EFM32®

... the world's most energy friendly microcontrollers

USB MSD Host Bootloader

AN052 - Application Note

Introduction

This application note describes how to implement a MSD (Mass Storage Device) host bootloader in a USB enabled EFM32 to update the EFM32 firmware from a binary image on an attached USB MSD device. Note that the bootloader described in this application note is NOT pre-programmed in any EFM32s.

This software provided with this application note can run on the following kits:

- EFM32GG-DK3750
- EFM32GG-STK3700

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - IAR and Keil IDE projects
  - IAR linker files for applications
  - Binary image for the bootloader

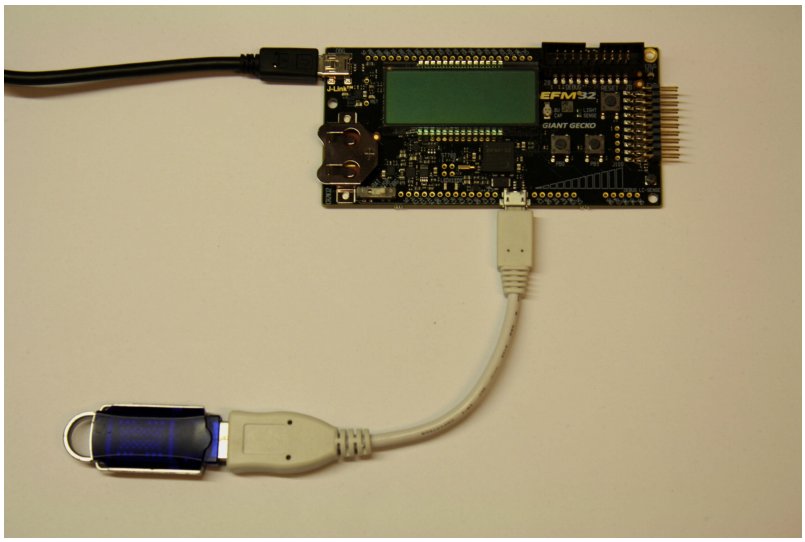
ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 Bootloader Operation

The bootloader described in this application note sets up the EFM32 as a USB host that can connect to a Mass Storage Device (MSD) like a memory stick etc. A new EFM32 firmware image can then be placed on the MSD device and when connected, the EFM32 will read the image from the device and write this to the application part of the EFM32 flash. Figure 1.1 (p. 2) shows how to connect the EFM32GG_STK3700 as a USB host to a Mass Storage Device through a female A to male micro A type USB cable.

Figure 1.1. Connecting the EFM32GG_STK3700 as a USB host to a Mass Storage Device



1.1 Starting the Bootloader

The EFM32 will enter bootloader mode after reset and wait for an USB MSD device to be plugged in if there is no valid application in application space. If the application space contains a valid application, the bootloader will invoke this application if no MSD device is plugged in within 2 seconds or if no valid binary file for firmware upgrade is found on the USB MSD device.

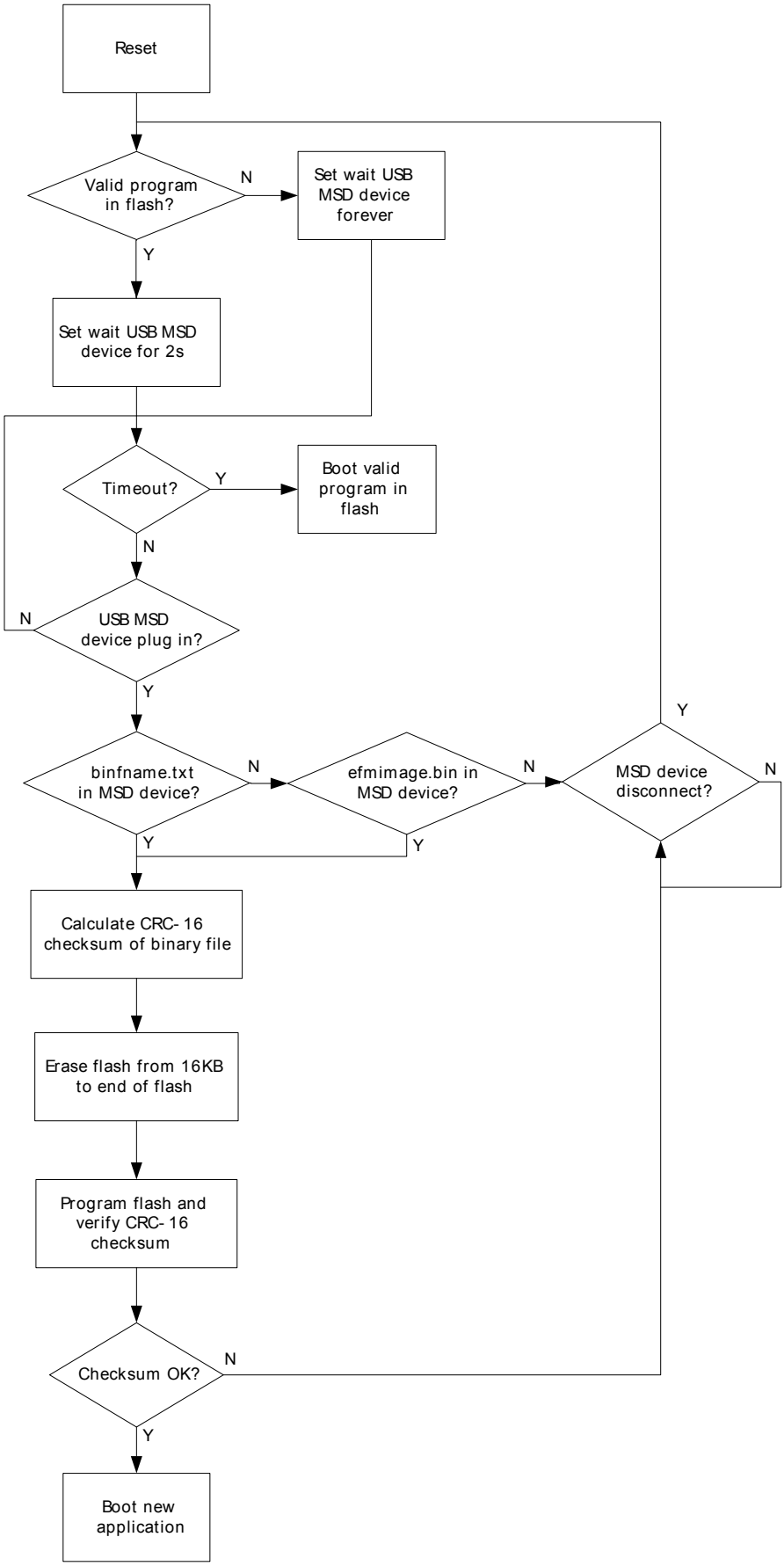
1.2 Firmware Upgrade

If the bootloader finds a valid firmware image on USB MSD device, the following tasks will be executed (flow chart shown in Figure 1.2 (p. 3)):

1. The bootloader searches the binfname.txt in root directory and then uses the file name inside to open the binary file in root directory.
2. If a binfname.txt file is not found, the bootloader searches for the default image file efmimage.bin in the root directory.
3. Calculate the CRC-16 checksum of the binary file (see flash.c for details).
4. Erase the flash from 16KB to end of flash, and then write binary data to flash.
5. Calculate the CRC-16 checksum of the flash area being programmed and verify against the one calculated from the binary file.
6. Start the new application if the CRC-16 checksum is OK.
7. If an error occurs during firmware upgrade, wait until the USB MSD device is plugged in again and repeat the process described above.

Ensure that the binary file size is less than total flash size minus 16KB. To minimize code size, long file name (LFN) is not supported for the binary file, so the name can be maximum 8 characters long followed by a 3 character file extension.

Figure 1.2. Firmware upgrade flow



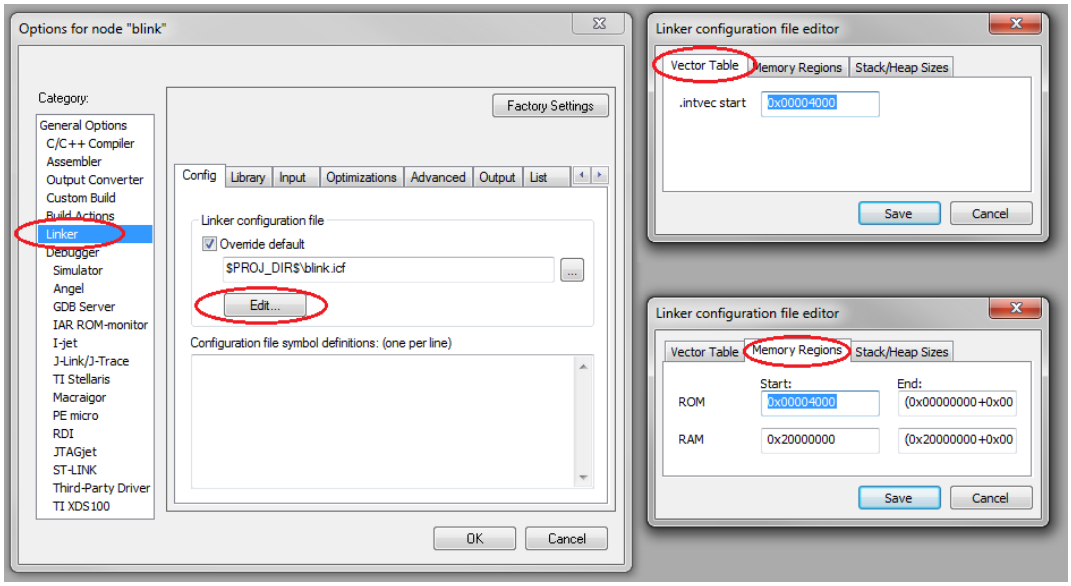
2 Creating Applications for use with the Bootloader

In order to allow future upgrades using the bootloader, applications images must be created with a starting address of 0x4000. The reason for this is that the bootloader itself occupies the flash area below this address. To achieve this, the default flash start address defined in the linker command file must be changed as described below.

2.1 Creating an Application with IAR Embedded Workbench For ARM

To create an application using IAR, use the linker files included with this application note for your project. The included linker files set up the starting address of Vector Table and Memory Regions to 0x4000 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 USB MSD Host Bootloader. Figure 2.1 (p. 4) shows how to configure the linker settings in an IAR project.

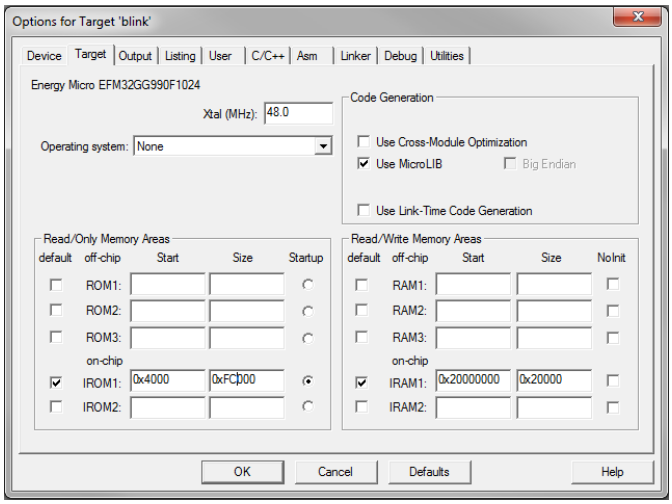
Figure 2.1. Application start address in IAR linker file



2.2 Creating an Application with Keil µVision 4/MDK-ARM

To create applications with Keil µVision 4/MDK-ARM, you must first change the target settings for your project. In the options dialog change IROM1 to a start at 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 µVision Help for details. Figure 2.2 (p. 5) shows how to configure the linker settings in a Keil project.

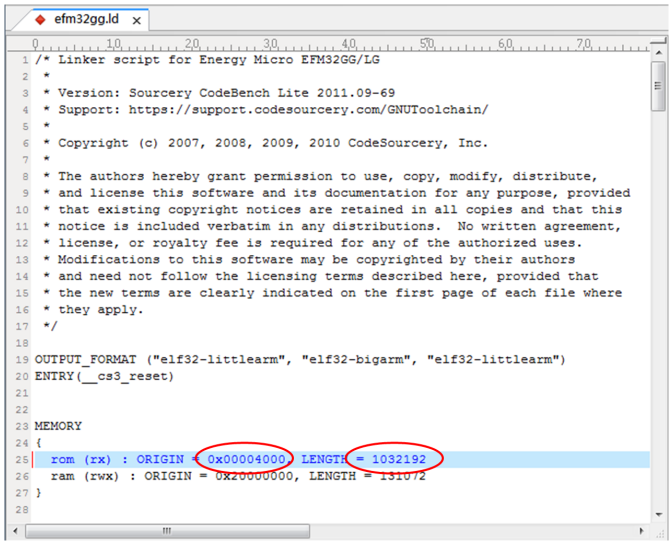
Figure 2.2. Setting up Keil µVision 4/MDK-ARM



2.3 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 linker file need to be modified. For application notes and example projects the location of the linker file is specified in the Makefile included with the software project. In the linker file MEMORY command, change the ROM ORIGIN to 0x00004000, 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



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.)

3 Customization Options for Bootloader

Precompiled binaries of the bootloader for the EFM32GG_STK3700 and the EFM32GG_DK3750 are included in the *bin*-folder of this application note. If you want to compile your own version you can change the configurations in below files to customize the bootloader.

3.1 config.h

Change this parameter if shorter or longer timeout is required.

```
/* Timeout in second for USB flash drive plug-in */

#define USB_WAIT_TIMEOUT 2
```

Change this parameter if the customized bootloader size is bigger or smaller than 16KB, it also requires changes on corresponding settings discussed in Section 3 above. Changes should be in steps of 2KB for LG and 4KB for GG parts.

```
/* The size of the bootloader flash image */

#define BOOTLOADER_SIZE (16*1024) /* 16 KB */
```

Change this parameter for default text file name.

```
/* Text file with binary file name */

#define TEXT_FILENAME "binfname.txt"
```

Change this parameter for default binary file name.

```
/* Default binary file for firmware upgrade if default text file is not
found */

#define BIN_FILENAME "efmimage.bin"
```

3.2 usart.h

The default setting for UART debug port is UART1 location 2 (TX at GPIO pin PB9) for DK and USART0 location 5 (TX at GPIO PC0, pin 3 of EXP header) for STK, baudrate is 115200, N, 8, 1.

Change below parameters for different baud rate and UART or USART:

#define USART_BAUD	0x1900
#define USART_USED	UART1
#define USART_CLK	CMU_HFPERCLKEN0_UART1
#define USART_PORTNUM	1
#define USART_TXPORT	gpioPortB
#define USART_TXPIN	9
#define USART_LOCATION	USART_ROUTE_LOCATION_LOC2

The printf statement is routed to UART debug port through retargetio.c in the drivers folder. Add messages with printf wherever debug information is required. In order to reduce the bootloader flash size, the debug output can be disabled by defining NOUART in the IDE project

To minimize the bootloader size to within 16KB the compiler should be set to optimize for size and the NDEBUG define must be set. Note that by default both NOUART and NDEBUG are set in the IDE projects, but can be removed in the following way:

- **IAR**: In the project options menu, select "C/C++ Compiler" and "Preprocessor". Remove NDEBUG and/or NOUART in the “Defined symbols:” box.
- **Keil**: In the project options menu, select "C/C++". Remove NDEBUG and/or NOUART in the “Preprocessor Symbols:” box.

To check the size of your compiled program, check the size of the read-only section in the resulting linker file (.map) after compilation.

4 Software Example

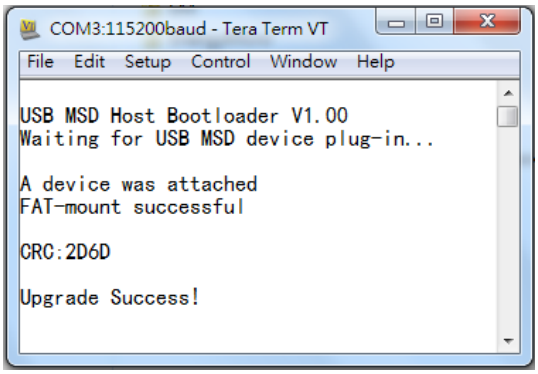
This application note includes two software projects for the bootloader, one for the EFM32GG_DK3750 and one for the EFM32GG_STK3700. The source files are identical for these two projects, it is just the project settings that differ. The RS232 port is only available on the DK. On the STK the UART debug TX signal can be picked up on a GPIO (PB9) without any configuration of the kit. Below are the steps needed to use the SW examples:

1. To use UART debugging on the Development Kit (EFM32GG_DK3750), press CFG->Peri->Select Serial (RS232) and change it to < UART > (UART LED will turn on) to enable RS232 connection (DB9 connector P9). Connect the DB connector through a RS232 cable to your PC and use any terminal program to monitor the output from UART port.
2. Download one of the binaries in the bin folder to the kit or compile and download one of the IDE projects.
3. Plug in an USB MSD device with binfname.txt and binary file or efmimage.bin for firmware upgrade in root directory.
4. Reset the MCU plug in board to start firmware upgrade.
5. Any errors during firmware upgrade will be displayed by the terminal program.
6. Error messages during firmware upgrade:
   - **FAT-mount failed:** Cannot mount the USB MSD device
   - **No text file:** Cannot find binfname.txt
   - **Text file read error:** Cannot read binfname.txt
   - **No binary file:** Cannot find the binary file in binfname.txt and efmimage.bin
   - **Flash write error:** Error during writing internal flash
   - **CRC verify error:** Program flash CRC does not match with binary file CRC
   - **Upgrade Fail:** One of above errors occur during firmware upgrade
   - **Please remove device:** Unplug USB MSD device to retry
   - **A malfunctioning device was attached, please remove device:** USB MSD device not recognized

Due to binary size, the Keil version does not support UART debug output unless start of application area is moved to a higher address.

Figure 4.1 (p. 8) shows the debug output during firmware upgrade.

Figure 4.1. Debug output during firmware upgrade



5 Further Improvements

To tailor the functionality of the bootloader to fit a specific use-case, several modifications can be made. Some possible modifications are listed below:

- The bootloader continuously waits USB MSD device to plug in if there is no valid application inside the flash. In order to reduce the current consumption, it can use GPIO (e.g. SWCLK) to invoke the bootloader after reset as in application notes AN0003 or AN0042.
- The firmware version and update history can be stored as log file on the USB MSD device.
- The firmware version or other related information can be stored in internal information block and this data will still keep after device erase operation.

6 Revision History

6.1 Revision 1.03

2013-09-03

New cover layout

6.2 Revision 1.02

2012-11-12

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

6.3 Revision 1.01

2012-08-21

Described default settings for NDEBUG and NOUART.

Minor typo fixes.

6.4 Revision 1.00

2012-08-10

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