# AN0015.0: EFM32 and EZR32 Wireless MCU Series 0 Watchdog

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

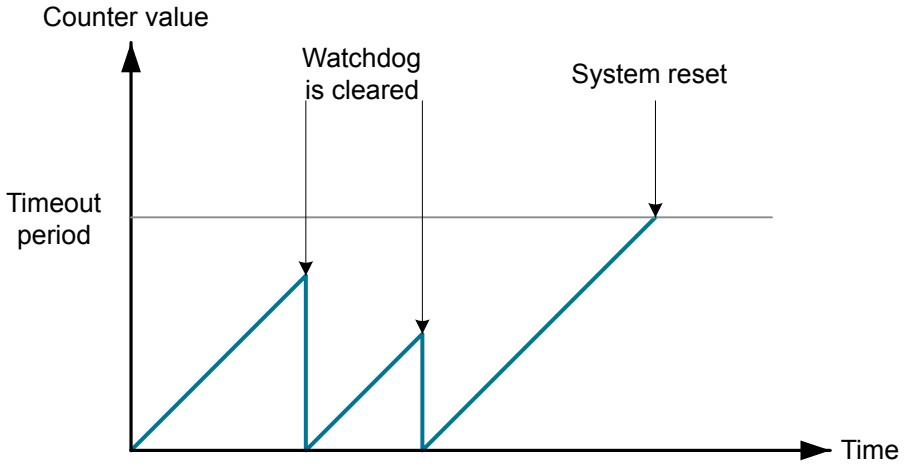
This document discusses initializing the Watchdog, a basic setup for operation, and ways to utilize the added Watchdog functionality in more advanced applications.

This application note includes:

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

**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 Energy Mode 4.
- 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.



# 1. Device Compatibility

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

EFM32 Series 0 consists of:

- EFM32 Gecko (EFM32G)
- EFM32 Giant Gecko (EFM32GG)
- EFM32 Wonder Gecko (EFM32WG)
- EFM32 Leopard Gecko (EFM32LG)
- EFM32 Tiny Gecko (EFM32TG)
- EFM32 Zero Gecko (EFM32ZG)
- EFM32 Happy Gecko (EFM32HG)

EZR32 Wireless MCU Series 0 consists of:

- EZR32 Wonder Gecko (EZR32WG)
- EZR32 Leopard Gecko (EZR32LG)
- EZR32 Happy Gecko (EZR32HG)

# 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 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. Refer to the WDOG chapter of the reference manual covering the device for more extensive specifications and descriptions.

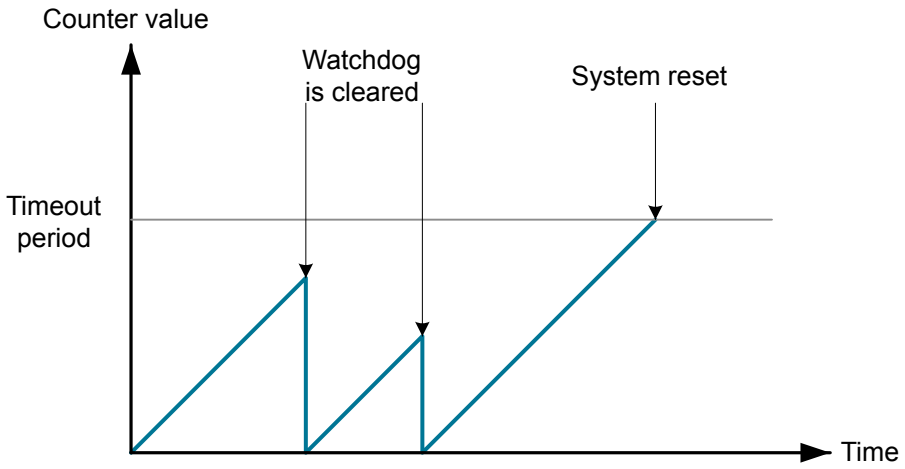



**Figure 2.1. Watchdog Timer Operation**

### 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 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 reset 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 in a timely manner. In this case, the Watchdog offers a safe way to return the system to a known state through system reset.

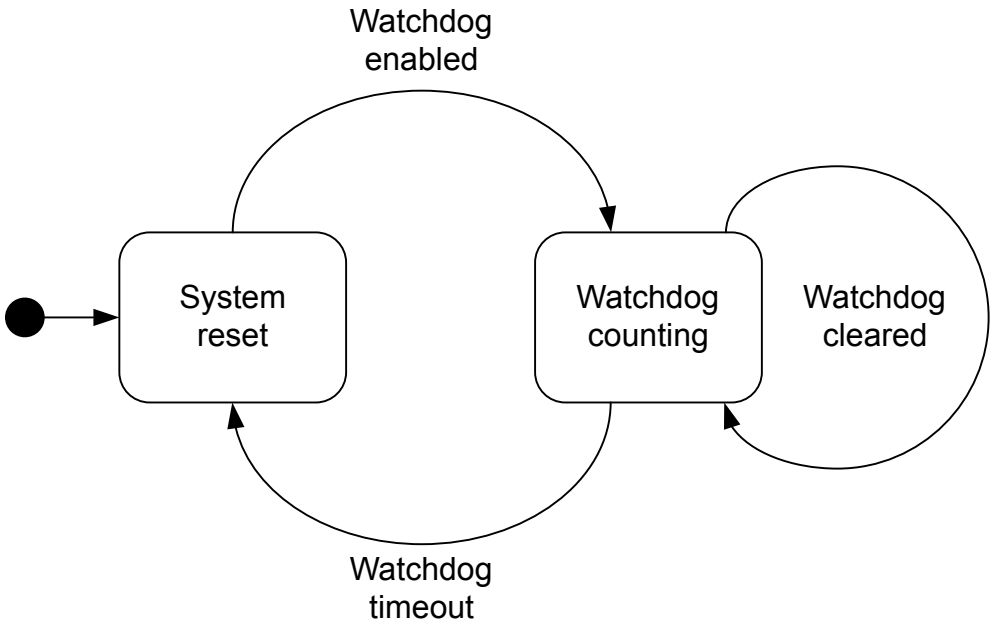



**Figure 2.2. Watchdog State Diagram**

# 3. Watchdog Configuration

## 3.1 Choice of Clock Source

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. For example, the EFM32LG Leopard Gecko data sheet specifies that the ULFRCO can vary between 0.7 and 1.75 kHz. This means that the timeout period may vary from -43% to +43% 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. 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 mean that even higher safety margins are needed than the -43% to +43% given above.

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 first need 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. 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.

For EFM32LG devices, the LFRCO (32.768kHz) can vary between 31.29 and 34.28 kHz (VDD= 3.0 V, $T_{AMB}$=25°C), which means that the timeout period may vary from -4.4% and 4.7% of the desired period. Over full supply and temperature range, the LFRCO can vary between 26.0 and 46.2 kHz, which means that the timeout period may vary from -29% and 26% of the desired period.

The LFXO clock (32.768kHz) 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.

In order to access the WDOG from the CPU, the HFCORECLK_LE clock gate must be enabled in the CMU (Clock Management Unit).

## 3.2 Timeout Periods

The Watchdog offers a wide range of timeout periods to select from during system design. The PERSEL field in WDOGn_CTRL 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 register values toggle if the Watchdog is to keep running in respectively EM2 and/or EM3, and whether or not the system is allowed to enter EM4. 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 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.

**Note:** 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.

## 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 clock. 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 the instruction is executed until the correct value enters the WDOG registers. To guarantee that the Watchdog timer always is cleared correctly, 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.

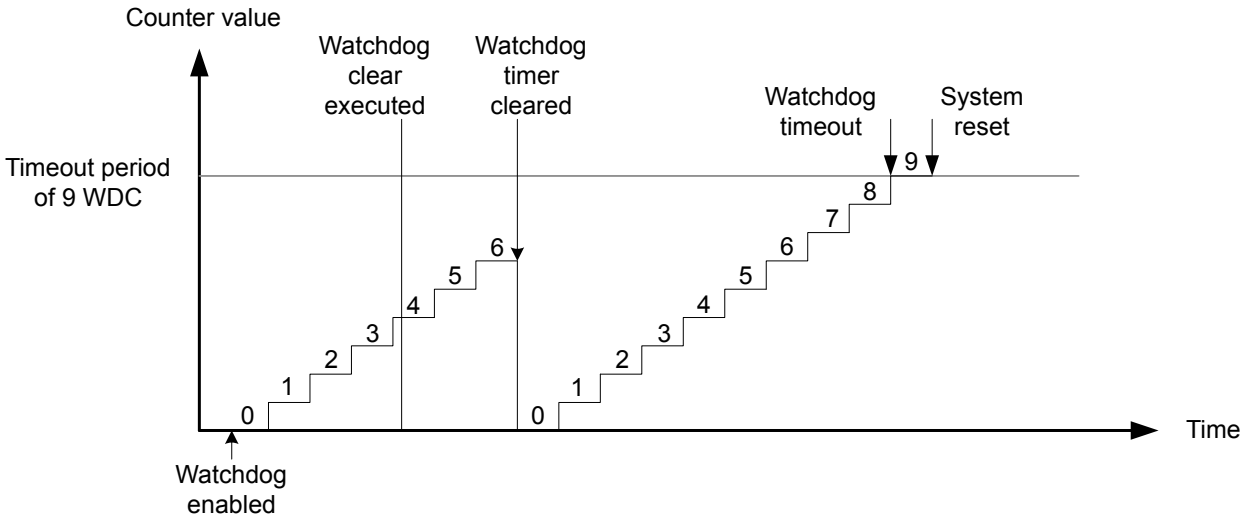



**Figure 3.1. Discrete Representation of the Watchdog Timer Operation**

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 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.

### 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:

- `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 cannot be altered. The register should be set using the function `void WDOG_Lock(void)` from emlib. The lock can only be disabled by a system reset.

# 4. Software Examples

The software example associated with this document demonstrates how to use the main functionality included in the Watchdog module of the device. The example can be used on a Starter Kit (STK). The examples are divided into two sets:

- Segment LCD for EFM32G, EFM32TG, EFM32LG, EFM32GG, and EFM32WG
- Memory LCD for EFM32ZG and EFM32HG

The LCD module is used to provide feedback. The examples uses PB1 to choose the example mode, and PB0 starts the mode. LED0 on the STK is used to indicate the status of the test.

## 4.1 Mode 0: Basic Watchdog Operation

This example mode uses the ULFRCO as the Watchdog clock source. The mode feeds the Watchdog every 50 ms using the usTimer driver. When running the example, LED0 toggles whenever the firmware feeds the Watchdog. The reset source number will be displayed on the LCD after the Watchdog causes a reset.

## 4.2 Mode 1: Lock Operation

This example mode uses the LFXO as the Watchdog clock source. The mode demonstrates the Watchdog lock feature. When running the example, LED0 toggles whenever software feeds the Watchdog. The reset source number will be displayed on the LCD after the Watchdog causes a reset.

# 5. Revision History

### 5.1 Revision 1.09

2017-09-12

Changed AN0015 to AN0015.0 for EFM32 MCU and EZR32 Wireless MCU Series 0 devices.

Updated content for all EFM32 and EZR Wireless MCU Series 0 devices.

Reorganized the example code structure.

Updated formatting and slightly modified wording in some places.

### 5.2 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

### 5.3 Revision 1.07

2013-10-14

New cover layout

### 5.4 Revision 1.06

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

### 5.5 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.

### 5.6 Revision 1.04

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

### 5.7 Revision 1.03

2011-10-21

Updated IDE project paths with new kits directory.

### 5.8 Revision 1.02

2011-05-18

Updated projects to align with new bsp version.

### 5.9 Revision 1.01

2010-11-16

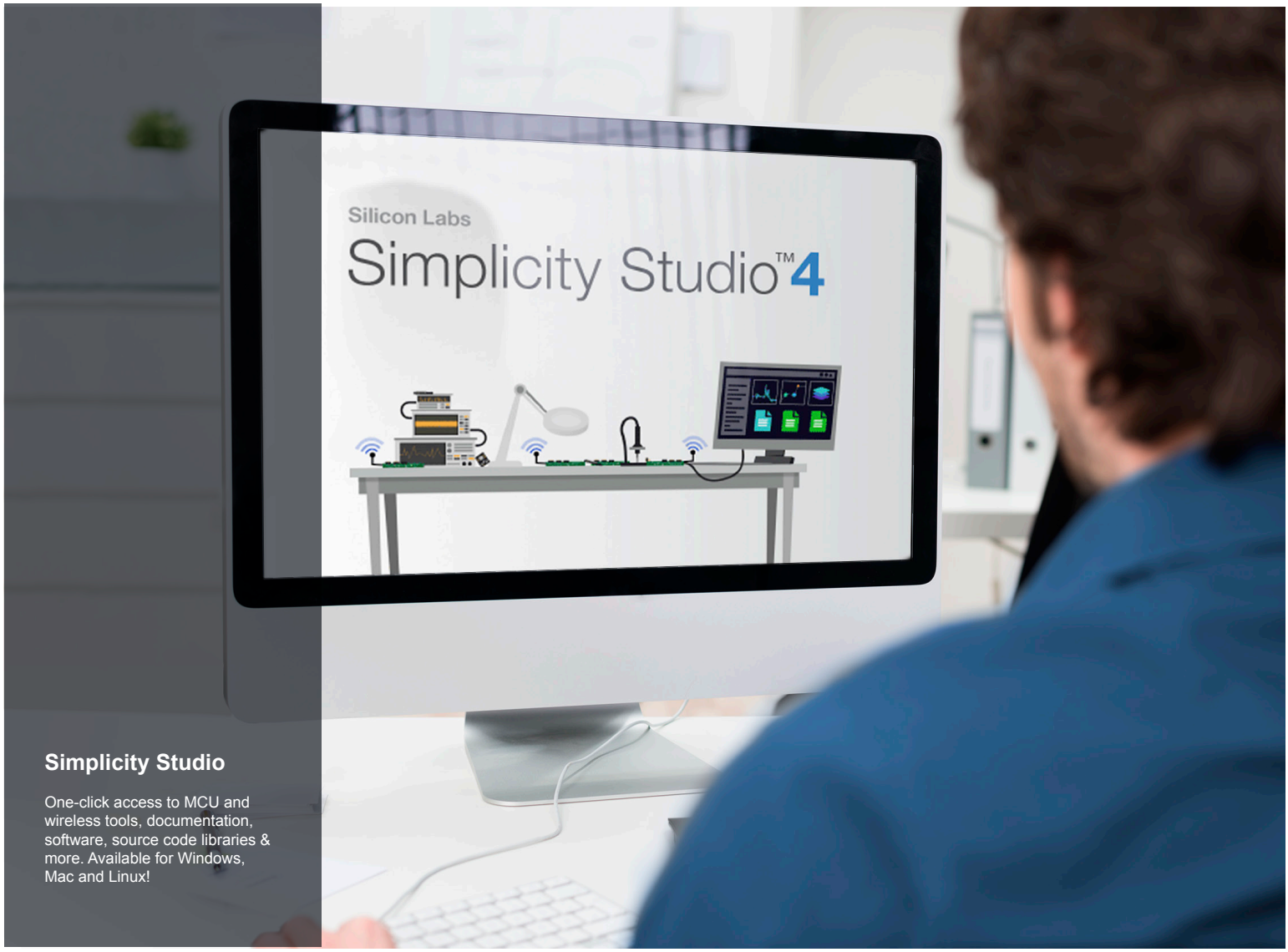
Changed example folder structure, removed build and src folders.

Updated chip init function to newest efm32lib version.

### 5.10 Revision 1.00

2010-09-22

Initial revision.




## Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

**IoT Portfolio**
*www.silabs.com/IoT*

**SW/HW**
*www.silabs.com/simplicity*

**Quality**
*www.silabs.com/quality*

**Support and Community**
*community.silabs.com*



SILICON LABS

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

**http://www.silabs.com**