UART Bootloader

AN0003 - Application Note

Introduction

This application note is intended for users of the EFM32 UART bootloaders. The bootloader enables users to program the through an UART without the need for a debugger. In addition to booting user applications, it offers a destructive write mode, which allows the user to overwrite the bootloader so that the entire flash space can be used for user applications. The contents of the flash can be verified through a CRC checksum and debug lock can be enabled to protect IP. Because the bootloader uses the established XMODEM-CRC protocol for data upload, any serial terminal program can be used to communicate with the bootloader.

The UART bootloader is preprogrammed in all EFM32 devices. Devices which include USB also include a USB bootloader. The bootloader for USB enabled devices is covered in application note "an0042_efm32_usb_uart_bootloader".

This application note includes:

- This PDF document
- Source files (zip)
  - Example c-code
  - IAR EW project
- IAR linker files for applications
- Binary images of the bootloader

1 Starting the UART Bootloader

1.1 Entering bootloader mode

To enter the bootloader DBG_SWCLK must be pulled high and the EFM32 must be reset. If DBG_SWCLK is low, the bootloader will check the application in the flash. If the application space contains a valid application the bootloader will continue to run the application. If there is not a valid application present, the bootloader will sleep in EM2 to to conserve power, while periodically checking the bootloader pins.

Note

DBG_SWDCLK has an internal pull-down. Leaving this pin unconnected will not invoke the bootloader.

Note

Earlier revisions of the bootloader used both SWDIO and SWDCLK pins to enter the bootloader. You can still enter the bootloader by pulling the SWDIO line, but debug locking will not work.

1.2 Initializing communication with the UART bootloader

The UART bootloader use GPIO pins E11 and E10 for UART communication. The UART use 1 stop bit, no parity and 8 data bits. To enable a wide variety of different terminals the bootloader uses autobaud. The autobaud functionality senses the baudrate used by the terminal program and adjusts accordingly. This is done by sending one uppercase 'U' to the bootloader. The bootloader senses the timing between bits and adjusts its own prescaler to match the sensed baudrate. The bootloader works with baudrates in the range from 57600 to 460800.

Note

EFM32GG, EFM32LG, EFM32G and EFM32TG parts use USART0 location 0.

EFM32TG108Fxx and EFM32ZGxxxFxx parts does not have USART0, also EFM32TG110Fxx does not support location 0 for USART0. These parts all use LEUART0, location 3. Note that this location overlaps with the regular SWD port. Therefore, when using these parts you should use a 4K pull-up on SWDCLK.

Once the bootloader has been initialized it will print the bootloader version and chip unique ID:

```
1.40 ChipID: F08AB6000B153525
```

Note

Bootloader versions prior to version 1.40 does not print the bootloader version, only the chip unique ID.

1.3 Command line interface

The command line interface uses single letter characters as commands. The following commands are supported:

u Upload application. This command lets the user upload an application to the flash, while keeping the bootloader intact. For an application to work correctly it must use a linker file which places the application start address at 0x800 for EFM32G, EFM32TG and EFM32ZG parts and at 0x1000 for EFM32GG, EFM32LG and EFM32WG parts. The application is transferred using the XMODEM-CRC protocol.

d Destructive upload. This command lets the user upload an application to flash, overwriting the bootloader. The application is transferred using the XMODEM-CRC protocol.

t Upload to user page. This command lets the user write to the user information page. The data is uploaded using the XMODEM-CRC protocol.

p Upload to lock page. This command lets the user write to the lock bits information page. The data is uploaded using the XMODEM-CRC protocol.

b Boot application. This command will start the uploaded application.

l Debug lock. This command sets the debug lock bit in the lock page. The EFM32 will be locked for debugging.

v Verify flash checksum. This command calculates the CRC-16 checksum of the entire flash and prints it. This is suitable for use in conjunction with the 'd' command.

c Verify application checksum. This command calculates the CRC-16 checksum of the application and prints it. This is suitable for use in conjunction with the 'u' command.

n Verify user page checksum. This command calculates the CRC-16 checksum of the user page and prints it. This is suitable for use in conjunction with the 't' command.

m Verify lock page checksum. This command calculates the CRC-16 checksum of the lock page and prints it. This is suitable for use in conjunction with the 'p' command.

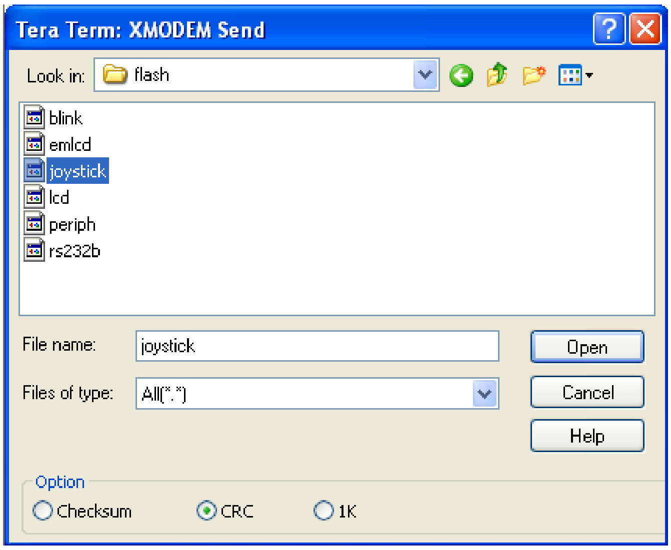
r Reset the EFM32

2 Uploading applications

To upload an application to the EFM32 either the 'u' or 'd' command must be used. After pressing the key use the terminal software built-in support for XMODEM-CRC to transfer the file. Any terminal software may be used, as long as it supports XMODEM-CRC transfers.

Figure 2.1 (p. 4) Shows an example of transferring a file using the built in transfer support in Tera Term.

Figure 2.1. Transferring a file using XMODEM-CRC with Tera Term on Windows XP.



2.1 Creating applications for use with the bootloader

There are two possibilities when uploading applications using the bootloader; destructive and regular upload. Destructive upload will overwrite the bootloader. No additional steps are required for creating applications in this case. Regular uploading keeps the bootloader. This allows future upgrades using the bootloader. However; the applications must be prepared for this to work. For applications to work with the bootloader they must be created with a starting address of 0x800 for EFM32G, EFM32TG and EFM32ZG parts and at 0x1000 for EFM32GG, EFM32LG and EFM32WG parts. The reason for this is that the bootloader itself occupies this flash area. To achieve this the linker file must be changed from the default flash start address of 0x0.

2.1.1 Creating an application with IAR

To create an application using IAR use the included linker files for your project. This will set up the correct starting address for the binary. In the project options menu, select "Output Converter" and "Generate additional output". Select the "binary" output format. The resulting binary can be used with the UART Bootloader.

2.1.2 Creating an application with Keil uVision 4/MDK-ARM

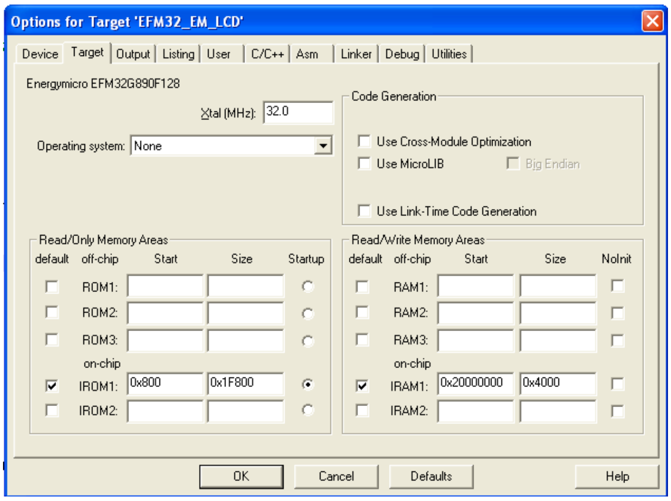
To create applications with Keil uVision 4/MDK-ARM, you must first change the target settings for your project. In the options dialog change IROM1 to a start of 0x800 or 0x1000 and subtract 0x800 or 0x1000 from the size field. (0x800 or 0x1000 depends on which part is being used).

To generate a binary output file, you can use the command line utility "fromelf.exe", that's usually installed under

```
C:\Keil\ARM\BIN40\fromelf.exe
```

See the "Realview Utilities Guide" in the uVision Help for details.

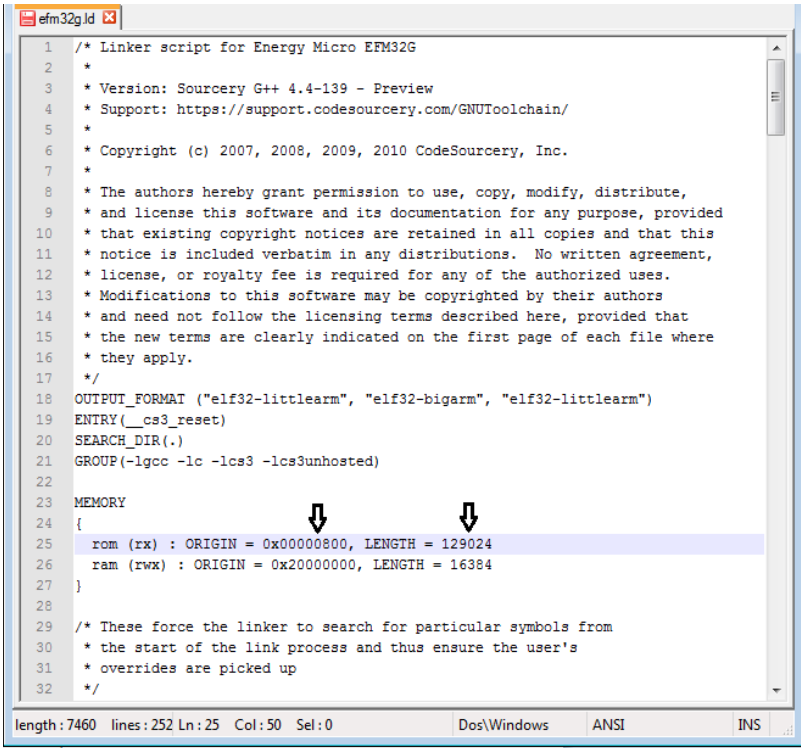
Figure 2.2. Setting up Keil uVision 4/MDK-ARM



2.1.3 Creating an application with Eclipse/GCC/Sourcery CodeBench

To create an application with Eclipse, GCC or Sourcery CodeBench that will work alongside the bootloader, the linkerfile need to be modified. For application notes and example projects the location of the linkerfile is specified in the Makefile included with the software project. In the linkerfile MEMORY command, change the rom ORIGIN to 0x00000800 or 0x00001000, the length should also be changed accordingly as in Figure 2.3 (p. 5) .

Figure 2.3. Application Start Address in Eclipse/gcc/cs linker file



efm32g.ld

```
/* Linker script for Energy Micro EFM32G
 *
 * Version: Sourcery G++ 4.4-139 - Preview
 * Support: https://support.codesourcery.com/GNUToolchain/
 *
 * Copyright (c) 2007, 2008, 2009, 2010 CodeSourcery, Inc.
 *
 * The authors hereby grant permission to use, copy, modify, distribute,
 * and license this software and its documentation for any purpose, provided
 * that existing copyright notices are retained in all copies and that this
 * notice is included verbatim in any distributions.  No written agreement,
 * license, or royalty fee is required for any of the authorized uses.
 * Modifications to this software may be copyrighted by their authors
 * and need not follow the licensing terms described here, provided that
 * the new terms are clearly indicated on the first page of each file where
 * they apply.
 */
OUTPUT_FORMAT ("elf32-littlearm", "elf32-bigarm", "elf32-littlearm")
ENTRY(__cs3_reset)
SEARCH_DIR(.)
GROUP(-lgcc -lc -lcs3 -lcs3unhosted)

MEMORY
{
  rom (rx)  : ORIGIN = 0x00000800, LENGTH = 129024
  ram (rwx) : ORIGIN = 0x20000000, LENGTH = 16384
}

/* These force the linker to search for particular symbols from
 * the start of the link process and thus ensure the user's
 * overrides are picked up
 */
```

length : 7460   lines : 252 Ln : 25   Col : 50   Sel : 0   Dos\Windows   ANSI   INS

Note

If you need to debug your application while using one of these linker files, you must explicitly set the position of the vector table in your code. This can be done with:

```
SCB->VTOR=0x800; // EFM32G, EFM32TG and EFM32ZG parts
```

or

```
SCB->VTOR=0x1000; // EFM32GG, EFM32LG and EFM32WG parts
```

In the released application this is not necessary as VTOR is set by the bootloader itself, before starting the application. (See Boot.c for details.)

2.2 Uploading applications

The 'u' command will upload an application. Use your terminal software to transfer the application binary to the chip. After completing the upload you might wish to verify the correctness by calculating the CRC-16 on the uploaded binary. This can be achieved by the 'verify application checksum' command (See Section 3.1 (p. 7)). To start the application from the bootloader use the 'boot' command ('b' - see Section 4.1 (p. 8)).

2.3 Destructive upload

The 'd' command will start a destructive upload. Use your terminal software to transfer the binary to the chip. Destructive upload differs from regular uploads in that it overwrites the bootloader. This enables you to upload another bootloader, or, if a bootloader is not needed, to reclaim the flash occupied by the bootloader. After completing the upload you might wish to verify the correctness by calculating the CRC-16 checksum. This can be achieved by the 'verify flash content' command (see Section 3.2 (p. 7)). To start the application, you can use the 'reset' command ('r' - see Section 4.2 (p. 8)).

2.4 Writing to the user information page

The 't' command enables you to write data to the user information page. Use your terminal software to transfer the user data to the user information page.

2.5 Writing to the lock bits information page

The 'p' command enables you to write data to the lock bits information page. Use your terminal software to transfer the user data to the user information page. This command enables you to lock pages in flash from writing and erasing, but does not protect contents. See the reference manual for details on lock bits.

3 Verify upload

Note

XMODEM-CRC transfers data in blocks of 128 bytes. If the binary's size is not a multiple of 128 bytes, the terminal program will pad the remaining bytes. Refer to the terminal program's documentation for details.

3.1 Verify application checksum

The 'c' command will calculate and print the CRC-16 checksum of the flash from base 0x800 or 0x1000 (beginning of application) to the end of flash space.

3.2 Verify flash content

The 'v' command will calculate and print the CRC-16 checksum of the flash from base 0x0 (beginning of flash space) to the end of the flash space.

4 Miscellaneous commands

4.1 Boot application

The 'b' command will boot the uploaded application in a similar manner as if the bootloader had not been enabled by pulling the debug pins high. The bootloader does this by first setting the Cortex-M3's vector table to the base of the application. Then, it reads out the first word in the new vector table and sets SP accordingly. Finally, it performs a vector reset by setting PC to the value defined by the reset vector.

Note

The bootloader configures TIMER, USART, CMU and GPIO during it's normal operation. These settings are kept when booting the application using this command. However, if the bootloader is not entered by asserting the bootloader pins, these registers are not modified. This is the typical situation.

4.2 Reset the Device;

The 'r' command resets the device. If this command is issued after a destructive upload, the new binary will be started. If this command is issued after a regular upload and the debug pins are not pulled high, the application will start. Otherwise, the bootloader will restart.

4.3 Debug lock

The 'l' command will lock the debug interface. After locking regular debugging facilities will not be accessible, only a device erase is possible through the debug interface.

Note

The device must be reset once before the debug interface is locked. This command will return 'OK' if the locking was successful, 'Fail' otherwise. If debug locking fails, please make sure that SWDIO is not connected and SWDCLK is tied high.

5 Compiling the Bootloader

Along with this application note is the source code for the bootloader itself. It is possible to use this source code to compile your own bootloader. A few remarks are important to be aware of when using this source code.

The compiled bootloader must fit within the configured area of flash. For Gecko, Tiny Gecko and Zero Gecko this is 0x800 bytes. For Leopard Gecko, Giant Gecko and Wonder Gecko it is 0x1000 bytes.

The source projects are only available for IAR. You can use the source files with another IDE, but you have to make sure the compiled bootloader fits within the allocated size.

In IAR, the source code must be compiled in Release configuration, see Figure 5.1 (p. 9). If compiled in Debug configuration the compiled program becomes too large.

For Tiny Gecko there are two projects. EFM32TG108Fxx and EFM32TG110Fxx devices should use bootloader-tg-small. Other Tiny Gecko devices should use bootloader-tg.

Figure 5.1. Compile in Release Mode

6 Revision History

6.1 Revision 1.67

2013-09-03

New cover layout

6.2 Revision 1.66

2013-07-31

Clarified which parts that use LEUART0.

6.3 Revision 1.65

2013-05-08

Added section about compiling the bootloader

6.4 Revision 1.64

2012-11-12

Fixed the note that wrongly stated; EFM32TG110 does not have USART0.

6.5 Revision 1.63

2012-04-25

Fixed missing comment ';' in assembly files.

6.6 Revision 1.62

2012-04-20

Added new prebuild binaries, previous ones did not work with devices with less than 32k flash.

Adapted software projects to new peripheral library naming and CMSIS_V3.

Added section on how to modify application code linker file for Eclipse/GCC/Sourcery CodeBench.

6.7 Revision 1.61

November 16th, 2011.

Added missing EFM32LG binaries.

Renamed application note.

6.8 Revision 1.60

September 22nd, 2011.

Added support for EFM32GG and EFM32LG parts.

No functional differences for other bootloaders.

6.9 Revision 1.50

June 7th, 2011.

Bootloader USART moved on TG110 and TG108 devices to F0, F1 which overlaps with the bootloader pins.

No functional differences for other bootloaders.

6.10 Revision 1.40

April 18th, 2011.

Bootloader pins changed from SWDIO and SWDCLK to only SWDCLK.

Debug locking fixed.

Added support for Tiny devices.

Print version number along with chip ID.

Added linker file to constrain flash and RAM usage automatically.

Moved part-specific configuration to config.h.

Moved some functionality to flash to reduce SRAM footprint.

6.11 Revision 1.3

September 27th, 2010.

Increased download speed by using DMA.

Better handling of high bitrates.

Added binary for testing.

6.12 Revision 1.22

September 17th, 2010.

Updated main heading on front page, no code changes.

Updated references for which devices this bootloader appnote applies to.

6.13 Revision 1.21

September 2th, 2010.

Updated main heading on front page, no code changes.

6.14 Revision 1.20

August 31th, 2010.

Improved speed for programming. Fixed bug where applications using the RTC would crash. This bug was introduced in version 1.10.

6.15 Revision 1.10

April 14th, 2010.

Bootloader now uses EM2 while idling.

6.16 Revision 1.00

January 8th, 2010.

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, the Silicon Labs logo, Energy Micro, EFM, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or 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