AN0059.0: UART Flow Control

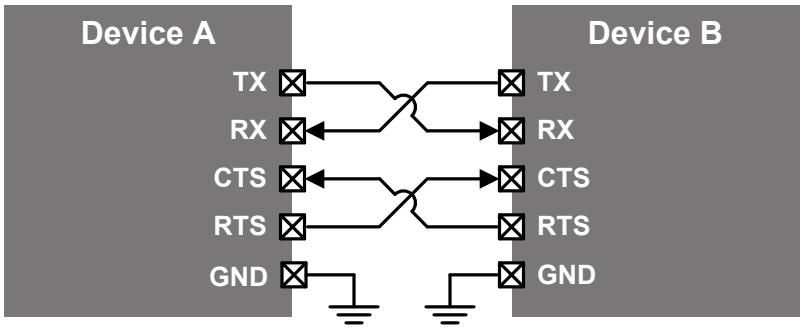
This application note describes how to implement hardware or software flow control for UART.

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, legacy hardware flow control, and software flow control.
- 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 MCU series 0 consists of:

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

EZR32 Wireless MCU series 0 consists of:

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

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 three 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.

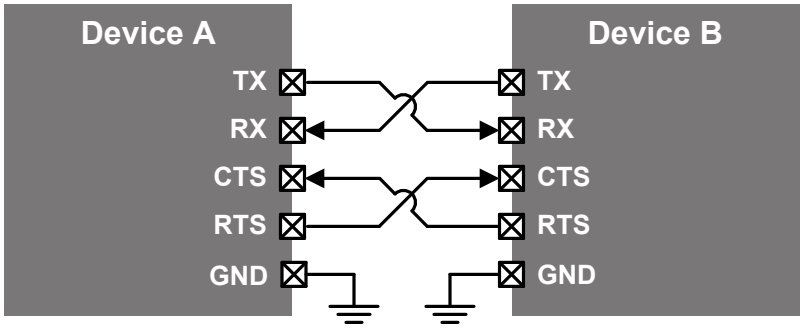


Figure 3.1. Hardware Flow Control

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.

3.2 Legacy Hardware Flow Control

A point of confusion when talking about hardware flow control is that the same names are used for different protocols. Hardware Flow Control sometimes refer to another method for flow control. In this document we shall refer to this second method as legacy Hardware Flow Control to differentiate it from the type discussed in 3.1 Hardware Flow Control. The name legacy is used because this method was actually the early method of implementing flow control.

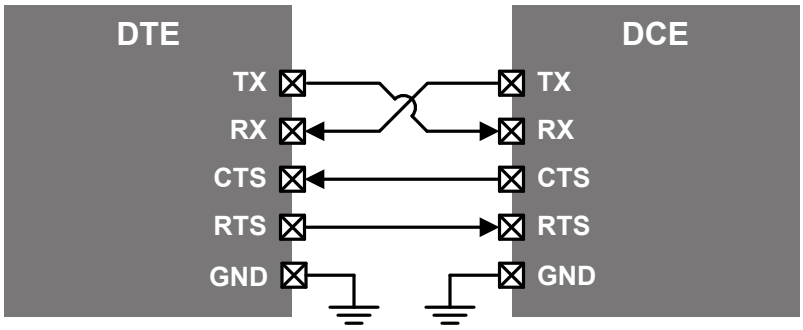


Figure 3.2. Legacy Hardware Flow Control

Legacy Hardware Flow Control still uses two extra wires named RTS and CTS, but the functionality is slightly different. In this scheme the flow control is unidirectional and there is a master/slave relationship (traditionally the master and slave are referred to as DTE (Data Terminal Equipment) and DCE (Data Communications Equipment)). When the master wants to transmit data to the slave it asserts the RTS line. The slave responds by asserting CTS. Transmission can then occur until the slave deasserts CTS, indicating that it needs a temporary halt in transmission. When the master has finished transmitting the entire message it will deassert RTS.

3.3 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 demo of Legacy hardware flow control master or Legacy hardware flow control slave are only included in the examples with emlib. The current demo name will be displayed on LCD.

4.1 Examples with emlib

The software examples in this application note define an application interface to UART Flow Control. The interface is defined in uart_flow_control_s0.c and uart_flow_control_s0.h.

The init function takes a configuration struct where several options can be specified, such as the pin location and polarity of the control signals.

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. At some defined buffer level the program will disable the transmission by deasserting RTS. It keeps RTS deasserted for some time to simulate a slow device.

Figure 4.1. Hardware Flow Control Example

Hardware Flow Control Example: After a certain time the receiver deasserts RTS and the transmitter responds by pausing transmission.

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.

4.1.3 Legacy HW Flow Control

This example works in the same way as the UART Flow Control Example, except that the Legacy Hardware Flow Control Scheme is now used. To test the application, download the example to EFM32 devices, and select master mode for one EFM32 and slave mode for another. Then connect the two according to Figure 3.2 Legacy Hardware Flow Control on page 4.

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 UART driver will assert RTS to start to receive new data, and deassert RTS after receiving is complete. Also the UART driver will read CTS before sending data to see if it is allowed to send data to the other device.

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.

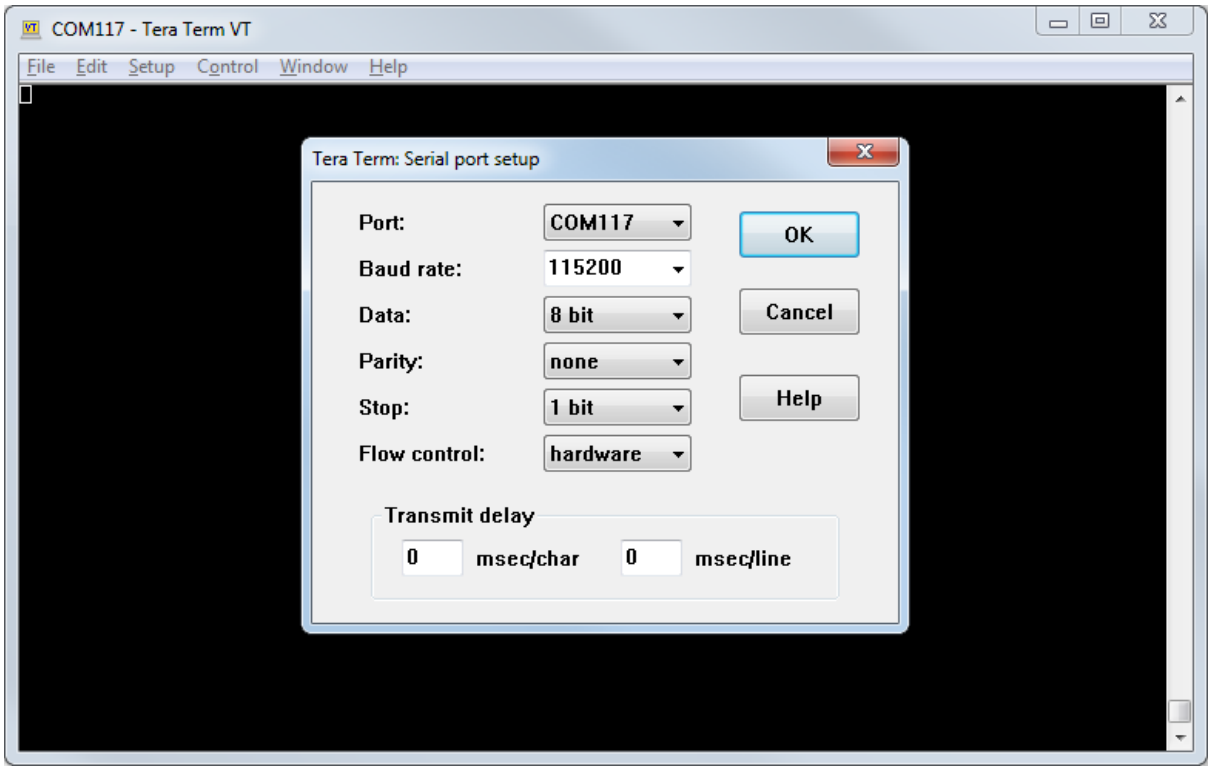


Figure 4.2. Setup the Serial Port

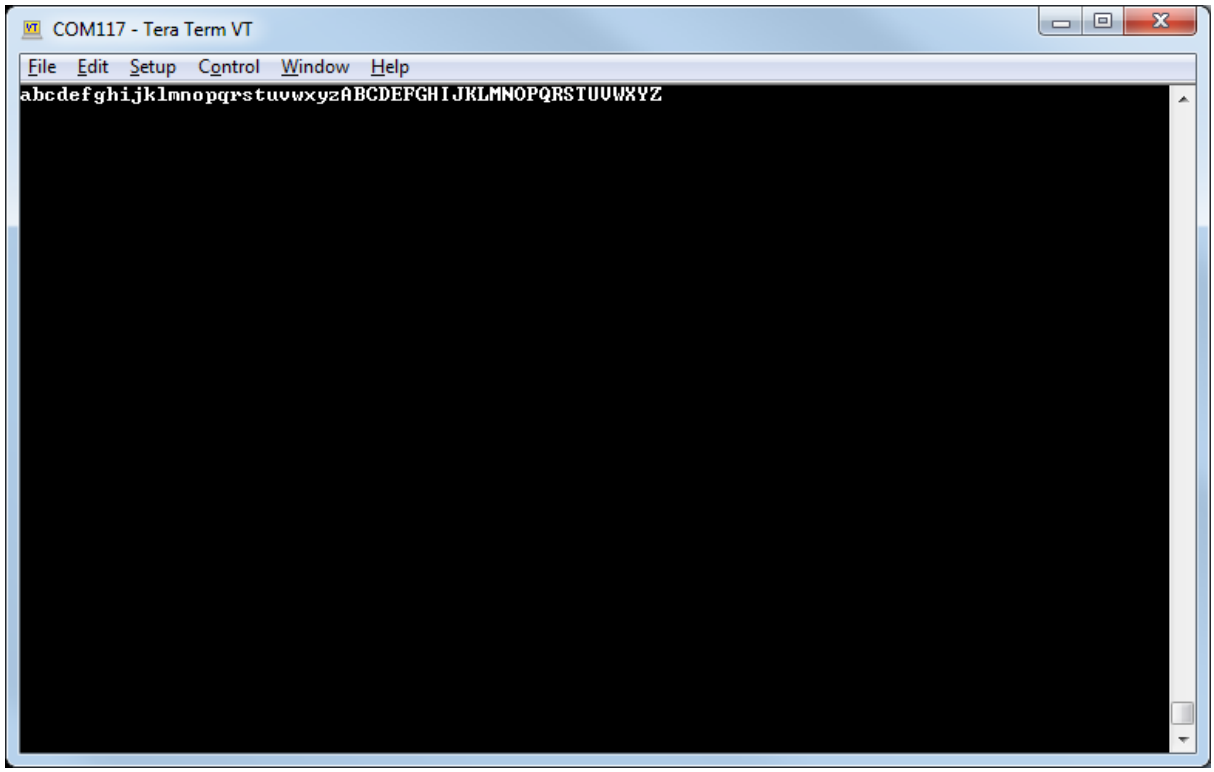


Figure 4.3. Test the Hardware Flow Control

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.

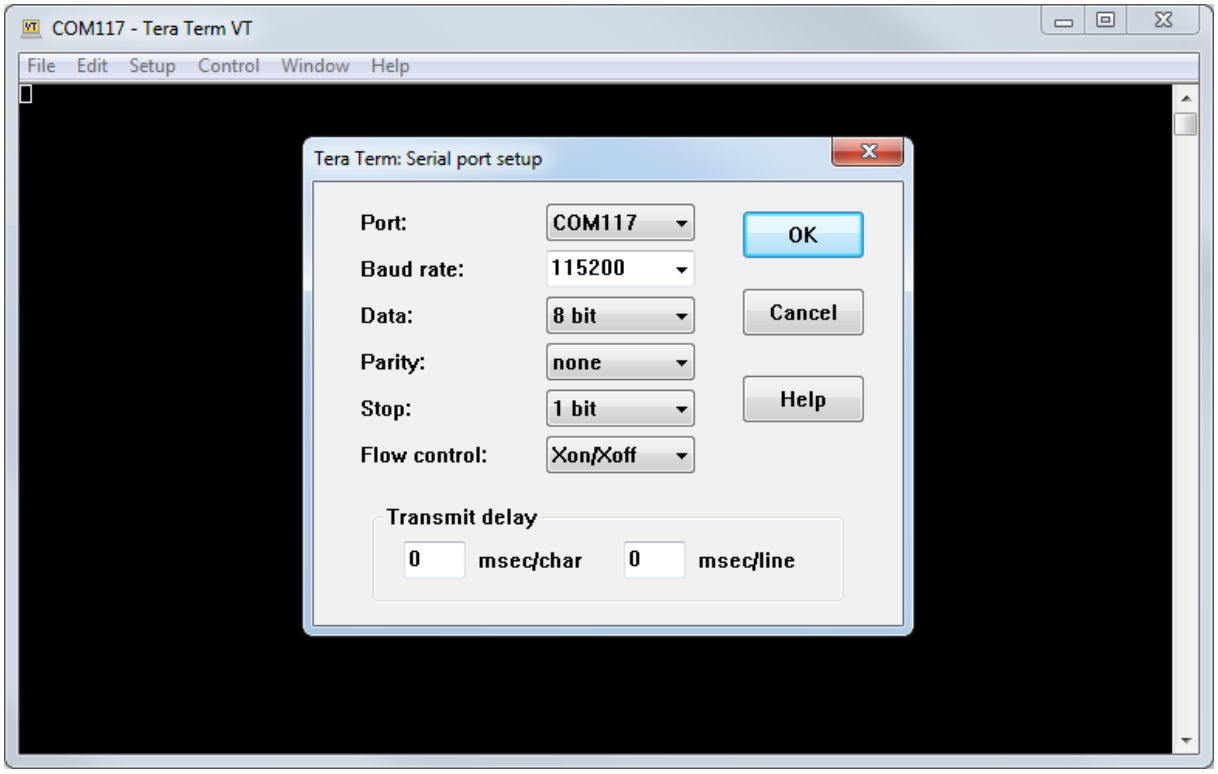


Figure 4.4. Setup the Serial Port

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 all EFM32 Series 0 and EZR32 Series 0 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

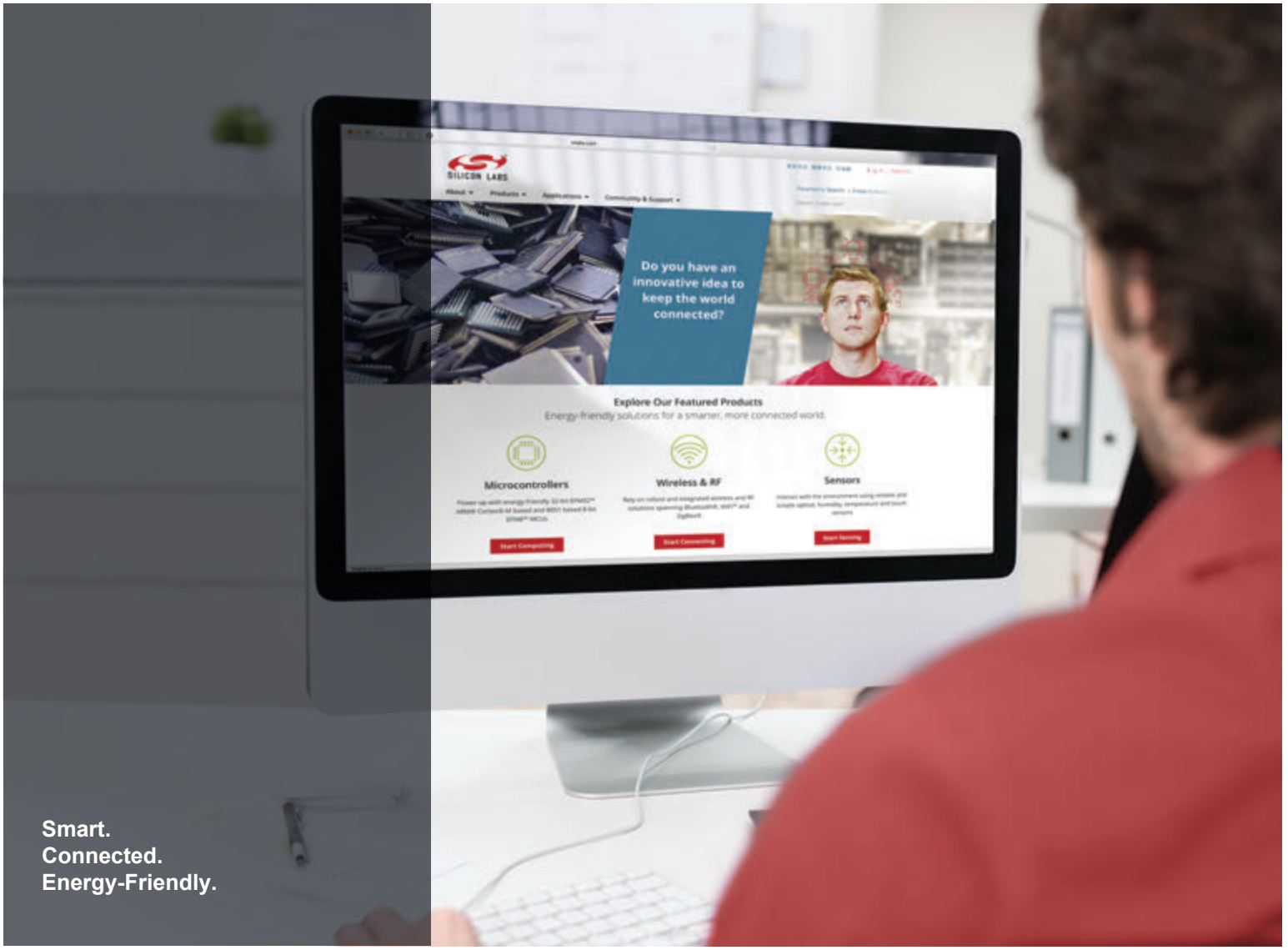
May, 2013

- 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

SILICON LABS®

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

http://www.silabs.com