APPLICATION NOTE

Atmel AVR2054: Serial Bootloader User Guide

Features

- Firmware programming over USART/SPI/TWI and other serial interfaces
- BitCloud® and RF4CE Over-the-Air Upgrade (OTAU) functionality support
- Atmel Bootloader PC tool for loading firmware from PC

Introduction

The serial bootloader is a stand-alone package consisting of two parts: embedded bootloader that should be loaded to the flash memory of a supported MCU and the PC based application that sends data to the embedded bootloader over serial link.

The package contains all necessary sources for modifying and compiling the embedded bootloader, including projects files for various development tool chains.

Chapter 1 Overview provides an overview of serial bootloader and package's contents. Chapter 2 Installation and getting started gives brief starting instructions including including serial ports and fuse settings. Following chapters cover more specific topics. Chapter 3 Embedded bootloader describes embedded bootloader application. Finally, Chapter 4 Bootloader PC tool gives instructions on how to create an SREC image and program a device with it, using the Atmel Bootloader PC tool.

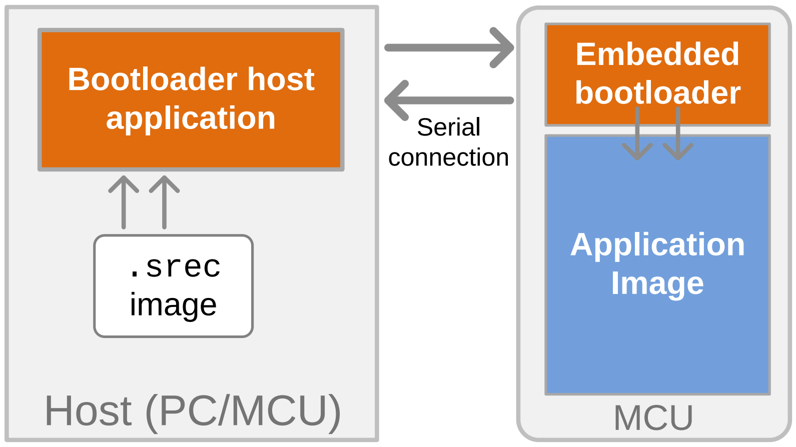
Table of Contents

1. Overview

The serial bootloader allows loading of firmware images to an MCU over the serial connection. It is intended for use with Atmel® wireless stacks, such as IEEE® 802.15.4 MAC [1] , RF4CE [2], BitCloud® [3], BitCloud Profile Suite [4] and LightWeightMesh [5], but can also be used with non-wireless applications. The package content is given in Section 1.3 Package content and structure while Chapter 2 Installation and getting started gives brief starting instructions on how to use the serial bootloader tool.

The concept of firmware programming with the serial bootloader is shown on Figure 1-1. The embedded bootloader preprogrammed to the MCU receives an application image over serial interfrace and writes it to the internal MCU flash. An application firmware image should be loaded in the Motorola S-record hexadecimal format (SREC) and is expected to be transferred by a host device. The host may be a PC or another MCU/MPU. It should be connected to the target MCU via supported serial interface and deliver the application image following the specific protocol.

Figure 1-1. General approach for using serial bootloader for MCU programming.



Embedded bootloader also supports some stack-specific functionality required to perform firmware Over-The-Air Upgrade (OTAU) procedure in ZigBee or RF4CE networks. Sections 3.7 and 3.8 provide more information on this topic.

Chapter 3 Embedded bootloader describes in details how the embedded bootloader works on different serial interfaces as well as how to configure and compile its source code.

For convenience the Serial Bootloader package includes pre-compiled images of the embedded bootloader images in different configurations. It also contains Atmel implementation of a PC host application - Bootloader PC tool that can be operated as a GUI application or as a command line tool (see Chapter 4 Bootloader PC tool for instructions how to use it).

The Bootloader PC tool may also be used to initiate an Over-the-Air Upgrade (OTAU) of a ZigBee® network by transferring a firmware image to the OTAU server device connected to the PC via a serial interface. See Section 3.7 as well as [8] for more details.

1.2 Supported platforms and interfaces

Embedded bootloader is supported on a set of Atmel microcontrollers shown in Table 1-1. It is implemented to be board-independent and works on Atmel development boards as well as custom boards. The PC tool however requires COM/USB enumerated ports and might not support board-specific functionalities.

Embedded bootloader can work with several different serial interfaces simultaneously as listed in the table Table 1-1 (except of TWI that needs to be used alone only). More details on the interface configurations are given in Section 2.1 Serial interface connections).

Table 1-1. Atmel MCUs and corresponding interfaces, boards and modules supported by Serial Bootloader tool.

Atmel MCU	Supported interfaces	Atmel modules	Wireless development boards/kits

ATmega256RFR2
ATmega128RFA1
	`USART0`
	`USART1`
	`TWID`
	`USB_FIFO`
		N/A	AT256RFR2-EK
			AT256RFR-XPRO(1)
			ATRF4CE-EK
			ATAVR128RFA1-EK1

ATxmega256A3	`USARTF0`
	`USARTD0`
	`SPIE`
	`TWIC`
	`TWIE`
		N/A	REB2xxED-EK
			STK600 + RZ600

ATxmega256D3	`USARTD0` `TWIC`	N/A	STK600 + RZ600

ATmega1281	`USART0`
	`USART1`
	`USB_FIFO`
	`TWID`
		ATZB-24-B0 [10]
		ATZB-24-A2 [10]
		ATZB-A24-UFL [11]
		ATZB-900-B0 [12]
			RCB230 / 231 / 212(2)

AT91SAM3S4C	`USB_DFU`	N/A	deRFusb-23E00 (2)

(1) Not supported by Atmel PC Bootloader tool.

(2) Non-Atmel products; see http://www.dresden-elektronik.de for details and purchase.

1.3 Package content and structure

The Serial Bootloader package contains:

- Embedded bootloader sources, including
  - Source code and header files
  - IAR projects
  - Atmel Studio projects
  - Makefiles for various configurations
- Pre-compiled firmware images of embedded bootloader
- Installation files for the Bootloader PC tool
- Documentation
- Drivers and utility firmware images

Table 1-2 describes the serial bootloader file structure with paths given relative to the package's root directory.

Table 1-2. Serial bootloader files and directories.

Path	Description
`\Documentation\`	Documentation folder
`\Embedded_Bootloader_images\`	Pre-compiled firmware images of embedded bootloader
`\Embedded_Bootloader_src\`	Embedded bootloader sources
`\Embedded_Bootloader_src\application\`	Embedded bootloader source code
`\Embedded_Bootloader_src\iar_projects\`	IAR Embedded Workbench project files for embedded bootloader. Project files for RF4CE version are not provided.

Path	Description
`\Embedded_Bootloader_src\as6_projects\`	Atmel Studio 6 project and solution files for the embedded bootloader
`\Embedded_Bootloader_src\makefiles\`	Makefiles for compiling the embedded bootloader with IAR or GCC compiler from the command line
`\PC_Bootloader_Setup\`	Installation files for the Bootloader PC tool
`\ThirdPartySoftware\`	Drivers and firmware for external tools
`\ThirdPartySoftware\XMEGA_A1_Xplained_Firmware\`	Pre-compiled firmware image of USART-to-TWI interface and driver for Xmega-A1-Xplained board
`\ThirdPartySoftware\USB_FIFO_FTDI_Driver\`	Driver for USB-FIFO of Sensor Terminal (STB) board [9]
`\ThirdPartySoftware\Bitcloud_DFU_bootloader.inf`	Driver for deRFusb-23E00

1.3.2 Precompiled bootloader firmware

Table 1-3 lists serial interfaces used in precompiled bootloader images available in `\Embedded_Bootloader_images\`.

Note: There are three types of embedded bootloader firmware distinguished by their functionality: common bootloader, the bootloader with OTAU support for Atmel BitCloud applications, and the bootloader for RF4CE applications. The latter two provide extra functionality in addition to common bootloader features. For details see Sections 3.7 and 3.8.

More details on the interface configurations are given in Section 2.1 Serial interface connections.

Table 1-3. Default serial port settings for embedded bootloader images.

MCU	Precompiled image(s)	Used MCU port	Comment

ATmega256RFR2
ATmega128RFA1
	`Bootloader_<MCU>.hex`
	`Bootloader_<MCU>_BitCloud_OTAU.hex`
	`Bootloader_<MCU>_RF4CE.hex`
		USART1	No board-specific behavior

	`Bootloader_<MCU>_KEY_REMOTE.hex`	USART0	No board-specific behavior. Can be used with RCB mount on Red Key Remote board and custom boards
	`Bootloader_<MCU>_STB.hex`	USB_FIFO	Only for use with RCBs mounted on Sensor Terminal Board (STB) [9].

	`Bootloader_<MCU>_TWI.hex`
	`Bootloader_<MCU>_BitCloud_OTAU_TWI.hex`
	`Bootloader_<MCU>_RF4CE_TWI.hex`
		TWID	No board-specific behavior.

ATXmega256A3	`Bootloader_<MCU>.hex`
	`Bootloader_<MCU>_BitCloud_OTAU.hex`
		USARTD0	No board-specific behavior.
			Can be used on STK600, REB-CBB and custom boards

	`Bootloader_<MCU>_TWI.hex`
	`Bootloader_<MCU>_BitCloud_OTAU_TWI.hex`
		TWIE (SDA-PE0 SCL-PE1)	No board-specific behavior

ATXmega256D3	`Bootloader_<MCU>.hex`
	`Bootloader_<MCU>_BitCloud_OTAU.hex`
		USARTD0	No board-specific behavior

MCU	Precompiled image(s)	Used MCU port	Comment

	`Bootloader_<MCU>_TWI.hex`
	`Bootloader_<MCU>_BitCloud_OTAU_TWI.hex`
		`TWIC` (SDA-PC0 SCL-PC1)	No board-specific behavior

ATmega1281	`Bootloader_<MCU>.hex`
	`Bootloader_<MCU>_BitCloud_OTAU.hex`
	`Bootloader_<MCU>_RF4CE.hex`
		`USART1`	No board-specific behavior

	`Bootloader_<MCU>_TWI.hex`
	`Bootloader_<MCU>_BitCloud_OTAU_TWI.hex`
	`Bootloader_<MCU>_RF4CE_TWI.hex`
		`TWID`	No board-specific behavior

AT91SAM3S4C		`USB DFU`	For USB sticks, attach the device to the USB port on a PC

2. Installation and getting started

Serial bootloader is typically used in the following way:

1. Unpack the package to a folder on the PC hard drive.
2. If embedded bootloader is not present on the target MCU device:
   a. Connect JTAG/ISP adapter to the target MCU device.
   b. If needed, configure on the MCU device the fuse bits for the corresponding bootloader feature support (see Section 2.2).
   c. Select or build a `.hex` image of the embedded bootloader with support of required serial interfaces and feature set. See Section 1.3.2 Precompiled bootloader firmware describing pre-compiled firmware images and Sections 3.2-3.3 describing how to configure and compile embedded bootloader source code.
   d. Program the target MCU with the `.hex` embedded bootloader image via JTAG/ISP insterface (see Section 3.4). Now the MCU has the embedded bootstrap on it.
3. Connect the MCU with the embedded bootstrap to the host device (for example a PC) via the serial interface supported in the MCU's embedded bootstrap (see Section 2.1 Serial interface connections).
4. If a PC is used as the host:
   a. If needed install the board- and connection-specific drivers (example VCP, USB, etc.) to get the serial connection to the connected MCU enumerated properly on the PC (see Section 2.1).
   b. Install the Bootloader PC tool from `\PC_Bootloader_Setup\`. To launch the Bootloader PC tool you will need to install Java® [16], if it is not installed on your PC.
   c. Launch the Bootloader PC tool, specify the connection settings to the target device, the application image in SREC format, and upload the image to the target MCU (see Section 4.1 for the detailed instructions).
5. If the host is not Atmel PC Bootloader tool:
   a. Implement own host application that will be able to upload the `.srec` images following the serial protocol supported by the embedded bootloader. See Section 3.5 and 3.6 for protocols description.
   b. Perfrom firmware upload according to own host program algorithm and requirements.

2.1 Serial interface connections

Embedded bootloader can work with USART, SPI, TWI and USB interfaces. The exact set of supported interfaces depends on a platform and is provided in Section 1.2 Supported platforms and interfaces while interfaces used by default in precompiled images are given in Section 1.3.2. MCU pins shall be connected to the host according to the pin assignement of the supported serial interface described in target MCU's datasheet.

Several different interfaces and ports can be enabled for bootloader use at the same time. As described in Section 3.5 embedded bootloader will scan them for required handshake packages one after another. Note however that TWI interfase can be operated only alone.

2.1.1 USART interface

Embedded bootloader supports different USART ports available on the target MCU as listed in Table 1-1. For USART connection between the MCU and the host is is required to connect only `TXD, RXD` and `GND` pins of the corresponding port. It is the responsibility of the user to ensure proper connection to the host device.

Default port configuration used for UART and hence also in precompiled firmware images is given in Table 2-1.

Table 2-1. Default COM port settings for embedded bootloader host

Baud rate	38400
Data bits	8
Parity	None
Stop bits	1
Flow control	None

For Atmel PC Bootloader tool to operate correctly as a host over UART it shall get the connection enumerated as a COM port and configured as described in Table 2-1.

2.1.2 USB-FIFO interface

USB-FIFO interface is a special connection to a PC via USB cable on the Sensor Terminal Board [9].

It requires installation of `ThirdPartySoftware\USB_FIFO_FTDI_Driver\` driver on the PC to get the connection enumerated as a COM port. Bootloader PC tool shall use COM port settings given in Table 2-1.

This interface is not recommended for reuse on custom boards.

2.1.3 TWI interface

When TWI interface is used the embedded bootstrap operates MCU in TWI slave mode with TWI slave bus address equal `0x55`.

Caution: TWI interface in the embedded bootloader must be used only alone and cannot be enabled together with any other serial interface.

When using Bootloader PC tool it is recommended to have Xmega-A1 Xplained board [15] as a TWI-to-USB bridge that provides connection to the PC and can get it enumerated as a COM port.

Atmel XMEGA-A1 Xplained kit contains ATxmega128A1 and AT32UC3B1256 devices. The preprogrammed AT32UC3B1256 device acts as a USB-to-UART gateway. The ATxmega128A1 device on this board shall be programmed with the firmware image `‘XplainedA1SerialToI2CBootLoaderBridge.hex’` from `\ThirdPartySoftware\XMEGA_A1_Xplained_Firmware\` to make it act as a USART-to-TWI bridge. TWI on PORTD is used by ATxmega128A1 and thus GPIO0 (SDA) and GPIO1 (SCL) of PORTD of XMEGA PORTD header of Xplained board shall be used to connect to the target MCU via TWI. The programming of firmware image to ATxmega128A1 device is explained in “Connecting the board” Section of AVR1927: XMEGA-A1 Xplained Getting Started Guide [14].

When connecting the XMEGA-A1 Xplained to PC, the operating system will request a driver file for installing the serial communication driver. This driver file `‘XPLAINED_Virtual_Com_Port.inf’` is available in `\ThirdPartySoftware\XMEGA_A1_Xplained_Firmware\` folder.

After VCP has enumerated PC Bootloader tool should be used with COM port settings given in Table 2-1.

2.2 Fuse bits settings

This section does not consider SAM3S because it does not have fuse bits.

An MCU must be aware of the size of the memory occupied by the bootloader if it resides in the flash and must start from the bootloader section rather than from the application section.

Recommended fuse bits for supported AVR microcontrollers are given in Table 2-2 and fuse bits for supported XMEGA microcontrollers are given in Table 2-3. The tables list all fuse bits and show the resulting fuse bytes as well.

Table 2-2. Fuse bits on ATmega1281, ATmega128RFA1 and ATmega256RFR2.

Option	Common bootloader	OTAU bootloader for Atmel BitCloud applications	Bootloader for RF4CE applications
BODLEVEL	Disabled	Disabled	Brown-out detection at VCC = 1.8V
OCDEN	Disabled	Disabled	Disabled
JTAGEN	Enabled	Enabled	Enabled
SPIEN	Enabled	Enabled	Enabled
WDTON	Disabled	Disabled	Disabled
EESAVE	Disabled	Disabled	Enabled
BOOTSZ	Boot Flash size=1024 words start address=$FC00	Boot Flash size=2048 words start address=$F800	Boot Flash size=2048 words start address=$F800
BOOTRST	Enabled	Enabled	Enabled
CKDIV8	Enabled	Enabled	Enabled
CKOUT	Disabled	Disabled	Disabled
SUT_CKSEL	Int. RC osc.; Start-up time: 6 CK + 65ms	Int. RC osc.; Start-up time: 6 CK + 65ms	Int. RC osc.; Start-up time: 6 CK + 0ms
Resulting fuse bytes	***0xFF 0x9C 0x62***	***0xFF 0x9A 0x62***	***0xFE 0x92 0x42***

Fuse bits configuration informs the MCU to start execution from the bootloader section in memory instead of the application's. On Atmel AVR microcontrollers the bootloader is stored in the flash memory in the same address space as the application. This reduces the amount of memory available for the application. The fuse bits are also used to specify the amount of memory exclusively reserved for the bootloader. Bootloader size may also vary depending on the bootloader type (for example, the OTAU bootloader requires larger memory allocation than a bootloader without OTAU support).

On XMEGAs, the embedded bootloader is kept in a separate storage, and so it is not needed to specify the size of the bootloader section (all 256kb of flash are available as the application memory).

Table 2-3. Fuse bits on Atmel ATxmega256A3 and Atmel ATxmega256D3.

Option	Required value
JTAGUSERID	0xFF
WDWP	8 cycles (8ms @ 3.3V)
WDP	8 cycles (8ms @ 3.3V)
DVSDON	OFF
BOOTRST	BOOTLDR
BODACT	Disabled
BODPD	Disabled
SUT	0ms

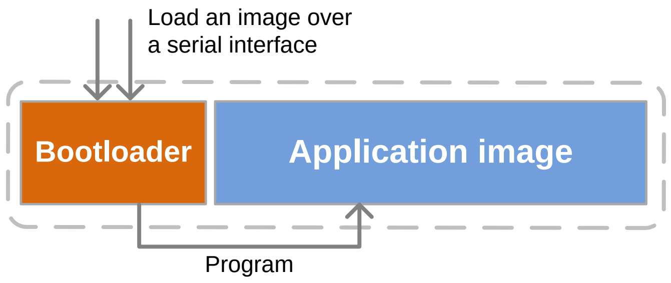
Option	Required value
WDLOCK	OFF
JTAGEN	ON
EESAVE	OFF
BODLVL	1.6V
Resulting fuse bytes	
FUSEBYTE0	*0xFF*
FUSEBYTE1	*0x00*
FUSEBYTE2	*0xBF*
FUSEBYTE4	*0xFE*
FUSEBYTE5	*0xFF*

3. Embedded bootloader

Embedded bootloader occupies a little amount of memory and serves the only purpose, to load an application image from a serial interface and write it to the MCU's internal flash and/or EEPROM. A simple communication protocol described in Sections 3.5 and 3.6 is used to ensure proper programming. Figure 3-1 illustrates bootloader organization.

The embedded bootloader application is provided as a set of pre-compiled firmware images for various configurations and the source code. The user can immediately start using boot loading by programming devices with the pre-compiled images. It is also possible to modify the source code of the embedded bootstrap and compile it using the appropriate toolchain as described in Sections 3.2-3.4 .

Figure 3-1. Embedded bootloader architecture.



In addition to the common bootloader that simply programs application firmware to the flash and EEPROM, embedded bootloader application may be built to support additional features like Over-the-Air Upgrade (OTAU) applied by BitCloud applications or RF4CE features. The OTAU bootloader is able to load an application image, which has been received by the application over the air and saved in an external memory device, and program it into the internal flash memory (see Section 3.7). RF4CE features are invoked by the application to enable a special type of the Over-the-Air upgrade when the application image is replaced in memory simultaneously with reception of the new image. For detail refer to Section 3.8.

3.2 Configuration options

Build configuration options for the embedded bootloader such as target MCU and supported feature set (BitCloud OTA or RF4CE) are defined in the embedded bootloader IDE projects and configurations if building with an IDE. For compilation with command line such options can be selected in the `Makefile` and will also be contained in makefiles present in the `makefiles/` directory.

Application configuration options for embedded bootloader are collected in the `configuration.h` file. Using this file it is possible to configure the following parameters:

- Type and port of the serial interfaces enabled for reception of new firmware image from the host. This are defined separately for each MCU with the `#ifdef` sections. For example, enabling only the USART1 port for an Atmel ATmega128RFA1 MCU is done like this:

```
#ifdef ATMEGA128RFA1
  // Use USART0
  #define USE_USART0 0
  //#define USE_USART0 1

  // Use USART1
  //#define USE_USART1 0
  #define USE_USART1 1
```

```
    // Use USB_FIFO
    #define USE_USB_FIFO 0
    //#define USE_USB_FIFO 1

    // Use TWI
    #define USE_TWIS 0
    //#define USE_TWIS 1

  #ifdef USE_TWIS
    // Use TWID
    //#define USE_TWIS_D 0
    #define USE_TWIS_D 1
  #endif

  #endif //#ifdef ATMEGA128RFA1
```

- Type of external flash device used to store the application image during an over-the-air upgrade procedure. This is done via `TYPE_EXTERNAL_MEMORY` parameter and is applicable only to configurations with BitCloud OTA feature enabled. For an overview of BitCloud OTA bootloader and more references see Section 3.7.

3.3 Compiling embedded bootloader application

The bootloader application can be compiled with Atmel Studio [6] or IAR [7] toolchains of versions given in Table 3-1.

For convenience the bootloader package includes precompiled `.hex` images of the embedded bootloader in most common configurations as described in Section 1.3.2.

Table 3-1. Supported toolchains and their versions

Toolchain	Comments
Atmel Studio v6.1 (with GCC compiler)	For AVR and Xmega devices only
IAR for AVR v6.20	For AVR and Xmega devices
IAR for ARM® v6.20.3.52642	For SAM devices

To compile the bootloader application in Atmel Studio, open witht the Atmel Studio the `.atsln` project file that corresponds to the target platform and feature set from the `\Embedded_Bootloader_src\as6_projects` directory. Then choose a particular build configuration from the list on the toolbar, and select `Build` form the `Build` menu.

To compile the bootloader application in IAR IDE, open with corresponding IAR toolchain the `.eww` project file from the `\Embedded_Bootloader_src\iar_projects` directory. Then choose a particular build configuration that corresponds to the target platform and feature set and select `Rebuild All` form the `Project` menu.

3.3.2 Compilation from the command line

When the embedded application is compiled using the command line, compilation employs `Makefile` placed in the `\Embedded_Bootloader_src\` folder to determine what makefile from the `makefiles` directory shall be used. In `Makefile`, specify `PROJECT_NAME` to choose among supported MCUs and `CONFIG_NAME` to select a particular configuration.

3.4 Programming embedded bootstrap

If not already present on the MCU the firmware part of embedded bootloader (`.hex` file) shall be uploaded to the device to enable programming over serial interface. Atmel Studio [6] is recommended for loading an embedded bootloader image. Note that to work correctly embedded bootloader requires specific fuse bit settings that depend on the target platform and bootloader functionality (see Section 2.2 Fuse bits settings).

Caution: Setting fuse bits incorrectly may cause improper device functionality.

3.5 Bootloader UART/SPI/TWI programming sequence

The embedded bootloader handles data received over UART/SPI/TWI interfaces using algorithm illustrated on Figure 3-2 and described in steps below.

1. After the device is reset, the embedded bootloader waits 200ms on each enabled interface for a `HANDSHAKE_REQ` data sequence to arrive. If no `HANDSHAKE_REQ` is received on none of the interfaces, the embedded bootloader jumps to the application's entry point in the flash memory.
2. If a `HANDSHAKE_REQ` sequence is received, bootstrap code sends a `HANDSHAKE_CONF` sequence over the same serial interface and expects reception of SREC records one by one.
3. After every valid reception of a SREC record the embedded bootloader responds with an `ACK` data sequence over the serial interface.
4. In case of any error during loading process, the embedded bootloader sends a `NACK` data sequence and then proceeds to step 1.

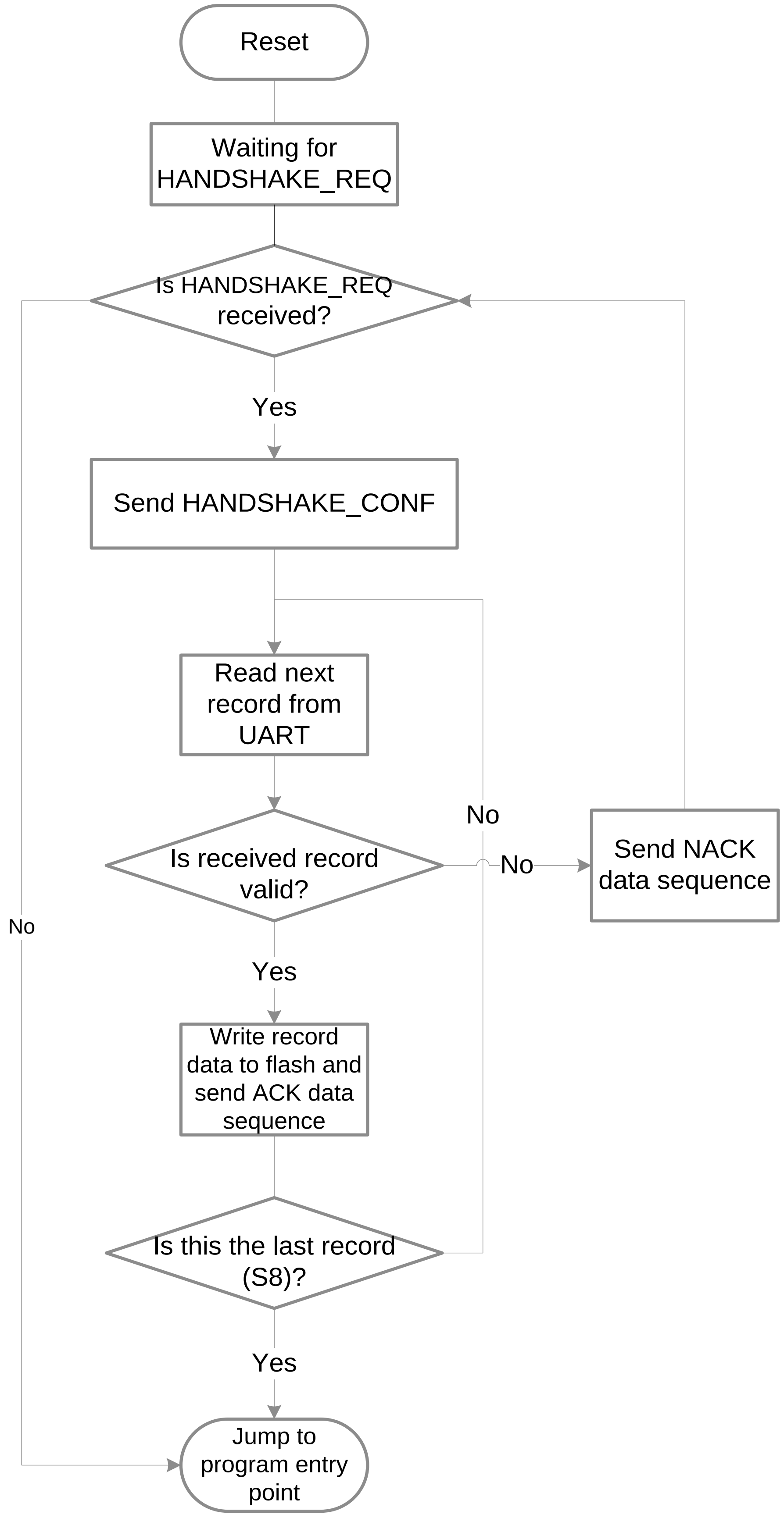
As the host side, Atmel Bootloader PC tool performs following actions to conform to the embedded bootloader behavior:

1. The Bootloader PC tool sends a `HANDSHAKE_REQ` data sequence on specified serial port for 30 seconds with 200ms interval, waiting for a `HANDSHAKE_CONF` data sequence between transmissions. Any reply except `HANDSHAKE_CONF` is ignored.
2. If `HANDSHAKE_CONF` is received, the PC bootloader starts sending data from the SREC file via the serial link. Each record of the SREC file is converted to binary representation before sending.
3. For every record sent the host expects an `ACK` over the serial link as a response. If a `NACK` sequence is received or a timeout occurs, the PC bootloader aborts.

Serial data sequences used between the device and the host while boot loading:

- `HANDSHAKE_REQ:  0xB2 0xA5 0x65 0x4B`
- `HANDSHAKE_CONF: 0x69 0xD3 0x0D 0x26`
- `ACK:            0x4D 0x5A 0x9A 0xB4`
- `NACK:           0x2D 0x59 0x5A 0xB2`

Figure 3-2. Embedded serial bootloader programming algorithm.


Reset
Waiting for HANDSHAKE_REQ
Is HANDSHAKE_REQ received?
Yes
No
Send HANDSHAKE_CONF
Read next record from UART
Is received record valid?
No
Send NACK data sequence
Yes
Write record data to flash and send ACK data sequence
Is this the last record (S8)?
No
Yes
Jump to program entry point

3.6 Bootloader USB DFU specifics

The embedded bootloader for Atmel SAM3S microcontrollers operates in a different way than shown on Figure 3-2 to conform to certain features of the SAM3S microcontroller and development boards, on which it is hosted.

Embedded bootloader for SAM3S uses Device Firmware Upgrade (DFU) standard. A DFU component should be included in the application (see Section 3.6.2.1). The procedure of uploading a new firmware image does not change for the user (see Section 4.1.2). The remaining part of this section describes implementation details of SAM3S embedded bootloader and is intended for users who will possibly modify the embedded bootloader code.

3.6.1 Changing the reset vector

The Atmel SAM3S MCU does not contain a special bootloader section in the flash memory[1]. The fuse settings do not indicate whether to start execution from the application section or from the bootloader section and where this section starts. So if the bootloader code is first programmed at the end of the flash memory and then the application code is loaded into the remaining part of the flash memory, the MCU will start code execution from the last loaded code after reset (that is, the application code).

In order for the embedded bootloader to work correctly it must receive control first after reset to be able to communicate with the host software and upload a new device image before the application code is executed.

To achieve this, the embedded bootloader for SAM3S, during uploading, reassigns the reset vector value (the MCU component that stores the address to start from reset), setting it to its own starting address. Thus the MCU will always start execution from the embedded bootloader, which, in turn, will pass control to the application section upon completion.

3.6.2 DFU impact

Another feature involves notifying embedded bootloader that the PC tool is ready to transfer a new firmware image to the device. For the Atmel AVR and XMEGA MCUs the device must be reset before the new software upload can take place. Upon reset the COM port assigned to the device on the PC is still enabled. The bootloader receives control and waits for the uploading requests from the PC tool. However, to reset a USB stick with SAM3S it should be removed and re-attached to the PC. This may cause the PC to assign a different COM port to the device and stop the serial communication with the device.

To prevent serial port reassignment when using SAM3S, the USB Device Firmware Upgrade components based on [13] should be used. DFU components should be included in both the embedded bootloader and the application code (see Section 3.6.1). The DFU component inserted in the application code should always be enabled along with other application components. Upon receiving a command indicating that a new firmware image is ready, it writes a special byte into the program memory and initiates a device reset. After reset, the DFU component included in the embedded bootloader checks this status byte and enters the new firmware upload mode.

If the embedded bootloader is the only image loaded in the program memory, it will constantly be ready to receive and upload a new firmware image from the host.

To program a device that contains the embedded bootloader with DFU support (such as the embedded bootloader for Atmel SAM3S) you may also use the third-party software, since DFU components provided with this software package closely adhere to the DFU standard.

3.6.2.1 Adding DFU support to an application

The DFU class for the application is implemented on top of the USB driver. Set the `DFU_SUPPORT` parameter to `1` in `configuration.h` file to enable DFU support:

[1] A dedicated bootloader ROM section is included by individual manufactures for their individual use.

```
#define DFU_SUPPORT 1
```

3.6.2.2 Not-responding application or absent DFU

The application loaded to the flash by embedded bootloader may not include a DFU component as well as any special code to deal with PC tool's requests. In other cases when the DFU component is installed, it may stop responding due to application failure. In these cases the device will not be able to process commands from the PC tool and thus initiate reset and perform the firmware upgrade.

To load an image using bootloader into such device, connect JTAG's PB7 pin to ground and reset the device. On an Atmel SAM3S USB stick, these are the first two pins of the JTAG port and so can be connected with a jumper wire.

3.6.2.3 Additional remarks

Below are some more features:

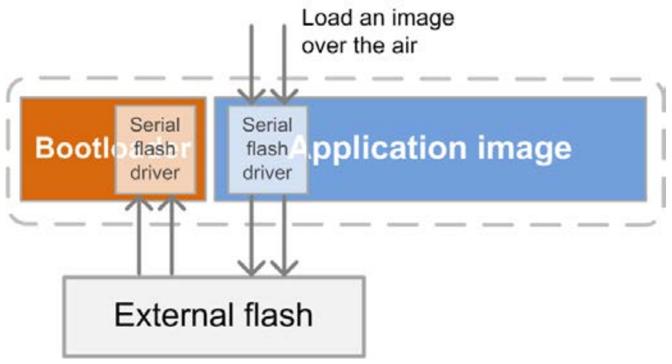
- If one loads a firmware image to a device via the embedded bootloader for SAM3S and then reads the image using JTAG it will not equal the initial binary image. The difference will be in the first four bytes where the bootloader replaced the reset vector.
- Both the embedded bootloader and an application cannot be loaded via JTAG. If the device is first programmed with the embedded bootloader and then with the application, by default the execution will start with the application code. And if the application is programmed first, the embedded bootloader once programmed will not be aware of the presence of the application.

3.7 BitCloud OTAU bootloader features

The OTAU bootloader is a version of embedded bootloader that can additionally be used to upload BitCloud [3] [4] and application to a device through Over-the-Air upgrade (OTAU). Unlike the common bootloader, the OTAU bootloader contains a driver, which is able to transfer an image from the external flash device to the MCU's flash memory as illustrated in Figure 3-3.

The OTAU bootloader is still able to write an application image received via a serial interface to the flash as a common embedded bootloader. But it has more functions. An application image, supporting OTAU typically includes the driver, which writes the parts of the new application image received during the upgrade to an external flash device. Once a new image is loaded, the application sets a new image available status bit in EEPROM and resets. On startup after the reset the bootloader checks this status bit and, if it is set, transfers the image from the external flash memory to the internal MCU's flash memory.

Figure 3-3. OTAU embedded bootloader architecture.



Over-the-Air Upgrade usually requires a special device that will distribute the application image through the network. To start an upgrade, such device must be connected to a PC with the Bootloader PC tool installed. The Bootloader PC tool contains the *OTAU* tab, which is used to initiate and control an upgrade. For further details on the topic of OTAU refer to [8] as well as BitCloud / BitCloud Profile Suite Quick Start Guides [3], [4].

3.8 Bootloader configuration for Atmel RF4CE

The embedded bootloader configuration to support Atmel RF4CE [2] applications is primarily the same as for the common embedded bootloader, but it also provides additional APIs to support Over-the-Air upgrade and the persistent data storage feature that can be used by RF4CE applications. The APIs provided by the serial bootloader and the additional APIs for RF4CE, both make use of self-programming functions. These extra APIs to support Over-the-Air upgrade and persistent data storage are written into the RWW section (the bootloader section). These APIs can also be used by other applications for the purposes as listed below:

- Storing the persistent data (for example, NIB) in the flash memory
- Clearing the persistent data (for example, clear NIB)

4. Bootloader PC tool

The Bootloader PC tool is a PC application that is used to load a firmware image to a device programmed with the embedded bootloader code. Connection to the MCU shall be enumerated as a COM or USB port depending on the interface used. Source firmware images shall be in the Motorola S-record (SREC) format. Such images are created during compilation further to `.hex` and `.elf` images. A device must be pre-programmed with an image of the embedded bootloader application, which will receive data sent by the Bootloader PC tool and write it to the device's flash.

Another use of the Bootloader PC tool is initiating Over-the-Air Upgrade. This document does not cover this usage of the tool for this purpose. A complete guide to Over-the-Air Upgrade of ZigBee networks is given in [8].

4.1 Using the Bootloader PC tool

To install the Bootloader PC tool, launch the installation file from the `\PC_Bootloader_Setup\` and proceed with instructions. After setup is completed both GUI and console versions of the bootloader are extracted to the provided installation path. The Bootloader PC tool is a Java application, and to launch it Java [16] must be installed on the host PC.

As a firmware image the Bootloader PC tool requires a file in the Motorola S-record hexadecimal format, also known as SREC or S19 format. Such file names have the `.srec` extension and can contain both flash memory and EEPROM images.

4.1.1 Generating SREC images

4.1.1.1 Using GCC tools

An SREC image can be generated from initial binary image with the help of a special tool supplied with a tool chain.

- For Atmel AVR use `avr-objcopy.exe` tool provided in AVR toolchain for Atmel Studio 6
- For ARM use `arm-elf-objcopy.exe` tool, which is a part of the Yagarto GCC toolchain

For example, to generate an SREC image from a .hex image for AVR run a command in the following format from the command line:

```
avr-objcopy.exe -O srec --srec-len 128 <srcFile> <destFile>
```

For ARM just replace the name of the AVR tool with `arm-elf-objcopy.exe`. Note that this tool takes an image in the `.elf` format as input.

4.1.1.2 Using IAR tools

If you use IAR toolchain to compile applications you may also specify a directive for the linker to create an SREC image:

- For command line compilation add `-Omotorola-s28=<path>/<fileName>.srec` to the linker flags
- For IAR Embedded Workbench
  - Open configuration's options and go to `Linker`
  - On the `Extra Output` tab check the `Generate extra output file` box, specify the image name and select `motorola-s28` as `Output format`

An image generated by the IAR compiler may be also converted to SREC format with the `avr-objcopy.exe` or `arm-elf-objcopy.exe`.

4.1.2 Loading application firmware to an MCU

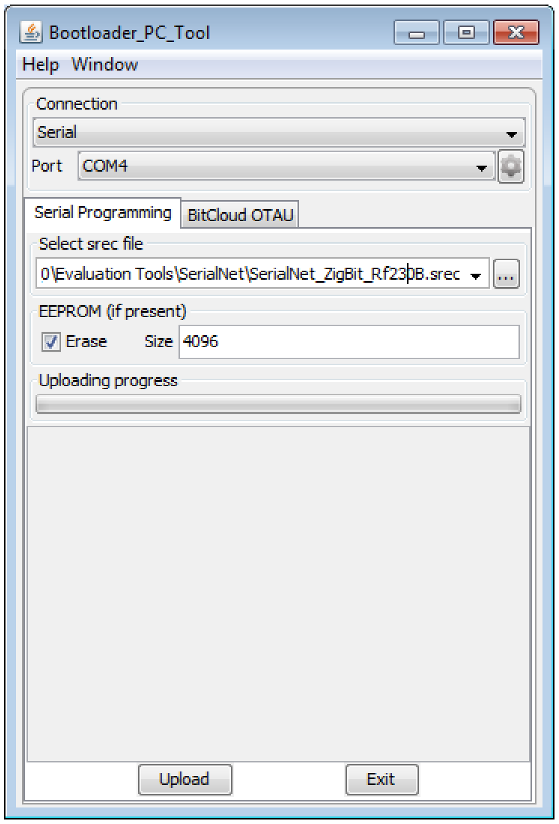
To program an MCU using serial bootloader the following steps shall be done:

1. Connect an MCU with the embedded bootloader firmware on it to a PC via serial connection. For detail refer to Section 2.1 as well as documentation of the development kit or the software provided with it.
2. For the GUI version of serial bootloader run the application by double-clicking the `bootloadergui.exe` file and the control dialog as shown on Figure 4-1 will appear. For the console version just start command line from the Bootloader PC tool installation directory (see Section 4.1.3 for more detail).
3. Specify uploading parameters:
    a. Select the connection type. For all devices except Atmel SAM3S select `Serial`, while for SAM3S select `USB`.
    b. Select the port from the drop-down list.

There is a restriction on the size of firmware downloadable by serial booting process. Serial bootloader cannot rewrite the area where the bootstrap code resides.

4. Press the `Upload` button if Bootloader PC GUI tool is used. For the console bootloader press `Enter` on the keyboard to start uploading.
5. Press the HW reset button on the device if requested. The Bootloader PC tool will be waiting for approximately 30 seconds for the button to be released. If this does not happen, programming will be aborted.
6. The Bootloader PC tool will indicate the programming progress. Once loading is finished successfully, the device will be restarted automatically. If loading fails, the Bootloader PC tool will indicate the reason. In case the new image upload fails (for example, because of random communication errors) the device should be re-programmed. It the reprogramming does not resolve the issue then the previously programmed code image in the device may be corrupted. The device should be erased and reprogrammed via JTAG.

Figure 4-1. The Bootloader PC tool main GUI window.



Connection: Serial
Port: COM4
Select srec file: 0\Evaluation Tools\SerialNet\SerialNet_ZigBit_Rf230B.srec
EEPROM (if present)
☑ Erase Size 4096
Uploading progress

4.1.3 Using the command line

MCU programming with PC Bootloader tool can be also done via command line interface as follows:

```
bootloader –f <file_name> –p <com_port> [-e <eeprom size>]
```

For example following line:

```
bootloader –f C:\work\SerialNet_ZigBit_Rf230B.srec –p COM4
```

can be used to load `C:\work\SerialNet_ZigBit_Rf230B.srec` image via port enumerated as `COM4.` Figure 4-2 shows a screenshot with an example of successful firmware upload using command line control.

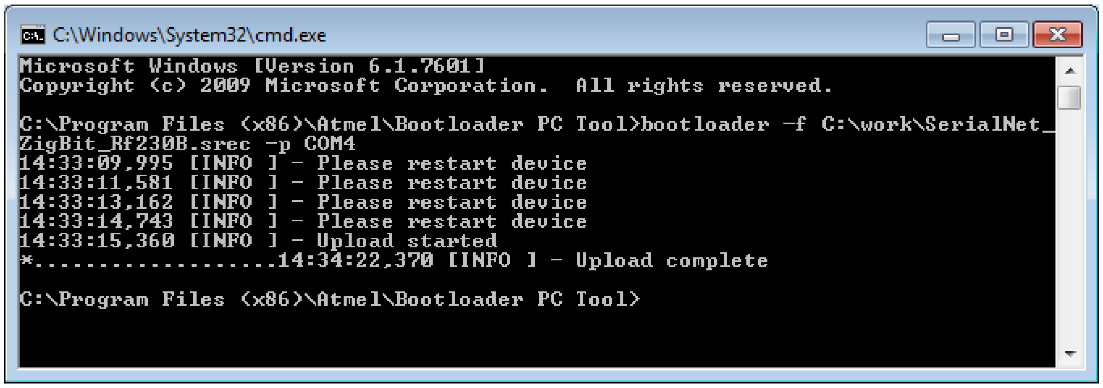
Note: With the command line COM-port settings will be automatically used as given in Table 2-1 and cannot be changed.

Besides, when the `-e <eeprom_size>` option is specified the EEPROM section is cleared.

To see help information run:

```
bootloader -h
```

Figure 4-2. Loading application firmware using command line control of PC Bootloader tool.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files (x86)\Atmel\Bootloader PC Tool>bootloader -f C:\work\SerialNet_
ZigBit_Rf230B.srec -p COM4
14:33:09,995 [INFO ] - Please restart device
14:33:11,581 [INFO ] - Please restart device
14:33:13,162 [INFO ] - Please restart device
14:33:14,743 [INFO ] - Please restart device
14:33:15,360 [INFO ] - Upload started
*.................14:34:22,370 [INFO ] - Upload complete

C:\Program Files (x86)\Atmel\Bootloader PC Tool>
```

4.1.3.2 Command line options

Table 4-1 describes set of command line options supported by the PC bootloader tool and corresponding GUI control elements (the GUI window is shown on Figure 4-1). Options can be entered in any order.

Table 4-1. Command-line options and corresponding GUI controls.

Command line option	GUI control	Description
-h	N/A	Print help information
-f	Select SREC file	Path to the firmware image of SREC format to be loaded to the MCU
-p	Select Serial port	COM port number
-e	EEPROM Erase	Clean up EEPROM section. EEPROM size shall be specified

5. Reference

[1] Atmel IEEE 802.15.4 wireless stack

[2] Atmel RF4CE wireless stack

[3] Atmel BitCloud ZigBee wireless stack

[4] Atmel BitCloud Profile Suite wireless stack

[5] Atmel LightWeightMesh wireless stack

[6] Atmel Studio Software

[7] IAR Embedded Workbench

[8] AVR2058; BitCloud OTAU User Guide; Atmel Corporation

[9] Atmel AVR2063: Sensor Terminal Board - Hardware User's Manual

[10] Atmel ZigBit® 2.4GHz wireless modules. ATZB-24-A2/B0 datasheet

[11] Atmel ZigBit 2.4GHz wireless modules. ATZB-A24-UFL/U0 datasheet

[12] Atmel ZigBit 700/800/900MHz wireless modules. ATZB-900-B0 datasheet

[13] Universal Serial Bus Device Class Specification for Device Firmware Upgrade

[14] AVR1927: XMEGA-A1 Xplained Getting Started Guide

[15] Atmel AVR1924: XMEGA-A1 Xplained Hardware User's Guide

[16] Java® Runtime Environment

6. Document revision history

Doc. Rev.	Date	Comment
A	15.02.2011	First release
B	15.11.2011	Document for separate package with OTA , RF4CE and DFU support
C	11.06.2013	Updated with 256RFR2 and TWI support

Enabling Unlimited Possibilities®

Atmel Corporation	**Atmel Asia Limited**	**Atmel Munich GmbH**	**Atmel Japan G.K.**
1600 Technology Drive	Unit 01-5 & 16, 19F	Business Campus	16F Shin-Osaki Kangyo Bldg.
San Jose, CA 95110	BEA Tower, Millennium City 5	Parkring 4	1-6-4 Osaki, Shinagawa-ku
USA	418 Kwun Tong Road	D-85748 Garching b. Munich	Tokyo 141-0032
Tel: (+1)(408) 441-0311	Kwun Tong, Kowloon	GERMANY	JAPAN
Fax: (+1)(408) 487-2600	HONG KONG	**Tel:** (+49) 89-31970-0	**Tel:** (+81)(3) 6417-0300
www.atmel.com	**Tel:** (+852) 2245-6100	**Fax:** (+49) 89-3194621	**Fax:** (+81)(3) 6417-0370
	Fax: (+852) 2722-1369		