EFM32®

... the world's most energy friendly microcontrollers

USB/UART Bootloader

AN0042 - Application Note

Introduction

This application note is intended for users of the bootloader in USB enabled EFM32s. The bootloader enables users to program the EFM32 through an UART or an USB CDC class virtual 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 USB/UART bootloader is preprogrammed in all EFM32s with an USB periheral.

This application note includes:

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

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 Starting the 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 run this application. If a valid application is not 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.

1.2 Initializing communication with the bootloader

Bootloader communication is initialized by transmitting an uppercase 'U' on the UART (which starts the autobaud algorithm), or by enumeration of the USB CDC device. Whichever happens first will govern the rest of the bootloader operation. The 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 terminal emulators the bootloader uses an autobaud algorithm (not applicable if using the USB serial port). Upon reception of an uppercase 'U' on the UART, the host baudrate will be measured and UART baudrate adjusts accordingly. The autobaud algorithm works with baudrates in the range from 2400 to 115200. Once the bootloader has completed the autobaud sequence, it will print the bootloader version and chip unique ID:

```
BOOTLOADER version x.yy, Chip ID F08AB6000B153525
```

If using the USB UART you will not see the version string. Issue command 'i' (see below) to check version information.

Note

If neither the autobaud algorithm nor USB enumeration completes within 30 seconds, the chip will be reset.

If using a Windows host and you want to use the USB CDC virtual UART, a USB CDC device driver must be installed. This is most easily done by opening the Device Manager, rigth-clicking on the device and selecting *Update Driver Software...* Do a manual install and browse to the location of the *EFM32-Cdc.inf* file.

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 0x4000. 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

i Print bootloader version and chip unique ID.

Note

The 'b' and 'r' commands will when using the USB CDC virtual UART, delay 7 seconds before the commands are actually performed. The first 5 seconds to allow an operator to disconnect the UART connection, then the bootloader performs an USB soft disconnect before waiting an additional 2 seconds to allow time for the host OS to tear down the USB CDC driver stack.

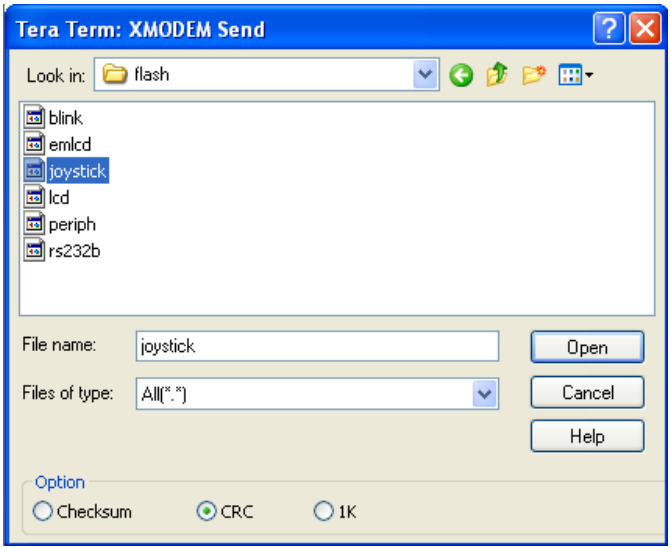
The reason for this procedure is that most terminal emulators are unaware of the concepts of USB CDC serial ports, which can be detached any time.

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 0x4000. 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

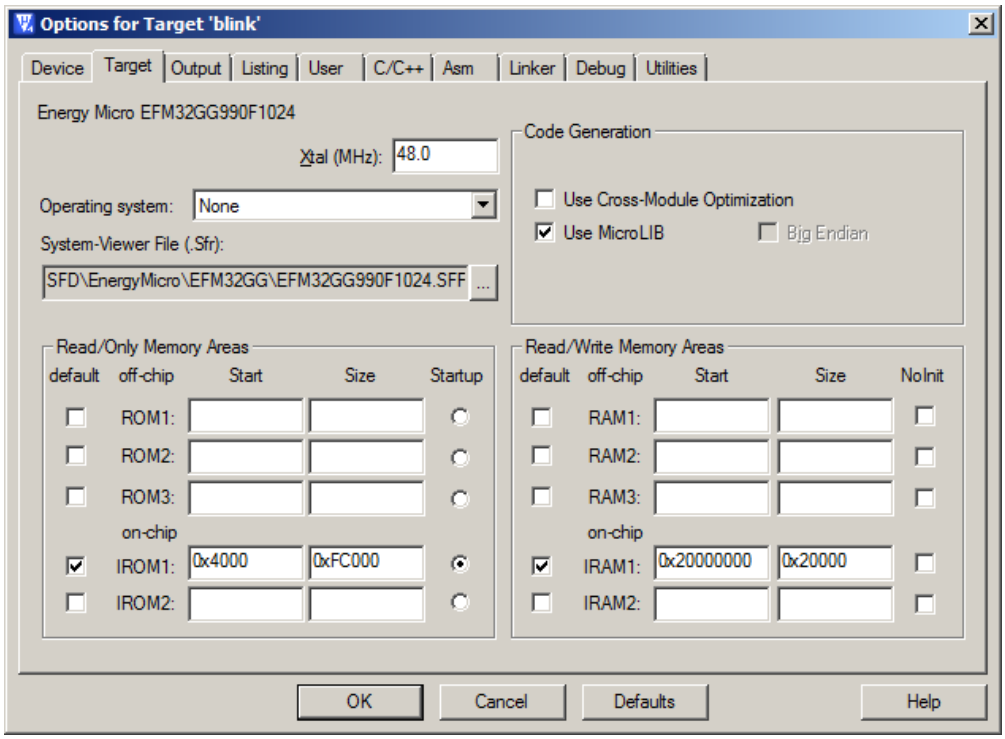
To create applications with Keil uVision 4/MDK-ARM, you must first change the target settings for your project. In the options dialog change IROM1 to a start of 0x4000 and subtract 0x4000 from the size field.

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



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=0x4000
```

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 Non-destructive application upload

The 'u' command will upload an application without overwriting the bootloader itself. 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 application 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 0x4000 (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.

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 Revision History

5.1 Revision 1.03

2013-10-14

New cover layout

5.2 Revision 1.02

2012-11-12

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

5.3 Revision 1.01

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

5.4 Revision 1.00

2011-11-17

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