EFM32
... the world's most energy friendly microcontrollers

Watchdog

AN0015 - Application Note

Introduction

This application note demonstrates how to use the Watchdog module on the EFM32 microcontrollers. This includes initializing the Watchdog, a basic setup for operation, and ways to utilize the added Watchdog functionality in more advanced applications.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

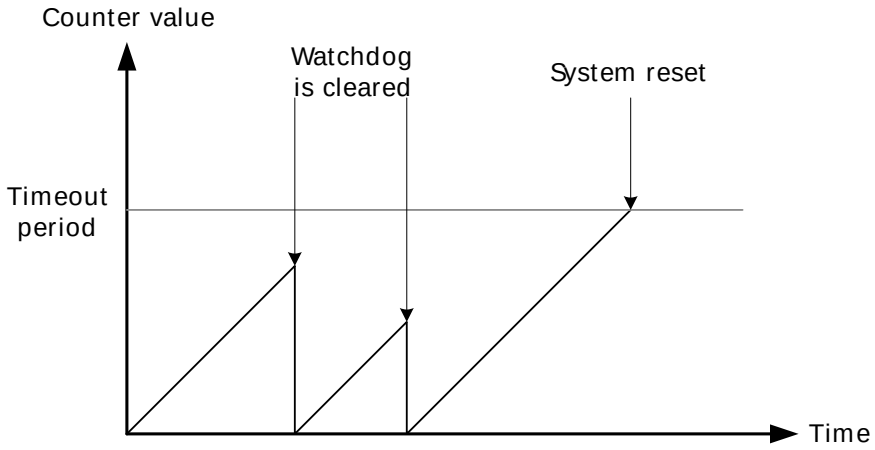
1 Watchdog Theory

1.1 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 an external event such as an ESD pulse, or by a software failure.

The Watchdog circuit is a timer which (when enabled) must be cleared by software regularly. If software does not clear it, a Watchdog reset is activated. This functionality provides recovery from a software stalemate. Please refer to the WDOG chapter of the reference manual covering the device for more extensive specifications and descriptions.

Figure 1.1. Watchdog Timer Operation



2 Watchdog Configuration

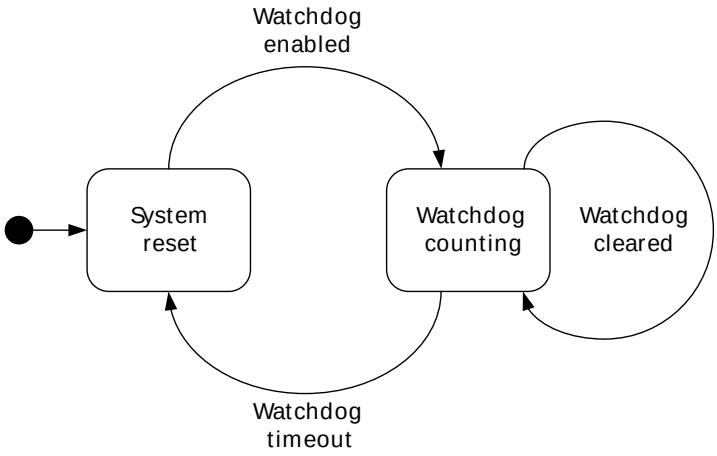
2.1 Watchdog Reset

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 the given deadline.

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

When the Watchdog timer reaches a certain 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 timely. In this case the Watchdog offers a safe way to return the system to a known state through system reset.

Figure 2.1. Watchdog state diagram



2.2 Choice of Clock Source

The WDOG can be configured to use one of three different clock sources: LFRCO, LFXO or ULFRCO. ULFRCO (Ultra Low Frequency RC Oscillator) is a separate and dedicated Watchdog 1 kHz RC oscillator that also runs in EM3. This oscillator is always enabled, 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. According to the EFM32WG datasheets the ULFRCO can vary between 0.8 and 1.5 kHz. This means that the timeout period may vary from -33% to +25% of the period wanted, device to device. This is important to remember when testing software on one device, that later will 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. Furthermore, depending on the application, the variations over temperature and supply voltage must also be taken into account when determining the Watchdog timeout period. These variations may result in that even higher safety margins are needed, than the -33% +25% given above. Please refer to the device specific datasheet for more extensive information on oscillator accuracy.

The other two clock sources available for the Watchdog timer, have higher accuracy, but in turn need more energy to operate. They are also shared with other system devices, and need first 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. See Section 2.6 (p. 4) and the software examples, for a method to avoid this problem. The highest accuracy can be obtained through the LFXO clock source. However all oscillators have uncertainty that should be accounted for during system design to guarantee predictable system behaviour.

It can also be noted that in order to access the WDOG from the CPU, the HFCORECLK_LE clock gate must be enabled in the CMU (Clock Management Unit).

2.3 Operation

The emlib provided by Silicon Laboratories Norway AS offers functions for easy configuration and operation of the WDOG.

- The function `void WDOG_Init(const WDOG_Init_TypeDef *init);` is intended run once during initialization, and should be used to configure all the WDOG settings. See the software examples for one suggested usage.
- The function `void WDOG_Enable(bool enable);` can be used to either disable or enable the WDOG after it has been initialized. If the WDOG is enabled prior to initialization, it will begin operation using its default settings which are defined in the "efm_wdog.h" file in the emlib.
- The function `void WDOG_Lock(void);` can be used to lock the WDOG->CTRL register from being altered. See Section 2.6 (p. 4) for more information.
- `void WDOG_Feed(void);` is the functions that clears the WDOG timer. This is the function that should be run regularly in software to avoid WDOG timeout and subsequent system reset.

2.4 Timeout Periods

The EFM32 Watchdog offers a wide range of timeout periods to select from during system design. A total of 16 levels offer periods between 9 to 256k cycles of the chosen Watchdog clock source. (i.e. from ~270 µs to ~256 s).

2.5 Energy Mode Integration

The EFM32 Watchdog contains extensive support for integration with the different Energy Modes, and energy consumption minimization. Different register values toggle if the Watchdog is to keep running in respectively EM2 and/or EM3, and if the system is allowed to enter EM4 or not. (e.g. the Watchdog can be set to keep counting in EM2, but not during EM3. When resuming from EM3, the Watchdog will keep counting from where it left of.) The registers are:

- `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.

EM2RUN and EM3RUN are not functional on EFM32G devices. In these devices, the Watchdog will keep running in EM2/EM3 regardless of the state of these bits.

2.6 Register Locking

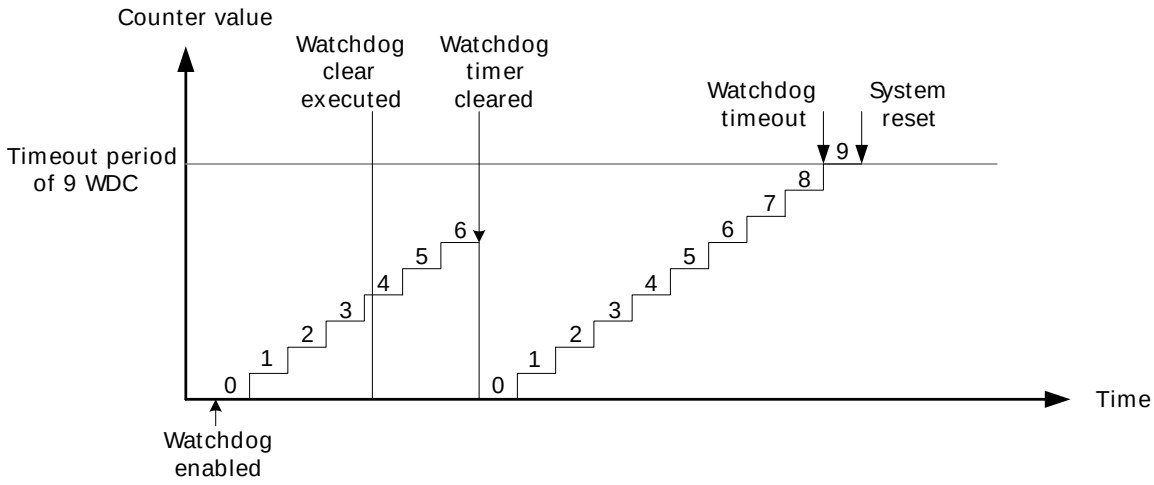
Also some functionality is added to keep other parts of the system from interfering with the Watchdog. Different register entries can be set to prevent disablement of the chosen clock source, and the Watchdog register can be locked so that no modifications can be made to it. The registers are:

- WDOG_CTRL_SWOSCBLOCK - when set, the Watchdog blocks all software from disabling the oscillator driving the WDOG timer.
- WDOG_CTRL_LOCK - when set, the WDOG->CTRL register is locked, and can not be altered. The register should be set using the function `void WDOG_Lock(void);` from the emlib. The lock can only be disabled by a system reset.

2.7 Watchdog Clearing Considerations

All the available Watchdog clocks are asynchronous to the CPU clock. Synchronization between the CPU and the Watchdog clock domains should therefor be taken into consideration when using the Watchdog. It generally takes 3 WDOGCLK cycles from the instruction is executed, till the correct value enters the WDOG registers. To guarantee that the Watchdog timer always is cleared timely, all clearings of the Watchdog timer should be written 4 WDOGCLK cycles before timeout (3 + 1 uncertainty). This is critical to remember especially when using short timeout periods. (e.g. when operating with a Watchdog period of 9 WDOGCLK cycles, the first clearing 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 covering the device.

Figure 2.2. 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 need be longer than the longest possible execution path through the software initialization and main loop. The execution path length must also include the expected interrupts and their handler's run times. This is to ensure that a large number of interrupts is not regarded by the Watchdog as a system stalemate.

2.8 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 covering the device..

3 Software Examples

These following software examples demonstrate how to use the main functionality included in the Watchdog module of the EFM32 microcontrollers. The examples can be used ether on a STK or a DVK. The LCD is used to give feedback, so on the DVK, the G890 MCU Board should be mounted. In the sections below more details about each example are given.

3.1 Example 1: Basic Operation

The first example shows an easy setup example for the Watchdog. Here the Watchdog is set up with settings close to the default ones. First some operations are done while timely resetting the Watchdog. Later the system is willingly put in stall, and a system reset is triggered by the Watchdog. On the new system startup, the Watchdog is identified as the source of the latest reset. Finally the system is stalled without enabling the Watchdog.

3.2 Example 2: More Extensive Functionality

This example demonstrates the basis of locking the clock source and Watchdog control registers, operating safely with the Watchdog in EM2 using the RTC and how clock sources other than the dedicated ULFRCO can easily be used as the Watchdog clock.

4 Revision History

4.1 Revision 1.08

2014-05-07

Updated example code to CMSIS 3.20.5

Changed to Silicon Labs license on code examples

Added example projects for Simplicity IDE

Removed example makefiles for Sourcery CodeBench Lite

4.2 Revision 1.07

2013-10-14

New cover layout

4.3 Revision 1.06

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

4.4 Revision 1.05

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

Added software projects for Tiny and Giant Gecko STK.

4.5 Revision 1.04

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

4.6 Revision 1.03

2011-10-21

Updated IDE project paths with new kits directory.

4.7 Revision 1.02

2011-05-18

Updated projects to align with new bsp version.

4.8 Revision 1.01

2010-11-16

Changed example folder structure, removed build and src folders.

Updated chip init function to newest efm32lib version.

4.9 Revision 1.00

2010-09-22

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

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

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4