AN0003: UART Bootloader

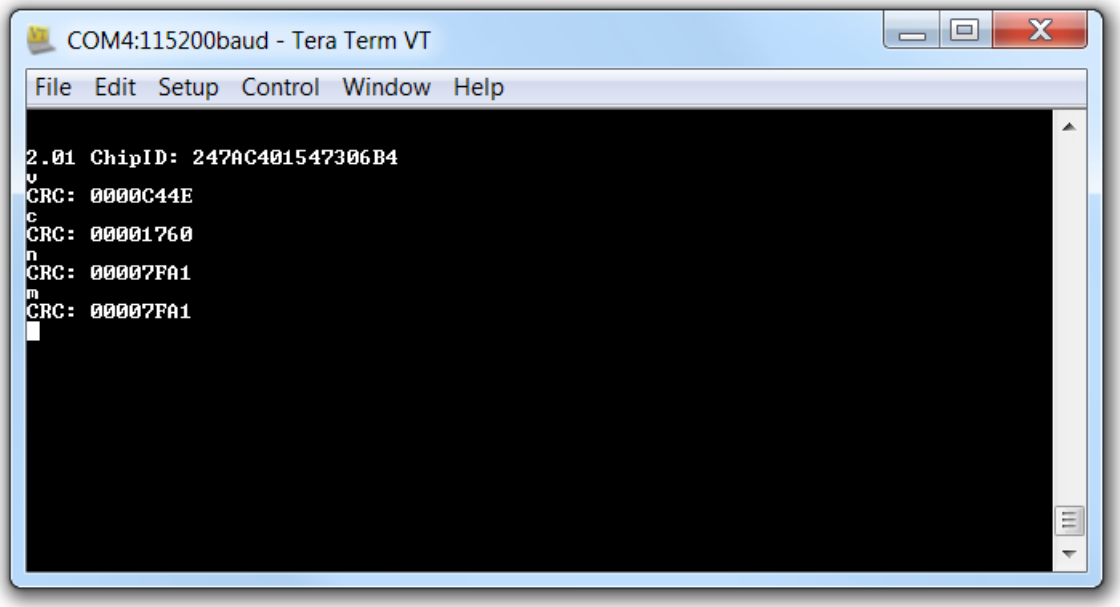
This application note is intended for users of the EFM32 UART bootloaders. The bootloader enables users to program the EFM32, EZR32, and EFM32 Gemstones devices through a UART without the need for a debugger.

In addition to booting user applications, the EFM32/EZR32 bootloader 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. In the EFM32 Gemstones devices, the bootloader resides in the reserved area of flash memory, and as such, the bootloader for these devices offers no destructive write mode. There is, however, a bit in the lock bit page that can be set to disable the bootloader in the EFM32 Gemstones devices.

In all devices, 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, EZR32, and EFM32 Gemstones devices. Devices which include USB also include a USB bootloader. The bootloader for USB enabled devices is covered in application note "AN0042 USB UART Bootloader", available through Simplicity Studio or from the Silicon Labs website at www.silabs.com.

KEY POINTS

- All EFM32 devices are pre-programmed with the bootloader, which:
- Can remain alongside customer applications to support field upgrades.
- Can be overwritten to maximize available flash area (EFM32 and EZR32 only) or be diasbled by a lock bit in the lock bit page (EFM32 Gemstones only).
- Communicates using an 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 (write-protect, lock the debug port), etc. the EFM32 device.
- Accepts file transfers using the XMODEM-CRC protocol.
- Is invoked after a reset while DBG_SWCLK is pulled high.



1. Starting the UART Bootloader

1.1 Entering Bootloader Mode

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

Note: Earlier revisions of the bootloader used both DBG_SWDIO and DBG_SWCLK pins to enter the bootloader. You can still enter the bootloader by pulling the DBG_SWDIO line low, but debug locking will not work.

1.2 Initializing Communication with the UART Bootloader

For the EFM32 and EZR32 devices, the UART bootloader generally uses GPIO pins PE11 (RX) and PE10 (TX) for UART communication (see note below for exceptions). Consult the device specific data sheet for pin locations.

Note: For EFM32G, EFM32GG, EFM32LG, and most EFM32TG parts, the bootloader communicates using USART0 in Location 0. However, some devices do not have a USART0 peripheral, and others. do but don't have a Location 0 option. For these and other reasons, the bootloaders in EFM32ZG, EFM32HG, and some EFM32TG (specifically EFM32TG108Fxx and EFM32TG110Fxx) parts all use LEUART0, location 3. 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.

For the EFM32 Gemstones devices, the bootloader uses the USART connection on port F0 (DBG_SWCLK) and F1 (DBG_SWDIO). These devices should also use a 4 kΩ pull-up on DBG_SWCLK.

The UART uses 1 stop bit, no parity, and 8 data bits. In addition, the bootloader uses autobaud to enable a wide variety of different terminals. The autobaud functionality senses the baudrate used by the terminal program and adjusts accordingly. This initialization done by sending one uppercase "U" to the bootloader immediately after opening a serial connection to the device. 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.

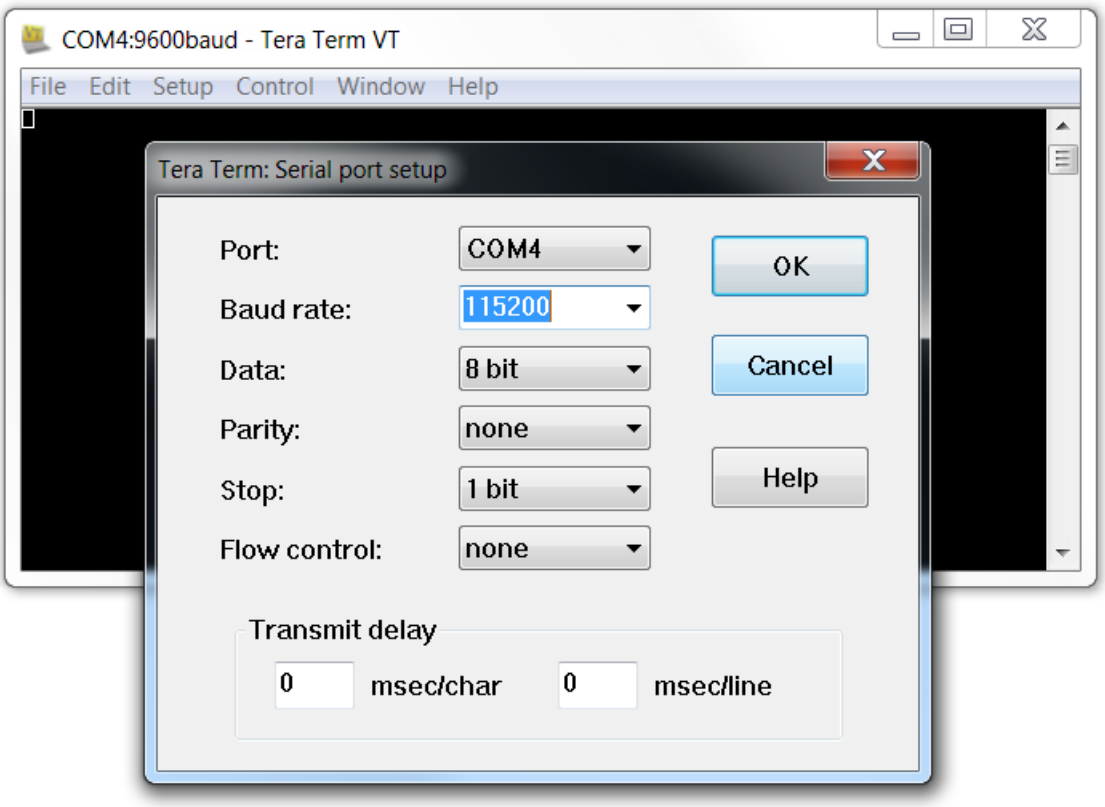


Figure 1.1. Configuring the serial port for UART bootloader in Tera Term on Windows 7.

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

```
1.40 ChipID: F08AB6000B153525
```

Note: Bootloader versions prior to version 1.40 do 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.

EFM32 and EZR32: 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, EFM32ZG and EFM32HG parts and at 0x1000 for EFM32GG, EFM32LG and EFM32WG parts. The application is transferred using the XMODEM-CRC protocol.

EFM32 Gemstones: This command lets the user upload an application to the flash. No special modification to the user application is necessary because the bootloader resides in reserved flash and cannot be overwritten. The application is transferred using the XMODEM-CRC protocol.

d

Destructive upload (EFM32 and EZR32 only). This command lets the user upload an application to flash, overwriting the bootloader. No modification of the start address of the user application is necessary. The application is transferred using the XMODEM-CRC protocol. Destructive upload is not an option on the EFM32 Gemstones devices.

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, EZR32, or EFM32 Gemstone 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 (EFM32 and EZR32) or the [**u**] command (EFM32 Gemstones). Please note that the [**v**] command and the [**c**] command will yeild the same result on the EFM32 Gemstones devices.

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.
Please note that the [**c**] command and the [**v**] command will yeild the same result on the EFM32 Gemstones devices.

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, EZR32, or EFM32 Gemstone.

2. Uploading Applications

To upload an application to the EFM32, EZR32, and EFM32 Gemstones, either the [**u**] (all devices) or [**d**] (EFM32 and EZR32 only) 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.

The figure below shows an example of transferring a file using the built in transfer support in Tera Term.

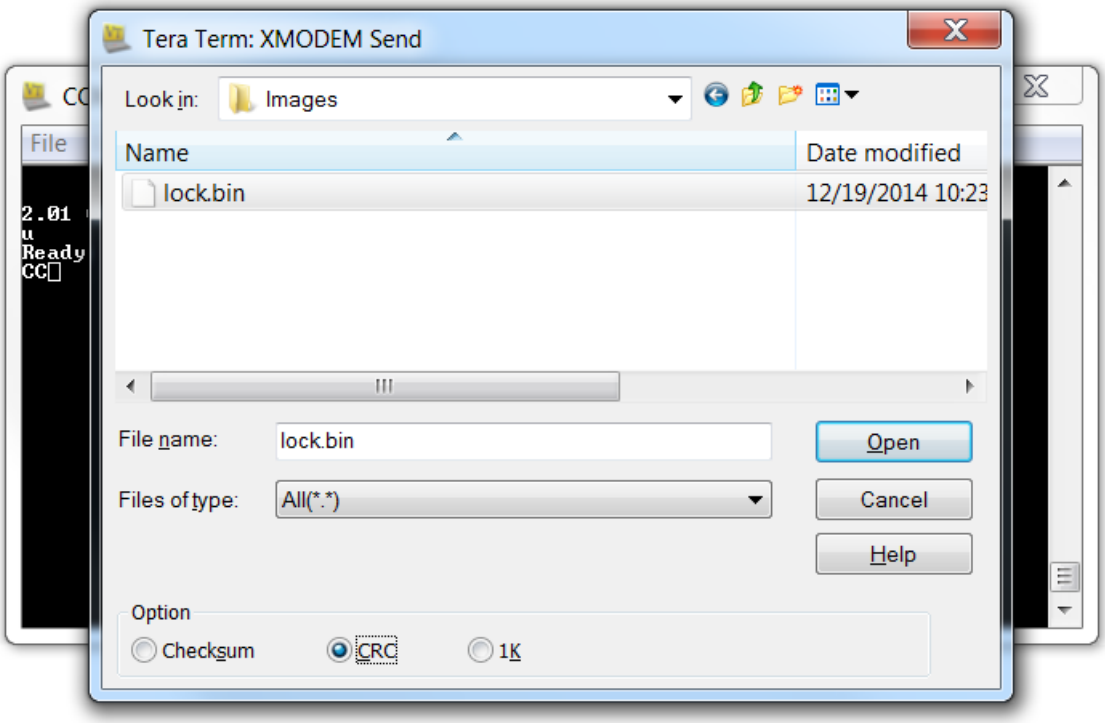


Figure 2.1. Transferring a file using XMODEM-CRC with Tera Term

2.1 Creating Applications for Use with the Bootloader - EFM32 and EZR32

Note: For information on creating applications for use with the EFM32 Gemstones devices, see 2.2 Creating Applications for Use with the Bootloader - EFM32 Gemstones.

There are two possibilities when uploading applications using the bootloader for EFM32 and EZR32 devices: 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, EFM32ZG and EFM32HG parts, and at 0x1000 for EFM32GG, EFM32LG and EFM32WG parts. The reason for this is that the bootloader itself occupies flash area between 0x0 and 0x7FF or 0x0FFF, respectively. For an application to coexist with the bootloader, the application 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=0x800; // EFM32G, EFM32TG, EFM32ZG and EFM32HG 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.

2.1.1 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.

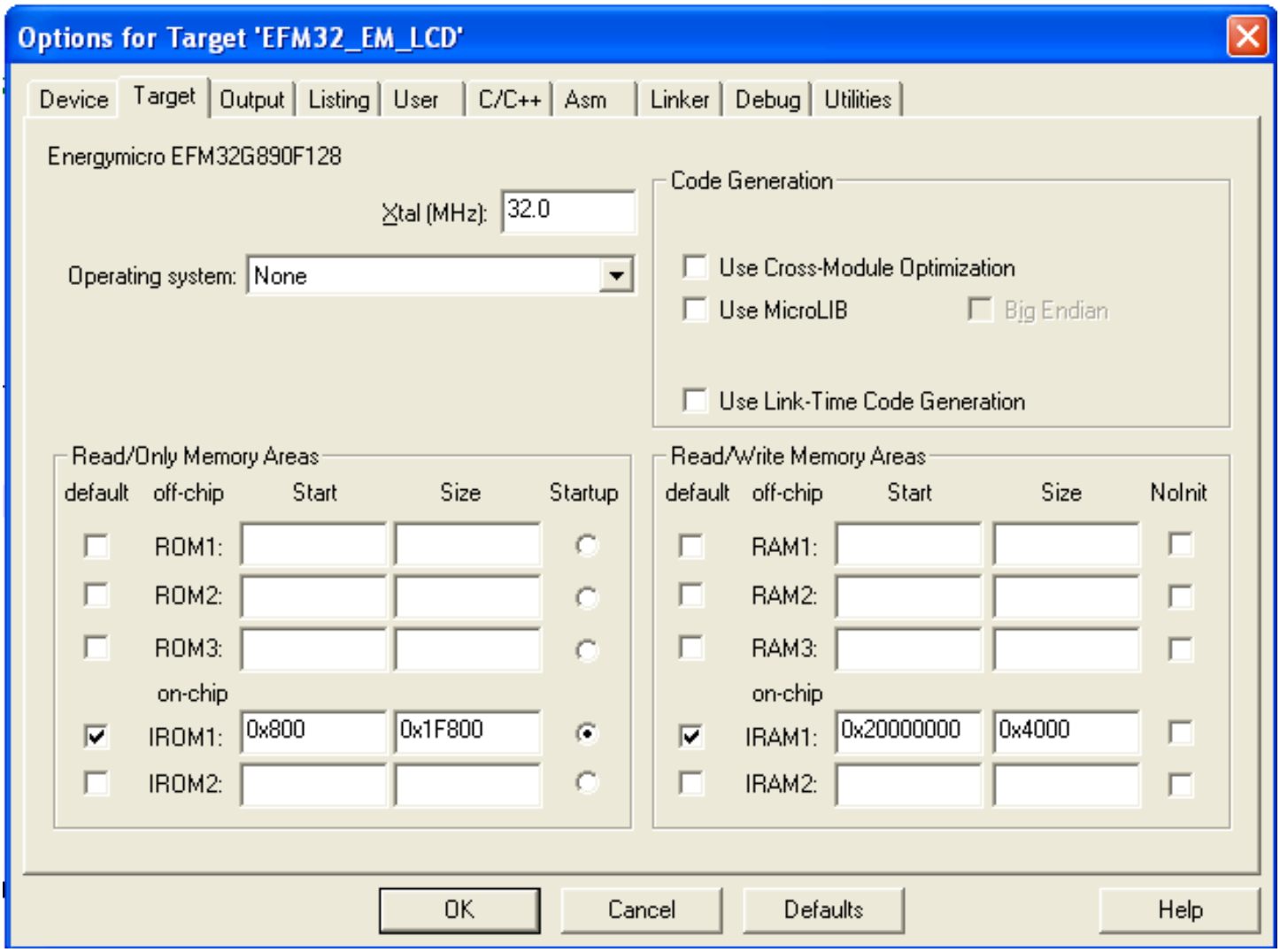


Figure 2.2. Setting up Keil uVision 4/MDK-ARM

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 needs 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 the following figure.

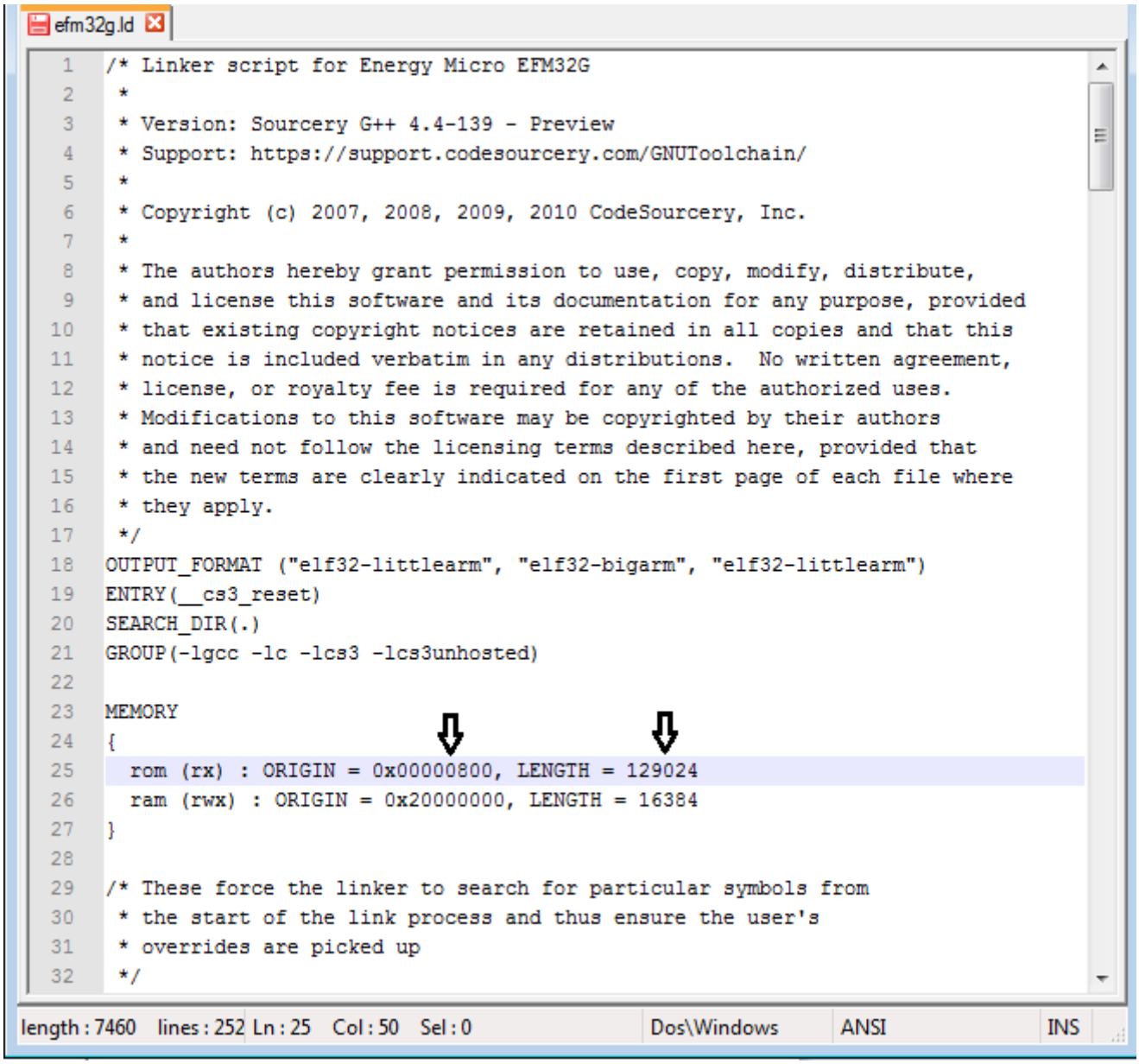


Figure 2.3. Application Start Address in Eclipse/gcc/cs linker file

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, EFM32ZG and EFM32HG 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).

2.1.4 Creating an application with Simplicity Studio

To create an application with Simplicity Studio that will work alongside the EFM32 or EZR32 bootloader, you must change the project properties to tell the linker where to locate the application in memory. To do this from the Simplicity Studio IDE, click on Project > Properties. In the Properties window, select the Tool Settings tab and then select Memory Layout. Select Override default flash options and then in the Origin field enter 0x800 (EFM32G, EFM32TG, EFM32ZG and EFM32HG) or 0x1000 (EFM32GG, EFM32LG and EFM32WG), and adjust the Length value accordingly (i.e. subtract 0x800 or 0x1000 from the original length, respectively), as in Figure 2.4 Changing Application Starting Address and Memory Length in Simplicity Studio on page 7. Because the linker file is automatically generated by Simplicity Studio, do not manually change these memory settings in the linker file as your changes will be overwritten in the next build.

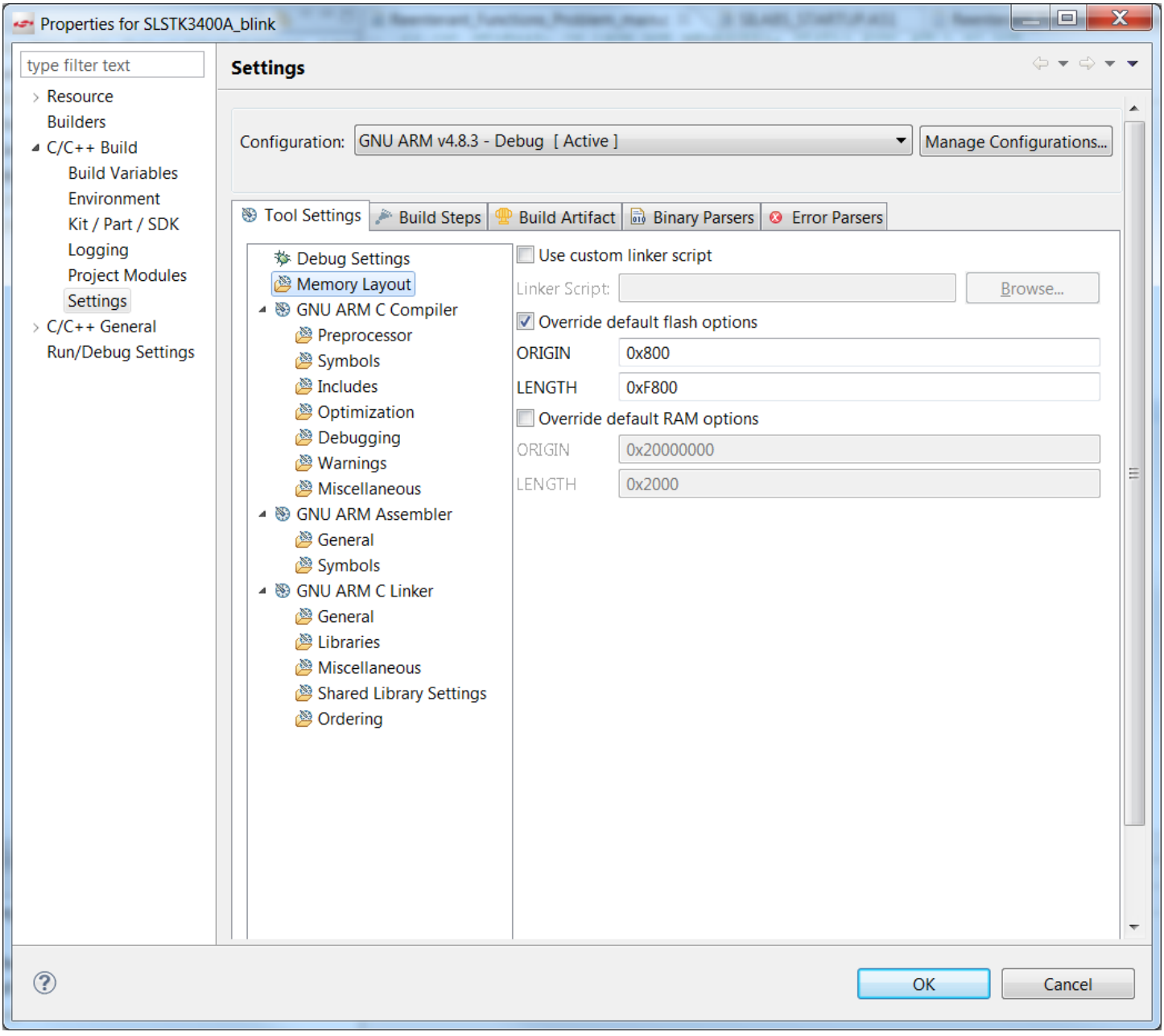


Figure 2.4. Changing Application Starting Address and Memory Length in Simplicity Studio

Note: If you need to debug your application after making these changes, you must explicitly set the position of the vector table in your code. This can be done with:

```
SCB->VTOR=0x800; // EFM32G, EFM32TG, EFM32ZG and EFM32HG 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 Creating Applications for Use with the Bootloader - EFM32 Gemstones

Because the bootloader for EFM32 Gemstones resides in reserved flash and cannot be overwritten, no additional steps are required for creating user applications for these devices. The bootloader on these devices can be disabled by clearing bit 122 (CLW0) in the Lock Bits page. See the device reference manual for more information on the Lock Bits Page configuration.

2.3 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 3.1 Verify Application Checksum). To start the application from the bootloader use the 'boot' command ([**b—**]see 4.1 Boot Application).

2.4 Destructive Upload - EFM32 and EZR32 Only

The [**d**] command will start a destructive upload on EFM32 and EZR32 devices. This option is not available on EFM32 Gemstones devices. 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 3.2 Verify Flash Content). To start the application, you can use the "reset" command ([**r**]—see 4.2 Reset the Device).

2.5 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.

2.6 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.

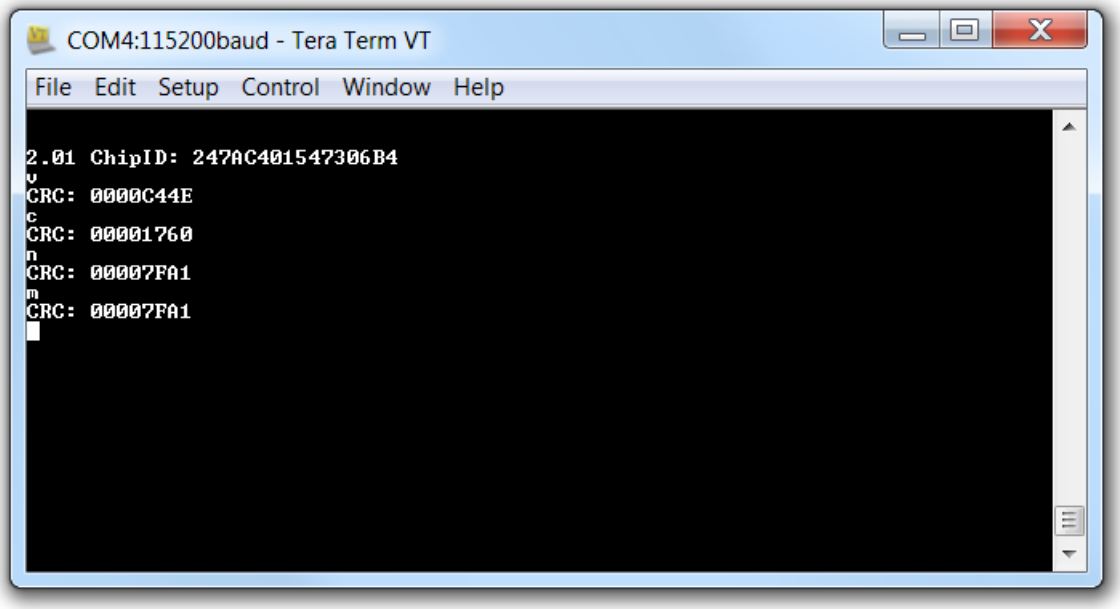


Figure 3.1. Example Tera Term session with bootloader checksum commands (initial "U" character not shown).

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.

3.3 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).

3.4 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).

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 DBG_SWDIO is not connected and DBG_SWCLK 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, Zero Gecko and Happy 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. If compiled in Debug configuration, the compiled program becomes too large. Check "High optimization for size" and enable "Multi-file compilation". "Discard unused publics" is also a helpful feature but will be enabled already by compiling in Release configuration.
- For Tiny Gecko, there are two projects. EFM32TG108Fxx and EFM32TG110Fxx devices should use bootloader-tg-small. Other Tiny Gecko devices should use bootloader-tg.

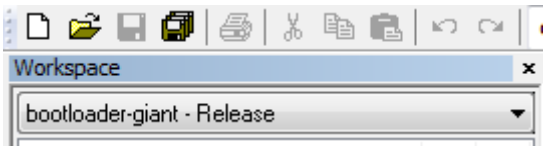


Figure 5.1. Compile in Release Mode

6. Revision History

6.1 Revision 1.70

2015-11-04

Added EFM32PG and EFM32JG devices.

Added instructions for changing memory settings in Simplicity Studio.

Differentiated instructions for EFM32 and EZR32 devices and EFM32 Gemstones devices.

6.2 Revision 1.69

2015-03-06

Added EFM32HG devices.

Updated formatting.

Clarified bootloader UART configuration for various families.

Added IAR compile guidance.

6.3 Revision 1.68

2014-05-07

Added binaries and linker files for EFM32 Wonder Gecko

Replaced references to CodeSourcery and Eclipse with generic GCC references

Changed to Silicon Labs license

6.4 Revision 1.67

2013-10-14

New cover layout

6.5 Revision 1.66

2013-07-31

Clarified which parts that use LEUART0.

6.6 Revision 1.65

2013-05-08

Added section about compiling the bootloader

6.7 Revision 1.64

2012-11-12

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

6.8 Revision 1.63

2012-04-25

Fixed missing comment ';' in assembly files.

6.9 Revision 1.62

2012-04-20

Added new pre-built 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.

6.10 Revision 1.61

November 16th, 2011.

Added missing EFM32LG binaries.

Renamed application note.

6.11 Revision 1.60

September 22nd, 2011.

Added support for EFM32GG and EFM32LG parts.

No functional differences for other bootloaders.

6.12 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.

6.13 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.

6.14 Revision 1.3

September 27th, 2010.

Increased download speed by using DMA.

Better handling of high bitrates.

Added binary for testing.

6.15 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.

6.16 Revision 1.21

September 2th, 2010.

Updated main heading on front page, no code changes.

6.17 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.

6.18 Revision 1.10

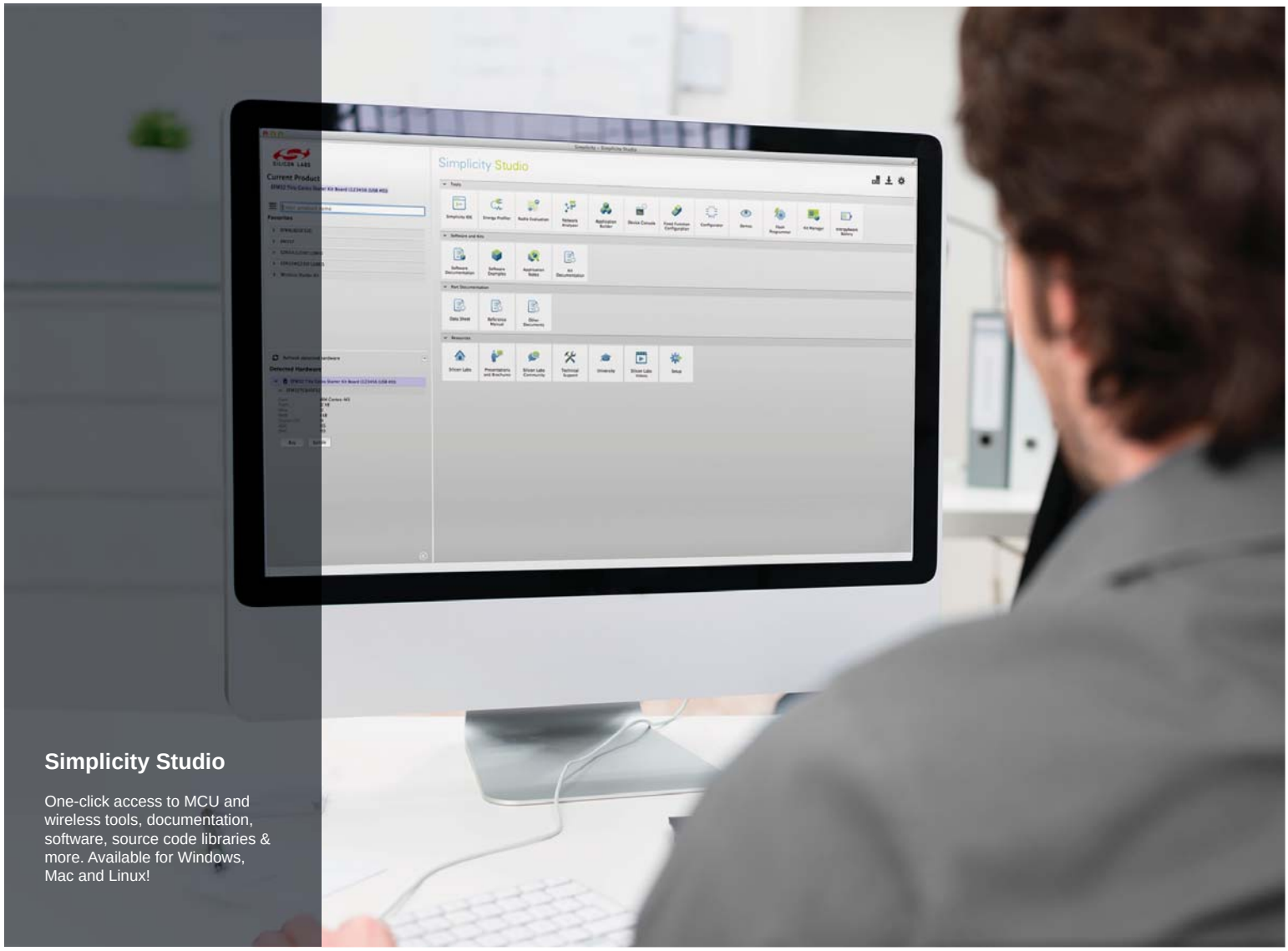
April 14th, 2010.

Bootloader now uses EM2 while idling.

6.19 Revision 1.00

January 8th, 2010.

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