AVR1310: Using the XMEGA Watchdog Timer

Features

- **Watchdog timeout programmable through fuses**
- **Watchdog settings lockable through fuses**
- **Watchdog with normal and window mode**
  - **Normal mode:**
    - **Programmable timeout period between 8 ms and 8 seconds**
  - **Window mode:**
    - **Programmable “closed window” period between 8 ms and 8 seconds**
  - **Total timeout period between 16 ms and 16 seconds, using window mode**
- **Example source code**
  - **Initialization of normal and window mode operation**
  - **Disabling and reconfiguration of Watchdog**

1 Introduction

Watchdog Timers are used to ensure that a system can recover from unforeseen failures in firmware or hardware. The Watchdog Timer will, if used correctly, be able to detect abnormalities in the program execution and respond by resetting the MCU. This brings the MCU to a well-defined and known state from where normal operation can be resumed.

The XMEGA™ AVR® family offers a very robust internal watchdog: Ordinary integrated Watchdog Timers often use the CPU clock as clock source, while the XMEGA Watchdog Timer’s clock source is independent from the CPU clock. This means that failure of the main clock would not affect the Watchdog Timer operation.

Further, the XMEGA Watchdog Timer does not only offer the “normal mode”, where the Watchdog Timer must be reset before a given timeout period. It also offers a “window mode” where the Watchdog Timer only can be reset within a limited period. In window mode, if the Watchdog Timer is reset too early (or too late), a system reset is triggered.

More information about using watchdog timers can be found in application note AVR132.

8-bit AVR® XMEGA Microcontrollers

Application Note

Rev. 8034B-AVR-04/09

2 Theory of operation

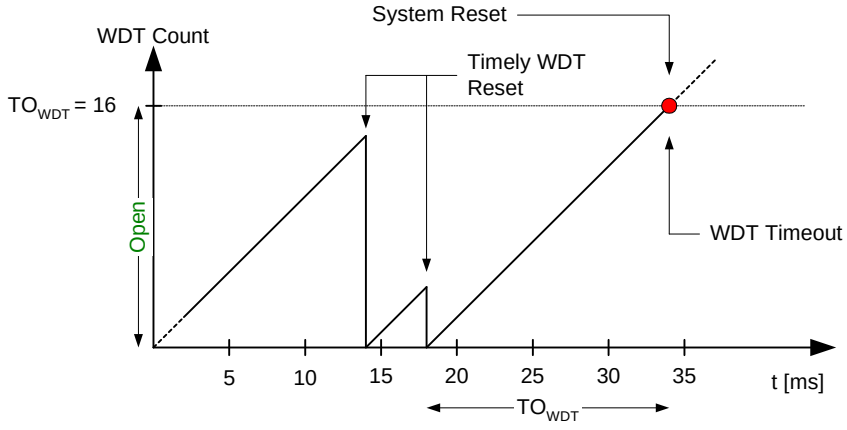
Before the XMEGA Watchdog Timer can be discussed it is important to get a few terms clear:

- The Watchdog Timer (WDT) is the peripheral module that can be configured to generate a system reset if the timer is reset too early or too late according to specified timeout periods. The timer value itself cannot be read or written, only reset.
- A Watchdog Timer Reset (WDT reset) is when the timer in the WDT is cleared (aka reset). This will make the timer start counting from zero again, and thus restart the timeout period.
- A “system reset” is when the AVR microcontroller is reset, resetting the CPU and I/O register to default values, and restart program execution from address 0x0000 (or boot section). The WDT can cause a system reset if it times out or, in window mode, if the WDT is reset too early.

2.1 Normal Mode

The WDT can be used in normal mode, which is when a single timeout period is set for the WDT; if the WDT is not reset before the timeout occurs the WDT will cause a system reset. Figure 2-1 illustrates this. The “Open” range along the “WDT Count” axis indicates that the WDT can be reset at any time before the WDT timeout (TO_{WDT}) expires (please refer to section 2.5 for exact timing details), as opposed to when using the window mode where the WDT cannot be reset until after the Watchdog Window timeout (TO_{WDTW}) expires.

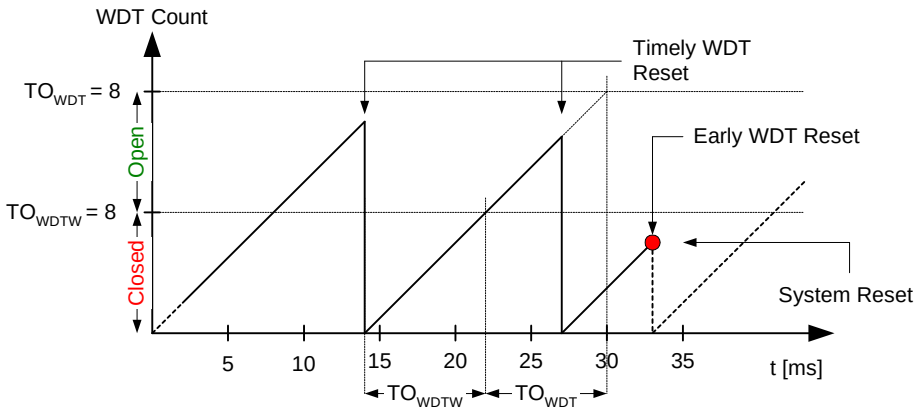
Figure 2-1. Timely and late (missing) Watchdog Timer reset in normal mode.



2.2 Window Mode

When the WDT is used in window mode, it uses two different timeout periods, a “closed” window timeout period (TO_{WDTW}) and the normal WDT timeout period (TO_{WDT}). The first defines a period from 8 ms to 8s where the WDT cannot be reset: if the WDT is reset in this period the WDT will cause a system reset. The normal WDT timeout period, which is also 8 ms to 8s, defines the duration of the “open” period, in which the WDT can (and should) be reset. The open period will always follow the closed period, and thus the total duration of the WDT timeout is the sum of the windowed and the normal timeouts. The closed and open periods used in window mode are illustrated in Figure 2-2.

Figure 2-2. Timely and early Watchdog Timer reset in Window mode.



2.3 Timer Clock

The WDT is clocked from an internal 1kHz ultra low power (ULP) RC oscillator. This oscillator can also be used by the RTC timer and is used by the Brown Out Detection circuit - if used in sampled mode. If any of these modules are configured to used the ULP oscillator the oscillator is running. The additional current consumption by enabling additional modules using the ULP RC oscillator is very low. Please refer to the datasheet for more information about power consumption.

It is important to be aware that the clock for the WDT is not very accurate. This is due to the fact that the oscillator is designed to draw very little power to be able to use the WDT even in long-life battery powered applications. The downside of low power oscillators is low accuracy. The typical accuracy of the clock for the WDT is +/-30% (please refer to datasheet for exact information on accuracy of the clock). This means that one have to be aware that the clock frequency can vary from one device to another. When designing software which uses the WDT the device-to-device variation must be kept in mind to ensure that the timeout periods used are valid for all devices, and not only the ones used in the lab during development.

Further, one has to consider that the clock source may vary over temperature and supply voltage – though this variation is significantly less than the +/-30% device-to-device variations. Please refer to the datasheet for more information on this topic.

2.4 Timeout Periods

The WDT offers a wide range of timeout periods, from 8 ms to 8 seconds (please refer to datasheet for details). The timeout period for the WDT period and the WDT window period is controlled by respectively the *WDT Period* (PER) bits and the *WDT Window Period* (WPER) bits located in the *WDT Control Register* (CTRL) and *WDT Window Control Register* (WINCTRL).

The WDT is reset when a valid write access to CTRL or WINCTRL register is performed (refer the description of the timed sequence required to write the control registers found in section 2.6.2).

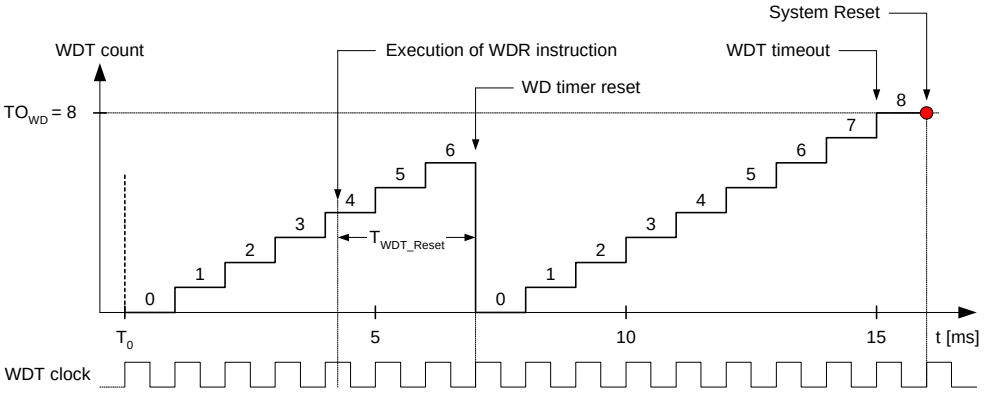
2.5 Digital timing of the Watchdog Timer

The WDT is operating in a different clock domain than the CPU and synchronization between the two domains should be considered when using the WDT.

It takes up to 2-3 WDT clock cycles to configure the WDT: when configuration settings are written to the WDT control registers (CTRL and WINCTRL), the new configuration becomes effective from the next WDT clock edge (rising edge of WDT clock in Figure 2-3), i.e. between 2 and 3 ms after the configuration is written. This means that the initial timeout period is up to 3 ms longer than the specified timeout. If the specified timeout is 8 ms, the actual timeout will be between 10 and 11 ms. This will mainly be relevant when using window mode and short timeout periods. This characteristic is not unique to the WDT; all asynchronous timers operate in this way due to synchronization between clock domains.

Again, the WDT is reset when a valid write access to the control registers is performed (please refer to section 2.6.2 for further details).

Figure 2-3. Digital timing of the XMEGA Watchdog Timer.



Another timing characteristic that one has to be aware of is that the duration between the execution of the Watchdog Timer Reset instruction (WDR) and the actual resetting of the WDT, is also subject to synchronization between clock domains: the WDT is reset on the third WDT clock edge after the WDR is executed (see T_{WDT_Reset} in Figure 2-3), which means that the WDT is reset between 2 ms and 3 ms after the WDR instruction is executed. Considering the use of a 8 ms timeout period, one therefore have to realize that the first WDR instruction should be executed within 5 ms after enabling the WD. Taking the +/-30% accuracy of the ULP oscillator into account as well, the WDR instruction must be executed within 3.5 ms or less. The interval between subsequent WDR instructions should be 4.9 ms or less (8 ms – 1 ms uncertainty – 30% oscillator uncertainty). The effect of this synchronization is minimized when the timeout period increases. This should however provide a strong hint that use of window mode in combination with short timeout periods requires very strict timing.

If the WDT causes a system reset, e.g. by timing out, the system reset occurs on the first following WDT clock edge (see Figure 2-3). This means that the system reset occurs 1 ms after the WDT timeout period has expired. This should normally not cause any problem, but it is useful to know if trying to measure the WD timeout period by monitoring the logic level of a pin. A better way to determine the actual frequency of the WDT clock is to use the XMEGA Real Time Clock timer, which can also be clocked by the ULP oscillator.

All of the above-mentioned conditions apply to both the WDT normal mode timeout and the WDT window mode timeouts.

2.6 Watchdog enabling and timing configuration

The WDT can be enabled and configured in two ways, either through fuses, which makes the AVR load the specified settings before leaving system reset. Alternatively, the WDT can be enabled at run-time, which means that the firmware writes the desired settings to the WDT control registers at run-time.

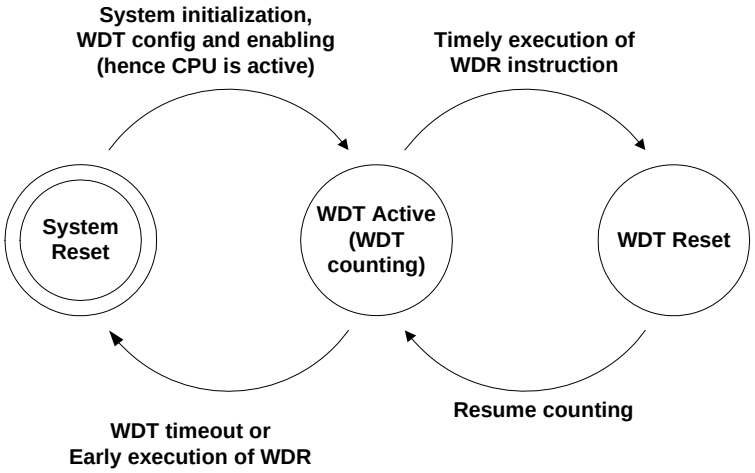
2.6.1 Fused enabling of the Watchdog

The WDT can be initialized automatically during system reset by writing the fuses corresponding to the normal mode and to window mode. These fuses will determine both the Watchdog Timer timeout period and the duration of the WDT closed window, and can also be used to enable the WDT at start-up. Please refer to the datasheet for more information about the fuses controlling the WDT.

Note that the window mode cannot be enabled through fuses, as it is an advantage to be able to enable it run-time to ensure proper timing when using window mode.

To provide maximum protection, it is possible to program the Watchdog Lock (WDLOCK) fuse, which ensures that the WDT cannot be disabled run-time - neither accidentally, nor on purpose. This can be desirable if considering that a system failure may have a duration in time that for some reason impairs the enabling of the Watchdog at run-time (in the firmware).

Figure 2-4. State chart for enabling WD through fuses.



2.6.2 Run-time enabling of the Watchdog Timer

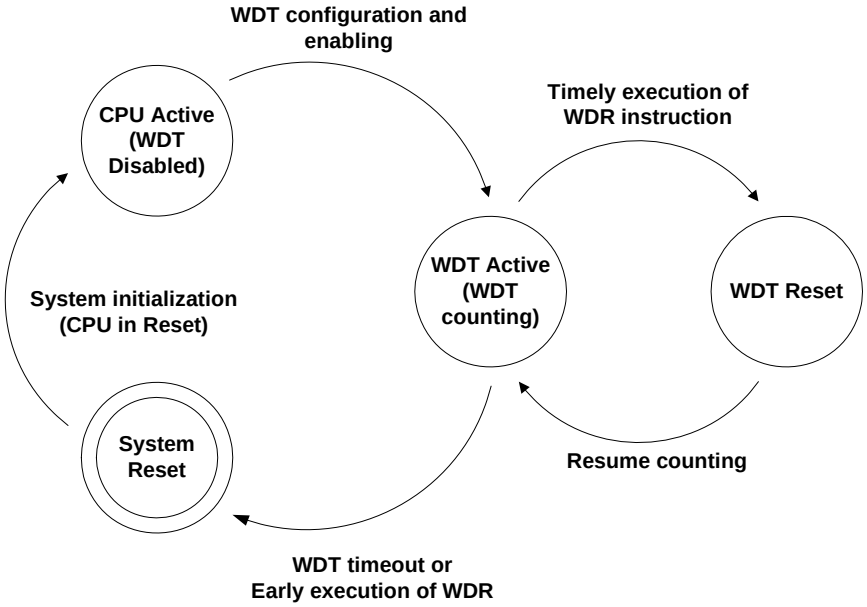
In some applications it can be required to be able to disable the WDT to reduce the power consumption in sleep mode to an absolute minimum. This can be the case in long life battery powered applications. If it is desirable to use the WDT in periods where the CPU is active, but not in sleep modes, it is convenient to be able to enable and disable the WDT, though this may reduce the protection offered by the WD. Note however that if other modules are using the ULP oscillator in sleep mode (BOD and RTC) the additional current consumption spend by leaving the WDT on is very low.

To enable the WDT in run-mode or to change the timeout period a timed sequence must be used: The timed sequence involved the Configuration Change Protection register (`CCP`), which controls access to certain registers. After writing the signature `0xD8` to the CCP, the WDT can be reconfigured within the next four instruction cycles. During these four instruction cycles the global interrupts are automatically disabled to ensure that the timed sequence is not invalidated by an interrupt. Note that the timeout is given in instruction cycles, and not clock cycles, a DMA transfers will therefore not affect the timing: DMA transfers are not counted as instruction cycles – only instructions executed by the CPU are counted.

Within four instruction cycles after writing the `0xD8` signature to the CCP, the WDT control registers (`CTRL` and `WINCTRL`) can be modified. The procedure for doing so is specified in the datasheet, and examples how to do it can be found in the code example for this application note.

Note that it is not possible to change the WDT or the WDT window mode configuration while the `WDLOCK` fuse is set, but it is possible to enable/disable the WDT window mode.

Figure 2-5. State chart for enabling WDT run-time.



2.7 Intended use of the Watchdog

A WDT is meant to save the day if the system has an unforeseen failure that is not handled in firmware or hardware, or if an external disturbance causes the system to fail. A well-used WDT would be able to generate a system reset, which the end-user will not or will barely notice. Seeing this in contrast to a product that needs power cycling every now and then to return to an operational state, the difference should be apparent: Whether the end-user is satisfied with the product or not.

In general it is recommended to issue a WDT reset from somewhere in the main loop of the firmware. Do not reset the WDT in interrupt service routines - unless the interrupt routine checks a series of flags that confirms correct execution of various parts of the firmware. If these simple rules are followed, the WDT is hard to misuse.

The WDT window mode is a bit more challenging to use than the normal mode, as it involves more strict control of the WDT reset timing. In window mode the WDT should be reset from somewhere within the main loop, never in interrupt service routines, as this would impair the closed window protection: Because the closed window defines the minimum expected duration of the main loop (or subsections of the main loop), it can be used to catch cases where parts of the main loop code is not executed, or cases where early exit from function calls happens. An example could be that some algorithm or other software failure causes an operation to finish too quickly. For instance, the expected duration of writing a value to the EEPROM is 4 ms, but if it completes in a few microseconds (inspection of a flag failed?), a subsequent WDT Reset would arrive too early. Another example could be corruption of the return stack or stack pointer itself, causing abnormal program execution.

Another case where the window mode offers good protection is if the code execution is stuck in a loop where the WDR instruction is executed repeatedly: if the WDT reset occurs more frequently than anticipated the WDT will “assume” that a failure has occurred and reset the system to bring it back to an operational state.

3 Examples

This application note includes a source code package with a basic Watchdog Timer driver implemented in C.

Note that this Watchdog Timer driver is not intended for use with high-performance code. It is designed as a library to get started with the XMEGA Watchdog Timer. For timing and code space critical application development, you should access the Watchdog Timer registers directly. Please refer to the driver source code and device datasheet for more details.

3.1 Files

The source code package consists of three files:

- *wdt_driver.c* – Watchdog Timer driver source file
- *wdt_driver.h* – Watchdog Timer driver header file
- *wdt_example.c* – Example code using the driver

For a complete overview of the available driver interface functions and their use, please refer to the source code documentation.

3.2 Doxygen Documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit http://www.doxygen.org. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the *readme.html* file in the source code folder.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
Avr32@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature