AN0015.2: EFR32 Series 2 Watchdog

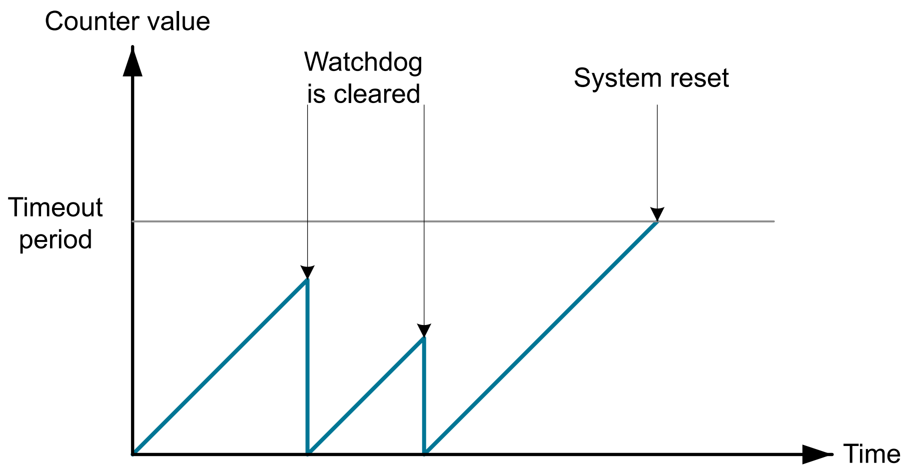
This application note demonstrates how to use the Watchdog module on EFR32 Series 2 devices. For watchdog information regarding EFM32 and EZR32 Wireless MCU Series 0 devices, refer to *AN0015.0: EFM32 and EZR32 Wireless MCU Series 0 Watchdog*. For watchdog information regarding EFM32 and EFR32 Series 1 devices, refer to *AN0015.1: EFM32 and EFR32 Series 1 Watchdog*.

This document discusses initializing the Watchdog, a basic setup for operation, and ways to utilize the added Watchdog functionality in more advanced applications.

For Watchdog example projects, see: https://github.com/SiliconLabs/peripheral_examples/tree/master/series2/wdog

KEY POINTS

- Clock source from selectable oscillators.
- Configurable timeout period.
- Selection to keep running or freeze when entering debug mode.
- Selection to block the CPU from entering EM4.
- Individual selection to keep running or freeze when entering EM2 DeepSleep or EM3 Stop.
- Selection to block the CMU from disabling the selected watchdog clock.
- Advanced features like timeout interrupt without MCU reset, PRS event monitoring or resets, and warning or window interrupt generation.



1. Device Compatibility

This application note supports multiple device families, and some functionality is different depending on the device.

EFR32 Wireless Gecko Series 2 consists of:

- EFR32BG21
- EFR32MG21
- EFR32BG22
- EFR32FG22
- EFR32MG22
- EFR32FG23
- EFR32SG23
- EFR32ZG23
- EFR32BG24
- EFR32MG24
- EFR32FG25
- EFR32BG27
- EFR32MG27
- EFR32FG28
- EFR32SG28
- EFR32ZG28

EFM32 Series 2 consists of:

- EFM32PG22
- EFM32PG23
- EFM32PG28

2. Watchdog Theory

2.1 General Theory

The purpose of the Watchdog timer is to generate a reset in case of a system failure to increase application reliability. The failure may be caused by a variety of events, such as an ESD pulse or a software failure.

The Watchdog circuit is a timer which, when enabled, must have its counter regularly cleared by software. If software does not clear the Watchdog timer's counter value, a Watchdog reset is activated. This functionality provides recovery from a software stalemate. Refer to the WDOG chapter of the reference manual covering the device for more extensive specifications and descriptions.

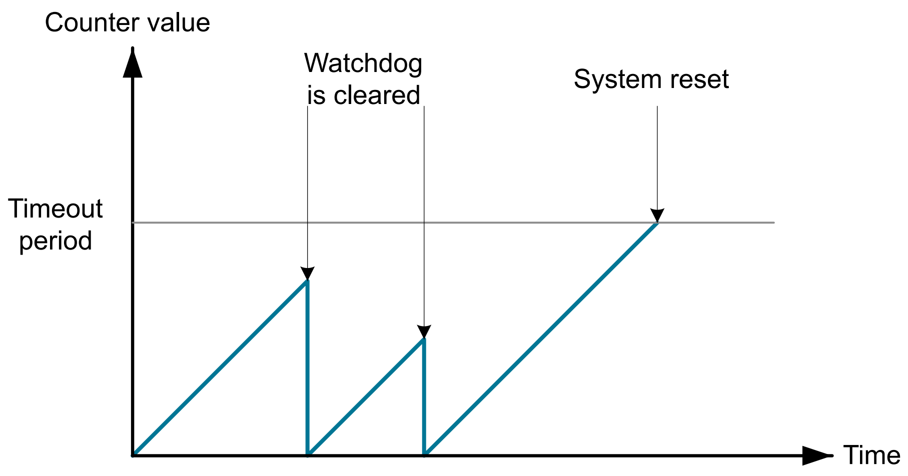


Figure 2.1. Watchdog Timer Operation

2.2 Watchdog Reset

The Watchdog timer on the EFM32 (WDOG) is a Low Energy Peripheral module that can be configured to generate a system reset in the case that it is not cleared by software before a configured deadline.

When the Watchdog is cleared (often referred to as feeding or petting the Watchdog), it means that the value counted to by the timer is reset to zero. This is done by software and is the normal procedure when the system is running correctly.

When the Watchdog timer's counter reaches a configured threshold value, it will generate a system reset. This indicates that the system is *not* running correctly and that the CPU has failed to reset the Watchdog in a timely manner. In this case, the Watchdog offers a safe way to return the system to a known state through system reset.

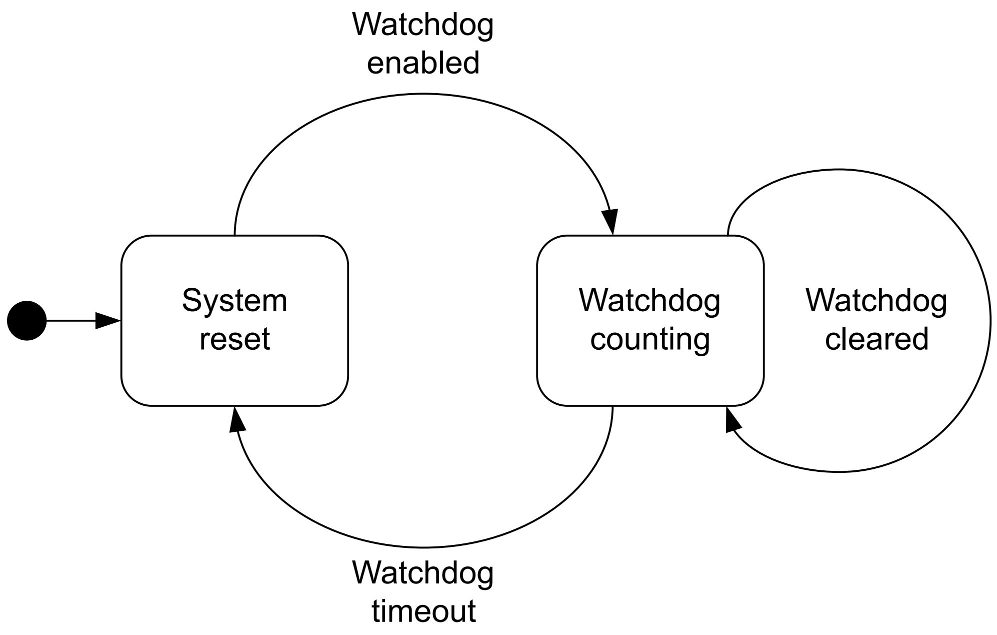


Figure 2.2. Watchdog State Diagram

3. Watchdog Configuration

3.1 Choice of Clock Source

On all devices, the WDOG can be configured to use the following clock sources: LFXO, LFRCO, ULFRCO, or HFCLKDIV1024 (a pre-scaled HCLK; divided by 1024). Users must clear EM2RUN and EM3RUN when the selected clock source is HFCLKDIV1024, as this clock is only available in EM0/EM1.

ULFRCO (Ultra Low Frequency RC Oscillator) is a separate 1 kHz RC oscillator that allows the Watchdog timer to run in EM3. This oscillator is enabled in EM0/EM1/EM2/EM3 (available in EM4, but Watchdog timer is not) and is included in all current consumption numbers.

It is important to take into consideration the ULFRCO accuracy. The oscillator offers extremely low energy consumption but has reduced accuracy. It is therefore advisable to time the Watchdog resets with considerable margin to avoid unwanted system resets. For example, the EFR32MG21 Mighty Gecko datasheet specifies that the ULFRCO can vary between 0.944 and 1.095 kHz. This means that the timeout period may vary from -8.6% to +5.9% of the desired period, device to device. This is important to remember when testing software on one device that will later run on other devices. A specific timeout period might give good results on one device but may generate unwanted system resets on a device with a slightly different ULFRCO frequency. The ULFRCO is available down to EM4.

The LFXO and LFRCO have higher accuracy, but in turn need more energy to operate. They are also shared with other system devices and first need be enabled in software. The LFXO clock (32.768 kHz) accuracy will be dependent on the crystal specification but is generally more accurate than either the LFRCO or the ULFRCO.

Refer to the specific device data sheet for more extensive information on oscillator accuracy.

All Watchdog oscillators have uncertainty that should be accounted for during system design to guarantee predictable system behavior. The clock sources for the Watchdog peripheral are also shared with other system devices and first need to be enabled in software. When using them as Watchdog clock sources, it is critical that they are not disabled in other parts of the software.

3.2 Timeout Periods

The Watchdog offers a wide range of timeout periods to select from during system design. The PERSEL field in WDOGn_CFG is used to divide the selected watchdog clock, and the timeout for the watchdog timer can be calculated with the formula:

$$T_{\mathrm{TIMEOUT}} = \frac{2^{3+\mathrm{PERSEL}} + 1}{f}$$

A total of 16 levels offer periods between 9 to 256k cycles of the chosen Watchdog clock source. (i.e., from ~274 µs for 32768 Hz clock source to ~262 s for 1000 Hz clock source).

3.3 Energy Mode Integration

The Watchdog contains extensive support for integration with the different Energy Modes and energy consumption minimization. Different configurations allow the Watchdog timer to continue running or to freeze when entering EM2 or EM3. The Watchdog can also be configured to prevent the CPU from entering EM4 by software request. For example, the Watchdog can be set to keep counting in EM2, but not during EM3. When resuming from EM3, the Watchdog will keep counting from the previous counter value. The EM2RUN and EM3RUN bits **must** be cleared when running from the HCLKDIV1024 clock source since HCLK is off while in these energy modes.

The registers are as follows:

- `WDOG_CTRL_EM2RUN` — toggles if the Watchdog timer is to keep running when the system has entered EM2.
- `WDOG_CTRL_EM3RUN` — toggles if the Watchdog timer is to keep running when the system has entered EM3.
- `WDOG_CTRL_EM4BLOCK` — toggles if the system is disabled by the Watchdog from entering EM4, where all the Watchdog clock sources are disabled.

3.4 Debug Functionality

The watchdog timer can either keep running or be frozen when the device is halted by a debugger. This configuration is done through the DEBUGRUN bit in WDOGn_CTRL. When code execution is resumed, the watchdog will continue counting from the previous value.

3.5 Watchdog Clearing Considerations

All the available Watchdog clocks are asynchronous to the CPU. When using clock sources other than HCLKDIV1024, synchronization between the CPU and the Watchdog clock domains should therefore be taken into consideration when using the Watchdog. It generally takes 3 WDOGCLK cycles from instruction execution until the correct value enters the WDOG registers. To guarantee that the Watchdog timer is always cleared properly, all clear operations of the Watchdog timer should be written 4 WDOGCLK cycles before timeout (3 + 1 clock of uncertainty). This is critical to remember especially when using short timeout periods. For example, when operating with a Watchdog period of 9 WDOGCLK cycles, the first clear should be no later than 3.3 WDOGCLK cycles (9 – 4 – 33%) after the Watchdog is enabled. The interval between all subsequent Watchdog instructions should be no more than 5.3 WDOGCLK cycles (9 – 1 uncertainty – 33%). When using a clock source different from the ULFRCO, the safety margin should be changed according to the accuracy of that oscillator. This should encourage the use of caution when implementing strict timing schemes for Watchdog resetting. For more information on accessing asynchronous registers and on oscillator accuracy, refer to the Reference Manual for the device.

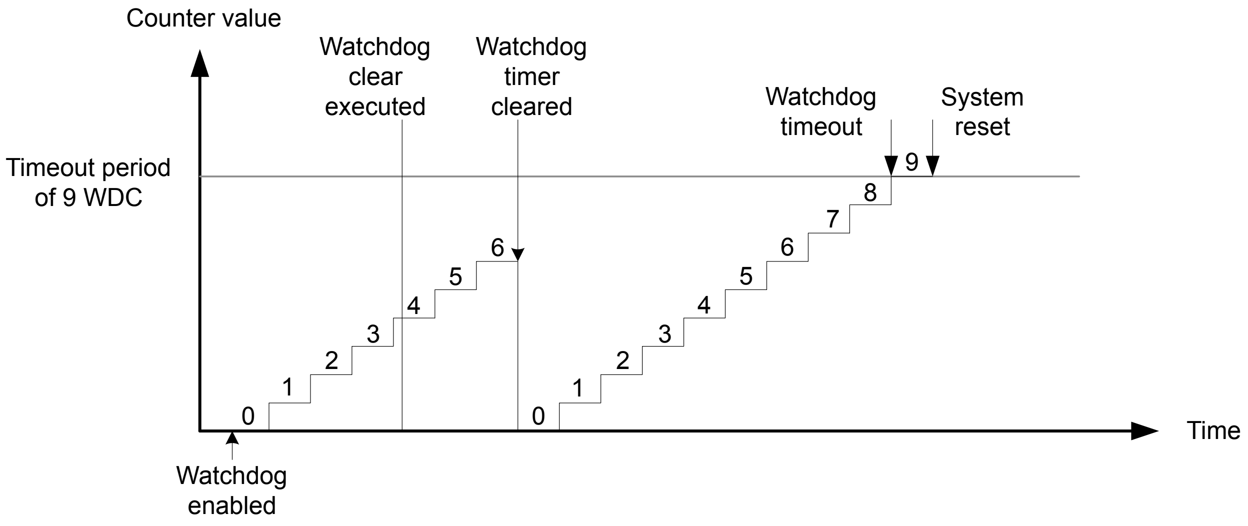


Figure 3.1. Discrete Representation of the Watchdog Timer Operation

To effectively determine when to issue a Watchdog reset in software, it is important to have good knowledge of its timing characteristics. The Watchdog timeout period needs to be longer than the longest possible execution path through the software initialization and main loop. The execution path length must also include any expected interrupts and their handler's run times. This ensures that a large number of interrupts is not regarded by the Watchdog as a system stalemate.

3.6 Reset Cause Detection

The RMU (Reset Management Unit) contains a register that holds information on what caused the last system reset. This register provides a method to detect if the Watchdog has triggered a reset. This information can be very useful in applications using the Watchdog, or to check if the Watchdog generates unwanted system resets. A basic use of the functionality is shown in the software examples. This can also be used to check if the Watchdog triggered a reset even in applications where the watchdog is not in use. This is a secure way to ensure safe operation even when the Watchdog is enabled accidentally. Alternatively, the watchdog register can be locked during startup. For more information, refer to the RMU and WDOG chapters in the Reference Manual for the device.

3.7 Register Locking

Some functionality is added to keep other parts of the system from interfering with the Watchdog. Different register entries can be set to prevent disablement of the chosen clock source, and the Watchdog register can be locked so that no modifications can be made to it. The registers are:

- WDOGn_LOCK() —Writing any value other than the unlock code, `44008 (0xABE8)`, to the WDOG_LOCK register locks the watchdog configuration; to unlock the configuration, write the unlock code to this register. The register should be set using the function `void WDOGn_Lock (WDOG_TypeDef *wdog)` from emlib as this function waits for synchronization to complete before locking the configuration. Similarly, `WDOGn_Unlock(WDOG_TypeDef *wdog)` waits for synchronization to complete before unlocking.
- `CMU_WDOGLOCK`—The watchdog clock source configuration can be locked separately using the CMU_WDOGLOCK register. Write any value other than the unlock code, `37879 (0x93F7)`, to lock the clock sources (CMU_WDOGnCLKCTRL), to unlock the configuration write the unlock code to this register.

It is important to note that writing to a locked configuration will cause a hard-fault exception.

3.8 Watchdog Timeout Interrupt

The reset behavior of the Watchdog timeout is configurable. When the WDOGRSTDIS field of the WDOGn_CTRL register is set to 1, the Watchdog reset is disabled, meaning that the Watchdog counter reaching the timeout value will not cause a device reset. If the TOUT field of the WDOG_IEN register is set to 1, then the Watchdog timeout can instead generate an interrupt, as shown in the following figure.

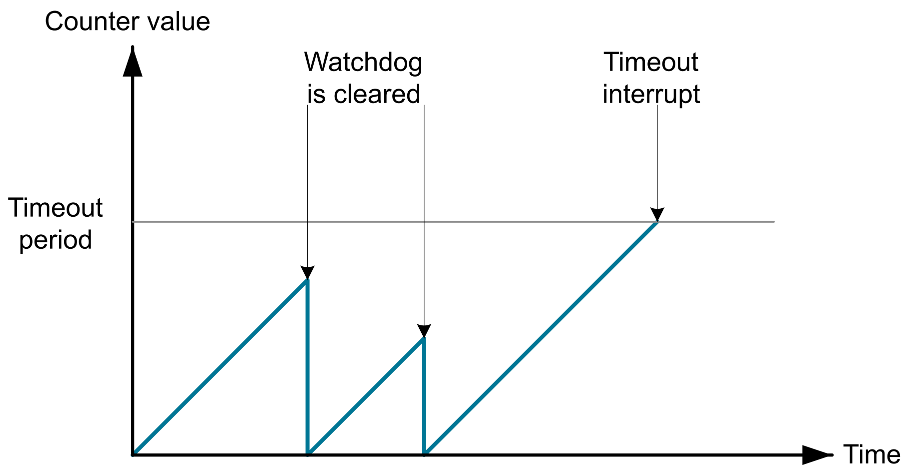


Figure 3.2. Watchdog Timeout Interrupt

3.9 PRS as a Watchdog Clear Source

The first PRS channel (selected by the PRS_CONSUMER_WDOG0_SRC0 register) can be used to clear the watchdog counter. To enable this feature, set the CLRSRC field of the WDOGn_CFG register to 1. The following figure shows how the PRS channel takes over the Watchdog clear function. Clearing the Watchdog with the PRS is mutually exclusive of clearing the Watchdog by software.

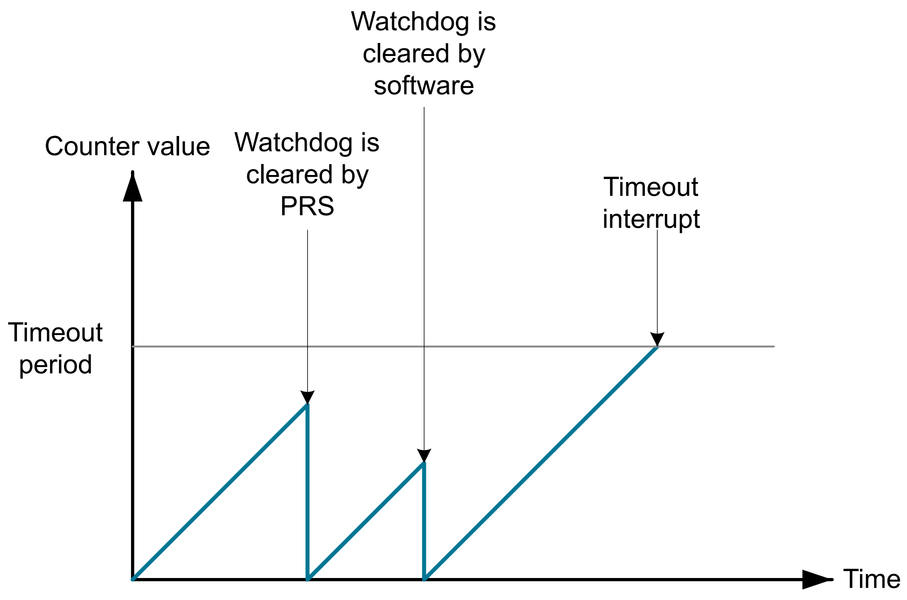


Figure 3.3. PRS as a Watchdog Clear Source

3.10 PRS Rising Edge Monitoring

PRS channels can be used to monitor multiple processes. If enabled, the PRS channels are checked every time the Watchdog timer is cleared, and any channel which has *not* seen an event can trigger an interrupt. The following figure illustrates these monitoring interrupts and shows where the PRS rising edge must occur.

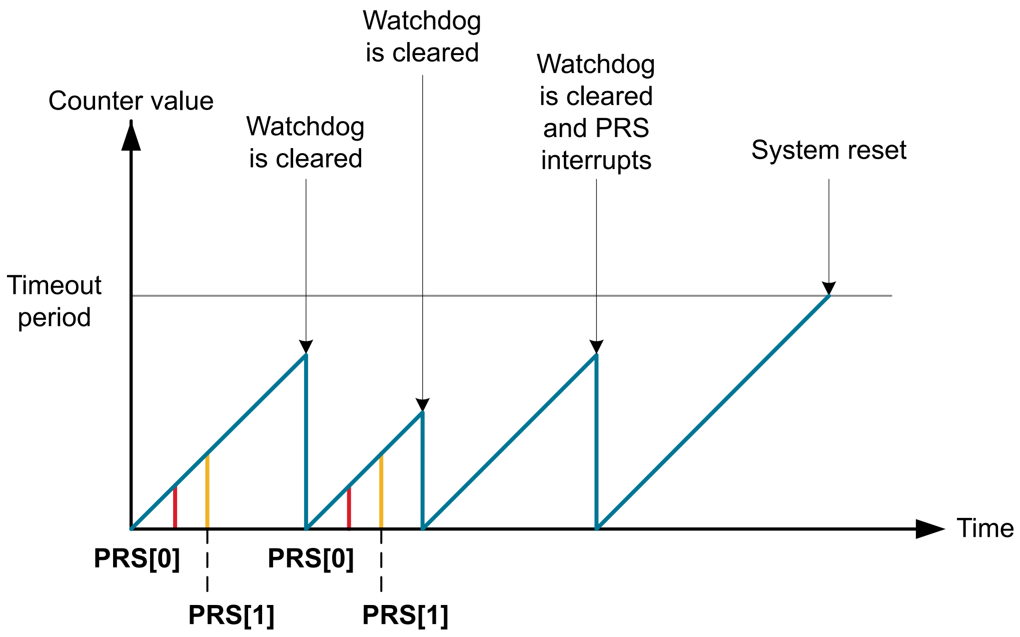


Figure 3.4. Watchdog PRS Edge Monitoring

3.11 Watchdog Warning Interrupt

The Watchdog implements a warning interrupt that can be configured to occur at approximately 25%, 50%, or 75% of the timeout period through the WARNSEL field of the WDOGn_CFG register. This interrupt can be used to wake up the CPU for clearing the watchdog. The warning point for the watchdog timer can be calculated with the formula:

$$T_{\mathrm{WARNING}} = \frac{2^{3+\mathrm{PERSEL}} \times \frac{\mathrm{WARNSEL}}{4} + 1}{f}$$

where f is the frequency of the selected clock.

The following figure illustrates the warning interrupt. Once the warning interrupt is enabled, the Watchdog will need to be cleared before the Watchdog counter counts up to the warning threshold (line in red), otherwise, the warning interrupt will fire.

When the watchdog is enabled, it is recommended to clear the Watchdog before changing WARNSEL.

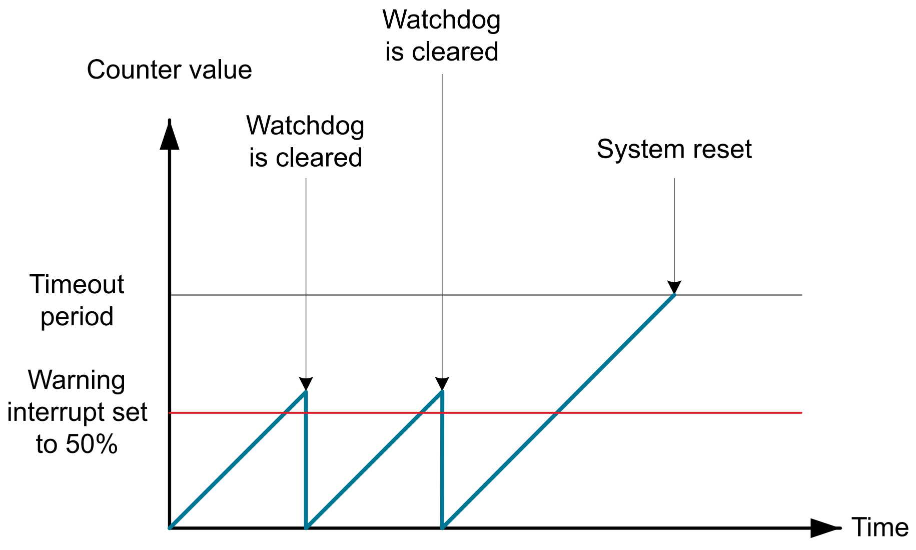


Figure 3.5. Watchdog Warning Interrupt Generation

3.12 Watchdog Window Interrupt

This interrupt occurs when the Watchdog is cleared below a certain threshold. This threshold is given by the formula:

$$T_{\mathrm{WINDOW}} = \frac{2^{3+\mathrm{PERSEL}} \times \frac{\mathrm{WINSEL}}{4} + 1}{f}$$

where f is the frequency of the selected clock.

This value will be approximately 12.5%, 25%, 37.5%, 50%, 62.5%, 75%, or 87.5% of the timeout value based on the WINSEL field of the WDOGn_CFG register. The following figure illustrates the window and the watchdog timeout period. Once the window interrupt is enabled, clearing the Watchdog too early causes the window interrupt to fire.

When the watchdog is enabled, it is recommended to clear the Watchdog before changing WINSEL.

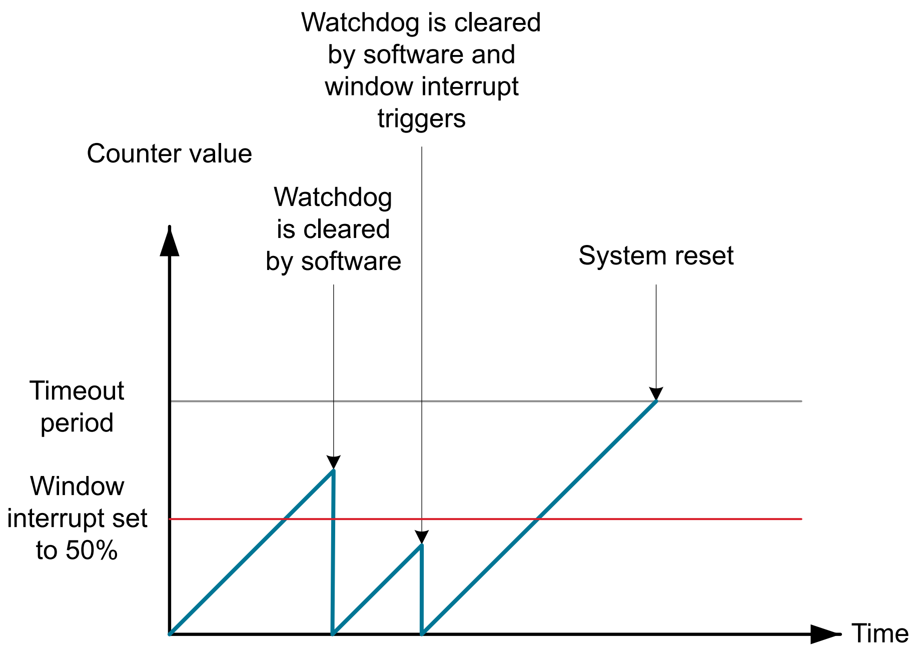

Watchdog is cleared by software and window interrupt triggers
Counter value
Watchdog is cleared by software
System reset
Timeout period
Window interrupt set to 50%
Time

Figure 3.6. Watchdog Window Interrupt Generation

4. Software Examples

Software examples for the Watchdog can be found in the Silicon Labs Peripheral Example repository on Github - https://github.com/SiliconLabs/peripheral_examples.

5. Revision History

Revision 1.0

October, 2023

- Added EFM32PG22, EFx32xG23, EFR32xG24, EFR32xG25, EFR32xG27 and EFx32xG28 to the supported parts list.
- Added URLs to the watchdog examples in the peripheral examples repository.

Revision 0.2

March, 2020

- Added new OPNs to 1. Device Compatibility.
- Updated clock sources and ULFRCO specifications for Series 2 device in accuracy example - 3.1 Choice of Clock Source.
- Corrected EM4H/EM4S to EM4 throughout document.
- Corrected register names in 3.8 Watchdog Timeout Interrupt and 3.9 PRS as a Watchdog Clear Source.
- Minor verbiage updates and typographical corrections, including capitalization, mis-spellings and punctuation marks, throughout document.

Revision 0.1

February, 2019

- Initial revision.

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com