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

UART Flow Control

AN0059 - Application Note

Introduction

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

This application note includes:

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

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 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* can not 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. What is referred to as hardware flow control uses extra wires. 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.

2 Flow Control Protocols

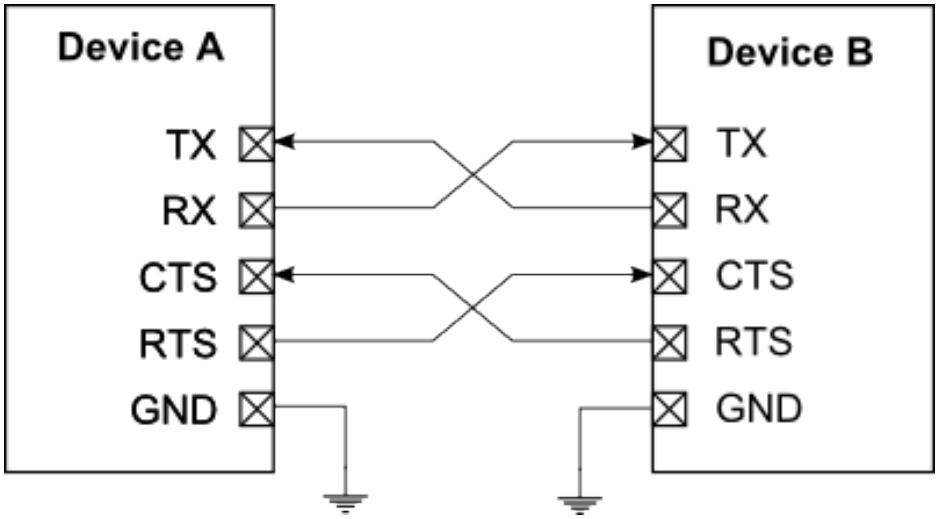
This chapter is describing the three most common ways to implement flow control.

2.1 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 2.1. Hardware Flow Control

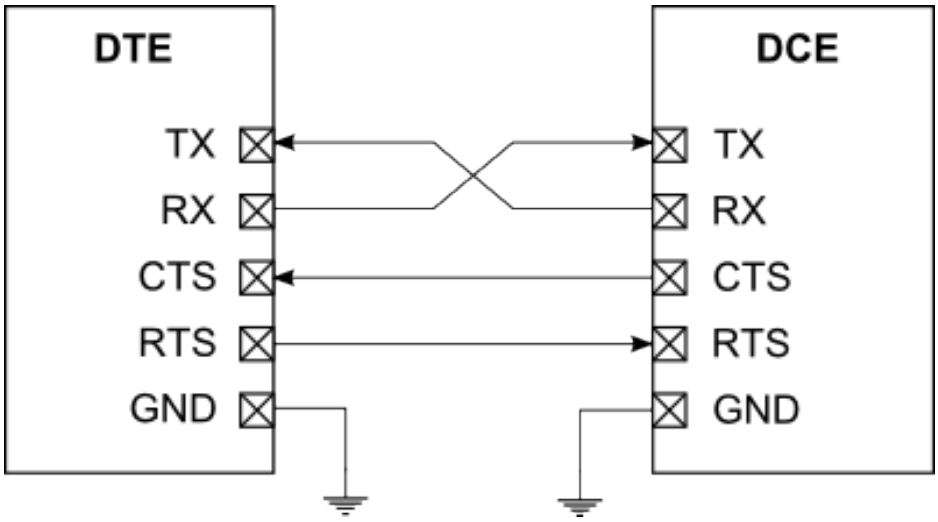


The flow control is bidirectional, meaning both devices can request a halt in transmission. If one of the devices never have to request a stop in transmission (i.e. it is fast enough to always receive data), the CTS signal on the other device can be tied to the *asserted* logic level. The RTS pin on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

2.2 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 Section 2.1 (p. 3). The name legacy is used because this method was actually the early method of implementing flow control.

Figure 2.2. Legacy Hardware Flow Control



Legacy Hardware Flow Control still use 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 CTE (Computer Terminal Equipment) and (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.

2.3 Software Flow Control

Software flow control does not use extra wires. Only 3 wires are required (RX,TX,GND). Transmission is started and stopped by sending special control flow characters. The control flow characters are sent over the normal TX and RX lines. The control flow 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*.

3 Software Examples

The software examples in this application note define an application interface to UART Flow Control. The interface is defined in uart_flow_control.{c,h}. The flow control functions are prefixed with `UART_FC_`.

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 `UART_FC_startTX()`. 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.

Figure 3.1. Hardware Flow Control Example

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

3.1 UART 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.

To test the application connect two EFM32's according to Figure 2.1 (p. 3) .

3.2 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 master example to one EFM32 and the slave example to another. Then connect the two according to Figure 2.2 (p. 4) .

4 Revision History

4.1 Revision 1.03

2014-05-07

Updated example code to CMSIS 3.20.5

Changed source code license to Silicon Labs license

Added project files for Simplicity IDE

Removed makefiles for Sourcery CodeBench Lite

4.2 Revision 1.02

2013-10-14

New cover layout

4.3 Revision 1.01

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

4.4 Revision 1.00

2013-01-03

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