# AN0059.1: UART Flow Control

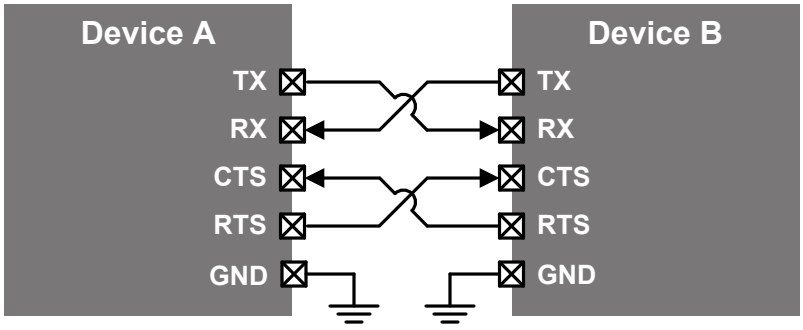
This application note describes how to implement hardware or software flow control for UART.

This application note includes the following:

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

**KEY POINTS**

- UART Flow Control is a strategy for the communication bewteen slow and fast devices without data losing.
- Introduce the protocols of hardware flow control and software flow control.
- The hardware flow control is natively supported by the EFM32 and Wireless SoC Series 1 family USART peripheral.
- UART hardware flow control is fully supported by UARTDRV, and UART software flow control is partially supported by the driver.



# 1. Device Compatibility

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

EFM32 Series 1 (EFM32xG1/EFM32xG12) consists of:

- EFM32 Pearl Gecko (EFM32PG1/EFM32PG12)
- EFM32 Jade Gecko (EFM32JG1/EFM32JG12)

EFR32 Wireless MCU Series 1 (EFR32xG1/EFR32xG12/EFR32xG13/EFR32xG14) consists of:

- EFR32 Blue Gecko (EFR32BG1/EFR32BG12/EFR32BG13)
- EFR32 Flex Gecko (EFR32FG1/EFR32FG12/EFR32FG13/EFR32FG14)
- EFR32 Mighty Gecko (EFR32MG1/EFR32MG12/EFR32MG13)

## 2. UART Flow Control Introduction

UART Flow Control is a method for slow and fast devices to communicate with each other over UART without the risk of losing data.

Consider the case where two units are communicating over UART. A transmitter T is sending a long stream of bytes to a receiver R. R is a slower device than T, and at some point R cannot keep up. It needs to either do some processing on the data or empty some buffers before it can keep receiving data.

R needs to tell T to stop transmitting for a while. This is where flow control comes in. Flow control provides extra signaling to inform the transmitter that it should stop (pause) or start (resume) the transmission.

Several forms of flow control exist. For example, hardware flow control uses extra wires, where the logic level on these wires define whether the transmitter should keep sending data or stop. With software flow control, special characters are sent over the normal data lines to start or stop the transmission.

# 3. Flow Control Protocols

This chapter describes the most common ways to implement flow control.

## 3.1 Hardware Flow Control

With hardware flow control (also called RTS/CTS flow control), two extra wires are needed in addition to the data lines. They are called RTS (Request to Send) and CTS (Clear to Send). These wires are cross-coupled between the two devices, so RTS on one device is connected to CTS on the remote device and vice versa. Each device will use its RTS to output if it is ready to accept new data and read CTS to see if it is allowed to send data to the other device.

As long as a device is ready to accept more data, it will keep the RTS line asserted. It will deassert RTS some time before its receive buffer is full. There might still be data on the line and in the other device transmit registers which has to be received even after RTS has been deasserted. The other device is required to respect the flow control signal and pause the transmission until RTS is again asserted.

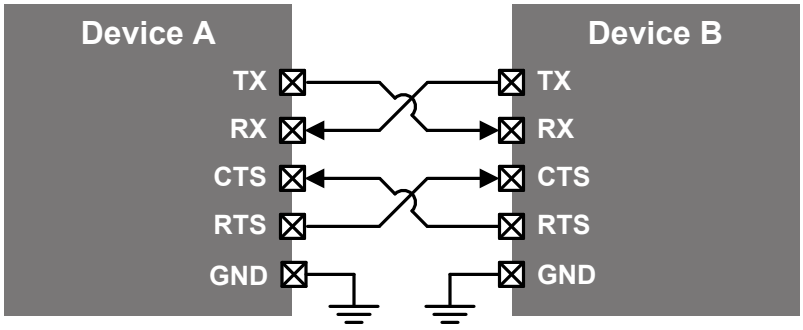



**Figure 3.1. Hardware Flow Control**

The flow control is bidirectional, meaning both devices can request a halt in transmission. If one of the devices never has to request a stop in transmission (i.e. it is fast enough to always receive data), the CTS signal on the other device can be tied to the asserted logic level. The RTS pin on the fast device can thus be freed up to other functions.

The hardware flow control is natively supported by the EFM32 and Wireless SoC Series 1 family. Use RTSPEN and CTSPEN in USARTn_ROUTEPEN to allocate the hardware flow control to GPIOs. RTS is an outgoing signal which indicates that RX buffer space is available to receive a frame. The link partner is being requested to send its data when RTS is asserted. CTS is an incoming signal to stop the next TX data from going out. When CTS is negated, the frame currently being transmitted is completed before stopping. CTS indicates that the link partner has RX buffer space available, and the local transmitter is clear to send. Also use CTSEN in USARTn_CTLX to enable the CTS input into the TX sequencer.

## 3.2 Software Flow Control

Software flow control does not use extra wires. Only 3 wires are required (RX, TX, and GND). Transmission is started and stopped by sending special flow control characters. The flow control characters are sent over the normal TX and RX lines. The flow control characters are typically the ASCII codes XON and XOFF (0x11 and 0x13). If device A sends XOFF to device B it means that B should halt transmission to A until B receives an XON character from A.

# 4. Software Examples

There are two kinds of examples provided regarding UART flow control. Examples with emlib and examples with emdrv (uartdrv).

For each kind of example, users can select the demo for Hardware flow control, Software flow control using the PB1 button on the STK board, and pressing PB0 button to run the demo. The current demo name will be displayed on LCD.

## 4.1 Examples with emlib

Since the hardware flow control be natively supported by USART peripheral, the init function will initialize the pin location of the flow control signals. The RTS and CTS is default low true, but the polarity can be inverted with RTSINV and CTSINV in USARTn_CTRLX.

Two callback functions are specified. The RX callback will be called every time a byte is received. The TX callback is called when a full message has been sent.

To send a message with flow control, use the function uartFcStartTx(). This method takes two parameters, a pointer to the message and the message length. The function will return immediately but start sending the message using interrupts. Transmission will automatically be turned on or off depending on the flow control signals. When the entire message has been sent, the TX callback routine is called.

### 4.1.1 UART Hardware Flow Control

This example uses HW Flow control between two EFM32's to send dummy messages back and forth. At regular intervals the program will send a long message and continually listen for messages.

The USART peripheral will deassert RTS pin on RX buffer full, and assert the RTS pin when the RX buffer space is available to receive a frame. When CTS is negated, the frame currently being transmitted is completed before stopping, and the next Tx data will be stopped.

To test the application connect two EFM32's according to Figure 3.1 Hardware Flow Control on page 4.

### 4.1.2 UART Software Flow Control

This example uses SW Flow control between two EFM32's to send dummy messages back and forth. At regular intervals the program will send a long message and continually listen for messages. At some defined buffer level the program will disable the transmission by sending XOFF, and send the XON to restart the transmission after processing the received data done.

## 4.2 Examples with emdrv

The UART driver supports the UART capabilities of the USART, UART, and LEUART peripherals. The driver is fully reentrant, and multiple driver instances can coexist. The driver does not buffer or queue data, but it queues UART transmit and receive operations. Both blocking and non-blocking transfer functions are available. Non-blocking transfer functions report transfer completion with callback functions. Transfers are done using DMA.

UART hardware flow control (CTS/RTS) is fully supported by the UART driver. However, UART software flow control (XON/XOFF) is partially supported by the driver. The application must monitor buffers and make decisions on when to send XON/ XOFF. Also the application should monitor the received data to pause or resume the transmission after receiving the flow control character XOFF or XON.

### 4.2.1 UART Hardware Flow Control

This example uses HW Flow control which is supported by the UART driver. At regular intervals the program will send a long message and then start to listen for messages until a message is received with the specific length.

The USART peripheral will deassert RTS pin on RX buffer full, and assert RTS pin when the RX buffer space is available to receive a frame. When CTS is negated, the frame currently being transmitted is completed before stopping, and the next Tx data will be stopped.

To test the application connect two EFM32's according to Figure 3.1 Hardware Flow Control on page 4, or use the terminal with a USB-to-UART bridge. After setting up the serial port with hardware flow control according to the figures as below, type 20 characters, and then the EFM32 will send a 52 bytes message.

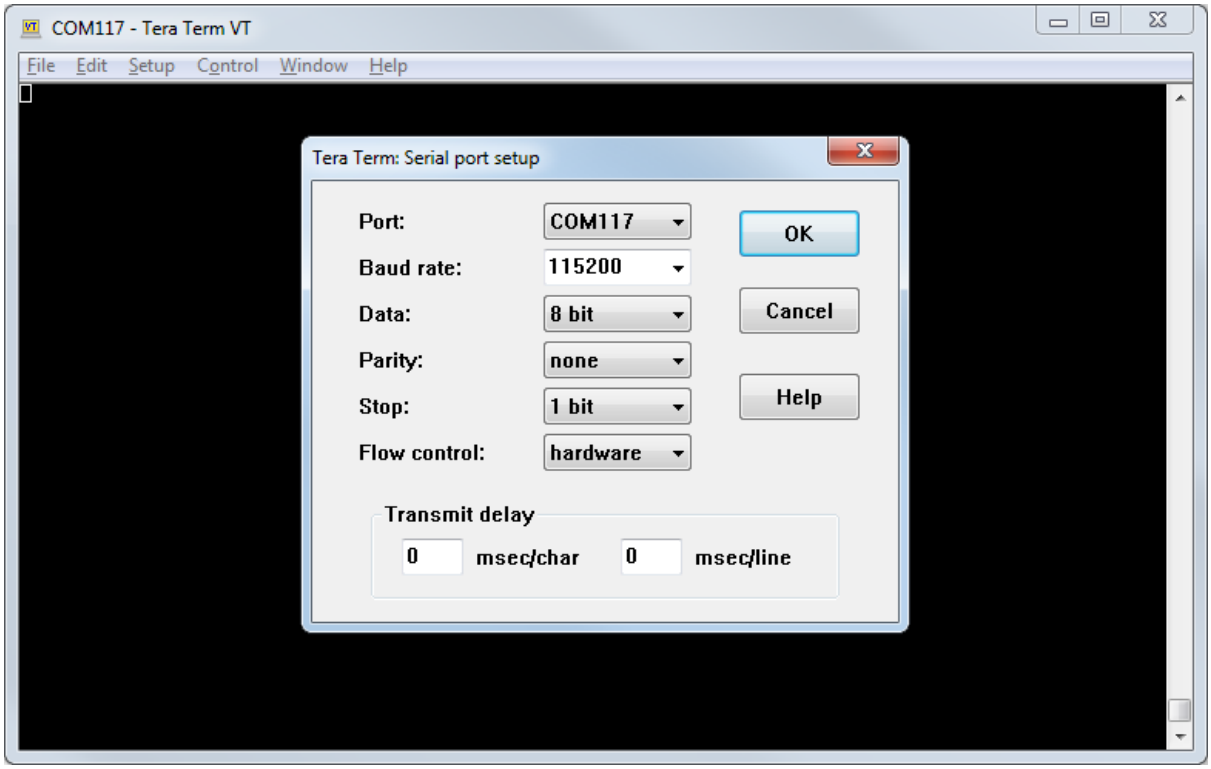



**Figure 4.1. Setup the Serial Port**

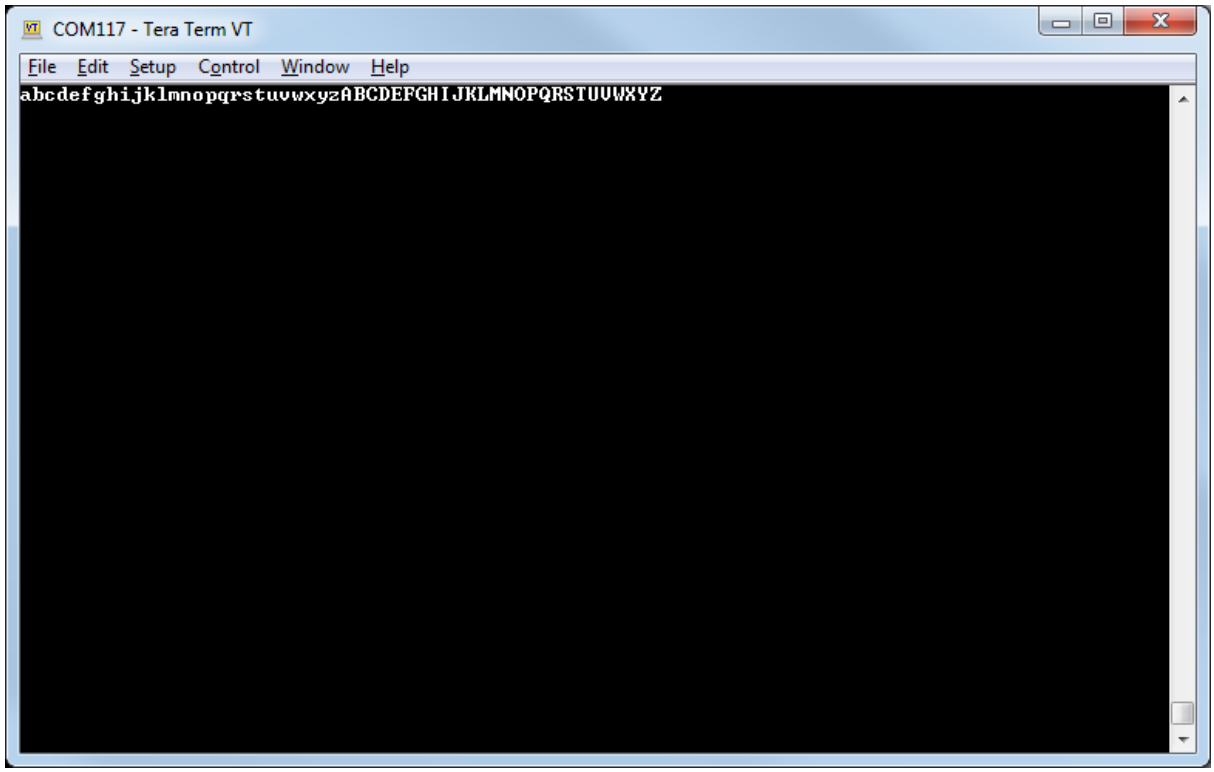



**Figure 4.2. Test the Hardware Flow Control**

### 4.2.2 UART Software Flow Control

UART software flow control (XON/XOFF) is partially supported by the driver, the application should monitor the received data during the data transmission, and pause or resume the transmitter after receiving XOFF or XON.

To test the application connect two EFM32's or use the terminal with a USB-to-UART bridge. After setting up the serial port with software flow control according to the figure as below, type 20 characters, and then the EFM32 will send a 52 bytes message.

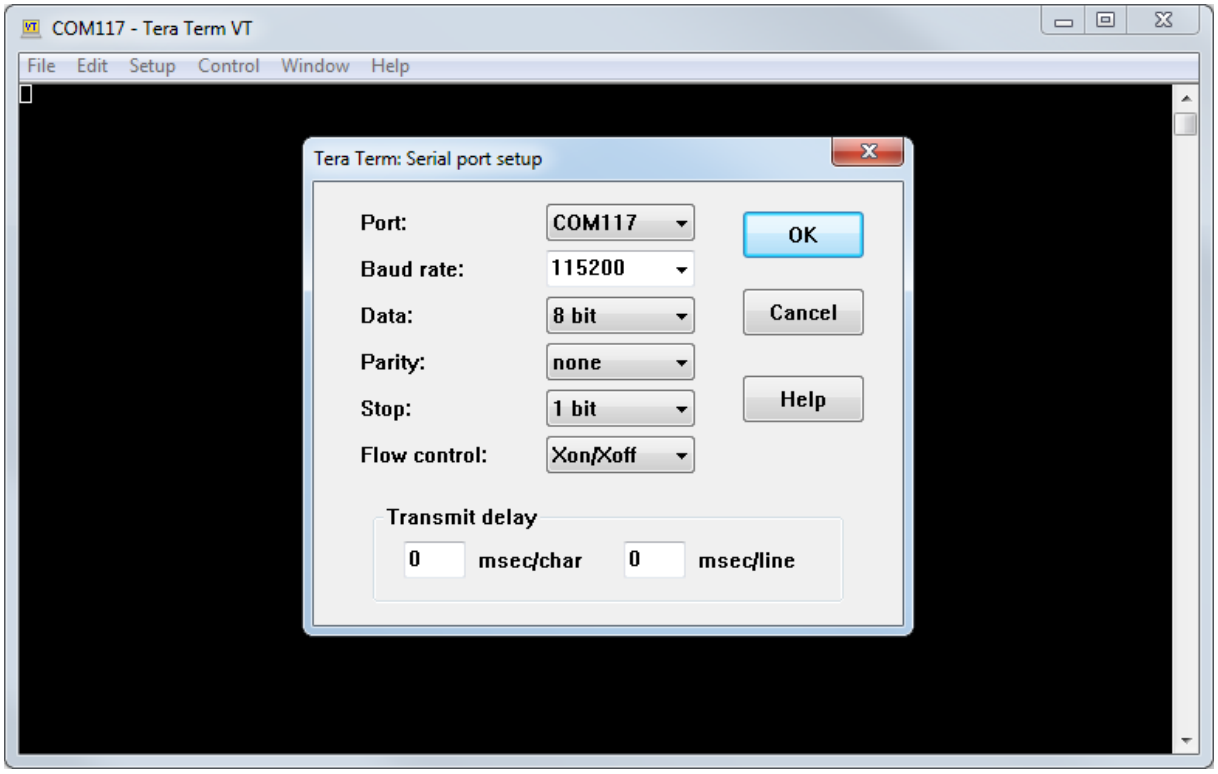



**Figure 4.3. Setup the Serial Port**

# 5. Revision History

### Revision 1.03

January, 2017

- Split AN0059 into AN0059.0 and AN0059.1 for MCU/Wireless Series 0 and MCU/Wireless Series 1, respectively.
- Added the "Device Compatibility" section.
- Added the "Examples with emdrv" section.
- Added support for EFM32 Series 1 and EFR32 Series 1 devices.
- Re-organized the example code structure.
- Added software projects to demo how to implement the UART flow control by using UARTDRV.

### Revision 1.02

September, 2013

- New cover layout

### Revision 1.01

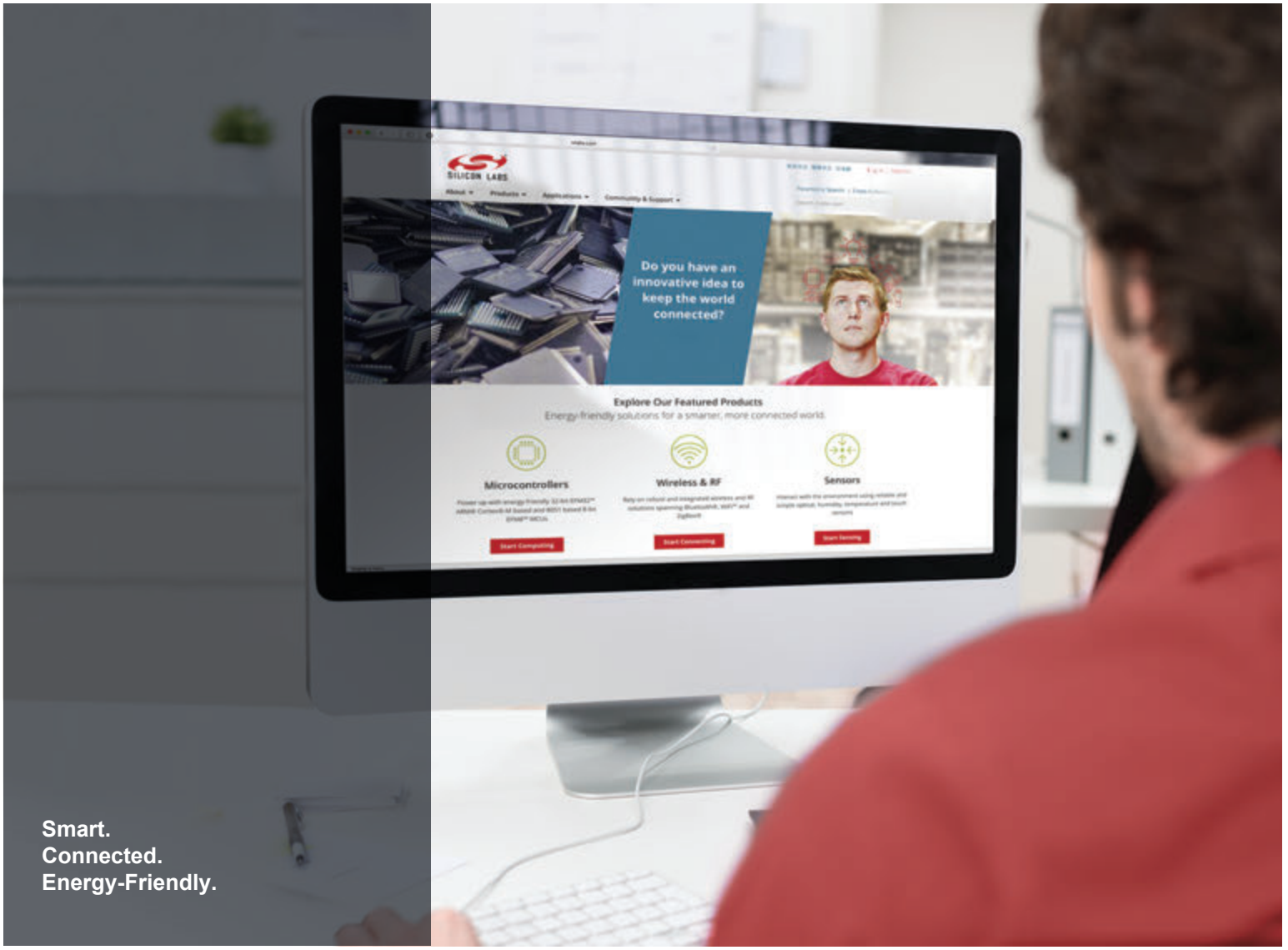
May, 2013

- Added software projects for ARM-GCC and Atollic TrueStudio.

### Revision 1.00

January, 2013

- Initial revision.




**Products**
*www.silabs.com/products*

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

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

**Disclaimer**
Silicon Labs 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 Labs 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 Labs 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 Labs 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 are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Labs. 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 Labs products are not designed or authorized for military applications. Silicon Labs 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.

**Trademark Information**
Silicon Laboratories Inc.® , Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOmodem®, Micrium, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri and others are trademarks or registered trademarks of Silicon Labs. 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.

SILICON LABS®

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

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