-bit)	32768ul
`.c0WarnLimit`	Set the Subcounter 0 warn limit (unsigned integer 32-bit)	0ul

`.c0LowerAction`	Set Subcounter 0 lower action to “no action”, “fault”, or “fault then reset”:
	CY_MCWDT_ACTION_NONE = 0ul
	CY_MCWDT_ACTION_FAULT = 1ul
	CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul
		CY_MCWDT_ACTION_NONE

(table continues...)

Table 4 **(continued) List of MCWDT parameters**

Parameters	Description	Value

`.c0UpperAction`	Set Subcounter 0 upper action to "no action", "fault", or "fault then reset":
	CY_MCWDT_ACTION_NONE = 0ul
	CY_MCWDT_ACTION_FAULT = 1ul
	CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul
		CY_MCWDT_ACTION_FAULT

`.c0WarnAction`	Set Subcounter 0 warn action to "no action", or "interrupt":
	CY_MCWDT_WARN_ACTION_NONE = 0ul
	CY_MCWDT_WARN_ACTION_INT = 1ul
		CY_MCWDT_WARN_ACTION_NONE

`.c0AutoService`	Configure to automatically clear MCWDT when Subcounter 0 value reaches WARN_LIMIT:
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_DISABLE

`.c0SleepDeepPause`	Enable to pause Subcounter 0 when the corresponding CPU is in DeepSleep:
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_ENABLE

`.c0DebugRun`	Set the debugger configuration. It needs when using debugger.
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_ENABLE

`.c1LowerLimit`	Set Subcounter 1 lower limit (unsigned integer 32-bit)	0ul
`.c1UpperLimit`	Set Subcounter 1 upper limit (unsigned integer 32-bit)	0ul
`.c1WarnLimit`	Set Subcounter 1 warn limit (unsigned integer 32-bit)	0ul

`.c1LowerAction`	Set Subcounter 1 lower action to "no action", "fault", or "fault then reset":
	CY_MCWDT_ACTION_NONE = 0ul
	CY_MCWDT_ACTION_FAULT = 1ul
	CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul
		CY_MCWDT_ACTION_NONE

`.c1UpperAction`	Set Subcounter 1 upper action to "no action", "fault", or "fault then reset":
	CY_MCWDT_ACTION_NONE = 0ul
	CY_MCWDT_ACTION_FAULT = 1ul
	CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul
		CY_MCWDT_ACTION_NONE

(table continues...)

Table 4 **(continued) List of MCWDT parameters**

Parameters	Description	Value

`.c1WarnAction`	Set Subcounter 1 warn action to "no action", or "interrupt":
	CY_MCWDT_WARN_ACTION_NONE = 0ul
	CY_MCWDT_WARN_ACTION_INT = 1ul
		CY_MCWDT_WARN_ACTION_NONE

`.c1AutoService`	Configure to automatically clear MCWDT when Subcounter 1 value reaches WARN_LIMIT:
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_DISABLE

`.c1SleepDeepPause`	Enable to pause Subcounter 1 when the corresponding CPU is in DeepSleep:
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_DISABLE

`.c1DebugRun`	Set the debugger configuration (required when using debugger)
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_DISABLE

(table continues...)

Table 4 **(continued) List of MCWDT parameters**

Parameters	Description	Value

`.c2ToggleBit`	Select the bit to observe for a toggle:
	CY_MCWDT_CNT2_MONITORED_BIT0 = 0ul
	CY_MCWDT_CNT2_MONITORED_BIT1 = 1ul
	CY_MCWDT_CNT2_MONITORED_BIT2 = 2ul
	CY_MCWDT_CNT2_MONITORED_BIT3 = 3ul
	CY_MCWDT_CNT2_MONITORED_BIT4 = 4ul
	CY_MCWDT_CNT2_MONITORED_BIT5 = 5ul
	CY_MCWDT_CNT2_MONITORED_BIT6 = 6ul
	CY_MCWDT_CNT2_MONITORED_BIT7 = 7ul
	CY_MCWDT_CNT2_MONITORED_BIT8 = 8ul
	CY_MCWDT_CNT2_MONITORED_BIT9 = 9ul
	CY_MCWDT_CNT2_MONITORED_BIT10 = 10ul
	CY_MCWDT_CNT2_MONITORED_BIT11 = 11ul
	CY_MCWDT_CNT2_MONITORED_BIT12 = 12ul
	CY_MCWDT_CNT2_MONITORED_BIT13 = 13ul
	CY_MCWDT_CNT2_MONITORED_BIT14 = 14ul
	CY_MCWDT_CNT2_MONITORED_BIT15 = 15ul
	CY_MCWDT_CNT2_MONITORED_BIT16 = 16ul
	CY_MCWDT_CNT2_MONITORED_BIT17 = 17ul
	CY_MCWDT_CNT2_MONITORED_BIT18 = 18ul
	CY_MCWDT_CNT2_MONITORED_BIT19 = 19ul
	CY_MCWDT_CNT2_MONITORED_BIT20 = 20ul
	CY_MCWDT_CNT2_MONITORED_BIT21 = 21ul
	CY_MCWDT_CNT2_MONITORED_BIT22 = 22ul
	CY_MCWDT_CNT2_MONITORED_BIT23 = 23ul
	CY_MCWDT_CNT2_MONITORED_BIT24 = 24ul
	CY_MCWDT_CNT2_MONITORED_BIT25 = 25ul
	CY_MCWDT_CNT2_MONITORED_BIT26 = 26ul
	CY_MCWDT_CNT2_MONITORED_BIT27 = 27ul
	CY_MCWDT_CNT2_MONITORED_BIT28 = 28ul
	CY_MCWDT_CNT2_MONITORED_BIT29 = 29ul
	CY_MCWDT_CNT2_MONITORED_BIT30 = 30ul
	CY_MCWDT_CNT2_MONITORED_BIT31 = 31ul
		CY_MCWDT_CNT2_MONITORED_BIT0

`.c2Action`	Set Subcounter 2 action to “no action” or “interrupt”:
	CY_MCWDT_CNT2_ACTION_NONE = 0ul
	CY_MCWDT_CNT2_ACTION_INT = 1ul
		CY_MCWDT_CNT2_ACTION_NONE

(table continues...)

Table 4 (continued) List of MCWDT parameters

Parameters	Description	Value

`.c2SleepDeepPause`	Enable to pause Subcounter 2 when the corresponding CPU is in DeepSleep:
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_DISABLE

`.c2DebugRun`	Set the debugger configuration (required when using debugger)
	CY_MCWDT_DISABLE = 0ul
	CY_MCWDT_ENABLE = 1ul
		CY_MCWDT_DISABLE

Code Listing 18 shows an example program of the MCWDT configuration part. For details of the interrupt and fault initial setting procedure, see the 'Interrupt and Fault Report Structure Section in AN219842 listed in Related documents.

Code Listing 18 Example program to configure MCWDT

```
cy_stc_sysint_irq_t irq_cfg =
{
    .sysIntSrc  = cpuss_interrupts_fault_1_IRQn,
    .intIdx     = CPUIntIdx2_IRQn,
    .isEnabled  = true,
};
cy_stc_mcwdt_config_t mcwdtConfig =    /*  (1) Configure MCWDT Parameters  */
{
    .coreSelect       = CY_MCWDT_PAUSED_BY_DPSLP_CM4_CM7_0,    /*  Select CPU to be used for
SleepDeepPause.  */
    .c0LowerLimit     = 0,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0UpperLimit     = 32768,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0WarnLimit      = 0,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0LowerAction    = CY_MCWDT_ACTION_NONE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0UpperAction    = CY_MCWDT_ACTION_FAULT,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0WarnAction     = CY_MCWDT_WARN_ACTION_NONE,    /*  Configure WDT Subcounter 0
Parameters  */
    .c0AutoService    = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0SleepDeepPause = CY_MCWDT_ENABLE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0DebugRun       = CY_MCWDT_ENABLE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c1LowerLimit     = 0,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1UpperLimit     = 0,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1WarnLimit      = 0,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1LowerAction    = CY_MCWDT_ACTION_NONE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1UpperAction    = CY_MCWDT_ACTION_NONE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1WarnAction     = CY_MCWDT_WARN_ACTION_NONE,    /*  Configure WDT Subcounter 1
Parameters  */
    .c1AutoService    = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1SleepDeepPause = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1DebugRun       = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c2ToggleBit      = CY_MCWDT_CNT2_MONITORED_BIT0,    /*  Configure WDT Subcounter 2
Parameters  */
    .c2Action         = CY_MCWDT_CNT2_ACTION_NONE,    /*  Configure WDT Subcounter 2
Parameters  */
    .c2SleepDeepPause = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 2 Parameters  */
    .c2DebugRun       = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 2 Parameters  */
};
int main(void)
{
    SystemInit();
    __enable_irq(); /* Enable global interrupts. */
    /*********************************************************************/
    /*****                 Fault report settings                    *****/
    /*********************************************************************/
    Cy_SysFlt_ClearStatus(FAULT_STRUCT1);    /* (2) Clear Fault Status. */
    Cy_SysFlt_SetMaskByIdx(FAULT_STRUCT1, CY_SYSFLT_SRSS_MCWDT1);    /* (3) Enable Fault MCWDT.
*/
    Cy_SysFlt_SetInterruptMask(FAULT_STRUCT1);    /* (4) Enable Fault Interrupt. */
    /*********************************************************************/
    /*****                   Interrupt setting                      *****/
    /********************************************************************/
    Cy_SysInt_InitIRQ(&irq_cfg);    /* (5) Setup Interrupt. */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, irqFaultReport1Handler);
    NVIC_SetPriority(CPUIntIdx2_IRQn, 0);    /* (6) Configure Interrupt Priority. */
    NVIC_EnableIRQ(CPUIntIdx2_IRQn);    /* (7) Enable Interrupt. */
    /********************************************************************/
    /*****                 Configuration for MCWDT                 *****/
    /********************************************************************/
    Cy_MCWDT_DeInit(MCWDT1);    /* (8) De-initialize MCWDT. */
    Cy_MCWDT_Init(MCWDT1, &mcwdtConfig);    /* (9) Initialize MCWDT. */
    Cy_MCWDT_Unlock(MCWDT1);    /* (10) Unlock MCWDT. */
    Cy_MCWDT_SetInterruptMask(MCWDT1, CY_MCWDT_CTR0);    /* (11) Enable MCWDT Interrupt. */
    Cy_MCWDT_Enable(MCWDT1,
                    CY_MCWDT_CTR0,    /* (12) Enable MCWDT Counter. */
                    0);
    Cy_MCWDT_Lock(MCWDT1);    /* (13) Unlock MCWDT. */
    for(;;)
    {
    :
    }
}
```

3.7.3 Example program to configure the MCWDT in driver part

Code Listing 19 to Code Listing 24 shows an example program to configure the MCWDT in the driver part.

The following description will help you understand the register notation of the driver part of SDL:

- Base signifies the pointer to the MCWDT register base address. counters specifies the Subcounter within the MCWDT. See Table 5
- To improve the performance of the register setting procedure, the SDL writes a complete 32-bit data to register. Each bit field is generated in advance in a bit-writable buffer and written to the register as the final 32-bit data

```
tempCNT2ConfigParams.stcField.u5BITS    = config->c2ToggleBit;
tempCNT2ConfigParams.stcField.u1ACTION                 = config->c2Action;
tempCNT2ConfigParams.stcField.u1SLEEPDEEP_PAUSE        = config->c2SleepDeepPause;
tempCNT2ConfigParams.stcField.u1DEBUG_RUN              = config->c2DebugRun;
base->unCTR2_CONFIG.u32Register                        = tempCNT2ConfigParams.u32Register;
```

See `cyip_srss_v2.h` under `hdr/rev_x/ip` for more information on the union and structure representation of registers.

Table 5 List of MCWDT parameters in driver part

Parameters	Description	Value
base	Specify the MCWDT number to configure its registers: MCWDT0 MCWDT1 MCWDT2 (only for CYT4) MCWDT3 (only for CYT6)	MCWDT1

(table continues...)

Table 5 (continued) List of MCWDT parameters in driver part

Parameters	Description	Value
counters	Specify the Subcounter to configure its registers: CY_MCWDT_CTR0: Subcounter 0 CY_MCWDT_CTR1: Subcounter 1 CY_MCWDT_CTR2: Subcounter 2 CY_MCWDT_CTR_Msk: All Subcounters	CY_MCWDT_CTR0

Code Listing 19 Example program to deinitialize MCWDT in driver part

```
/*  (8) De-initializes the MCWDT block, returns register values to their default state. */
void Cy_MCWDT_DeInit(volatile stc_MCWDT_t *base)
{
    Cy_MCWDT_Unlock(base);   /* Unlock MCWDT Registers */

    // disable all counter
    for(uint32_t loop = 0ul; loop < CY_MCWDT_NUM_OF_SUBCOUNTER; loop++)
    {
        base->CTR[loop].unCTL.u32Register = 0ul;
    }
    base->unCTR2_CTL.u32Register    = 0ul;

    for(uint32_t loop = 0ul; loop < CY_MCWDT_NUM_OF_SUBCOUNTER; loop++)
    {
        while(base->CTR[loop].unCTL.u32Register != 0x0ul); // wait until enabled bit become 1
        base->CTR[loop].unLOWER_LIMIT.u32Register = 0x0ul;
        base->CTR[loop].unUPPER_LIMIT.u32Register = 0x0ul;
        base->CTR[loop].unWARN_LIMIT.u32Register  = 0x0ul;
        base->CTR[loop].unCONFIG.u32Register      = 0x0ul;
        base->CTR[loop].unCNT.u32Register         = 0x0ul;
    }

    while(base->unCTR2_CNT.u32Register != 0ul); // wait until enabled bit become 1
    base->unCPU_SELECT.u32Register  = 0ul;
    base->unCTR2_CONFIG.u32Register = 0ul;
    base->unSERVICE.u32Register     = 0x00000003ul;
    base->unINTR.u32Register        = 0xFFFFFFFFul;
    base->unINTR_MASK.u32Register   = 0ul;

    Cy_MCWDT_Lock(base);   /* Lock MCWDT Registers */
}
```

Code Listing 20 Example program to initialize MCWDT in driver part

```
/*  (9) Initializes the MCWDT block according to the MCWDT configuration  */
cy_en_mcwdt_status_t Cy_MCWDT_Init(volatile stc_MCWDT_t *base, cy_stc_mcwdt_config_t const
*config)
{
    cy_en_mcwdt_status_t ret = CY_MCWDT_BAD_PARAM;
    if ((base != NULL) && (config != NULL))   /*  Validate configuration parameter */
    {
        Cy_MCWDT_Unlock(base);   /*  Unlock MCWDT Registers */
        un_MCWDT_CTR_CONFIG_t  tempConfigParams     = { 0ul };
        un_MCWDT_CTR2_CONFIG_t tempCNT2ConfigParams = { 0ul };

       /*  Configure CPU to be used for SLEEPDEEP_PAUSE.  */
        base->unCPU_SELECT.u32Register                     = config->coreSelect;


       /*  Configure Subcounter 0  */
        base->CTR[0].unLOWER_LIMIT.stcField.u16LOWER_LIMIT = config->c0LowerLimit;
        base->CTR[0].unUPPER_LIMIT.stcField.u16UPPER_LIMIT = config->c0UpperLimit;
        base->CTR[0].unWARN_LIMIT.stcField.u16WARN_LIMIT   = config->c0WarnLimit;
        tempConfigParams.stcField.u2LOWER_ACTION           = config->c0LowerAction;
        tempConfigParams.stcField.u2LOWER_ACTION           = config->c0LowerAction;
        tempConfigParams.stcField.u2UPPER_ACTION           = config->c0UpperAction;
        tempConfigParams.stcField.u1WARN_ACTION            = config->c0WarnAction;
        tempConfigParams.stcField.u1AUTO_SERVICE           = config->c0AutoService;
        tempConfigParams.stcField.u1SLEEPDEEP_PAUSE        = config->c0SleepDeepPause;
        tempConfigParams.stcField.u1DEBUG_RUN              = config->c0DebugRun;
        base->CTR[0].unCONFIG.u32Register                  = tempConfigParams.u32Register;


       /*  Configure Subcounter 1.  */
        base->CTR[1].unLOWER_LIMIT.stcField.u16LOWER_LIMIT = config->c1LowerLimit;
        base->CTR[1].unUPPER_LIMIT.stcField.u16UPPER_LIMIT = config->c1UpperLimit;
        base->CTR[1].unWARN_LIMIT.stcField.u16WARN_LIMIT   = config->c1WarnLimit;
        tempConfigParams.stcField.u2LOWER_ACTION           = config->c1LowerAction;
        tempConfigParams.stcField.u2UPPER_ACTION           = config->c1UpperAction;
        tempConfigParams.stcField.u1WARN_ACTION            = config->c1WarnAction;
        tempConfigParams.stcField.u1AUTO_SERVICE           = config->c1AutoService;
        tempConfigParams.stcField.u1SLEEPDEEP_PAUSE        = config->c1SleepDeepPause;
        tempConfigParams.stcField.u1DEBUG_RUN              = config->c1DebugRun;
        base->CTR[1].unCONFIG.u32Register                  = tempConfigParams.u32Register;


       /*  Configure Subcounter 2.  */
        tempCNT2ConfigParams.stcField.u5BITS             = config->c2ToggleBit;
        tempCNT2ConfigParams.stcField.u1ACTION           = config->c2Action;
        tempCNT2ConfigParams.stcField.u1SLEEPDEEP_PAUSE  = config->c2SleepDeepPause;
        tempCNT2ConfigParams.stcField.u1DEBUG_RUN        = config->c2DebugRun;
        base->unCTR2_CONFIG.u32Register                  = tempCNT2ConfigParams.u32Register;

        Cy_MCWDT_Lock(base);       /*  Lock MCWDT Registers */
```

```
            ret = CY_MCWDT_SUCCESS;
        }

        return (ret);
    }
```

Code Listing 21 Example program to unlock MCWDT registers in driver part

```
    #define CY_MCWDT_LOCK_CLR0      (1ul)
    #define CY_MCWDT_LOCK_CLR1      (2ul)
    __STATIC_INLINE void Cy_MCWDT_Unlock(volatile stc_MCWDT_t *base)
    {
        uint32_t interruptState;

        interruptState = Cy_SysLib_EnterCriticalSection();

       /*  (10) Unlock MCWDT Registers when Interrupts are disabled. */
        base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_CLR0;
        base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_CLR1;

        Cy_SysLib_ExitCriticalSection(interruptState);
    }
```

Code Listing 22 Example program to enable MCWDT interrupt in driver part

```
    __STATIC_INLINE void Cy_MCWDT_SetInterruptMask(volatile stc_MCWDT_t *base, uint32_t counters)
    {
        if (counters & CY_MCWDT_CTR0)
        {
            base->unINTR_MASK.stcField.u1CTR0_INT = 1ul;    /* (11) Enable the MCWDT Subcounter
    Interrupt.  */
        }
        if (counters & CY_MCWDT_CTR1)
        {
            base->unINTR_MASK.stcField.u1CTR1_INT = 1ul;
        }
        if (counters & CY_MCWDT_CTR2)
        {
            base->unINTR_MASK.stcField.u1CTR2_INT = 1ul;
        }
    }
```

Code Listing 23 Example program to enable MCWDT counter in driver part

```
__STATIC_INLINE void Cy_MCWDT_Enable(volatile stc_MCWDT_t *base, uint32_t counters, uint16_t
waitUs)
{
    if (counters & CY_MCWDT_CTR0)
    {
        base->CTR[0].unCTL.stcField.u1ENABLE = 1ul;   /*  (12) Enable the MCWDT Subcounter.  */
    }
    if (counters & CY_MCWDT_CTR1)
    {
        base->CTR[1].unCTL.stcField.u1ENABLE = 1ul;
    }
    if (counters & CY_MCWDT_CTR2)
    {
        base->unCTR2_CTL.stcField.u1ENABLE = 1ul;
    }
    Cy_SysLib_DelayUs(waitUs);
}
```

Code Listing 24 Example program to lock MCWDT registers in driver part

```
#define CY_MCWDT_LOCK_SET01      (3ul)
__STATIC_INLINE void Cy_MCWDT_Lock(volatile stc_MCWDT_t *base)
{
    uint32_t interruptState;

    interruptState = Cy_SysLib_EnterCriticalSection();

    base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_SET01;   /* (13) Lock MCWDT Registers
when interrupts are disabled.  */

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

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, 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. Refer to Figure 7 which shows an example 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 Use case.

3.8.2 Example flow to clear the MCWDT

Figure 14 shows an example flow to clear the MCWDT.

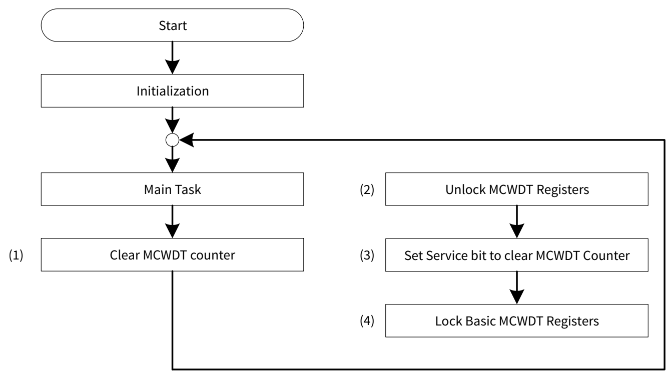

Figure 14 **Example flow to clear MCWDT**

3.8.3 Example program to clear the MCWDT

Code Listing 25 shows an example program to clear the MCWDT.

Code Listing 25 Example program to clear MCWDT

```
int main(void)
{
:
for(;;)
{
        :
Cy_MCWDT_ClearWatchdog(MCWDT1, CY_MCWDT_COUNTER0);   /*  (1) Clear the MCWDT counter. */
}
}
```

Code Listing 26 shows an example program to clear the MCWDT counter in the driver part.

Code Listing 26 Example program to clear MCWDT counter in driver part

```
void Cy_MCWDT_ClearWatchdog(volatile stc_MCWDT_t *base, cy_en_mcwdtctr_t counter)
{
    Cy_MCWDT_Unlock(base);   /*  (2) Unlock MCWDT Registers . */
    Cy_MCWDT_ResetCounters(base, (1u << (uint8_t)counter), 0u);
    Cy_MCWDT_Lock(base);   /*  (4) Lock MCWDT Registers . */
}

__STATIC_INLINE void Cy_MCWDT_ResetCounters(volatile stc_MCWDT_t *base, uint32_t counters,
uint16_t waitUs)
{
    if (counters & CY_MCWDT_CTR0)
    {
        base->unSERVICE.stcField.u1CTR0_SERVICE = 1ul;   /*  (3) Set the Service bit to clear
the MCWDT counter. */
    }
    if (counters & CY_MCWDT_CTR1)
    {
        base->unSERVICE.stcField.u1CTR1_SERVICE = 1ul;
    }
    if (counters & CY_MCWDT_CTR2)
    {
        // No reset functionality for CTR2
    }
    Cy_SysLib_DelayUs(waitUs);
}
```

3.9 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 13 and Code Listing 18. As an example, the fault structure 1 is used.

For details of the fault setting procedure, see the 'Fault Report Structure' section in AN219842 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.

3.9.1 Use case

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

3.9.2 Example flow of MCWDT fault handler

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

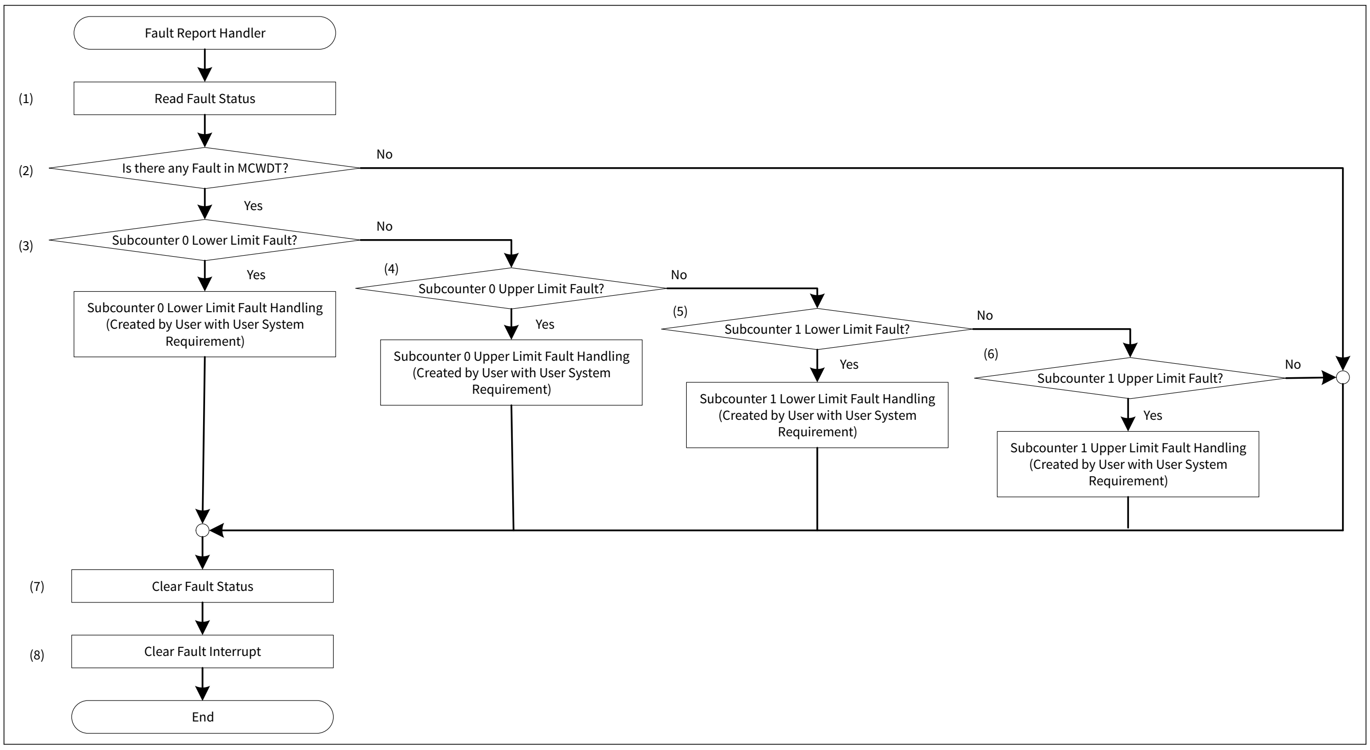
Fault Report Handler
(1)
Read Fault Status
(2)
Is there any Fault in MCWDT?
No
Yes
(3)
Subcounter 0 Lower Limit Fault?
No
Yes
Subcounter 0 Lower Limit Fault Handling (Created by User with User System Requirement)
(4)
Subcounter 0 Upper Limit Fault?
No
Yes
Subcounter 0 Upper Limit Fault Handling (Created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

3.9.3 Example program of MCWDT fault handler

Code Listing 27 shows an example program of the MCWDT fault handler.

Code Listing 27 Example program of MCWDT fault handler

```
void irqFaultReport1Handler(void)
 {
    cy_en_sysflt_source_t status;
    uint32_t faultData;

    /* Read FAULT status from FAULT structure */
    status = Cy_SysFlt_GetErrorSource(FAULT_STRUCT1);   /*  (1) Read Fault Status Register 
(FAULT_STRUCT1_STATUS) */

    /* Evaluation of FAULT status */
    if(status != CY_SYSFLT_NO_FAULT)
    {
        */ MCWDT1 FAULT */
        if(status == CY_SYSFLT_SRSS_MCWDT1)   /* (2) Check if any Fault in MCWDT1 
(FAULT_STRUCT1_STATUS.SRSS_MCWDT1) */
        {
            /* Read and evaluate FAULT source from FAULT structure */
            faultData = Cy_SysFlt_GetData0(FAULT_STRUCT1);   /* Check Fault Data Register 
(FAULT_STRUCT1_DATA0.[0-3]) */
            if(faultData & 0x00000001ul)   /* (3) Check if Subcounter 0 Lower Limit Fault  */
            {
                 // Subcounter 0 lower limit fault handling created by user
            }
            else if(faultData & 0x00000002ul)   /* (4) Check if Subcounter 0 Upper Limit Fault 
*/
            {
                 // Subcounter 0 upper limit fault handling created by user
            }
            else if(faultData & 0x00000004ul)   /* (5) Check if Subcounter 1 Lower Limit Fault 
*/
            {
                 // Subcounter 1 lower limit fault handling created by user
            }
            else if(faultData & 0x00000008ul)   /* (6) Check if Subcounter 1 Upper Limit Fault 
*/
            {
                 // Subcounter 1 upper limit fault handling created by user
            }
        }
    }
    /* Clear FAULT interrupt */
    Cy_SysFlt_ClearStatus(FAULT_STRUCT1);   /* (7) Clear Fault Status (FAULT_STRUCT1_STATUS = 
0) */
    Cy_SysFlt_ClearInterrupt(FAULT_STRUCT1);   /* (8) Clear Fault Interrupt (FAULT_INTR.FAULT = 
1) */
 }
```

3.10 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).

3.11 MCWDT registers

Table 6 MCWDT registers

Name	Description
MCWDTx_CTL	MCWDT Subcounter 0/1 control register
MCWDTx_LOWER_LIMIT	MCWDT Subcounter 0/1 lower limit register
MCWDTx_UPPER_LIMIT	MCWDT Subcounter 0/1 upper limit register
MCWDTx_WARN_LIMIT	MCWDT Subcounter 0/1 warn limit register
MCWDTx_CONFIG	MCWDT Subcounter 0/1 configuration register
MCWDTx_CNT	MCWDT Subcounter 0/1 count register
MCWDT2_CTR2_CTL	MCWDT Subcounter 2 control register
MCWDT2_CTR2_CONFIG	MCWDT Subcounter 2 configuration register
MCWDT2_CTR2_CNT	MCWDT Subcounter 2 count register
MCWDT2_LOCK	MCWDT lock register
MCWDT2_SERVICE	MCWDT service register
MCWDT2_INTR	MCWDT interrupt register
MCWDT2_SET	MCWDT interrupt set register
MCWDT2_MASK	MCWDT interrupt mask register
MCWDT2_MASKED	MCWDT interrupt masked register
CLK_SELECT	Clock selection register
CLK_ILO_CONFIG	ILO 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 7 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 that 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 8 Definitions, acronyms, and abbreviations

Terms	Definitions
AHB	Advanced high-performance bus
CPU	Central processing unit
CPUSS	CPU subsystem
ECO	High-frequency 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	Low-frequency watch crystal oscillator
WDT	Watchdog timer
WIC	Wakeup interrupt controller

6 Related documents

The following are the TRAVEO™ T2G family series datasheets and technical reference manuals. Contact Infineon support to obtain these documents.

- Device datasheet
  - CYT2B6 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G Family
  - CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G Family
  - CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G Family
  - CYT2BL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™™ T2G family
  - CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
  - CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
  - CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family (Doc No. 002-33466)
  - CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
  - CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G Family
  - CYT4EN datasheet 32-bit Arm®Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-30842)
  - CYT2CL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- Body Controller Entry Family
  - TRAVEO™T2G automotive body controller entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2BL (Doc No. 002-29852)
- Body Controller High family
  - TRAVEO™T2G automotive body controller high family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT6BJ (Doc No. 002-36068)
- Cluster 2D Family
  - TRAVEO™ T2G automotive cluster 2D architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4EN (Doc No. 002-35181)
- Cluster entry family
  - TRAVEO™ T2G automotive cluster entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive cluster entry registers technical reference manual (TRM) for CYT2CL
- Application Notes
  - AN219842 – How to Use Interrupt in TRAVEO™ T2G

7 Other references

Infineon provides the sample driver library (SDL) including the initialization code as sample software to access various peripherals. SDL also serves as a reference to customers for drivers that are not covered by official AUTOSAR™ products. The SDL cannot be used for production purposes because it does not qualify to automotive standards. Code snippets in this application note are part of the SDL. Contact Infineon support to obtain the SDL.

Revision history

Document revision	Date	Description of changes
**	2018-08-21	Initial release.
*A	2018-10-29	Changed target part number (from CYT2B5/B7 series to CYT2B series) in all instances across the document.
*B	2019-02-28	Added target part number (CYT4B series) in all instances across the document.
*C	2019-10-01	Added target part number (CYT4D series) in all instances across the document.
*D	2020-03-02	Changed target part number (from CYT2B/CYT4B/CYT4D series to CYT2/CYT4 series) in all instances across the document. Added target part number (CYT3 series) in all instances across the document.
*E	2020-06-04	Added the flow to Section 2.7, 2.8, 3.7, 3.8, 3.9. Updated the example codes in Section 2.7, 2.8, 3.7, 3.8, 3.9. Added the AN219842 to Section 6. Added Section 7 (containing the information of the Sample Driver Library).
*F	2021-03-15	Updated Figure 8. Updated to Infineon template.
*G	2023-11-09	Template update; no content update
*H	2024-12-02	Added to CYT6BJ

Edition 2024-12-02

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ova1681539638600