AN0042: USB/UART Bootloader

This application note is intended for users of the bootloader in USB-enabled EFM32 or EZR32 devices.

The bootloader enables users to program the EFM32 or EZR32 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 most EFM32 or EZR32 devices with a USB peripheral. For more information on device compatibility, see 1. Device Compatibility.

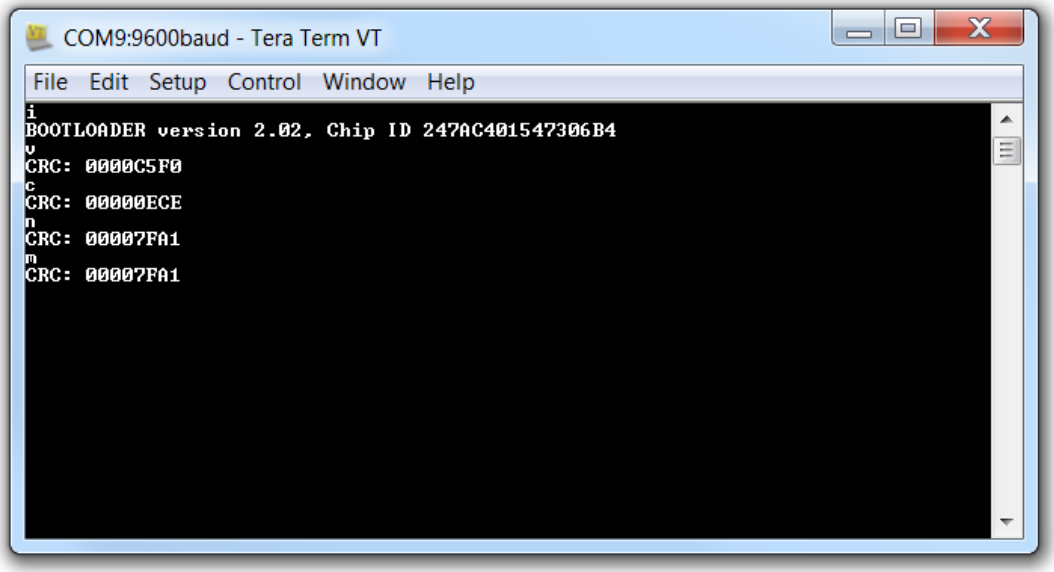
Note: EFM32HG devices only have 8 KB of RAM. The bootloader is significantly optimized to fit in these devices.

This application note includes the following:

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

KEY POINTS

- All EFM32 or EZR32 devices with an USB peripheral are pre-programmed with the USB/UART bootloader, which:
- Can remain alongside customer applications to support field upgrades, or be overwritten to maximize available flash area.
- Communicates using either an USB-based virtual or a physical UART interface (see section 1.2 for part-specific peripheral configuration and supported baudrates).
- Supports single-character commands to: program (upload/overwrite), verify (calculate checksums), secure (writeprotect, lock the debug port), etc. the EFM32 or EZR32 device.
- Accepts file transfers using the XMODEM-CRC protocol.
- Is invoked after reset while DBG_SWCLK is pulled high.



1. Device Compatibility

This application note supports multiple device families, and some functionality is different depending on the device.

This bootloader is preprogrammed in all MCU and Wireless MCU Series 0 devices that have the USB peripheral. Series 0 devices which do not include the USB peripheral are pre-programmed with a UART bootloader discussed in application note *AN0003: UART Bootloader*, available through Simplicity Studio (www.silabs.com/simplicity) or from the Silicon Labs website at www.silabs.com.

MCU Series 0 consists of:

- EFM32 Gecko (EFM32G)
- EFM32 Giant Gecko (EFM32GG)
- EFM32 Wonder Gecko (EFM32WG)
- EFM32 Leopard Gecko (EFM32LG)
- EFM32 Tiny Gecko (EFM32TG)
- EFM32 Zero Gecko (EFM32ZG)
- EFM32 Happy Gecko (EFM32HG)

Wireless MCU Series 0 consists of:

- EZR32 Wonder Gecko (EZR32WG)
- EZR32 Leopard Gecko (EZR32LG)
- EZR32 Happy Gecko (EZR32HG)

MCU and Wireless SoC Series 1 devices are not supported by this bootloader, including EFM32GG11.

MCU Series 1 consists of:

- EFM32 Jade Gecko (EFM32JG1/EFM32JG12/EFM32JG13)
- EFM32 Pearl Gecko (EFM32PG1/EFM32PG12/EFM32PG13)
- EFM32 Giant Gecko (EFM32GG11)

Wireless SoC Series 1 consists of:

- EFR32 Blue Gecko (EFR32BG1/EFR32BG12/EFR32BG13)
- EFR32 Flex Gecko (EFR32FG1/EFR32FG12/EFR32FG13)
- EFR32 Mighty Gecko (EFR32MG1/EFR32MG12/EFR32MG13)

2. Starting the Bootloader

2.1 Entering Bootloader Mode

To enter the bootloader, DBG_SWCLK must be pulled high and the EFM32 or EZR32 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_SWCLK has an internal pull-down. Leaving this pin unconnected will not invoke the bootloader.

2.2 Initializing Communication with the Bootloader

Bootloader communication is initialized either by transmitting an uppercase "U" on the physical UART interface (which starts the auto-baud algorithm), or by enumeration of the USB CDC virtual UART device. Whichever happens first will govern the rest of the bootloader operation.

The bootloader generally uses GPIO pins E11 and E10 for UART communication (RX and TX, respectively, for USART0 in Location 0—see note below for exceptions). The UART uses 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 physical UART, the host baudrate will be measured and UART baudrate adjusts accordingly. Navigate to [**an0042_efm32_usb_uart_bootloader**]>[**binaries**]>[**notes.txt**] in the AN0042 software package (available here - https://www.silabs.com/support/resources) for a part-specific range of baud rates compatible with the autobaud algorithm.

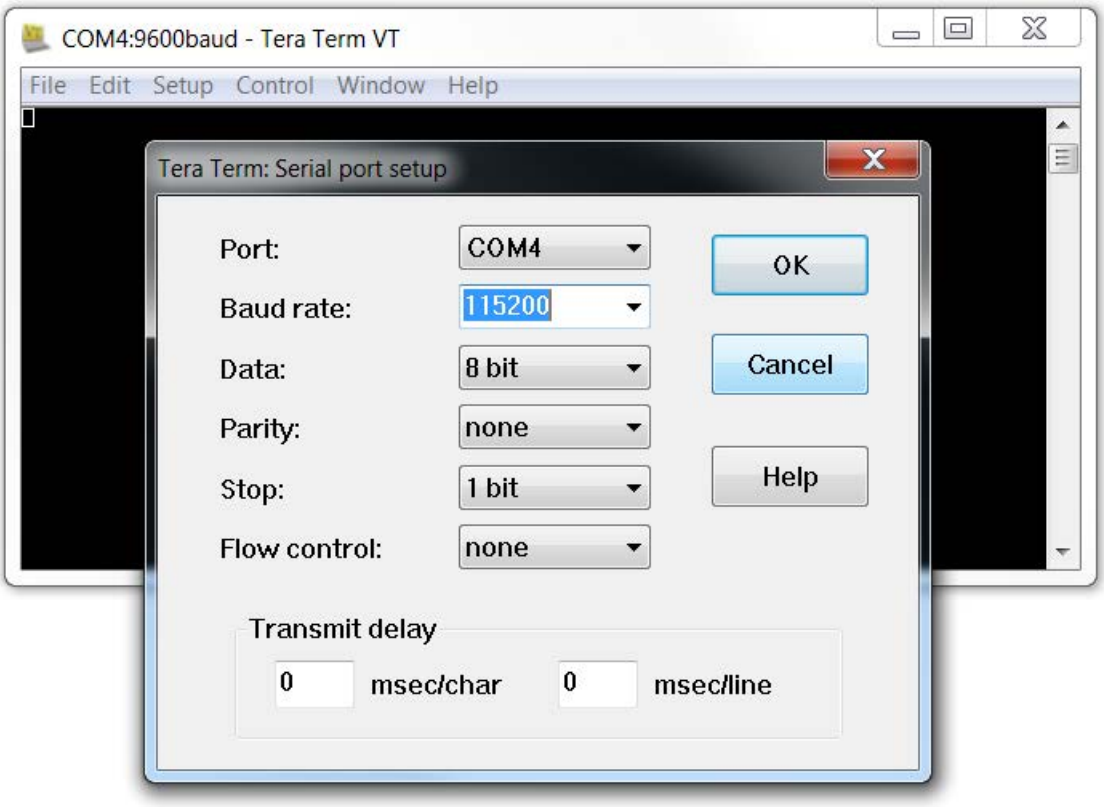


Figure 2.1. Configuring the UART serial port for bootloader communication in Tera Term on Windows 7

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]** 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, right-clicking on the device and selecting [**Update Driver Software...**]. Do a manual install and browse to the location of the `SiLabs-CDC.inf` file.

Note: Bootloaders in most EFM32 or EZR32 devices are configured to use peripheral USART0 in Location 0 as the physical UART interface. However, some devices do not have a USART0 peripheral or Location 0 option. For these and other reasons, the bootloaders in EFM32HG devices instead use LEUART0 in Location 3, and those in EZR32LG devices use USART1 in Location 2. See the device data sheets for pinout locations of these signals. Note that this location overlaps the regular SWD port, which, as discussed above, is used to enter the bootloader. Therefore, when using these parts, you should use a 4 kΩ pull-up on DBG_SWCLK.

Note: Some of the OPNs of HG, LG, WG (both EFM32 and EZR) and EFM32GG do not include USB functionality. These OPNs use the UART Bootloader and follow the guidelines mentioned in *AN0003: UART Bootloader*. To find out which OPNs support USB functionality, please refer to the device datasheet.

Note: Navigate to [**an0042_efm32_usb_uart_bootloader**]>[**binaries**]>[**bootloader-matrix.txt**] in the AN0042 software package (available here - https://www.silabs.com/support/resources) for the latest version of the Bootloader per family.

2.3 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 or EZR32 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 or EZR32

i

Print bootloader version and chip unique ID.

Note:

The [**b**] and [**r**] commands will delay 7 seconds before the commands are actually performed when using the USB CDC virtual UART. The first 5 seconds 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.

3. Uploading Applications

To upload an application to the EFM32 or EZR32, 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.

To send a file through XMODEM-CRC in Tera Term, navigate to [**File**]>[**Transfer**]>[**XMODEM**]>[**Send...**]. The following figure shows an example of transferring a file using the built in transfer support in Tera Term.

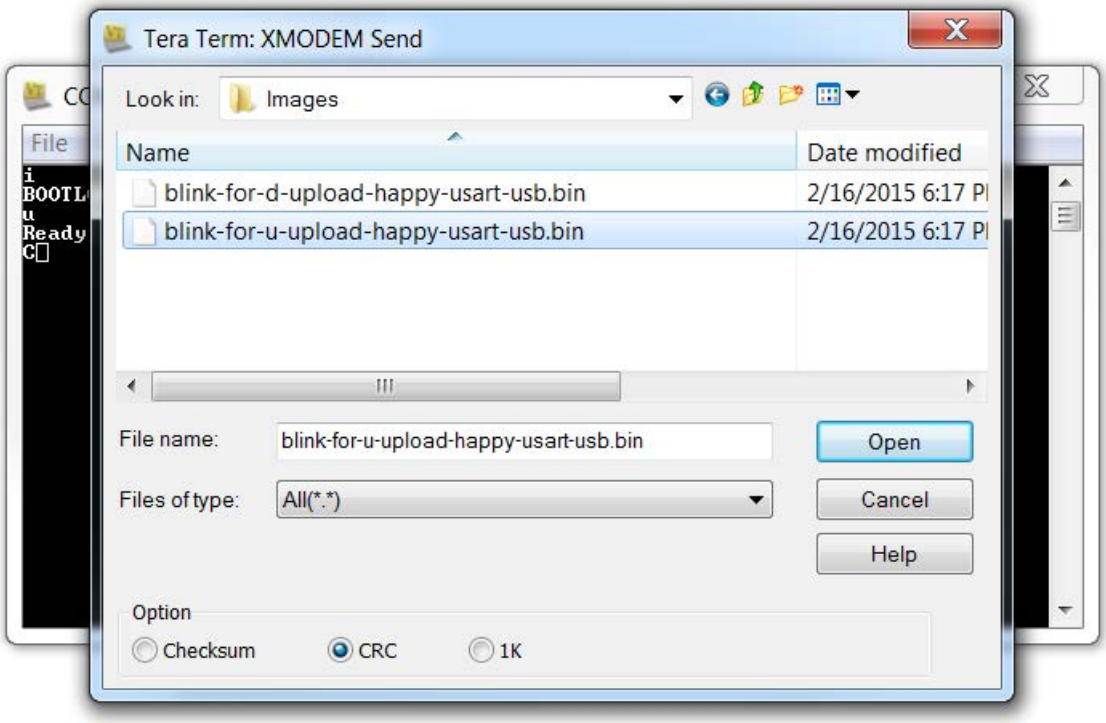


Figure 3.1. Transferring a File using XMODEM-CRC with Tera Term on Windows 7

3.1 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 the flash area below this address. To achieve this, the linker file must be changed from the default flash start address of 0x0.

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
```

This is not necessary in the released application, as VTOR is set by the bootloader itself before starting the application (see `Boot.c` for details).

3.1.1 Creating an Application with IAR

To create an application using IAR, navigate to [**an0042_efm32_usb_uart_bootloader**]>[**iar_linker_files**] in the AN0042 software package (available here - https://www.silabs.com/support/resources) and use the part-specific linker files for your project. This will set up the correct starting address for the binary. Another option is to right-click on the IAR project and Select [**Options..**]. Navigate to the [**Linker**] tab and ensure that the [**Override default**] option is checked.

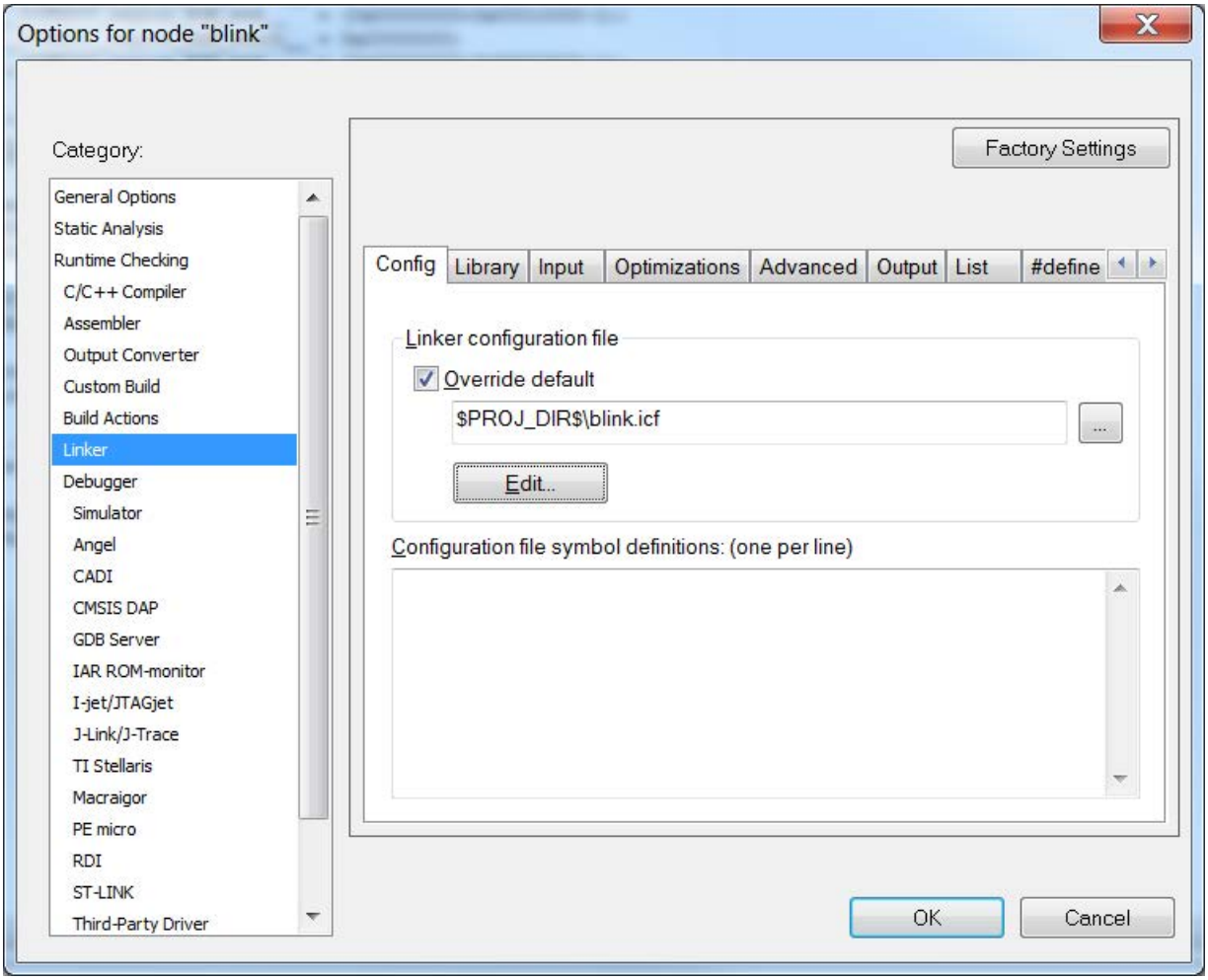


Figure 3.2. IAR Project Options

Create an .icf file within the Project Directory and click [**Edit**]. Set the starting address of the Vector Table in the [**Vector Table**] tab to [**0x4000**]. Change the starting address of ROM in the [**Memory Regions**] tab to [**0x4000**]. Click [**Save**] when done. Click [**OK**] in the [**Options**] window.

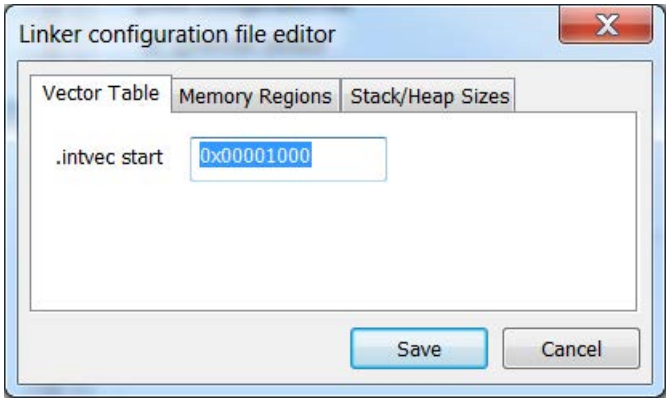


Figure 3.3. Change Vector Table Address

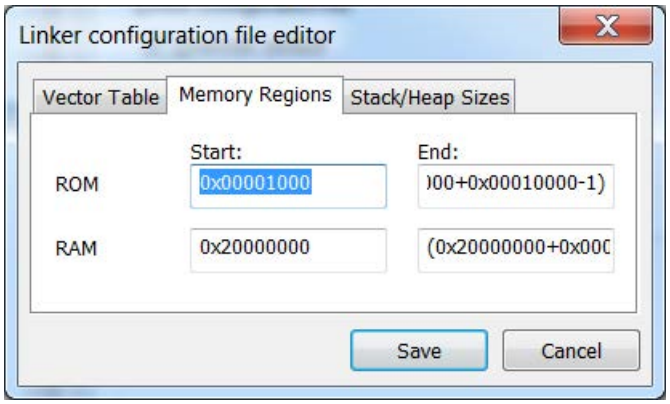


Figure 3.4. Change ROM Starting Address

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.

3.1.2 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.

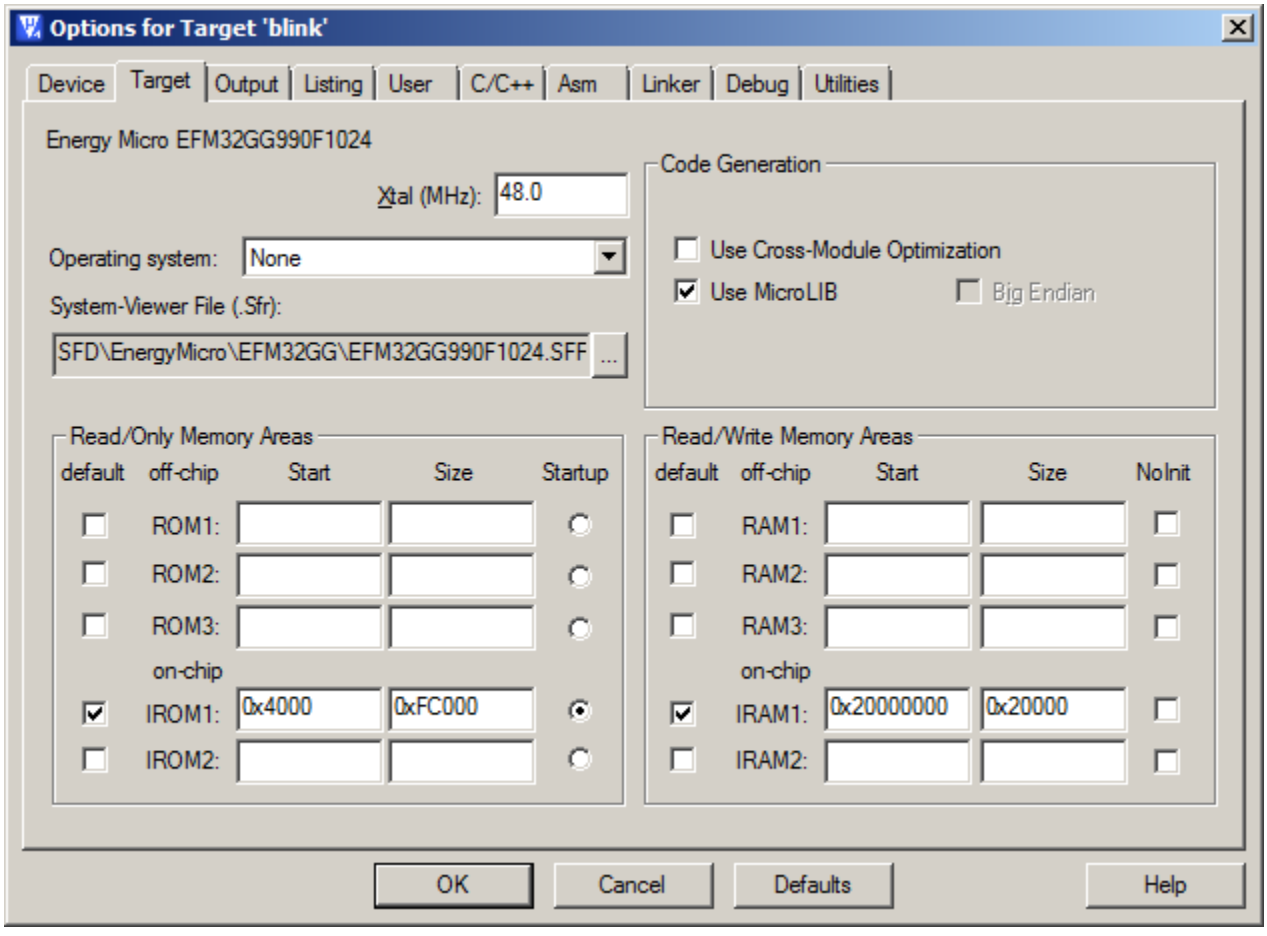


Figure 3.5. Setting up Keil uVision 4/MDK-ARM

3.1.3 Creating an Application with Simplicity Studio

To create applications with Simplicity Studio, change the memory layout settings for the project. In the [**Properties**]>[**C/C++ Build**]>[**Settings**]>[**Tool Settings**]>[**Memory Layout**] dialog:

1. Check [**Override default flash options**].
2. Change [**ORIGIN**] to a start of [**0x4000**].
3. Subtract [**0x4000**] from the length field.

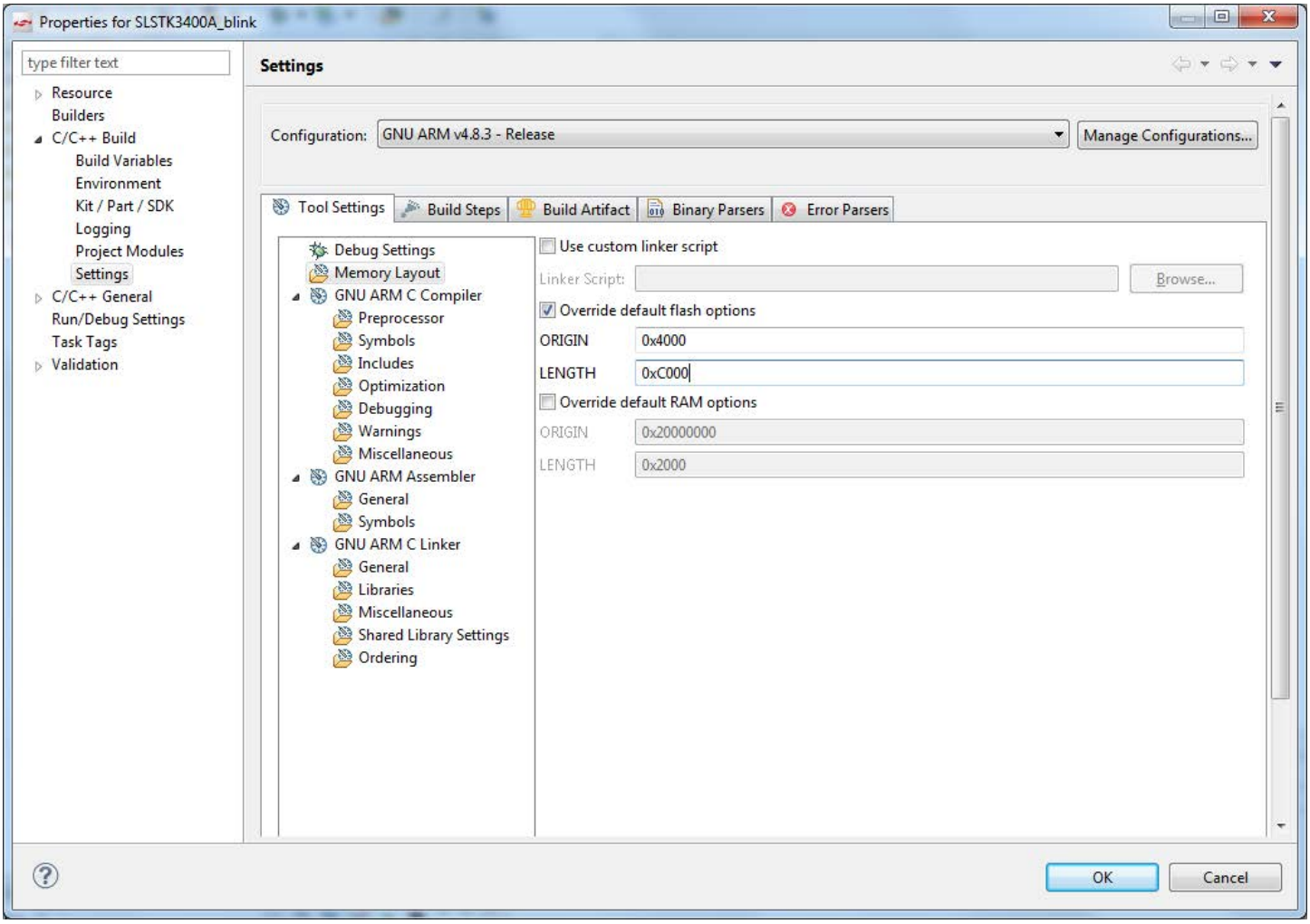


Figure 3.6. Setting up Simplicity Studio

The application will now start after the bootloader in memory.

3.2 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 4.1 Verify Application Checksum). To start the application from the bootloader, use the boot command ([**b—**]see 5.1 Boot Application).

3.3 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 4.2 Verify Flash Content). To start the application, you can use the reset command ([**r—**]see 5.2 Reset the Device).

3.4 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.

3.5 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.

4. 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.

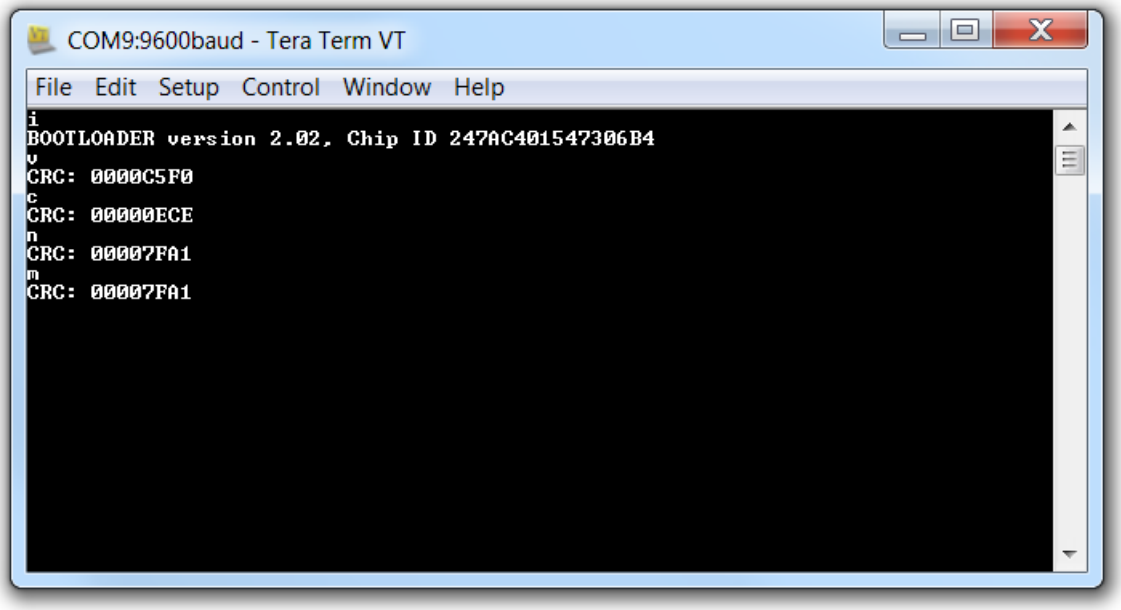


Figure 4.1. Example Tera Term session with bootloader version/ChipID and checksum commands.

4.1 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.

4.2 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.

4.3 Verify User Page Checksum

The [**n**] command will calculate and print the CRC-16 checksum of the User Data (UD) Page (page 0 of the Information flash block).

4.4 Verify Lock Page Checksum

The [**m**] command will calculate and print the CRC-16 checksum of the Lock Bits (LB) Page (page 1 of the Information flash block).

5. Miscellaneous Commands

5.1 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: When using the UART bootloader, the letter [**b**] will not print on the serial terminal, but the user should be able to see the application boot after pressing the [**b**] command.

5.2 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.

5.3 Debug Lock

The [**l**] command will lock the debug interface. After locking, regular debugging facilities will not be accessible, and 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 DBG_SWDIO is not connected and DBG_SWCLK is tied high.

6. Revision History

Revision 1.22

2017-10-05

- Updated the baudrates compatible with the autobaud algorithm.
- Updated the language about entry into Bootloader mode on devices that share Bootload entry pins with DBG SWCLK.
- Added a note about HG OPNs that do not have USB functionality.
- Updated the language about XMODEM transfer in Teraterm.
- Added a note about the difference in Serial Terminal display when using the USB Bootloader and USB-UART Bootloader.
- Added a note about the location of the text file showing the latest version of the Bootloader.
- Added some screenshots to explain Creating Applications with IAR in detail.

Revision 1.21

2017-06-16

- Added 1. Device Compatibility.
- Updated the language on the front page about devices that are pre-programmed with this bootloader.

Revision 1.20

2016-08-24

- Added EZR32HG and EZR32WG.
- Added Keil and Simplicity Studio support, including examples and 3.1.3 Creating an Application with Simplicity Studio.
- Added a note to the front page about the bootloader size and EFM32HG devices.

Revision 1.11

2015-03-06

- Added Happy Gecko and EZR32LG.
- Updated format.
- Added part-specific bootloader UART configuration details.

Revision 1.10

2014-05-07

- Changed to Silicon Labs license on code examples.
- Added EFM32 Wonder Gecko linker files.

Revision 1.03

2013-10-14

- New cover layout.

Revision 1.02

2012-11-12

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

Revision 1.01

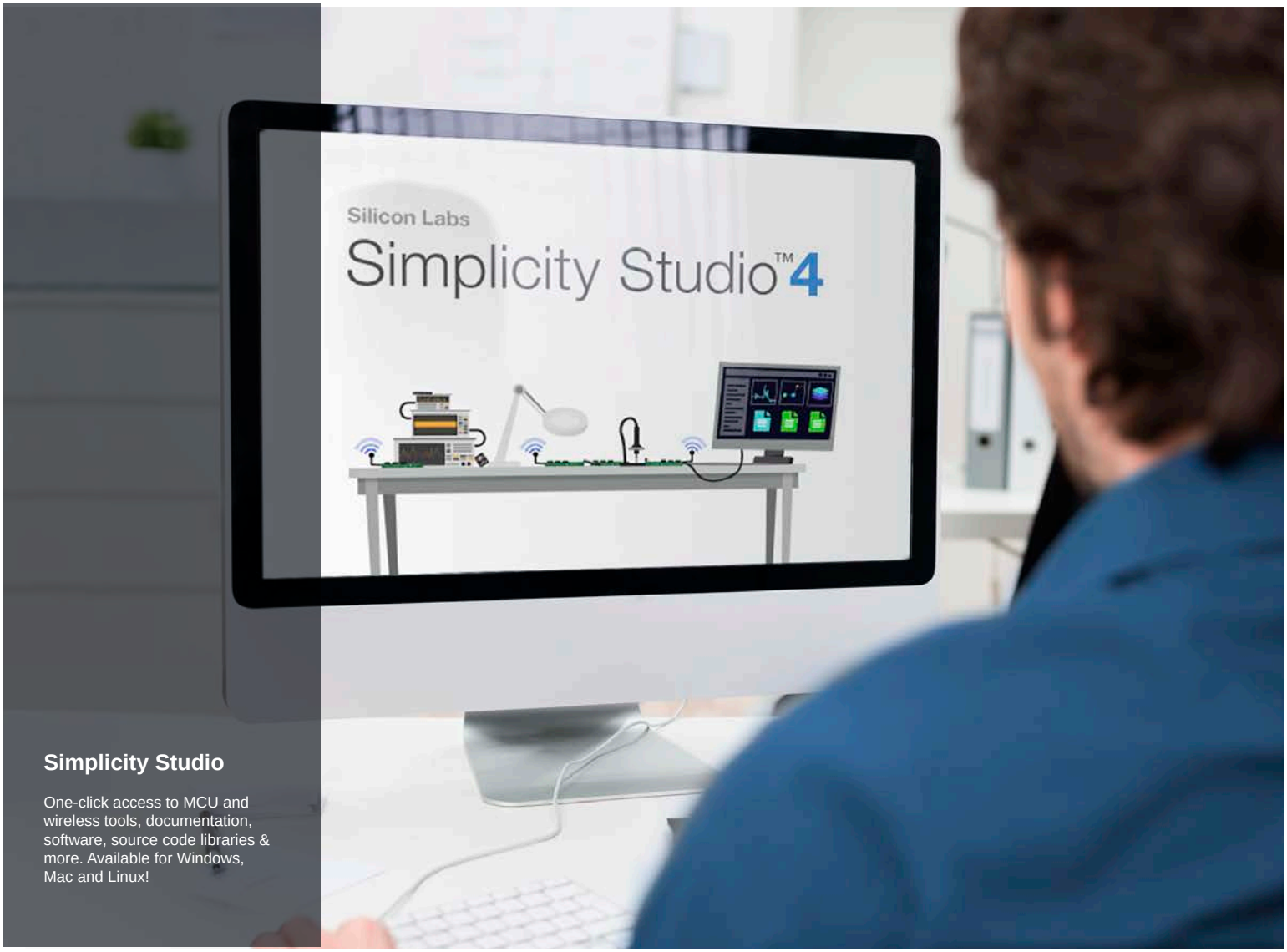
2012-04-20

- Adapted software projects to new peripheral library naming and CMSIS_V3.

Revision 1.00

2011-11-17

- Initial revision.



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com