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

USART/UART - Asynchronous mode

AN0045 - Application Note

This application note describes how to configure the EFM32 UART or USART to operate in asynchronous mode.

An included software example for the EFM32GG-DK3750 Giant Gecko Development Kit shows how to implement interrupt driven receive and transmit, utilizing the on-board RS-232 transceiver.

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 Universal Asynchronous Receive Transmit (UART)

1.1 Basic Theory

A UART is a well established standard for low cost, low speed serial communications over a simple 2-wire (plus ground) interface.

Asynchronous communications differs from synchronous communications in that synchronization between transmitter and receiver are encoded into the transmitted signal, rather than using a separate wire to transfer the transmitter clock to the receiver.

Embedding the synchronization information in the data reduces the cost of cables and connectors, and may also be beneficial on a space constrained PCB or if one wants to keep the pin-usage low. On the other side adding synchronization information to the datastream increases overhead, causing the effective data rate to be lower than the baud rate.

Normally, asynchronous communication modes facilitate somewhat lower data rates compared with synchronous modes. Some of the reason is the above mentioned overhead, but also because asynchronous communications may impose stronger requirements on the transceivers and the transmission lines between receiver and transmitter.

Low cost and low power transceivers usually don't have advanced clock recovery mechanisms, but simply rely on the combination of oversampling and that the receiver and transmitter clock frequencies are sufficiently close.

1.2 RS-232

UART does not specify any electrical characteristics such as signal levels etc. Instead, several separate electrical interface standards can be applied. Most common is RS-232, but other well known standards include RS-422, RS-485, and also some standards that don't use electrical signalling such as IrDA.

In this application note, the included software example uses the RS-232 transceiver that is included on the Development Kit.

1.3 Using the EFM32 UART/USART

The information necessary to configure and use the UART/USART modules on an EFM32 microcontroller are contained in the device family reference manual. This application note also presents some further details and clarifications.

1.3.1 Clock Source

Often, the HFRCO is too unprecise to be used for communications. So using the HFXO with an external crystal is recommended when using the EFM32 UART/USART.

In some cases, the internal HFRCO can be used. But then careful considerations should be taken to ensure that the clock performance is acceptable for the communication link.

1.3.2 Baud Rate Calculation

The baud rate is given by the following expression:

Baud rate

$$br = f_{HFPERCLK} \cdot \frac{1}{OVS} \cdot \frac{1}{1 + \frac{DIV}{4}} \tag{1.1}$$

Where

- br is baud rate,
- $f_{HFPERCLK}$ is the frequency of the HFPERCLK branch of the high frequency clock tree (See figure on CMU Overview in device family specific reference manual),
- OVS is the oversampling factor, and
- DIV is the configurable part of the fractional divider in the UART/USART module.

When rearranged, one can compute a clock divider setting that will obtain a wanted baud rate by the following formula:

Clock divisor

$$DIV = 4 \cdot \left(\frac{f_{HFPERCLK}}{br \cdot OVS} - 1 \right) \quad (1.2)$$

The clock divider is a fractional divider dividing by (1+DIV/4) where DIV is a 15 bit value ranging from 0 to 32767. I.e. the clock can be divided by a factor from 1 to 8192.75. Depending on the configurable oversampling factor the baud rate is given by a further division by a factor of 4, 6, 8 or 16. This results in a baud rate that is the clock frequency divided by 4 to 131,084. If the HFXO is run at 32 MHz, baudrates between 8 Mbps and 244.11 bps can be generated as long as the HFPERCLK prescaler is set to 1.

It is worth noting that the equations in this application note differs somewhat from the reference manual. The reason is that the reference manual refers to CLKDIV which is the entire 32-bit register value, of which only the 15-bit wide bitfield DIV is actually used to control the fractional divider. In this document, the bitfield DIV is consistently used.

2 Software Example

The included software example is made for the EFM32 Giant Gecko Development Kit, EFM32GG-DK3750. However, with minor modifications the project will also work on our other EFM32 development kits. It can also be ported to the starter kits. But because the starter kits don't include RS-232 line drivers, please ensure that signal levels are compatible before establishing a communication link between two parties. Connecting the EFM32 UART directly to a PC serial port will damage the EFM32.

The kit's on-board RS-232 transceiver is used to demonstrate a possible interrupt based asynchronous mode configuration of an EFM32 U(S)ART peripheral.

The example uses interrupt driven transmit and receive. When transmitting a block of data, the data is first copied into a transmit queue. The U(S)ART TXBL interrupt is enabled. When the UART is ready to transmit, the TXBL interrupt goes high. The interrupt handler function fetches one byte from the transmit queue and copies it to the UART transmit buffer (UARTn->TXDATA). While transmitting, the CPU is free to perform other tasks. In the example project, the MCU spends this time in Sleep Mode (EM1).

The same principle is used on receive. When an RXDATAV interrupt is received, the Rx interrupt handler copies the incoming data to a receive queue.

2.1 Kit Configuration

The development kit's on-board RS-232 line driver is used. This transceiver is normally disconnected from the MCU, so before it can be used, it must be enabled by software. To do this, the kit library functions are used. The kit libraries are included in the kit software packages that can be installed via Simplicity Studio. Documentation is found under EM_BSP_COMMON in Software Documentation in Simplicity Studio.

The RS-232 transceiver is connected to UART1, location 2 on the EFM32.

2.2 Instructions

A serial cable and terminal emulator software is required to try this example. On Windows, the Open Source terminal Tera Term can be used.

First, connect a serial cable between a computer and the 9-pin RS-232 connector on the development kit. Configure the serial port as follows

- Baud rate = 115 200
- Data bits = 8
- Parity = none
- Stop bits = 1
- Flow control = none

before opening a connection with the terminal emulator.

One should also configure the terminal emulator to handle new line in the same way as the SW example. In Tera Term the proper configuration is to use LF on receive and CR+LF on transmit. If this can't be configured on the chosen emulator, the example can of course be altered to match the emulator settings.

When connected, start typing. After entering some characters, press '.' which is predefined as a "termination character" causing the MCU to echo the contents of the RX queue back out on the UART.

2.3 Transmit

Transmit is handled by two functions: **uartPutData()** and **UART1_TX_IRQHandler()**.

uartPutData() copies data to send into a transmit queue. The queue is implemented as a circular buffer. The data is copied into the queue starting at the write index (wrI). When finished, the pending byte counter is updated. Finally the TX interrupt for the UART is enabled.

UART1_TX_IRQHandler() reacts when the TXBL interrupt goes high, signalling that the UART transmit buffer is empty. When this happens, one byte is copied from the read index (rdI) position in the TX queue into the UART transmit buffer. The read index is updated, and the pending byte counter is decremented. If the transmit queue becomes empty, the TXBL interrupt is disabled.

2.4 Receive

In the same way, receive is also handled by two functions: **UART1_RX_IRQHandler** and **uartGetData**.

UART1_RX_IRQHandler() reacts on the RXDATAV interrupt, meaning that the UART RX buffer contains valid data. When this happens, the incoming byte is copied from the UART RX buffer into the RX queue. The queue write index (wrI) is updated, and the pending byte counter is incremented. The IRQ handler will also disable the TXBL interrupt if the transmit queue becomes empty.

uartGetData() pulls a number of bytes from the receive queue. The copy starts at the read index (rdI). When data is copied, the read index is updated and the pending byte counter is decremented.

Also, for the sake of the example, the RX interrupt handler checks if the received byte is a predefined termination character.

3 Revision History

3.1 Revision 1.04

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

Corrected flags for interrupt enabling and clearing in software example

3.2 Revision 1.03

2013-10-14

New cover layout

Removed unnecessary read of IF in TX IRQ Handler

3.3 Revision 1.02

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

3.4 Revision 1.01

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

3.5 Revision 1.00

2012-06-28

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 Equations

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