Atmel AVR2052: Atmel BitCloud Quick Start Guide

Features

- **Introduces Atmel® BitCloud® Software Development Kit (SDK)**
- **Introduces WSNDemo application**

1 Introduction

This document is intended for engineers and software developers evaluating the Atmel BitCloud ZigBee® PRO stack.

The BitCloud SDK and the supported kits serve as the perfect vehicle to evaluate the performance and features of Atmel microcontrollers and radio transceivers as devices in a low-power, Atmel ZigBee-compliant wireless sensor network. The SDK provides a complete software and documentation toolkit for prototyping, developing, and debugging custom applications on top of the BitCloud application programming interface (API), as well as easily demonstrating ZigBee technology in action.

This document describes how to start quickly with the BitCloud SDK by installing development environment and programming devices with sample applications. It also provides a short introduction to the BitCloud API. To find more detailed information concerning application development, refer to [2].

8-bit Atmel Microcontrollers

Application Note

Rev. 8200N-AVR-05/12

2 Overview

Atmel BitCloud is a full-featured, professional-grade, embedded software ZigBee stack from Atmel. The stack provides a software development platform for reliable, scalable, secure wireless applications running on Atmel microcontrollers and radio transceivers. BitCloud is designed to support a broad ecosystem of user-designed applications addressing diverse requirements while enabling a full spectrum of software customization.

The BitCloud stack is fully compliant with the Atmel ZigBee PRO and ZigBee standards for wireless sensing and control. It provides an augmented set of APIs, which, while maintaining full compliance with the standard, offer extended functionality designed with developers' convenience and ease-of-use in mind.

2.1 Supported platforms

The following hardware platforms are supported by the BitCloud SDK.

Table 2-1. Supported hardware platforms.

Name in this document	Platform (MCU + RF)	Supported modules	Supported external flash for OTAU	Supported evaluation kit	SDK package
ZigBit®	ATmega1281 + AT86RF230	ATZB-24-B0 ATZB-24-A2 ATZB-A24-UFL ATZB-900-B0	AT25F2048, AT45DB041	N/A	BitCloud for ZigBit
megaRF	ATmega128RFA1	N/A	AT45DB041 (only 264-byte pages)	RCB128RFA1 and ATAVR128RFA1-EK1 with ATSTK600 boards	BitCloud for megaRF
XMEGA®	ATxmega256A3 / ATxmega256D3 + AT86RF231 or ATxmega256A3 / ATxmega256D3 + AT86RF212 or ATxmega256A3 / ATxmega256D3 + AT86RF230	N/A	AT45DB041 (only 264-byte pages)	ATxmega card hosted on STK®600 and RZ600 radio boards ATREB212ED-EK ATREB231ED-EK ATREB232ED-EK	BitCloud for AVR® XMEGA
UC3	AT32UC3A0512 + AT86RF231	N/A	N/A	EVK1105	BitCloud for AVR UC3 32-bit MCUs
SAM3S	ATSAM3S4C + AT86RF231 or ATSAM3S4C + AT86RF230 or ATSAM3S4C + AT86RF212	N/A	N/A	SAM3S-EK and RZ600 radio boards Atmel RF212USB-RD Atmel RF231USB-RD	BitCloud for SAM3S-EK

Name in this document	Platform (MCU + RF)	Supported modules	Supported external flash for OTAU	Supported evaluation kit	SDK package
SAM7X	AT91SAM7X256 + AT86RF231 or AT91SAM7X256 + AT86RF230 or AT91SAM7X256 + AT86RF212	N/A	N/A	AT91SAM7X-EK and RZ600 radio boards	BitCloud for SAM7X-EK

Please note that this document describes the use of the Atmel BitCloud SDK with the specific modules and evaluation kits listed in Table 2-1. Operation of BitCloud on supported MCU/RF combinations realized in a custom hardware application is outside the scope of this document.

3 Getting started

This document describes how to quickly get Atmel BitCloud running. A typical starting scenario consists of the following:

- Setting up the development environment. Note that platform-specific details are moved to the corresponding appendix chapters, as described in Table 3-1
- Programming several devices with demonstration firmware. This topic is also described in the corresponding appendix chapter
- Monitoring activity of a network formed by the devices, with a sample wireless sensor network (WSN) monitor PC application provided with the SDK. This topic is described in detail in Chapter 4
- Starting the development of new or modifying existing applications based on the BitCloud C API, as shown in Chapter 5

The BitCloud SDK is available for several platforms, as described in Section 2.1, and so before proceeding, select the SDK version that matches your target platform.

The majority of instructions for setting up the BitCloud stack and applications depend on the specific platform and evaluation kit. To get started, proceed to the platform-specific sections listed below.

Table 3-1. Hardware-specific getting started sections.

For platform	Refer to Section
ZigBit	8.1
megaRF	9.1
UC3	10.1
XMEGA	11.1
SAM7X	12.1
SAM3S	13.1

After completing the installation, try running the WSNDemo application by programming the devices with ready-to-use images, as described in Section 4.2.

4 WSNDemo application

4.1 Overview

The network and radio frequency performance of the hardware components is demonstrated with the WSNDemo application, which is based on the Atmel BitCloud API. This application consists of the embedded firmware, which supports functions for coordinator, router, and end device, and the GUI visualization application, WSNMonitor, which is run on a PC. In WSNDemo, the nodes communicate based on a proprietary messaging protocol.

The SDK includes the WSNMonitor PC application in binary format, and the WSNDemo embedded application is available in binary format and source code.

The source code for the WSNDemo application can be modified and extended, making it possible to develop WSN applications for a variety of application scenarios. WSNMonitor is described in Section 4.5. Some more details on WSNMonitor and WSNDemo are given in [1].

With WSNDemo installed, the devices are organized into a set of nodes forming a ZigBee PRO network.

With Atmel ZigBit development boards, end devices, routers, and the coordinator read the sensor data from onboard light and temperature sensors, and forward collected data to the WSNMonitor application for visualization. On Atmel XMEGA, Atmel SAM7X, Atmel SAM3S, Atmel megaRF, and Atmel UC3 platforms, zero values are sent to the network coordinator to emulate sensor data and demonstrate data transmission.

End devices follow a duty cycle (that is, the microcontroller and radio transceiver are put to sleep periodically) and wake up to transmit data to the coordinator. Using the serial connection, the coordinator transmits the received packets, along with its own sensor data (or emulated sensor data), to the WSNMonitor application. Those transmitted values are displayed on WSNMonitor panes as temperature, light, and battery level measurements.

WSNMonitor also visualizes network topology by drawing a tree of nodes that have joined the network. For each of the nodes, parameters like node address, node sensor information, and link quality data are displayed.

RSSI indicates a link's current condition and is measured in dBm. The RSSI resolution is 3dBm. *LQI,* a numeric parameter defined within the 0 to 255 range, is used to measure the link quality. Larger values mean a better link, while values close to zero indicate a poor connection.

In WSNDemo, the number of routers and end devices is limited only by the network parameter settings.

In reference to the WSNDemo application, Section 4.3 describes how to set up and use the boards. The user interface is described in Section 4.5.

The application is delivered with source code that demonstrates how to develop a wireless network application using the Atmel BitCloud API, and provides a number of useful programming templates for common application tasks. Development of custom applications is described in Section 4.5.

4.2 Programming the boards

As a first step, WSNDemo images should be loaded onto the boards. The locations of the WSNDemo image files are platform specific, and are provided in the “Installing the SDK” subsection of the sections specified in Table 3-1.

The programming instructions and the sets of pre-built application images provided with the SDK also depend on the target platform. Table 4-1 provides references to the sections that describe how to program each target platform and evaluation kit.

Table 4-1. Platform-specific programming sections.

For platform	Refer to Section
ZigBit / ZigBit Amp / ZigBit 900	8.2
megaRF	9.2
UC3	10.2
XMEGA	11.2
SAM7X	12.2
SAM3S	13.2

Running any ZigBee or ZigBee PRO application, WSNDemo included, requires that every device in the network have a unique, 64-bit MAC address. See the appropriate sections in Table 4-1 for how MAC addresses are assigned for each type of supported board. In order to make initial setup easier, there are a number of pre-compiled images provided with the SDK that can be used right away without any modification.

Atmel ZigBit platform does not require manual assignment of MAC addresses, as the evaluation boards are equipped with a dedicated unique ID chip or EEPROM, which the BitCloud stack uses automatically on startup. Also, note that the default images are configured to use a particular extended PAN ID and channel mask. To change these parameters, the user must modify the `configuration.h` file included in the project and rebuild the application. Special care must be taken by the user when configuring an application so that each compiled image contains a unique MAC address and all images share the same extended PAN ID and channel mask.

4.3 Running WSNDemo

Each target platform requires its own set of instruction for getting the WSNDemo application to run. Table 4-2 provides references to the sections with platform-specific instructions.

Table 4-2. Platform-specific WSNDemo sections.

For platform	Refer to Section
ZigBit	8.4
megaRF	9.4
UC3	10.4
XMEGA	11.4
SAM7X	12.4
SAM3S	13.4

4.4 Network startup

The coordinator organizes the wireless network automatically. Upon starting, every node informs the network of its role.

When the coordinator is powered on, it switches to an active state even though no child node is present. This is normal, and it indicates that the coordinator is ready and that child nodes can join the network with the coordinator's extended PAN ID. By default, the coordinator uses extended PAN ID `0xAAAAAAAAAAAAAAAA`, which is recognized by all routers. A short PAN ID is chosen at random. The extended PAN ID can be modified by the user through the application's `configuration.h` file.

NOTE

If the coordinator is absent or has not been turned on, the routers and end devices will remain in the network search mode. In this mode, routers scan the channels specified in the channel mask in search of a network with the specified extended PAN ID.

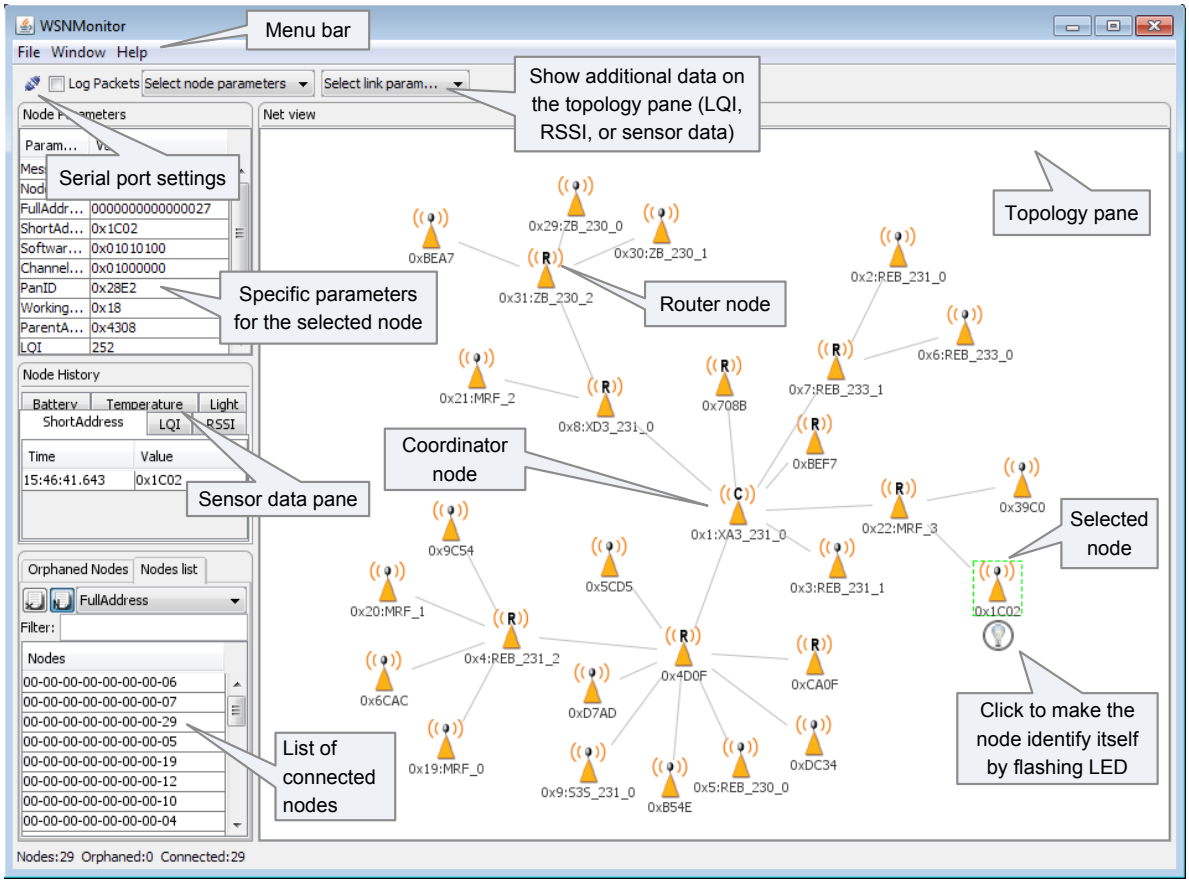
By default, the channel mask for all application images provided with the SDK contains a single channel. In rare cases, if the frequency corresponding to the radio channel is busy, the coordinator node may stay in the network search mode. If this happens, it may become necessary to change the application's channel mask to select another channel by changing the application's `configuration.h` file and recompiling the application.

Network health can be monitored by looking at the individual node's LED state (LED states are platform specific, and are described in platform-specific sections noted in Table 4-2) or through the WSNMonitor application described in the next section.

4.5 WSNMonitor

WSNMonitor is a PC counterpart to the WSNDemo embedded application, and can be used to display ZigBee network topology and other information about a wireless sensor network. A typical WSNMonitor screen is shown in Figure 4-1. It contains topology, sensor data, and node data panes and application toolbars.

Figure 4-1. WSNMonitor GUI.



The *topology pane* displays the network topology in real time, which helps the user monitor the formation of and dynamic changes in the network while nodes join, send data across, or leave the network. The network topology is constructed on the basis of next-hop information for each of the nodes, and each link is also tipped with RSSI and LQI values. Each of the nodes displayed is depicted by an icon, with the node's address or name below and sensor readings to the right of the icon, if required by settings.

The *sensor data pane* displays data coming from onboard sensors of the selected node (see Section 4.5.2). It is presented in graph and table form. Other parameters can be observed for each node in table form. The *node data pane* includes a *sensor selection combo-box*, which is used to switch between sensor types.

By default in the topology pane, nodes are labeled with their short addresses. However, another title can be assigned to any desired node by a double click. If "Cancel" is pressed in the opened window, the node's title is set back to the short address.

4.5.1 Identifying nodes

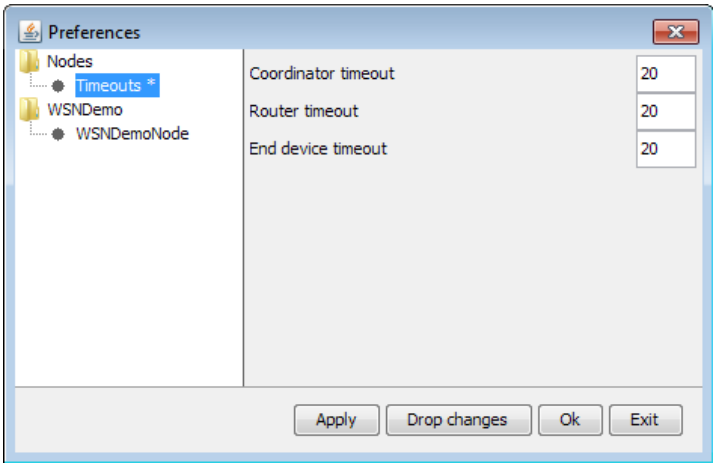
When a user clicks a node in the topology pane a button that can be used to identify the node appears under the node's icon. When the user clicks this button WSNMonitor sends a command, which is delivered to the coordinator through the

serial connection and wirelessly to the target node. The target node, receiving the command, blinks with its LED for several seconds.

4.5.2 Node timeouts

The `Window/Preferences` menu of WSNMonitor contains a number of parameters used to control application behavior. Timeouts are used to tune visualization of coordinator, routers, and end devices as the nodes disappear from the network each time a connection is lost, power is down, or a reset has occurred. A node timeout corresponds to the time the WSNMonitor application waits for a packet from a particular node before assuming that the node is no longer part of the network. Note that this value does not correspond to the frequency with which data are transmitted by each type of device. To get smooth topology visualization, setting timeouts to 20 seconds is recommended for coordinator and router, and 30 seconds is recommended for an end device. Assuming a default application configuration, these timeouts cover three periods between sending a packet, and so at least three packets would need to be lost before a node is removed from the WSNMonitor topology pane.

Figure 4-2. WSNMonitor preferences menu.



4.5.3 Sensor data visualization

Each board sends temperature/light/battery sensor readings (or emulated values) to the coordinator, which in turn sends it to the PC. WSNMonitor displays the readings from onboard sensors next to a node icon inside the topology pane. A corresponding option can be selected in the node/link parameters from the quick settings toolbar.

The user can select any node in the topology pane to monitor the node's activity and see the node data in one of three different forms:

- Text table
- Chart
- The onboard sensor's data displayed next to each node in the topology pane. These values are also tipped with arrows indicating whether the value increased or decreased in relation to the previous sample

NOTE

A given node is selected when it is clicked on and a dashed frame is visible around it.

The same values are shown on the sensor data pane, enabling the user to observe how the values change over a period of time.

The sensor data pane includes a sensor selection combo-box. Use the button on the sensor control toolbar to display the desired types of sensor data.

4.5.4 Over-the-air upgrade

Over-the-air upgrade (OTAU) can be executed on supported hardware platforms by loading a special version of the WSNDemo application with OTA support. This demonstration also requires an additional in-network device to perform the role of the upgrade server, which is the device that sends new firmware images to other devices on the network. For this purpose, a device programmed with the Runner application is used. Such device is able to receive and process commands from a PC over the serial link. To control the Runner device, the Bootloader PC tool included into the Atmel Serial Bootloader package is used (see Chapter 6). Pre-built images of the Runner application for various configurations are provided with the SDK and are located in the `\Evaluation tools\Runner` directory.

The details of compiling and running WSNDemo with OTAU support differ for each platform. Please refer to the appropriate section listed in the table below for platform-specific instructions.

Table 4-3. Platform-specific settings for WSNDemo with OTAU support.

For platform	Refer to Section
ZigBit	8.4.3
megaRF	9.4.3
XMEGA	11.4.3

After WSNDemo device(s) with OTAU support are configured, programmed and joined to the network, follow the step-by-step instructions below to execute an over-the-air upgrade:

1. Connect the Runner device to the PC.
2. Start the Bootloader PC tool (from the Microsoft® Windows® Start Menu, select `Atmel > Bootloader > Bootloader`):
    a. On the `OTAU` tab, select the network parameters, as shown in Figure 4-3.
    b. Specify `Connection` settings to match the port where the Runner device is connected.
    c. Click the `Init` button to force the program to collect information about the node's configuration; for example, security settings or network parameters.
    d. Click the `Set keys` button to manually set keys; for example network key, TC link key, public and private keys.
    e. The utility will automatically populate the list of devices that support OTA functionality (that is, applications that include OTA cluster). By default, only devices programmed with WSNDemo with OTA support should be shown in the list, as seen in Figure 4-4.
3. Start the Image Converter PC utility (from Windows Start Menu, select `Atmel > Bootloader > Converter`):
    a. Select the `*.srec` image(s) you wish to upload to a remote, OTA-capable device over the air.
    b. Fill in image metadata information in the fields below and click `Convert`.
4. Return to the Bootloader PC tool:
    a. Click on the `Update` button next to the device information.
    b. In the Open File dialog, select the `*.zigbee` file that was converted in step 3.

c. Click OK, and the upload process will begin. It usually takes 5-10 minutes to upload a single image to a router device; sleeping end device upgrade may last considerably longer.
d. Once the image is uploaded, the progress bar indicating the upload progress will be replaced with the button. The device will not reset and switch to the new image, until the command for this is sent by pressing this button.

Figure 4-3. Bootloader PC tool's main screen.

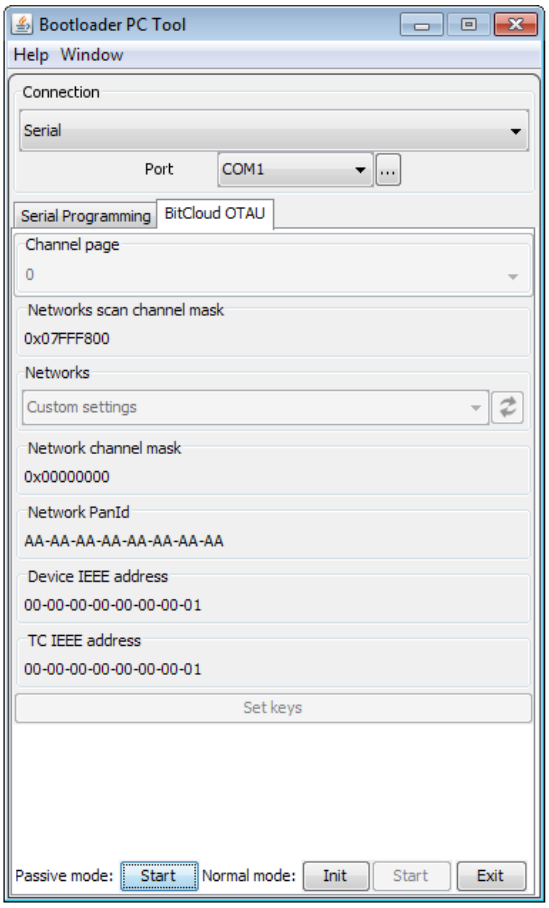
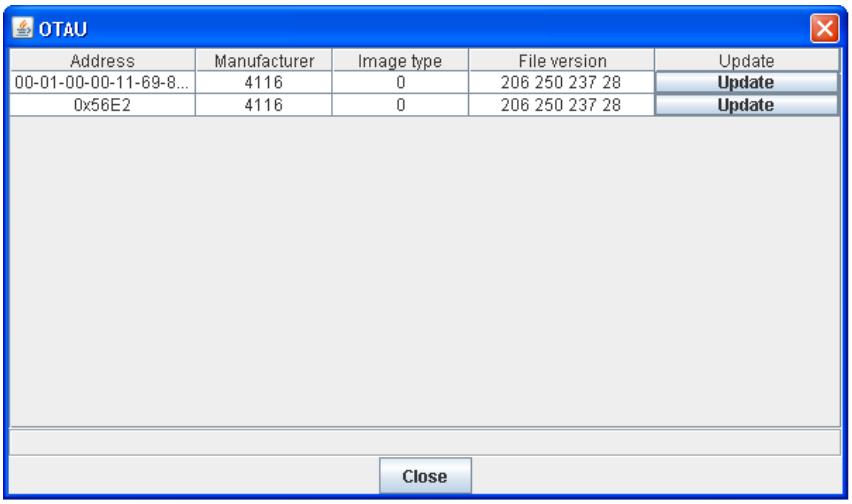


Figure 4-4. Bootloader PC tool's devices screen.



For complete description of OTAU implementation and usage in BitCloud, refer to [35].

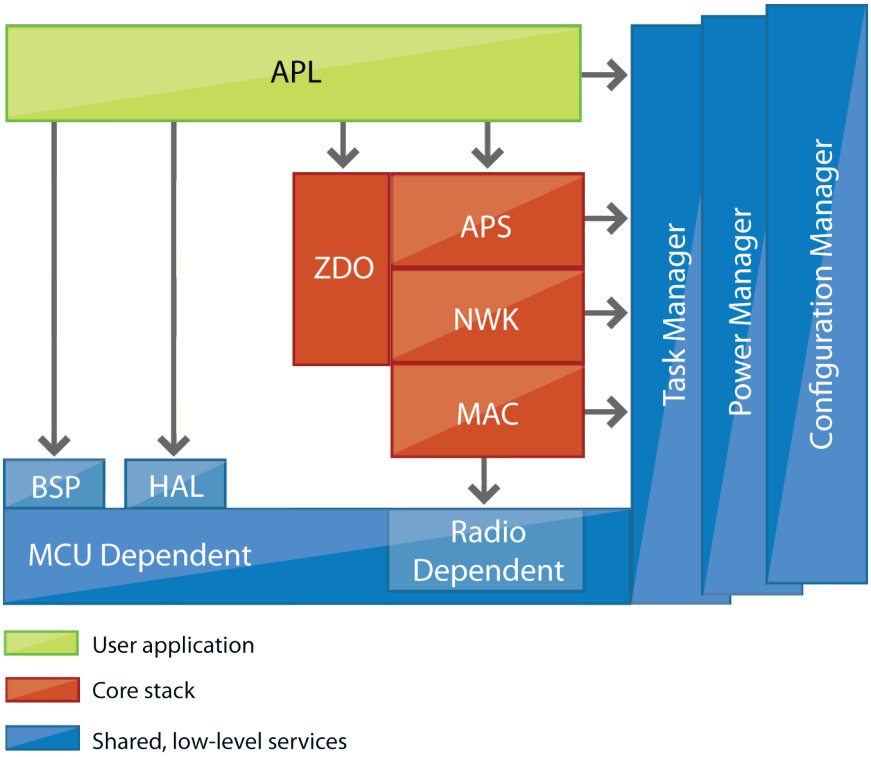
5 Developing custom applications with the Atmel BitCloud API

5.1 API overview

The BitCloud internal architecture follows IEEE® 802.15.4 and the ZigBee-defined convention for splitting the networking stack into layers. Besides the core stack containing the protocol implementation, the BitCloud stack contains additional layers implementing shared services (for example, task manager, configuration manager, and power manager) and hardware abstractions (for example, hardware abstraction layer (HAL) and board support package (BSP)). The APIs contributed by these layers are outside the scope of core stack functionality. However, these essential additions to the BitCloud API significantly reduce application complexity and simplify the development effort. The BitCloud Stack Documentation [1] provides detailed information on the stack's C API and its use.

The main structure of the BitCloud stack is presented in Figure 5-1.

Figure 5-1. BitCloud block diagram.



The topmost layer of the core stack, APS, provides the highest level of networking-related APIs visible to an application. ZDO provides a set of fully compliant ZigBee Device Object APIs, which enable main network management functionality (for example, start, reset, formation, join). ZDO also implements ZigBee Device Profile commands, including Device Discovery and Service Discovery.

There are three service "planes," including task manager, configuration manager, and power manager. These services are available to the user application, and may also be utilized by lower stack layers. The task manager is the stack component which

mediates the use of the MCU among internal stack components and user application. The task manager utilizes a proprietary, priority, queue-based algorithm specifically tuned for a multilayer stack environment and the demands of time-critical network protocols. Power management routines are responsible for gracefully shutting down all stack components, saving the system state when preparing to sleep, and restoring system state when waking up. The configuration manager is used by the internal stack components and the user application to provide a common way to store and retrieve network parameters like the extended PAN ID and channel mask.

The hardware abstraction layer (HAL) includes a complete set of APIs for using on-module hardware resources (for example, EEPROM, app, sleep, and watchdog timers) as well as the reference drivers for rapid design-in and smooth integration with a range of external peripherals (for example, IRQ, TWI, SPI, UART, 1-Wire®), where the hardware interface is supported by the platform. The board support package (BSP) includes a complete set of drivers for managing standard peripherals (for example, sensors, UID chip, sliders, and buttons) placed on development boards, such as those provided with Atmel ZigBit, Atmel ZigBit Amp, and Atmel ZigBit 900 evaluation kits.

Please refer to [1] and [2] for a more detailed description of the Atmel BitCloud API and its features.

5.2 Development tools

A development toolchain consists of:

1. An integrated development environment (for example, Atmel AVR Studio® or IAR Embedded Workbench®), where sample applications may be modified, compiled, and debugged,
2. A corresponding compiler toolchain (for example, AVRGCC, IAR™ or YAGARTO), which provides everything necessary to compile application source code into binary images, and
3. A programming device (for example, JTAG), which may be used to program and debug the application on a target platform.

IAR Embedded Workbench for Atmel AVR [13] can be used to develop and debug applications for AVR-based platforms, including ZigBit, megaRF, and Atmel XMEGA. IAR Embedded Workbench for ARM and AVR32 can be used to develop and debug applications on ARM-based platforms and 32-bit AVR platforms, respectively. All IAR IDEs support editing of application source code, compilation, linking object modules with libraries, and application debugging.

Atmel AVR Studio 5.1 [3] can be used to develop and debug applications for AVR-based platforms. AVR Studio 5.1 is equipped with the GCC toolchain and do not require external tools to compile BitCloud applications, though some additional utilities are needed to compile the HAL component of the BitCloud stack (see [37]).

Eclipse IDE [23] and/or IAR Embedded Workbench for ARM [22] may be used to develop and debug applications based on the BitCloud API on Atmel SAM7X. Eclipse IDE can be integrated with the YAGARTO compiler toolchain [24] to support seamless application programming and debugging on SAM7X, all inside Eclipse IDE.

In AVR Studio, each application has corresponding project files identified by the `.cproj` extension. A project file contains the necessary information about build configuration. Sample applications are provided with project files for AVR Studio 5.1 located in the `as5_projects` subdirectory of the application's root directory. For IAR Embedded Workbench, each application has a corresponding `.eww` file, which can be

double-clicked to open the application's project and located in `iar_projects` directory. For detailed instructions on how to compile and debug applications using the supported tools, refer to Section 5.2.2.

Platform-specific sections that describe development tool installation and provide setup instructions are listed in Table 3-1.

5.2.1 Reference applications

All Atmel BitCloud SDKs are supplied with the WSNDemo reference application provided in source code. WSNDemo is presented in detail in Chapter 4. To better understand the application payload format between network nodes and between the coordinator and the PC, the user can refer to Chapter 14, Appendix B-1: WSNDemo over-the-air protocol and Chapter 15, Appendix B-2: WSNMonitor serial protocol.

Additional sample applications are available for some platforms, as indicated in Table 5-1. The user is encouraged to browse reference application source code as a reference for the customer application being built. In many cases, reference application source code can be used in the target application with only minor modifications.

Table 5-1. Reference applications.

Application	Brief description	ZigBit	megaRF	UC3	XMEGA	SAM7X	SAM3S
WSNDemo	Featured SDK application demonstrating network functionality of software and additional network visualization with WSNMonitor. See Chapter 4	X	X	X	X	X	X
Blink	Introduces the simplest application that uses timer and LEDs. When started, the application makes all the LEDs blink synchronously with a certain period	X	X	X	X	X	X
Lowpower	Shows how to collect data from low-power, sleeping devices employing the simplest power management strategy	X					
Peer2peer	Shows how to organize the simplest peer-to-peer link. A simple buffering strategy is employed to avoid byte-by-byte data transfer	X			X	X	
PingPong	Shows how to process multiple, simultaneous data transmissions. Each node is waiting for a wireless message, and then passes it to the next node	X				X	
ThroughputTest	Measures wireless UART bandwidth of ZigBit, ZigBit Amp, and ZigBit 900 boards	X				X	

For more details on sample applications available for a specific platform, refer to [1].

Once the SDK is installed, the source code for the WSNDemo application can be found in the `<SDK-Root>\Applications\WSNDemo` directory. For other sample applications (where available), the source code can be found in the appropriate `<SDK-Root>\Applications\<application-name>` directory.

Network parameters and their default values are defined in the `configuration.h` file, which is located in the application root directory. For example, for the WSNDemo application, this file can be found in `<SDK-Root>/Applications/WSNDemo/`.

5.2.2 Supported toolchains

The following development environment options are available for each of the supported platforms.

Table 5-2. Platform-specific compilation options.

For platform	AVR Studio 5.1	Eclipse + YAGARTO	IAR Embedded Workbench
ZigBit	X		X
megaRF	X		X
UC3			X
XMEGA	X		X
SAM7X		X	X
SAM3S		X	X

A set of ready to use makefiles for various platform configurations is available for the reference applications. Additionally, the root directory of every application contains a special makefile, which allows switching among different configurations by specifying the configuration name. Custom applications should follow the same structure.

In order to compile an application in each of the available development environments, the following steps should be taken.

5.2.2.1 Atmel AVR Studio 5.1

- *Command line:* Compile the application by running the `make` utility.
- *IDE:* Open an appropriate project file from the `as5_projects` directory with Atmel AVR Studio 5.1, and execute `Build/Rebuild All` from the main menu.

Once the build process is completed, `.hex`, `.srec`, `.bin`, and `.elf` application images will be generated.

5.2.2.2 Eclipse + YAGARTO

- *Command line:* Compile the application by running the `make` utility.
- *IDE:*

1. In Eclipse framework, create a new C project of makefile type with an arbitrary name but with a location in the application directory (for example, the `<SDK-Root>\Applications\WSNDemo` directory).
2. Configure project properties as follows:
    a. Select C/C++ Build, Discovery Options.
    b. In the “Compiler invocation command” section, browse to the compiler (`C:\Program Files\yagarto\bin\arm-elf-gcc.exe`).
    c. Select C/C++ Build, Settings.
    d. Check GNU Elf Parser.
    e. Select GNU Elf Parser.
    f. Browse to the addr2line and c++filt programs (`C:\Program Files\yagarto\bin\arm-elf-addr2line.exe`) (`C:\Program Files\yagarto\bin\arm-elf-c++filt.exe`).
    g. Apply and click `OK`.

While building, the output of the make command will appear in the console pane on the bottom of the window. Once the build process is completed, `.hex`, `.srec`, `.bin`, and `.elf` image files will be generated.

5.2.2.3 IAR Embedded Workbench

- *Command line:* Compile the application by running the `make` utility. Some of the `.hex`, `.srec`, `.bin`, and `.elf` image files will then be generated, depending on the platform configuration that has been chosen.
- *IDE:* Open the `.eww` file in the `iar_projects` subdirectory of the appropriate application directory (for WSNDemo, the `WSNDemo.eww` file from the `<SDK-Root>\Applications\WSNDemo\iar_projects` subdirectory) with IAR Embedded Workbench, and execute the `Rebuild All` item from the `Project` menu. By default, the `.a90` file (for WSNDemo, `WSNDemo.a90`) will be generated in the `iar_projects/Debug/exe` subdirectory (for WSNDemo, in the `./Applications/WSNDemo/iar_projects/Debug/exe` directory) with format as specified in Linker Output Options of the IAR project.

5.3 Reserved hardware resources

Hardware resources provided by the supported hardware include microcontroller peripherals, buses, timers, IRQ lines, I/O registers, etc. Many of these interfaces have corresponding APIs in the hardware abstraction layer (HAL) of the Atmel BitCloud stack. When building custom applications on top of the BitCloud API, the user is strongly encouraged to use the high-level APIs instead of the low-level register interfaces to ensure that its resource use does not overlap with that of the stack.

The hardware resources in BitCloud reserved for internal use by the stack are listed in the platform-specific sections specified in Table 5-3. These resources must not be accessed by the application code. Please note that the lists of the reserved hardware resources differ for each device.

Table 5-3. Platform-specific reserved resources.

For platform	Refer to Section
ZigBit / ZigBit Amp / ZigBit 900	8.5
ATmega128RFA1	9.5
UC3	10.5
XMEGA	11.5
SAM7X	12.5
SAM3S	13.5

6 Serial Bootloader and OTAU

For some platforms devices may be programmed using Atmel Serial Bootloader, which comes as a separate software package. The package can be downloaded from the Atmel website. It includes pre-built firmware images of the embedded bootloader, which shall be loaded to the device's flash via JTAG, the source code of the embedded bootloader application, and the installation file for the Bootloader PC tool. The package is also necessary for Over-the-air upgrade, because all participating devices shall be programmed with a special version of the embedded bootloader, with extended functionality, and the whole process is initiated from the Bootloader PC tool.

Using Serial Bootloader as well as OTAU requires different fuse settings (see Setting fuse bits sections in the corresponding appendices). Once a device is programmed with the embedded bootloader image (for some platforms the embedded bootloader is already present in the device's flash memory), the application firmware is loaded to the device, using the Bootloader PC tool. The device shall be connected to the PC via a serial link. The Bootloader PC tool may be used as both a command line tool and a GUI application. Both versions have the same options and take an application image in the Motorola SREC format. Such images are generated along with other types of binaries for all reference applications.

Some hardware-specific details are given in the corresponding sections in the appendices. For detailed description of the Serial Bootloader package, the list of supported platforms, instructions on generating SREC images etc., please refer to *AVR2054: Serial Bootloader User Guide* [11]. OTAU is fully described in AVR2058*: OTAU User Guide* [35].

7 Basic troubleshooting

In case of any operational problem with your setup, please check the following:

1. Check the power first, and make sure that all of your equipment is properly connected.
2. Verify that the PC conforms to the minimum system requirements (see the System Requirements subsection of the appropriate Appendix's part; references are given in Table 3-1).
3. Verify that the PC USB or UART interface is working and that the correct drivers are installed.
4. Check the hardware kit documentation, and that you have set up the hardware according to specific kit instructions (see the Hardware Setup subsection of the appropriate Appendix's part; references are given in Table 3-1).
5. Make sure you have programmed the right images and set the correct fuse values (see the Programming subsection of the appropriate Appendix's part; references are given in Table 4-1). Resetting the node may be required.

Table 7-1 presents some typical problems you may encounter while working with the development kit and their possible solutions.

Table 7-1. Typical problems and solutions.

Problem	Solution
WSNMonitor fails to start	Make sure the Java® machine is properly installed on the PC
No node is shown on the topology pane in WSNMonitor	Check that WSNMonitor is using the proper COM port, and if not, change it and restart the program

Problem	Solution
WSNMonitor shows NO DATA in the sensor data graph pane	No node is selected. Select the required node by clicking on it
Node titles displayed on the topology pane do not show node destinations	The displayed titles do not necessarily relate to the node functions, but they can be redefined by the user at any time. These names are stored in the node title file (see Section 4.5) along with MAC addresses mapped to the nodes

8 Appendix A-1: ZigBit specifics

8.1 Getting started

The Atmel BitCloud SDK supports Atmel ZigBit modules, which include Atmega1281 MCU, and Atmel ATZB-EVB-24-A2 MeshBean and ATZB-EVB-A24-SMA MeshBean Amp evaluation boards as evaluation and development platforms. The SDK also supports Atmega1281 with RCB212/230/231 as evaluation platform.

8.1.1 Required hardware

Before installing and using the Atmel BitCloud SDK, make sure that all necessary hardware is available:

- Two or more Atmel ZigBit modules, operating in the same frequency band, mounted on PCBs with a UART RS-232 controller for serial communication
- Optionally:
    - Atmel AVR JTAGICE mkII or Atmel JTAGICE 3 when using the JTAG interface for programming and debugging
    - ISP programmer if the ISP interface is used for programming and debugging

8.1.2 Hardware setup

No special pre-usage assembly is required for MeshBean boards. Please note that the boards can be powered in one of the three ways:

- By a pair of AA-size batteries;
- Via the USB port (once connected for data transfer, see also Section 8.1.4);
- Via an AC/DC adaptor.

The nominal voltage is 3V for MeshBean and 3.3V for MeshBean Amp. Using an AC/DC adaptor automatically disconnects the AA batteries. Using the USB port disconnects the AC/DC adaptor.

8.1.2.1 Configuring jumpers on MeshBean boards

This section defines settings for some of the jumpers used on the MeshBean board. For more information on jumper settings and interface pin-outs, refer to [9].

Table 8-1. Jumper settings for P4 port: ZigBit power source.

Jumper position	Description
Pins 1 and 2 (USB) connected	The board is powered by USB
Pins 1 and 3 (BAT/EXT) connected	The board is powered by external source or battery

Table 8-2. Jumper settings for P4 port: disable VDD.

Jumper position	Description
Pins 5 and 6 (DISABLE VDD) connected	VDD is disabled
Pins 5 and 6 (DISABLE VDD) are not connected	VDD is not disabled

Table 8-3. Jumper settings for P4 port: current measurement.

Jumper position	Description
Pins 9 and 10 (FOR CM) connected	This position is used for normal operation.
Pins 9 and 10 (FOR CM) are not connected	In this position, the ZigBit module is not powered while remaining parts of the board are powered. This position is used to measure current consumption of the ZigBit module

WARNING *Any other position of jumpers for P4 port their omission may cause permanent damage of the MeshBean board.*

8.1.2.2 OTAU hardware setup

If you wish to demonstrate over-the-air upgrade functionality, an external flash memory device supported by the Atmel BitCloud SDK and OTAU embedded bootloader (for example, AT25F2048 or AT45DB041) must be attached to the Atmel ZigBit board, as shown in Table 8-4.

For details on OTAU demonstration, see Section 8.4.3.

Table 8-4. External flash and ZigBit/MCU pin assignment.

DataFlash® pin	ATZB-XX pin	Internal ATmega1281 MCU pin
MISO	USART0_RXD (38)	PE0
MOSI	USART0_TXD (39)	PE1
CLOCK	USART0_EXTCLK (40)	PE2
CS	ADC_INPUT3 (30)	PE3

8.1.3 System requirements

Before using the SDK, please ensure that the following system requirements are met by your PC and development environment.

Table 8-5. System requirements for ZigBit, ZigBit Amp, and ZigBit 900.

Parameter	Value	Note
CPU	Intel Pentium III, or higher, 800MHz	
RAM	128MB	
Free space on hard disk	50MB	
JTAG emulator	Atmel AVR JTAGICE mkII or Atmel JTAGICE 3 emulator with cable	Required to upload firmware onto the boards and debug through JTAG (see Section 8.2.4)

Parameter	Value	Note
Operating system	Windows 2000/XP	
IDE	Atmel AVR Studio 5.1.208 OR IAR Embedded Workbench AVR 6.11 (with IAR C/C++ Compiler for AVR v6.11.1.50453 (1))	Required to upload firmware images through JTAG (see Section 8.2.4) and develop applications using the API (see Section 5.2)
Java virtual machine	Java Runtime Environment (JRE) 6, or later	Required to run the WSNMonitor application

Note: 1. Users are strongly recommended to use the specified version of AVR Studio and IAR C/C++ Compiler for AVR. Other versions are not supported, and may not work.

8.1.4 Installing the SDK

Proceed with the following installation instructions.

1. Download the archive to your PC, and unpack it into an empty folder. The following SDK folders and files will be created with <SDK-Root> as the top-level SDK folder.

Table 8-6. The SDK file structure.

Directory/file	Description
`<SDK-Root>\Documentation`	Documentation for Atmel BitCloud software
`<SDK-Root>\Evaluation Tools\WSNDemo (Embedded)`	Ready-to-use image files for evaluating WSNDemo. Refer to Section 8.3 for the description of the images
`<SDK-Root>\Evaluation Tools\WSNDemo (WSN Monitor)\WSNMonitorSetup.exe`	WSNMonitor installer
`<SDK-Root>\Evaluation Tools\SerialNet`	Ready-to-use image files for the SerialNet application. Refer to [10] for more information on SerialNet
`<SDK-Root>\BitCloud\Components`	Header files for the BitCloud stack
`<SDK-Root>\BitCloud\Components\BSP`	Source, header, and library files for the BitCloud BSP
`<SDK-Root>\BitCloud\lib`	Library files for the BitCloud stack
`<SDK-Root>\Applications`	Source files for the sample applications

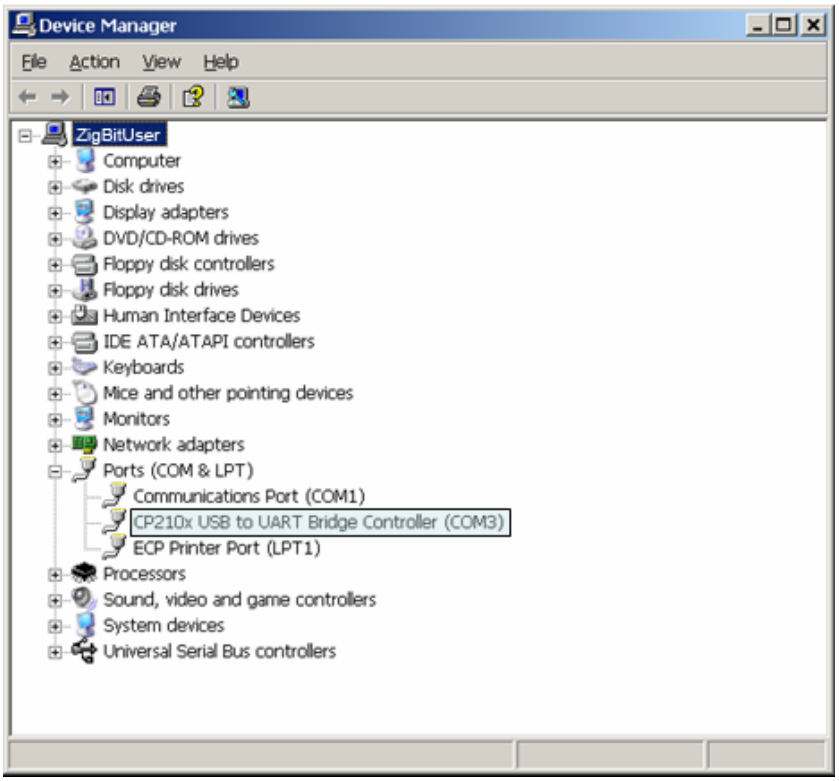
2. Install the desired IDE:
    a. For IAR Embedded Workbench AVR:
        i. Install IAR Embedded Workbench for Atmel AVR [13], if not already installed on your PC.
        ii. Add a Windows environment variable named IAR_AVR_HOME, and set its value to the IAR Embedded Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 6.11`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click New below the System variables list, and enter Variable Name

and Variable Value. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.

b. For Atmel AVR Studio:
   i. Install AVR Studio [3], if not already installed on your PC.
   ii. Add paths to the folder containing the AVRGCC compiler to the Path Windows environment variable. The compiler is located in the `\extensions\Atmel\AVRGCC\3.3.1.27\AVRToolchain\bin` directory of the AVR Studio installation directory.

3. The board can be connected to the host PC via the USB port using a USB 2.0 A/Mini-B cable. USB is a familiar connection option. Furthermore, it provides a convenient way to link multiple boards to a single PC, and no battery is required once a board is powered via USB.
4. Alternatively, the board can be connected to the host PC via the serial port using a serial cable. Please note that the logical USB and serial (RS-232) ports share the same physical port on the board, and, thus, cannot be used at the same time. Keep in mind that the connection mode is controlled by setting a jumper on a MeshBean board. Refer to Section 8.1.2 for the description of connectors and jumpers on MeshBean boards.
5. If you plan to use a USB connection, install the USB-to-UART Bridge VCP driver. To install the driver, please do the following:
   a. Download the driver from https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx.
   b. Attach the MeshBean board to the USB port of your PC. Windows should detect the new hardware. Follow the instructions provided by the driver installation wizard.
   c. Make sure that the driver is installed successfully and the new COM port is present in the device list. Check that the device is correctly shown in the Device Manager window, as in Figure 8-1.

Figure 8-1. Correctly installed COM port for MeshBean device.



6. Download and install Java Runtime Environment [12], if not already installed on your PC.
7. Demonstration of OTAU functionality and programming devices with Serial Bootloader require the Serial Bootloader package, which should be downloaded from the Atmel website.

8.2 Programming the boards

A firmware image file can be uploaded into Atmel ZigBit devices in one of the following ways: using the Serial Bootloader package, using the JTAG interface, or using the ISP programmer interface.

8.2.1 Setting fuse bits

Table 8-7 provides default fuse bit configurations loaded into all ZigBit modules during production. It also describes some use cases when certain fuse bits require values different from the default ones. Based on their own application-specific requirements, users can change the fuse bits as well. See [31] for detailed fuse bit descriptions.

NOTE

Modifying fuse bit settings is possible only using the JTAG or ISP programming interfaces, and cannot be done with Serial Bootloader.

Table 8-7. Fuse bit settings for ZigBit, ZigBit Amp, and ZigBit 900.

Option	Default value in ATZB-XX devices	Comments
BODLEVEL	`Brownout detection disabled`	Can be changed according to application-specific requirements

Option	Default value in ATZB-XX devices	Comments
OCDEN	Disabled	Can be changed during application development. Must be disabled for final products
JTAGEN	Enabled	Must always be enabled
SPIEN	Enabled	Must always be enabled
WDTON	Disabled	Can be changed according to application requirements
EESAVE	Disabled	Can be changed according to application requirements
BOOTSZ	`Boot Flash size=1024 words start address=$FC00`	Specifies section size, in words (2 bytes), reserved in flash memory for the embedded bootloader Applied only if `BOOTRST` is enabled Shall be changed to `Boot Flash size=2048 words start address=$F800` if OTAU support is needed on the device
BOOTRST	Enabled	Shall be enabled if device needs to be programmed with Serial Bootloader, or if OTAU support is required. Can be disabled in other cases
CKDIV8	Enabled	
CKOUT	Disabled	
SUT_CKSEL	`Int. RC Osc.; Startup time: 6 CK + 65ms`	Can be changed according to application-specific requirements, but external clock source shall not be used by sleeping devices

Default fuse bit configuration allows use of Serial Bootloader, and results in the following fuse bytes (extended fuse byte, high fuse byte, low fuse byte):

```
0xFF, 0x9C, 0x62
```

To work with OTAU functionality, special embedded bootloader firmware with OTAU support shall be loaded to the device via the JTAG interface. Due to its large size, `BOOTSZ` fuse shall be changed to `Boot Flash size=2048 words start address=$F800`.

If embedded bootloader support is not needed and devices are programmed only with JTAG or ISP, then `BOOTRST` can be disabled, leading to the following fuse bytes:

```
0xFF, 0x9D, 0x62
```

8.2.2 Extended (MAC) address assignment

For proper operation, all nodes in a ZigBee network shall have unique, 64-bit MAC address values. At startup, the Atmel BitCloud software assigns the MAC address to an Atmel ZigBit device as follows. If at compile time the `CS_UID` parameter is set to 0, the BitCloud stack attempts to load a MAC address from a dedicated UID chip available on the MeshBean board via the 1-Wire interface. If there is no such UID, then a zero MAC address will be assigned to the device. Note that for proper operation, all nodes in the network shall have unique MAC address values. Hence, if

an address cannot be obtained automatically from an external source, separate firmware images shall be created for each device with a unique `CS_UID` parameter specified in the application configuration every time an image is compiled.

8.2.3 Programming with Serial Bootloader

All ZigBit modules are shipped preprogrammed with the embedded bootloader image needed to use Serial Bootloader. Firmware images for the embedded bootloader as well as the Bootloader PC tool, which is needed to load the application image from a PC to the device, are included in the Atmel Serial Bootloader software package available for downloading from the Atmel website.

To program a board using Serial Bootloader, proceed as follows:

1. Connect the serial port on the PC to the ATZB module pins, as specified in Table 8-8. The MeshBean evaluation board shall be connected using USB or the extension slot, depending on the position of jumper J3 (see Section 8.1.2.1).

Table 8-8. Host UART and ZigBit/MCU pin connection.

UART pin on host device	ATZB-XX pin	ATmega1281 MCU pin
RXD	UART1_RXD (14)	PD3
TXD	UART1_TXD (13)	PD2
GND	DGND (any)	D_GND

2. Install and run the Bootloader PC tool from the command line or use the GUI. Specify the target image file in `.srec` format and the COM port, and launch the firmware upload.
3. If requested, reset the board, using the reset button.
4. The Bootloader PC tool indicates the programming progress. Once an upload is successfully completed, the board should restart automatically. If an upload fails, the embedded bootloader should indicate the reason. In rare cases, the booting process can fail due to communication errors between the board and the PC. If this happens, attempt booting again, or try using the conventional serial port instead of USB. If booting fails, the program recently written to the board will be corrupted, but the board can be reprogrammed, as the embedded bootloader remains intact.

See [11] for additional details about the Serial Bootloader package.

WARNING *Using JTAG to program the microcontroller will erase the embedded bootloader, making the loading of application images with Serial Bootloader impossible until the embedded bootloader firmware is reprogrammed to the device.*

8.2.4 Programming with JTAG

Refer to Atmel AVR Studio [3] or IAR Embedded Workbench [13] documentation for a description of how the images can be programmed to the devices using JTAG.

8.2.5 Programming with ISP

Refer to AVR Studio [3] documentation for a description of how firmware images can be programmed to a device using the in-system programming (ISP) interface. For Atmel ZigBit devices, the ISP frequency shall be set to 6.478kHz, and the ISP programmer shall be connected to the module as specified in Table 8-9.

Table 8-9. ISP programmer and ZigBit/MCU pin connections.

ISP programmer pin	ATZB-XX pin	ATmega1281 MCU pin
MOSI	USART0_RXD (38)	PE0
MISO	USART0_TXD (39)	PE1
SCK	SPI_SCLK (1)	PB1
RESET	RESET (8)	RSTn_In
VCC	D_VCC (any)	D_VCC
GND	DGND (any)	D_GND

8.3 Pre-built images

The SDK comes with a set of ready-to-use binary images of the WSNDemo application. It includes a set of images for different roles, which are preconfigured with distinct MAC addresses so they can be used for creating a small ZigBee network right away. The image name is formed according to the following scheme:

```
<App_name>_<rf_chip>_<region>_<MCU>.<extension>
```

`App_name` stands for the name of the application. For ZigBit devices, the WSNDemo, Runner, and SerialNet applications are available. To specify the node role, put `Coord`, `Router`, or `EndDev` for coordinator, router, and end device, respectively. To program with JTAG, use files with a `.hex` extension. `.srec` files are loaded with the serial bootloader. `Region` is an optional parameter that specifies the radio frequency used by the image according to the frequency band of the specified region. Possible options are `US`, `EU`, and `China`. For example, to use JTAG to program a device based on an RF212 radio with an image of the WSNDemo application for use in the US, use the file with the name

```
WSNDemo_Rf212_US.hex
```

The ready-to-use binary images retrieve MAC address automatically from the UID chip, ensuring that unique MAC addresses are assigned to all network nodes. Also note that the default WSNDemo application images are configured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and channel mask with:

- Channel 0x0F enabled for ZigBit and Atmel ZigBit Amp (WSNDemoApp.hex),
- Channel 0x00 and channel page 0 (WSNDemoApp_EU.hex) or channel 0x01 and channel page 0 (WSNDemoApp_RF212US.hex) for Atmel ZigBit 900,
- Channel 0x01 and channel page 5 (WSNDemoApp_China.hex) for ZigBit 900.

8.4 Running WSNDemo

8.4.1 Starting WSNDemo

To start WSNDemo, proceed as follows:

1. Set up the hardware, as described in Section 8.1.2.
2. Install the Atmel BitCloud SDK, as described in Section 8.1.4.
3. Program one device with the coordinator image file and other with either the router or end device images using one of the methods described in Section 8.2.
4. Connect the coordinator node to the PC using the USB port on the coordinator board.
5. Power on the coordinator node.
6. Run WSNMonitor (see Section 4.5).
7. Power on and reset the rest of the nodes.

8.4.2 Monitoring WSNDemo activity

Network activity can be monitored in two ways:

- Observing color LEDs on MeshBean boards (see Table 8-10)
- Monitoring network topology and configuration through WSNMonitor installed on a PC

Table 8-10. LED indication for MeshBean boards used in WSNDemo.

Node state	LED1 (red)	LED2 (yellow)	LED3 (green)
Searching for network	Blinking	OFF	OFF
Joined to network	ON		
+ receiving data		Blinking	
+ sending data to UART (coordinator only)			Blinking
Sleeping (end device only)	OFF	OFF	OFF

NOTE

When WSNDemo runs on a board other than a MeshBean, special care must be taken to reconfigure the application to output the networking state to available user interfaces (for example, LEDs).

8.4.3 Over-the-air upgrade configuration

Over-the-air upgrade functionality of the Atmel BitCloud SDK can be demonstrated using the Atmel ZigBit platform. This demonstration requires additional hardware, namely:

- A dedicated device running the Runner application and performing the function of an over-the-air upgrade client
- One of the supported external Atmel DataFlash devices is connected to the ZigBit module, as described in Section 8.1.2, and used to store uploaded application images

NOTE

Only those modules that have a DataFlash device connected to them should be programmed with the OTAU-capable WSNDemo application. The rest can run the stock WSNDemo firmware.

Once the external DataFlash device is connected to the board, the user should configure and install devices as follows:

1. Load the WSNDemo application configured and compiled with APP_USE_OTAU defined as 1 in the `configuration.h` file:
    a. Program the embedded bootloader `bootloaderOTAU_ATmega1281.hex` image file from the Serial Bootloader package, according to platform-specific instructions.
    b. The application image should be converted to `*.srec` format and installed using the Bootloader PC tool from the same package, as described in Section 4.5.3.The device is now able to perform as an OTA client, as defined in [30]. The above process should be repeated for every node that the user intends to upgrade over the air.
2. Program another device with the Runner application available in the `<SDK-Root>\Evaluation Tools\Runner\` directory of the SDK. The device is now able to perform as an OTA server, as defined in [30]. Once the images are programmed and WSNDemo devices are joined to the network, follow instructions given in Section 4.5.3 to update the firmware over the air.

8.5 Reserved hardware resources

Table 8-11. Hardware resources reserved by the stack on ZigBit, ZigBit Amp, and ZigBit 900 modules.

Resource	Description
Processor main clock	8MHz from internal RC oscillator or external radio frequency
SPI	Radio interface
ATmega ports PB0, PB1, PB2, PB3, PB4, PA7, PE5	Radio interface
ATmega port PC1	Interface for amplifier (if present)
ATmega ports PG3, PG4	Asynchronous timer interface
Timer/Counter 2	Asynchronous timer
Timer/Counter 4	System timer
External IRQ4	Wake up on DTR
External IRQ5	Radio interface
EEPROM	Storage for user settings accessible via persistent data server
PE0, PE1, PE2, PF3	External Atmel DataFlash, when OTAU functionality is used

9 Appendix A-2: ATmega128RFA1 specifics

9.1 Getting started

The Atmel BitCloud SDK supports two different development platforms with Atmel ATmega128RFA1: ATRF4CE-EK [14] and Atmel ATAVR128RFA1-EK1 [15] with Atmel ATSTK600 boards [16]. The instructions below highlight the differences between the two platform configurations, where present.

9.1.1 Required hardware

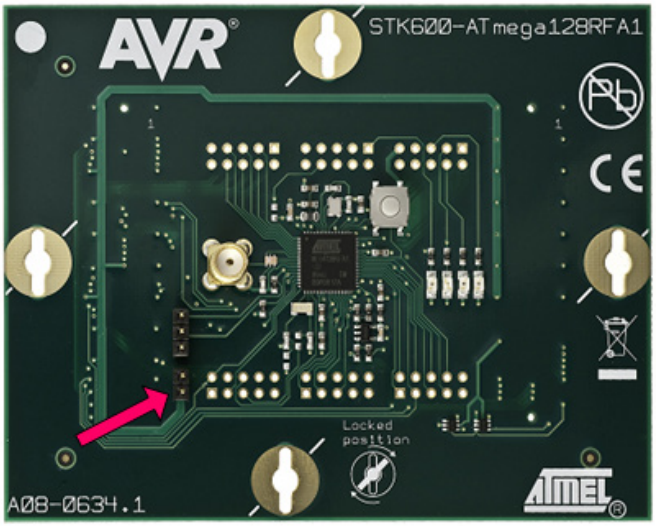
Before installing and using the BitCloud SDK for ATmega128RFA1, make sure that all necessary hardware is available:

- For ATRF4CE-EK:
    - Two or more RCB128RFA1 boards with 2.4GHz antennas and AAA batteries
    - One or more RCB breakout boards (other boards from the kit may be used, too)
    - One RS-232 interface cable for the RCB breakout board
    - Atmel AVR JTAGICE mkII or Atmel JTAGICE 3
- For ATAVR128RFA1-EK1:
    - Two or more ATSTK600 boards, each with an Atmega128RFA1 top card and 2.4GHz antenna
    - AVR JTAGICE mkII or Atmel JTAGICE 3

9.1.2 Hardware setup

For the RCB128RFA1, please refer to [14] for hardware setup instructions.

Figure 9-1. Jumper setting for STK600-ATmega128RFA1 top card.



ATAVR128RFA1-EK1 requires the following setup:

1. Make sure a jumper is present at the pins indicated by the red arrow in Figure 9-1.
2. Assemble the Atmel ATAVR128RFA1-EK1 on top of an Atmel STK600 board.
3. To communicate over the RS-232 port, connect pins PD2 and PD3 to pins RXD and TXD, respectively, of the RS-232 SPARE port on the board. This is required by the sample applications provided with the Atmel BitCloud SDK and Serial Bootloader. Custom applications are free to choose other pins.

Before continuing any further operations, perform the steps required to get started with the Atmel ATSTK600 [16]. Refer to AVR Studio Help [3] for details on that subject. At the very least, make sure that the Atmel STK600 firmware is up to date, and configure the voltage provided by the STK600 for the Atmel ATmega128RFA1 top card. For that, perform the following steps:

1. Attach the STK600 to a PC using a USB cable.
2. In Atmel AVR Studio, open the `Tools > Program AVR > Connect…` dialogue.
3. Choose the correct Platform = STK600, and press Connect.
4. Update the STK600 firmware, if suggested.
5. Go to the HW Settings tab.
6. Specify 3.3V in the VTarget field, and press Write.

The actions above must be performed only once for each ATSTK600 board.

9.1.2.1 OTAU hardware setup

If you wish to demonstrate OTA upgrade functionality, a serial Atmel DataFlash device (AT45DB041) should be connected to the ATmega128RFA1 pins, as specified in Table 9-1.

The STK600 board already has built-in DataFlash, and so to use OTAU on an STK600 it is sufficient to connect the `SI`, `SO`, `SCK`, and `/CS` pins under the DataFlash pin heading to the corresponding MCU pins noted in Table 9-1.

Table 9-1. External flash and MCU pin assignment.

DataFlash (STK600) pin	ATmega128RFA1 MCU pin
MOSI (SI)	PE1
MISO (SO)	PE0
CLOCK (SCK)	PE2
CS (CS)	PG5

9.1.3 System requirements

Before using the SDK, please ensure that the following system requirements are met by your PC and development environment.

Table 9-2. System requirements for ATmega128RFA1.

Parameter	Value	Note
CPU	Intel Pentium III, or higher, 1GHz	
RAM	512MB	
Free space on hard disk	200MB	
JTAG emulator	Atmel AVR JTAGICE mkII or Atmel JTAGICE 3 emulator with cable	Required to upload firmware onto the boards and debug through JTAG
Operating system	Windows 2000/XP	

Parameter	Value	Note
IDE	Atmel AVR Studio 5.1.208 OR IAR Embedded Workbench AVR 6.11 (with IAR C/C++ Compiler for AVR v6.11.1.50453 [1])	Required to upload firmware images through JTAG (see Section 9.2.4) and develop applications using the API (see Section 5.2)
Java virtual machine	Java Runtime Environment (JRE) 6, or later	Required to run WSNMonitor application

Note: 1. Users are strongly recommended to use the specified version of AVR Studio and IAR Compiler. Other versions are not supported, and may not work.

9.1.4 Installing the SDK

Proceed with the following installation instructions:

1. Download the archive to your PC and unpack it into an empty folder with no blank spaces present in the directory path. The following SDK folders and files will be created.

Table 9-3. The SDK file structure.

Directory/file	Description
`<SDK-Root>\Documentation`	Documentation for the Atmel BitCloud software
`<SDK-Root>\Evaluation Tools\WSNDemo (Embedded)`	Ready-to-use image files for evaluating WSNDemo. Refer to Section 9.3 for a description of the images.
`<SDK-Root>\Evaluation Tools\WSNDemo (WSN Monitor)\WSNMonitorSetup.exe`	WSNMonitor installer
`<SDK-Root>\BitCloud\Components\`	Header files for the BitCloud stack
`<SDK-Root>\BitCloud\Components\BSP\`	Source, header, and library files for the BitCloud BSP
`<SDK-Root>\BitCloud\lib`	Library files for the BitCloud stack
`<SDK-Root>\Applications`	Source files for the sample applications

2. Install desired IDE:
    a. For IAR Embedded Workbench AVR:
        i. Install IAR Embedded Workbench for Atmel AVR [13], if not already installed on your PC.
        ii. Add a Windows environment variable named IAR_AVR_HOME, and set its value to the IAR Embedded Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 6.11`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click New below the System variables list, and enter Variable Name and Variable Value. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.
    b. For Atmel AVR Studio:

i. Install AVR Studio [3], if not already installed on your PC.
ii. Add paths to the folder containing the AVRGCC compiler to the Path Windows environment variable. The compiler is located in the `\extensions\Atmel\AVRGCC\3.3.1.27\AVRToolchain\bin` directory of the AVR Studio installation directory.

3. Download and install Java Runtime Environment [12], if not already installed on your PC.
4. Demonstration of OTAU functionality and programming devices with Serial Bootloader require the Serial Bootloader package, which should be downloaded from the Atmel website.

9.2 Programming the boards

A firmware image file can be uploaded into the Atmel ATmega128RFA1 device in one of the following ways: using the Serial Bootloader package or using the JTAG interface.

9.2.1 Setting fuse bits

Table 9-4 provides the default fuse bit configuration loaded into ATmega128RFA1 devices during production. It also describes some use cases when certain fuse bits require values different from the default ones. Based on their own application-specific requirements, users can change the fuse bits as well. See [32] for detailed fuse bit descriptions.

NOTE

Modifying fuse bit settings is possible only using the JTAG programming interface, and cannot be done using the Serial Bootloader package tools.

Table 9-4. Fuse bit settings for ATmega128RFA1 devices.

Option	Deafult value in ATmega128RFA1 devices	Comments
BODLEVEL	`Brownout detection disabled`	Can be changed according to application-specific requirements
OCDEN	Disabled	Can be changed during application development. Must be disabled for final products
JTAGEN	Enabled	Must be always enabled
SPIEN	Enabled	Must be always enabled
WDTON	Disabled	Can be changed according to application requirements
EESAVE	Disabled	Can be changed according to application requirements

Option	Deafult value in ATmega128RFA1 devices	Comments
BOOTSZ	`Boot Flash size=4096 words start address=$F000`	Specifies section size, in words (2 bytes), reserved in flash memory for the embedded bootloader. Applied only if `BOOTRST` is enabled. Shall be changed to `Boot Flash size=1024 words start address=$FC00` if it is to work with the serial bootloader without OTAU support. Shall be changed to `Boot Flash size=2048 words start address=$F800` if OTAU support is required on the device.
BOOTRST	Disabled	Shall be enabled if the device needs to be programmed with the serial bootloader or if OTAU support is required. Can be disabled in other cases
CKDIV8	Enabled	
CKOUT	Disabled	
SUT_CKSEL	`Int. RC osc.; Start-up time: 6 CK + 65ms`	Can be changed according to application-specific requirements, but external clock source shall not be used by sleeping devices

By default, programming with Serial Bootloader is not allowed, and so firmware can be uploaded only over the JTAG interface. Default fuse bytes equal (in this order: extended fuse byte, high fuse byte, low fuse byte):

```
0xFF, 0x99, 0x62
```

To turn on bootloader support, first, the `BOOTRST` fuse must be enabled and `BOOTSZ` must be set to `Boot Flash size=1024 words start address=$FC00`, which leads to the following fuse bytes:

```
0xFF, 0x9C, 0x62
```

Second, a firmware image of the embedded bootloader (without OTAU support) shall be programmed to the Atmel ATmega128RFA1 using JTAG. After that, application images can be loaded to the device using the Bootloader PC tool from the Serial Bootloader package, as described in Section 9.2.3.

Enabling OTAU bootloader support requires the `BOOTSZ` fuse to be set to `Boot Flash size=2048 words start address=$F800`, and, hence, the following fuse bytes values:

```
0xFF, 0x9A, 0x62
```

9.2.2 Extended (MAC) address assignment

For proper operation, all nodes in a ZigBee network shall have unique, 64-bit MAC address values. At startup, the Atmel BitCloud software assigns the MAC address to the ATmega128RFA1 device as follows:

- If at compile time the `CS_UID` parameter is set to 0, BitCloud attempts to load the MAC address from the dedicated external EEPROM chip available on the RCB128RFA1 as well as on the Atmel ATAVR128RFA1-EK1 board via the SPI interface.
- If there is no such chip present, then a zero MAC address will be assigned to the device.

Note that for proper operation, all nodes in the network shall have unique MAC address values. Hence, if an address cannot be obtained automatically from an external source, separate firmware images shall be created for each device with a unique `CS_UID` parameter specified in the application configuration.

9.2.3 Programming with Serial Bootloader

Programming using Serial Bootloader requires that the embedded bootloader code is loaded to the device via JTAG. Firmware images for the embedded bootloader as well as the Bootloader PC tool, which is needed to load the application image from a PC to the device, are included in the Atmel Serial Bootloader software package available for downloading from the Atmel website.

Images that shall be loaded to ATmega128RFA1 via JTAG may be found under the `\Embedded_Bootloader_images\Atmega128rfa1` directory in the package:

- The `Bootloader_ATmega128RFA1_RCB_BB_RS232.hex` file for RCB128RFA1
- The `Bootloader_ATmega128RFA1.hex` file for ATAVR128RFA1-EK1

In both cases, the fuse bits should be configured properly; namely, the `BOOTRST` fuse should be enabled.

If the embedded bootloader is loaded, the following steps should be executed to upload the application image file to the board:

1. Assemble the board and connect it to a PC:
    a. For the ATRF4CE-EK:
        a. Assemble the RCB128RFA1 and RCB breakout boards (RCB-BB) together.
        Connect the RS-232 interface cable to the J1 extender on the RCB-BB and to the COM1 port on the PC.
    b. For the Atmel ATAVR128RFA1-EK1:
        b. Assemble the Atmel STK600 board and the ATAVR128RFA1-EK1 card, as described in Section 9.1.2.
        c. Connect PC COM1 port to the RS-232 SPARE port on the board. The RS-232 SPARE pins should be connected to the MCU pins, as specified in Table 9-5.

Table 9-5. Host UART and MCU pin connections.

UART pin on host device	ATmega128RFA1 MCU pin
RXD	PD3
TXD	PD2
GND	D_GND

2. Install and run the Bootloader PC tool from the command line or use the GUI. Specify the target image file in `.srec` format and the COM port, and launch the firmware upload.
3. Reset the board using the reset button, if requested.
4. The Bootloader PC tool indicates the programming progress. Once an upload is successfully completed, the board should restart automatically. If an upload fails,

the Bootloader PC tool should indicate the reason. In rare cases, the booting process can fail due to communication errors between the board and the PC. If this happens, attempt booting again or try using the conventional serial port instead of USB. If booting fails, the program recently written to the board will be corrupted, but the board can be reprogrammed, as the embedded bootloader remains intact.

See [11] for additional details about the Serial Bootloader package.

WARNING *Using JTAG to program the microcontroller will erase the embedded bootloader, making the loading of application images with Serial Bootloader impossible until the embedded bootloader firmware is reprogrammed to the device.*

9.2.4 Programming with JTAG

Refer to [14] and [16] for a description of how the images can be programmed to corresponding development boards using JTAG.

9.3 Pre-built images

The SDK comes with a set of ready-to-use binary images of the WSNDemo application. It includes a set of images for different roles, which are preconfigured with distinct MAC addresses so they can be used for creating a small ZigBee network right away. The image name is formed according to the following scheme:

```
WSNDemo_<board>_<MCU>_<role>.<extension>
```

To specify the node role, put `Coord`, `Router`, or `EndDev` for coordinator, router and end device, respectively. To program with JTAG, use files with a `.hex` extension. `.srec` files are loaded with the serial bootloader. For example, to use JTAG to program a device with the Atmel ATmega128RFA1 MCU on the Atmel STK600 board acting as coordinator, use the file with the name

```
WSNDemo_Stk600_Atmega128rfa1_Coord.hex
```

9.4 Running WSNDemo

9.4.1 Starting WSNDemo

To start WSNDemo, proceed as follows:

1. Set up the hardware, as described in Section 9.1.2.
2. Install the Atmel BitCloud SDK, as described in Section 9.1.4.
3. Program one device with the coordinator image file and other with either the router or end device images, as described in Section 9.2.
4. Connect the coordinator node to the PC using the serial interface.
5. Power on the coordinator node.
6. Run WSNMonitor (see Section 4.5).
7. Power on and reset the rest of the nodes.

9.4.2 Monitoring WSNDemo activity

Network activity can be monitored in two ways:

- Observing LEDs on the development boards, as described in Table 9-6. LED Dx labels correspond to the RCB128RFA1 board, while labels naming colors belong to the Atmel ATAVR128RFA1-EK1 board.

- Monitoring the network topology through WSNMonitor installed on a PC (see Section 4.5).

Table 9-6. LED indication for RCB128A1 (and ATAVR128RFA1-EK1) boards used in WSNDemo.

Node state	LED D2 (red)	LED D3 (yellow)	LED D4 (green)
Searching for network	Blinking	OFF	OFF
Joined to network	ON		
+ receiving data		Blinking	
+ sending data to UART (coordinator only)			Blinking
Sleeping (end device only)	OFF	OFF	OFF

9.4.3 Demonstrating OTA upgrade functionality

OTA upgrade functionality of the BitCloud SDK can be demonstrated using the megaRF platform by using the serial Atmel DataFlash device found on the STK600 board. Once the DataFlash device is connected to the Atmel ATmega128RFA1, as defined in Section 9.1.2, the user should configure and install devices as follows:

1. Load the WSNDemo application configured and compiled with `APP_USE_OTAU` defined as 1 in the `configuration.h` file:
    a. Program the embedded bootloader `bootloaderOTAU_ATmega128RFA1.hex` file from the Serial Bootloader package.
    b. The application image should be converted to `*.srec` format and installed using the Bootloader PC tool from the same package, as described in Section 4.5.3.
    c. The device is able to act as an OTA client, as defined in [30].
2. Program another device with the Runner application, which is available in the `<SDK-Root>\Evaluation Tools\Runner\` directory of the SDK. The Runner node acts as OTA server, as defined in [30].

Once the images are programmed and WSNDemo devices are joined to the network, follow the instructions given in Section 4.5.3 to update firmware over the air.

9.5 Reserved hardware resources

Table 9-7. Hardware resources reserved by the stack on ATmega128RFA1.

Resource	Description
Processor main clock	8MHz from internal RC oscillator
TRX24	Radio
ATmega ports PG3, PG4	Asynchronous timer interface
Timer/Counter 2	Asynchronous timer
Timer/Counter 4	System timer
External IRQ4	Wake up on DTR
EEPROM	Storage for user settings accessible via persistent data server
PE0..PE2, PG5	External DataFlash, when OTAU functionality is used

10 Appendix A-3: UC3 specifics

10.1 Getting started

10.1.1 Required hardware

Before installing and using the Atmel BitCloud SDK for the Atmel 32-bit AVR UC3, make sure that all necessary hardware is available:

1. Two or more Atmel EVK1105 boards [18]
2. Two or more radio extender boards [17]
3. Connectors for each EVK1105 board:
    a. Five two-wire cables
    b. One USB 2.0 A/Mini-B cable
4. Atmel AVR JTAGICE mkII or Atmel JTAGICE 3

10.1.2 Hardware setup

To prepare the hardware:

1. Install the J12 and J16 extension headers on the board (if not already installed).
2. Install the JTAG pin header on the board (if not already installed).
3. Use several two-wire cables to connect J16 pins to the corresponding pins on the REB231 boards, as indicated in Table 10-1.

Table 10-1. EVK1105 to radio extender board REB231 pin mapping.

EVK1105 J16 pin	REB231 pin
1	30
2	29
3	28
4	27
5	38
6	26
8	25
9	22
10	20

10.1.3 System requirements

Before using the SDK, please ensure that the following system requirements are met by your PC and development environment.

Table 10-2. System requirements for UC3.

Parameter	Value	Note
CPU	Intel Pentium III, or higher, 1GHz	
RAM	512MB	
Free space on hard disk	200MB	

Parameter	Value	Note
JTAG emulator	AVR JTAGICE mkII or Atmel JTAGICE 3 emulator with cable	Required to upload firmware onto the boards and debug through JTAG (see Section 10.2.3)
Operating system	Windows 2000/XP	
IDE	IAR Embedded Workbench 32-bit AVR (with IAR C/C++ Compiler for 32-bit AVR 3.30.1.40051/W32 (1)) and Atmel AVR32 GNU Toolchain v2.4.2	Required to upload firmware images through and develop applications using the API (see Section 5.2). AVR32 GNU Toolchain is needed only to install the USB VCP driver.
Java Virtual Machine	Java Runtime Environment (JRE) 6, or later	Required to run the WSNMonitor application

Note: 1. Users are strongly encouraged to use the specified versions of IAR C/C++ Compiler for 32-bit AVR. Other versions are not supported, and may not work.

10.1.4 Installing the SDK

Proceed with the following installation instructions:

1. Download the archive to your PC and unpack it into an empty folder with no blank spaces present in the directory path. The following SDK folders and files will be created.

Table 10-3. The SDK file structure.

Directory/file	Description
`<SDK-Root>\Documentation`	Documentation for the Atmel BitCloud software
`<SDK-Root>\Evaluation Tools\WSNDemo (Embedded)`	Ready-to-use image files for evaluating WSNDemo. Refer to Section 10.3 for the description of the images
`<SDK-Root>\Evaluation Tools\WSNDemo (WSN Monitor)\`	Contains the WSNMonitor installer
`<SDK-Root>\BitCloud\Components\`	Header files for the BitCloud stack
`<SDK-Root>\BitCloud\Components\BSP\`	Source, header, and library files for the BitCloud BSP
`<SDK-Root>\BitCloud\lib`	Library files for the BitCloud stack
`<SDK-Root>\Applications\`	Source files for the sample applications

2. Install IAR Embedded Workbench for Atmel AVR32 [21], if not already installed on your PC. Be sure to install only the supported version of IAR Embedded Workbench, as specified in Table 10-2.
    a. Add a Windows environment variable named IAR_AVR32_HOME, and set its value to the IAR Embedded Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 5.4_0`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click New below the System variables list, and enter Variable Name and Variable Value. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.

3. Install AVR32 GNU Toolchain [20], if not already installed on your PC.
4. Download and install Java Runtime Environment [12], if not already installed on your PC.
5. Install the USB VCP driver on the Atmel EVK1105 to allow it to communicate with your PC.
    a. Connect JTAG to the UC3B JTAG header, and power on the board.
    b. From `ThirdPartySoftware\EVK1105_UC3B_VCP`, run the `program_evk1105_at32uc3b-isp-cdc-1.0.1.cmd` Windows command script.
    c. The USB VCP driver should now be installed on the board.
6. Attach the EVK1105 board to the USB port of your PC using the USB 2.0 A/Mini-B cable. Windows should detect the new hardware. Follow the instructions provided by the driver installation wizard. When prompted, choose to install the driver from the specific location, and select the driver located in the `ThirdPartySoftware` folder of the SDK.

10.2 Programming the boards

A firmware image file can be uploaded into the Atmel UC3 device using JTAG. Programming the device using the serial bootloader is not supported.

10.2.1 Setting fuse bits

Table 10-4 provides the fuse bit configuration that should be loaded into Atmel AT32UC3A0512 devices before programming the application image with JTAG.

NOTE

Modifying fuse bit settings is possible only using the JTAG programming interface, and cannot be done with the Serial Bootloader package tools.

Table 10-4. Fuse bit settings for AT32UC3A0512.

Option	Default value on AT32UC3A	Values to program
BODLEVEL	000000b (0)	Brownout detection at VCC=1.92V (63)
BODHYST	Enabled	Enabled (1)
BODEN	Disabled	Disabled (3)
LOCK0 – LOCK15	Unlocked (1)	Unlocked (1)
EPFL	Enabled (1)	External instruction fetch enabled (1)
BOOTPROT	011b (4)	No bootloader (7)
GF29	Enabled (1)	Enabled (1)
GF30	Enabled (1)	Enabled (1)
GF31	Enabled (1)	Enabled (1)

10.2.2 Extended (MAC) address assignment

For proper operation, all nodes in a ZigBee network shall have unique MAC address values. For UC3 devices, a unique `CS_UID` parameter must be specified for each node in the application configuration, and then the application image must be built separately for each board.

10.2.3 Programming with JTAG

An image file from an existing project for IAR Embedded Workbench for Atmel AVR32 can be uploaded into the boards using a JTAG emulator as follows:

1. Assemble the board and connect it to the PC.
2. Connect the JTAG to the UC3A JTAG header. Power on the board and AVR JTAGICE mkII or Atmel JTAGICE 3.
3. Start IAR Embedded Workbench for Atmel AVR32.
4. From `File > Open > Workspace`, navigate to and open the desired IAR project (for example, the `WSNDemoApp.eww` file in the `Sample Applications\WSNDemo\iar\avr32` folder).
5. Select `Project > Download` and `Debug`.
6. Once the firmware is loaded, select `Debug > Stop Debugging`.
7. Unplug the JTAG from the UC3A JTAG header.
8. Reset the Atmel EVK1105.

Alternatively, it is possible to load a ready-to-use image file in .elf format using Atmel AVR32 GNU Toolchain [20] by running following command in the console:

```
avr32program program -finternal@0x80000000,256Kb -cxtal -e -v -O0x80000000 <filename.elf>
```

Make sure the proper fuse options in the `JTAGICE mkII -> Fuse` (`JTAGICE 3 -> Fuse`) handler menu of IAR Embedded Workbench for Atmel AVR32 are set, as described in Table 10-4.

10.3 Pre-built images

The SDK comes with a set of ready-to-use binary images of the WSNDemo application. It includes a set of images for different roles preconfigured with distinct MAC addresses so they can be used for creating a small ZigBee network right away. The image name is formed according to the following scheme:

`WSNDemo_<role>_<rf_chip>.<extension>`

To specify node role put `Coord`, `Router`, or `EndDev` for coordinator, router, and end device respectively. For example, to program a device acting as coordinator, use the file with the name

`WSNDemo_Coord_Rf231.elf`

NOTE

Default images are preconfigured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and operate on channel `0x0F` (for RF231 and RF230 radios) or channel `0x01` and channel page 0 (for RF212 radio).

10.4 Running WSNDemo

10.4.1 Starting WSNDemo

To start WSNDemo, do the following:

1. Set up the hardware, as described in Section 10.1.2.
2. Install the Atmel BitCloud SDK, as described in Section 10.1.4.
3. Program one device with the coordinator image file and the other with either the router or end device images, as described in Section 10.2.
4. Connect the coordinator node to the PC using the serial interface.
5. Power on the coordinator node.
6. Run WSNMonitor (see Section 4.5).
7. Power on and reset the rest of the nodes.

10.4.2 Monitoring WSNDemo activity

Network activity can be monitored in two ways:

- Observing LEDs on the development boards, as described in Table 10-5. The LEDx label corresponds to the EVK1105 board.
- Monitoring network topology information through WSNMonitor installed on a PC (see Section 4.5).

Table 10-5. LED indication for WSNDemo on an Atmel EVK1105 board.

Node state	LED0	LED1	LED2
Searching for network	Blinking	OFF	OFF
Joined to network	ON		
+ receiving data		Blinking	
+ sending data to UART (coordinator only)			Blinking
Sleeping (end device only)	OFF	OFF	OFF

10.5 Reserved hardware resources

Table 10-6. Hardware resources used by the stack on an Atmel AT32UC3A0512.

Resource	Description
Processor main clock	48MHz from external quartz
32-bit AVR ports A9, A20, A11, A12, A13	Radio interface
Timer Channel 0	Timer
32-bit AVR ports B30, B31	Sleep / reset

11 Appendix A-4: ATxmega specifics

11.1 Getting started

The Atmel BitCloud SDK supports two different hardware platforms with Atmel XMEGA microcontrollers: ATxmega card hosted on Atmel STK600 and RZ600 boards, and Atmel REB2*xx*ED-EK – with REB-CBB boards. The instructions below highlight the differences between the two platform configurations, where present.

11.1.1 Required hardware

Before installing and using the Atmel BitCloud SDK, make sure that all necessary hardware is available for the kit you would like to use:

- For Atmel STK600:
    - Two or more Atmel ATSTK600s. For each, additionally:
        - Routing card for the Atmel AVR ATxmega256A3 and ATxmega256D3 microcontrollers
        - Selected MCU (either ATxmega256A3 or ATxmega256D3)
        - Atmel RZ600 radio board
    - Atmel AVR JTAGICE mkII (an XMEGA PDI adapter for AVR JTAGICE mkII is required for the ATxmega256D3 device) or Atmel JTAGICE 3
- For REB212ED-EK/REB231ED-ED/REB232ED-EK:
    - Two or more REB-CBBs with ATxmega256A3
    - Radio extender board REB212/REB231/REB232/REB233 or REB231FE2 for each REB-CBB board
    - Atmel AVR JTAGICE mkII or Atmel JTAGICE 3

11.1.2 Hardware setup

For REB2*xx*ED-EK, please refer to [34] for hardware setup instructions.

Atmel STK600 boards require the following setup to use with ATxmega:

1. Attach the ATxmega socket card and routing card to the STK600.
2. If RZ600 radio boards are used, attach the board to `PORTC` on the STK600. Be sure to note the type of radio stick connected (RF231, RF230, or RF212) to ensure compatible firmware is chosen in the later steps.
3. Connect LEDs to the ATxmega expansion board. By default, applications provided with the SDK assume that the LEDs are connected through `PORTE` (defined in `<SDK-Root>\BitCloud\Components\BSP\ATML_STK600\include\bspLeds.h`). Connect `PORTE` on the ATxmega board to the LEDS connector on the STK600 using a 10-wire cable.
4. Route the ATxmega UART to the RS-232 port on the STK600 board. The current revision of the SDK uses asynchronous mode without hardware flow control (`RXD` and `TXD` pins only) on `PORTD` USART 0. Use two-wire cable to connect `PORTD` pins 2 and 3 with the RS-232 SPARE connector: `PD2` to `RXD` and `PD3` to `TXD`.
5. Before performing any further operations, perform the steps required to get started with the ATSTK600 [16]. Refer to AVR Studio Help for details on that subject. At the least, make sure that the STK600 firmware is up to date, and

configure the voltage provided by the STK600 for the ATxmega socket card. For that, perform the following steps:

a. Attach the STK600 to a PC using a USB cable.
b. In Atmel AVR Studio, open `Tools > Program AVR > Connect…` dialogue.
c. Choose the correct Platform = STK600, and press `Connect`.
d. Update the STK600 firmware, if suggested.
e. Go to the `HW Settings` tab.
f. Specify the required voltage in the VTarget field, and press `Write`. Please note that the voltage should be less than 3.6V (for example, 3.0V).

You need to perform this procedure once for each ATSTK600 board.

11.1.2.1 OTAU hardware setup

If you wish to demonstrate OTA upgrade functionality, a serial Atmel DataFlash device (AT45DB041) shall be connected to the Atmel ATxmega256A3/Atmel ATxmega256D3 pins specified in Table 11-1.

The Atmel STK600 board already has built-in DataFlash, and so to use OTAU on an STK600, it is sufficient to connect the `SI`, `SO`, `SCK`, and `/CS` pins under the DataFlash pin heading to the corresponding MCU pins noted in Table 11-1.

Table 11-1. External flash and MCU pin assignment.

DataFlash pin	ATxmega256A3/D3 MCU pin
MOSI	PD5
MISO	PD6
CLOCK	PD7
CS	PD4

11.1.3 System requirements

Before using the SDK, please ensure that the following system requirements are met by your PC and development environment.

Table 11-2. System requirements for ATxmega256A3/D3.

Parameter	Value	Note
CPU	Intel Pentium III, or higher, 800MHz	
RAM	128MB	
Free space on hard disk	50MB	
JTAG emulator	Atmel AVR JTAGICE mkII or Atmel JTAGICE 3 emulator with cable	Required to upload firmware onto the boards and debug through JTAG (see Section 11.2.4)
Operating system	Windows 2000/XP	
IDE	Atmel AVR Studio 5.1.208 OR IAR Embedded Workbench AVR 6.11 (with IAR C/C++ Compiler for AVR v6.11.1.50453 (1))	Required to upload firmware images through JTAG and develop applications using the API (see Section 5.2)

Parameter	Value	Note
Java virtual machine	Java Runtime Environment (JRE) 6, or later	Required to run the WSNMonitor application

Note: 1. Users are strongly encouraged to use the specified versions of AVR Studio and IAR C/C++ Compiler for AVR. Other versions are not supported, and may not work.

11.1.4 Installing the SDK

Proceed with the following installation instructions.

1. Download the archive to your PC and unpack it into an empty folder. The following SDK folders and files will be created.

Table 11-3. The SDK file structure.

Directory/file	Description
`<SDK-Root>\Documentation`	Documentation for the Atmel BitCloud software
`<SDK-Root>\Evaluation Tools\WSNDemo (Embedded)`	Ready-to-use image files for evaluating WSNDemo. Refer to Section 11.3 for a description of the images
`<SDK-Root>\Evaluation Tools\WSNDemo (WSN Monitor)\WSNMonitorSetup.exe`	WSNMonitor installer
`<SDK-Root>\BitCloud\Components`	Header files for the BitCloud stack
`<SDK-Root>\BitCloud\Components\BSP\`	Source, header, and library files for the BitCloud BSP
`<SDK-Root>\BitCloud\lib`	Library files for the BitCloud stack
`<SDK-Root>\Applications\`	Source files for the sample applications

2. Install the selected IDE.
    a. For IAR Embedded Workbench AVR:
        i. Install IAR Embedded Workbench for Atmel AVR [13], if not already installed on your PC.
        ii. Add a Windows environment variable named IAR_AVR_HOME, and set its value to the IAR Embedded Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 6.11`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click New below the System variables list, and enter Variable Name and Variable Value. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.
    b. For Atmel AVR Studio:
        i. Install AVR Studio [3], if not already installed on your PC.
        ii. Add paths to the folder containing the AVRGCC compiler to the Path Windows environment variable. The compiler is located in the

\extensions\Atmel\AVRGCC\3.3.1.27\AVRToolchain\bin directory of the AVR Studio installation directory.

3. Download and install Java Runtime Environment [12], if not already installed on your PC.
4. Demonstration of OTAU functionality and programming devices with Serial Bootloader require the Serial Bootloader package, which should be downloaded from the Atmel website.

11.2 Programming the boards

A firmware image file can be uploaded into the ATxmega device in one of the following ways: using the Serial Bootloader package or using the JTAG interface.

11.2.1 Setting fuse bits

Table 11-4 provides the default fuse bit configuration loaded into Atmel ATxmega256A3/Atmel ATxmega256D3 devices during production. It also describes some use cases when certain fuse bits require values different from the default ones. Based on their own application-specific requirements, users can change the fuse bits as well. See [33] for detailed fuse bit descriptions.

NOTE

Modifying fuse bit settings is possible only using the JTAG programming interface, and cannot be done with the Serial Bootloader package tools.

Table 11-4. Fuse bit settings for ATxmega256A3/D3 device.

Option	Deafult value in ATxmega256A3/D3 devices	Comments
JTAGUSERID	0xFF	Can be changed according to application-specific requirements
WDWP	8 cycles (8ms @ 3.3V)	Can be changed during application development. Must be disabled for final products
WDP	8 cycles (8ms @ 3.3V)	Must be always enabled
DVSDON	OFF	Must be always enabled
BOOTRST	Application reset	Must be set to BOOTLDR if device needs to be programmed with Serial Bootloader or if OTAU support is required
BODACT	BOD Disabled	Can be changed according to application requirements
BODPD	BOD Disabled	
SUT	0ms	
WDLOCK	OFF	
JTAGEN	ON	
EESAVE	OFF	Can be changed according to application-specific requirements, but external clock source shall not be used by sleeping devices.
BODLVL	1.6V	
Resulting bytes:		
FUSEBYTE0	*0xFF*	

Option	Deafult value in ATxmega256A3/D3 devices	Comments
FUSEBYTE1	*0x00*	
FUSEBYTE2	*0xFF*	
FUSEBYTE4	*0xFE*	
FUSEBYTE5	*0xFF*	

By default, programming with Serial Bootloader is not allowed, and so firmware can be uploaded only over the JTAG interface. To turn on bootloader support, the `BOOTRST` fuse must be set to `BOOTLDR` which results in the following fuse bytes:

```
0xFF, 0x00, 0xBF, 0xFE, 0xFF
```

Second, a firmware image of the embedded bootloader (without OTAU support) shall be programmed to the Atmel ATxmega256A3/Atmel ATxmega256D3 using JTAG. After that, application images can be loaded to the device using the Bootloader PC tool from the Serial Bootloader package, as described in Section 11.2.3.

11.2.2 Extended (MAC) address assignment

For proper operation all nodes in ZigBee network shall have unique MAC address values. On the Atmel XMEGA platform, a unique `CS_UID` parameter must be specified for each node in the application configuration, and then the application image must be built separately for each board.

11.2.3 Programming with Serial Bootloader

Programming using Serial Bootloader requires that the embedded bootloader code is loaded to the device via JTAG. Firmware images for the embedded bootloader as well as the Bootloader PC tool, which is needed to load the application image from a PC to the device, are included in the Atmel Serial Bootloader software package available for downloading from the Atmel website.

Images that shall be loaded to the MCU via JTAG may be found under `\Embedded_Bootloader_images\` in the `Atxmega256a3/Atxmega256d3` directory in the package:

- The `Bootloader_ATXmega256A3.hex` file for ATxmega256A3
- The `Bootloader_ATXmega256D3.hex` file for ATxmega256D3

In both cases, the fuse bits should be configured properly (see Table 11-4).

If the embedded bootloader is loaded, the following steps should be executed to upload the application image file to the board:

1. Assemble the board and connect it to the PC:
    a. For STK600:
        a. Assemble the Atmel STK600 board and ATxmega routing card, as described in Section 11.1.2.
        b. Connect PC COM1 port to the RS-232 SPARE port of the board. RS-232 SPARE pins should be connected to the MCU pins specified in Table 11-5.
    b. For REB-CBB:
        a. Attach the radio extender board to the REB-CBB and connect a power cable to the PWR port on the REB-CBB.

b. Connect the RS-232 interface cable to the USARTD0 port on the REB-CBB and to the COM1 port on the PC.

Table 11-5. Host UART and MCU pin connections.

UART pin on host device	ATxmega256A3/D3 MCU pin
RXD	PD2
TXD	PD3

2. Install and run the Bootloader PC tool from the command line or use the GUI. Specify the target image file in `.srec` format and the COM port, and launch the firmware upload.
3. Reset the board using the reset button, if requested.
4. The Bootloader PC tool indicates the programming progress. Once an upload is successfully completed, the board should restart automatically. If an upload fails, the Bootloader PC tool should indicate the reason. In rare cases, the booting process can fail due to communication errors between the board and the PC. If this happens, attempt booting again or try using the conventional serial port instead of USB. If booting fails, the program recently written to the board will be corrupted, but the board can be reprogrammed, as the embedded bootloader remains intact.

See [11] for additional details about the Serial Bootloader package.

WARNING *Using JTAG to program the microcontroller will erase the embedded bootloader, making the loading of application images with Serial Bootloader impossible until the embedded bootloader firmware is reprogrammed to the device.*

11.2.4 Programming with JTAG

Refer to Atmel AVR Studio [3] and IAR Embedded Workbench [13] documentation for the description of how the images can be programmed to the boards using JTAG.

- *For STK600:* the JTAGICE device must be connected to a particular JTAG connector on the STK600 board. This connector is not marked with any label, and is situated next to the connector marked as JTAG (in the blue background area), but closer to the ATxmega expansion board.

  Note that the Atmel ATxmega256D3 devices support only PDI programming. The PDI connector can be found next to the JTAG connector on the STK600 board. A special XMEGA PDI adapter must be used with JTAG for programming ATxmega256D3 devices.
- *For REB-CBB*: connect the JTAGICE device to the DBG port on a REB-CBB board.

Make sure that the board and ATxmega socket card are powered on (refer to Section 11.1.2 for the description of additional steps required for that) and the right options set in the *Fuses* tab (see Table 11-4) before uploading the image through JTAG. Don't forget to select the correct device (the Atmel ATxmega256A3 or ATxmega256D3) in the Device and Signature Bytes field on the Main tab of the programming dialogue. After programming the image, reset the board.

11.3 Pre-built images

The SDK comes with a set of ready-to-use binary images of the WSNDemo application. It includes a set of images for different roles, which are preconfigured with

distinct MAC addresses so they can be used for creating a small ZigBee network right away. The image name is formed according to the following scheme:

```
WSNDemo_<role>_<board>_<rf_chip>_<region>_<MCU>.<extension>
```

To specify the node role, put `Coord`, `Router`, or `EndDev` for coordinator, router, and end device, respectively. To program with JTAG, use files with a `.hex` extension. `.srec` files are loaded with the serial bootloader. `Region` is an optional parameter that specifies the radio frequency used by the image according to the frequency band of the specified region. Possible options are `US`, `EU`, and `China`. For example, to use JTAG to program a device with an ATxmega256A3 MCU on an STK600 board and an RF212 radio acting as coordinator for use in the US, use the file with name

```
WSNDemo_Coord_Stk600_Rf212_US_Atxmega256a3.hex
```

NOTE

Default images are preconfigured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and operate on channel `0x01` and channel page 0 for RF212 radio and on channel `0x0F` for other radio types.

11.4 Running WSNDemo

11.4.1 Starting WSNDemo

To run the WSN Demo application, proceed as follows:

1. Set up the hardware, as described in Section 11.1.2. Make sure to connect the LEDs and UART as described.
2. Install the Atmel BitCloud SDK, as described in Section 11.1.4.
3. Program one device with the coordinator image file and other with either the router or end device images, as described in, as described in Section 11.2.
4. Attach the Atmel STK600 #1 (coordinator) RS-232 connector to a PC COM port.
5. Run WSNMonitor (see Section 4.5).
6. Power on and reset the rest of the nodes.

11.4.2 Monitoring WSNDemo activity

Network activity can be monitored in two ways:

- Observing the STK600 boards' color LEDs (see Table 11-6)
- Monitoring the network topology through the WSNMonitor installed on a PC

Table 11-6. LED indication for the STK600 boards used in WSNDemo.

Node state	LED0 state	LED1 state	LED2 state
Searching for network	Blinking	OFF	OFF
Joined to network	ON		
+ receiving data		Blinking	
+ sending data to UART (coordinator only)			Blinking
Sleeping (end device only)	OFF	OFF	OFF

11.4.3 Demonstrating OTA upgrade functionality

OTA upgrade functionality of the BitCloud SDK can be demonstrated using the Atmel XMEGA platform by using the serial Atmel DataFlash device found on the STK600 board. Once the DataFlash device is connected to the Atmel ATxmega256A3, as defined in Section 11.1.2, the user should configure and install devices as follows:

1. Load the WSNDemo application configured and compiled with APP_USE_OTAU defined as 1 in the `configuration.h` file.
    a. Program the embedded bootloader image: `bootloaderOTAU_ATXmega256A3.hex` or `bootloaderOTAU_ATXmega256D3.hex` from the Serial Bootloader package.
    b. The application image should be converted to `*.srec` format and installed using the Bootloader PC tool from the same package, as described in Section 4.5.3.
    c. The device is able to act as an OTA client, as defined in [30].
2. Program another device with the Runner application available in the `<SDK-Root>\Evaluation Tools\Runner\` directory in the SDK. The Runner node acts as an OTA server, as defined in [30].

Once the images are programmed and WSNDemo devices are joined to the network, follow the instructions given in Section 4.5.3 to update firmware over the air.

11.5 Reserved hardware resources

Table 11-7. Hardware resources reserved by the stack on ATxmega256A3.

Resource	Description
Processor main clock	16MHz from internal 32MHz RC oscillator
Asynchronous clock	1kHz from 32kHz ultra-low-power internal oscillator
SPIC (Port C)	Radio interface
ATxmega ports PC0, PC2, PC3, PC4, PC5, PC6, PC7	Radio interface
RTC	Asynchronous timer
Timer/Counter C1	System timer
PORTC INT0	Radio interface
EEPROM	Storage for user settings accessible via persistent data server
PD4..PD7	External DataFlash, when OTAU functionality is used

12 Appendix A-5: AT91SAM7X-EK specifics

12.1 Getting started

12.1.1 Required hardware

Before installing and using the Atmel BitCloud SDK, make sure that all necessary hardware is available for the kit you would like to use:

- Two or more Atmel AT91SAM7X-EK kits. For each, additionally:
    - Atmel RZ600 radio board with 10-pin squid cable
- Atmel AT91SAM-ICE™ JTAG emulator

12.1.2 Hardware setup

Please refer to [25] for setup instructions.

Table 12-1. SAM7X-EK to RZ600 radio board pin mapping.

SAM7X-EK J16 pin	Radio board pin	Pin name
A10	1	Reset
N/A	2	Misc
A31	3	Interrupt
A9	4	Sleep transmit
A15	5	Chip select
A18	6	MOSI
A17	7	MISO
A19	8	SCK
C32	9	GND
C31	10	VCC

12.1.3 System requirements

Before using the SDK, please ensure that the following system requirements are met by your PC and development environment.

Table 12-2. System requirements for AT91SAM7X-EK.

Parameter	Value	Note
CPU	Intel Pentium III, or higher, 800MHz	
RAM	128MB	
Free space on hard disk	50MB	
JTAG emulator	AT91SAM-ICE emulator with cable	Required to upload firmware onto the boards and debug through JTAG
Operating system	Windows 2000/XP	

Parameter	Value	Note
IDE	Eclipse IDE for C/C++ , YAGARTO toolchain (GCC 4.6.2 [1]) and AT91-ISP v1.12 OR IAR Embedded Workbench for ARM 6.30 (with IAR C/C++ Compiler for ARM v6.30.6.53336/W32 [1])	Required to develop applications using the API (see Section 5.2) and upload firmware images through JTAG
Java virtual machine	Java Runtime Environment (JRE) 6, or later	Required to run WSNMonitor application

Note: 1. Users are strongly encouraged to use the specified version of the YAGARTO and/or IAR toolchains. Other versions are not supported, and may not work.

12.1.4 Installing the SDK

Proceed with the following installation instructions.

1. Download the archive to your PC and unpack it into an empty folder. The following SDK folders and files will be created.

Table 12-3. The SDK file structure.

Directory/file	Description
`<SDK-Root>\Documentation`	Documentation for the Atmel BitCloud software
`<SDK-Root>\Evaluation Tools\WSNDemo (Embedded)`	Ready-to-use image files for evaluating WSNDemo. Refer to Section 12.3 for a description of the images
`<SDK-Root>\Evaluation Tools\WSNDemo (WSN Monitor)\WSNMonitorSetup.exe`	WSNMonitor installer
`<SDK-Root>\BitCloud\Components`	Header files for the BitCloud stack
`<SDK-Root>\BitCloud\Components\BSP\`	Source, header, and library files for the BitCloud BSP
`<SDK-Root>\BitCloud\lib`	Library files for the BitCloud stack
`<SDK-Root>\Applications\`	Source codes for the sample applications
`<SDK-Root>\Third Party Software\6119.inf`	USB-to-serial converter driver

2. Install the selected IDE
    a. For Eclipse IDE and YAGARTO:
        a. Install Eclipse IDE for C/C++ [23], if not already installed on your PC.
        b. Install YAGARTO ARM cross-compiler toolchain [24], if not already installed on your PC. Be sure to install only the supported version of the YAGARTO toolchain, as specified in Table 12-2.
    b. For IAR Embedded Workbench ARM:
        a. Install IAR Embedded Workbench for ARM [22], if not already installed on your PC.

b. Add a Windows environment variable named IAR_ARM_HOME, and set its value to the IAR Embedded Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 6.30`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click New below the System variables list, and enter Variable Name and Variable Value. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.

3. Install AT91-ISP [26], if not already installed on your PC.
4. To install the USB-to-serial converter driver, attach the Atmel AT91SAM7X-EK device to your PC and wait for Windows' request for a specific driver for the device. If the device already has an assigned driver, or if Windows assigns driver to it automatically, go to Start/Control Panel/System/Hardware/Device Manager, double-click the device, and select "Update Driver…." In any case, choose the "Install from a list or specific location" option and point to the 6119.inf file provided with this SDK. Please refer to Section 4.9.1 of [8] for further details and basic troubleshooting options.

- Download and install Java Runtime Environment [12], if not already installed on your PC.

12.2 Programming the boards

A firmware image file can be uploaded into AT91SAM7X-EK device using the JTAG interface. Programming device using the serial bootloader is not supported.

12.2.1 Extended (MAC) address assignment

For proper operation, all nodes in a ZigBee network shall have unique MAC address values. On the SAM7X platform, a unique `CS_UID` parameter must be specified for each device in the application configuration, and then the application image must be built separately for each board.

12.2.2 Programming with JTAG

Atmel AT91SAM-ICE JTAG emulator and SAM Boot Assistant [27] (available in AT91-ISP package) shall be used for programming.

Connect the AT91SAM7X-EK board via the SAM-ICE JTAG emulator to a PC. After that, run Atmel SAM-BA® on the PC and choose \jlink\ARMX as the connection type and AT91SAM7X-EK as the target board. In the opened window select the flash memory tab and in the "Send File Name" field, provide the path to the application firmware image in `.bin` format. Keep the default memory settings and press the "Send file" button. After programming the image, reset the AT91SAM7X-EK board. For details about SAM-BA configuration and usage, please refer to SAM-BA User Guide [27].

NOTE

The Atmel BitCloud stack and applications can be configured to store stack and application parameters in nonvolatile memory (typically, EEPROM). This functionality cannot be demonstrated with the stock SAM7X-EK kit, as there is no external EEPROM on these boards. It is also not enabled in the reference applications and pre-built binaries provided with the SDK.

12.3 Pre-built images

The SDK comes with a set of ready-to-use binary images of the WSNDemo application. It includes a set of images for different roles, which are preconfigured with distinct MAC addresses so they can be used for creating a small ZigBee network right away. The iImage name is formed according to the following scheme:

```
WSNDemo_<role>_<rf_chip>_<region>.<extension>
```

To specify node role, put `Coord`, `Router`, or `EndDev` for coordinator, router, and end device, respectively. `Region` is an optional parameter that specifies the radio frequency used by the image according to the frequency band of the specified region. Possible options are `US`, `EU`, and `China`. For example, to use JTAG to program an Atmel AT91SAM7X-EK device with an RF212 radio acting as coordinator for use in the US, use the file with the name

```
WSNDemo_Coord_Rf212_US.bin
```

NOTE

Default images are preconfigured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and operate on channel `0x0F` (for RF231 and RF230 radios) or channel `0x01` and channel page 0 (for RF212 radio).

12.4 Running WSNDemo

12.4.1 Starting WSNDemo

To run WSN Demo application, proceed as follows:

1. Set up the hardware, as described in Section 12.1.2.
2. Install the Atmel BitCloud SDK, as described in Section 12.1.4.
3. Program one device with the coordinator image file and other with either the router or end device images, as described in Section 12.2.
4. Connect the AT91SAM7X-EK #1 (coordinator) to the PC USB port.
5. Run WSNMonitor (see Section 4.5).
6. Power on and reset the rest of the nodes.

12.4.2 Monitoring WSNDemo activity

Network activity can be monitored in two ways:

- Observing the AT91SAM7X-EK board's color LEDs (see Table 12-4);
- Monitoring network topology through WSNMonitor installed on a PC.

Table 12-4. LED indication for AT91SAM7X-EK boards used in WSNDemo.

Node state	LED1 (red)	LED2 (yellow)	LED3 (green)
Searching for network	Blinking	OFF	OFF
Joined to network	ON		
+ receiving data		Blinking	
+ sending data to UART (coordinator only)			Blinking

12.5 Reserved hardware resources

Table 12-5. Hardware resources reserved by the stack on AT91SAM7X256.

Resource	Description
SPI0	Radio interface
ARM ports PA8, PA9, PA14, PA16, PA17, PA18, PA30	Radio interface
Timer2	Microsecond timer
Timer0	System timer
External IRQ0	Radio interface
SPI1, PA21, PA22, PA23, PA24	For external EEPROM

13 Appendix A-6: SAM3S specifics

13.1 Getting started

The Atmel BitCloud SDK supports two different hardware platforms with SAM3S microcontrollers: Atmel SAM3S-EK with radio boards from the RZ600 kit, and Atmel RF231USB-RD. The instructions below highlight the differences between the two platform configurations, where present.

13.1.1 Required hardware

Before installing and using the Atmel BitCloud SDK, make sure that all necessary hardware is available for the kit you would like to use:

- For SAM3S-EK:
    - Two or more SAM3S-EK boards
    - Atmel RZ600 radio board with AT86RF230, AT86RF231 or AT86RF212 transceiver – for each SAM3S-EK board
    - Atmel AT91SAM-ICE JTAG emulator
- For RF231USB-RD:
    - Two or more RF231USB boards
    - Atmel AT91SAM-ICE JTAG emulator

13.1.2 Hardware setup

- For SAM3S-EK, connect the RZ600 radio board transceiver to the J16/ZigBee connector present on the SAM3S-EK board.

  For other hardware setup instructions, please refer to [29].
- For RF231USB-RD hardware instructions, please refer to [36].

13.1.3 System requirements

Before using the SDK, please ensure that the following system requirements are met by your PC and development environment.

Table 13-1. System requirements for SAM3S-EK.

Parameter	Value	Note
CPU	Intel Pentium III, or higher, 800MHz	
RAM	128MB	
Free space on hard disk	50MB	
JTAG emulator	AT91SAM-ICE emulator with cable	Required to upload firmware onto the boards and debug through JTAG
Operating system	Windows 2000/XP	

Parameter	Value	Note
IDE	Eclipse IDE for C/C++ , YAGARTO toolchain (GCC 4.6.2 (1)) and AT91-ISP v1.12 OR IAR Embedded Workbench for ARM 6.30 (with IAR C/C++ Compiler for ARM v6.30.6.53336/W32 (1))	Required to develop applications using API (see Section 5.2) and upload firmware images through JTAG
Java virtual machine	Java Runtime Environment (JRE) 6, or later	Required to run the WSNMonitor application

Note: 1. Users are strongly encouraged to use the specified version of IAR toolchain. Other versions are not supported, and may not work.

13.1.4 Installing the SDK

Proceed with the following installation instructions.

1. Download the archive to your PC and unpack it into an empty folder. The following SDK folders and files will be created.

Table 13-2. The SDK file structure.

Directory/file	Description
`<SDK-Root>\Documentation`	Documentation for the Atmel BitCloud software
`<SDK-Root>\Evaluation Tools\WSNDemo (Embedded)`	Ready-to-use image files for evaluating WSNDemo. Refer to Section 13.3 for the description of the images
`<SDK-Root>\Evaluation Tools\WSNDemo (WSN Monitor)\WSNMonitorSetup.exe`	WSNMonitor installer
`<SDK-Root>\BitCloud\Components`	Header files for the BitCloud stack
`<SDK-Root>\BitCloud\Components\BSP\`	Source, header, and library files for the BitCloud BSP
`<SDK-Root>\BitCloud\lib`	Library files for the BitCloud stack
`<SDK-Root>\Applications\`	Source codes for the sample applications

2. Install the selected IDE
    a. For Eclipse IDE and YAGARTO:
        a. Install Eclipse IDE for C/C++ [23], if not already installed on your PC.
        b. Install YAGARTO ARM cross-compiler toolchain [24], if not already installed on your PC. Be sure to install only the supported version of the YAGARTO toolchain, as specified in Table 13-2.
    b. For IAR Embedded Workbench ARM:
        a. Install IAR Embedded Workbench for ARM [22], if not already installed on your PC.
        b. Add a Windows environment variable called IAR_ARM_HOME, and set its value to the IAR Embedded

Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 6.30`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click `New` below `System variables` list and enter `Variable Name` and `Variable Value`. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.

3. Install AT91-ISP [26], if not already installed on your PC.
4. Download and install Java Runtime Environment [12], if not already installed on your PC.

13.2 Programming the boards

A firmware image file can be uploaded into the Atmel SAM3S-EK device using the JTAG interface and using the Serial Bootloader package tools.

13.2.1 Extended (MAC) address assignment

For proper operation, all nodes in a ZigBee network shall have unique MAC address values. On the SAM3S platform, a unique `CS_UID` parameter must be specified for each device in the application configuration, and then the application image must be built separately for each board.

NOTE

The BitCloud stack and applications can be configured to store stack and application parameters in nonvolatile memory (typically, EEPROM). This functionality cannot be demonstrated with a stock SAM3S-EK kit, as there is no external EEPROM on these boards. It is also not enabled in reference applications and pre-built binaries provided with the SDK.

13.2.2 Programming with JTAG

Atmel AT91SAM-ICE JTAG emulator and SAM Boot Assistant [27] (available in AT91-ISP package) shall be used for programming.

- Connect a SAM3S-EK board or a RF231USB board via the Atmel SAM-ICE JTAG emulator to a PC (a RF231USB board is connected to SAM-ICE JTAG emulator via SAM-ICE adapter)
- Run SAM-BA on the PC, and choose \jlink\ARMX as the connection type, SAM3S4-EK as the target board, and use the default JLink speed
- In the opened window, select the flash memory tab, and in the "`Send File Name`" field provide the path to application firmware image in `.bin` format
- Keep the default memory settings and press the "`Send file`" button
- After programming the image reset the board

For details about Atmel SAM-BA configuration and usage, please refer to SAM-BA User Guide [27].

13.2.3 Programming with Serial Bootloader

Programming using Serial Bootloader requires that the embedded bootloader code is loaded to the device via JTAG. Firmware images for the embedded bootloader as well as the Bootloader PC tool, which is needed to load the application image from a PC to the device, are included in the Atmel Serial Bootloader software package available for downloading from the Atmel website.

The images that shall be loaded to the MCU via JTAG may be found under `\Embedded_Bootloader_images\` in the `At91sam3s4c` directory in the package.

If the embedded bootloader is loaded, the following steps should be executed to upload the application image file to the board:

1. Assemble the board and connect it to the PC (a USB stick is just attached to a PC's USB port).
2. Install and run the Bootloader PC tool from the command line or use the GUI. Specify the target image file in `.srec` format and the COM port, and launch the firmware upload.
3. The Bootloader PC tool indicates the programming progress. Once an upload is successfully completed, the board should restart automatically. If an upload fails, the Bootloader PC tool should indicate the reason. In rare cases, the booting process can fail due to communication errors between the board and the PC. If this happens, attempt booting again or try using the conventional serial port instead of USB. If booting fails, the program recently written to the board will be corrupted, but the board can be reprogrammed, as the embedded bootloader remains intact.

See [11] for additional details about the Serial Bootloader package.

IMPORTANT

Since the embedded bootloader for SAM3S uses the DFU standard, the application should enable the DFU support (turned on in the application's `configuration.h` file). Otherwise after loading the application via Serial Bootloader tools loading another image will not be possible.

WARNING

Using JTAG to program the microcontroller will erase the embedded bootloader, making the loading of application images with Serial Bootloader impossible until the embedded bootloader firmware is reprogrammed to the device.

13.3 Pre-built images

The SDK comes with a set of ready-to-use binary images of the WSNDemo application. It includes a set of images for different roles, which are preconfigured with distinct MAC addresses so they can be used for creating a small ZigBee network right away. The image name is formed according to the following scheme:

`WSNDemo_<role>_<board>_<rf_chip>_<region>_<MCU>.<extension>`

To specify node role, put `Coord`, `Router`, or `EndDev` for coordinator, router, and end device, respectively. `Region` is an optional parameter that specifies the radio frequency used by the image according to the frequency band of the specified region. Possible options are `US`, `EU`, and `China`. For example, to use JTAG to program a SAM3S-EK device with an RF212 radio acting as coordinator for use in the US, use the file with the name

`WSNDemo_Coord_Rf212_US.bin`

NOTE

Default images are preconfigured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and operate on channel `0x0F` (for RF231 and RF230 radios) or channel `0x01` and channel page 0 (for RF212 radio).

13.4 Running WSNDemo

13.4.1 Starting WSNDemo

To run WSN Demo application, proceed as follows:

1. Set up the hardware, as described in Section 13.1.2.
2. Install the Atmel BitCloud SDK, as described in Section 13.1.4.
3. Program one device with the coordinator image file and other with either the router or end device images, as described in Section 13.2.
4. Connect the SAM3S-EK programmed as coordinator to a PC COM port.
5. Run WSNMonitor (see Section 4.5).
6. Power on and reset the rest of the nodes.

13.4.2 Monitoring WSNDemo activity

Network activity can be monitored in two ways:

- Observing the SAM3S-EK board's color LEDs (see Table 13-3);
- Monitoring network topology information through WSNMonitor installed on a PC.

Table 13-3. LED indication for SAM3S-EK boards used in WSNDemo.

Node state	LED D2 (blue)	LED D3 (yellow)	LED D4 (red)
Searching for network	Blinking	OFF	OFF
Joined to network	ON		
+ receiving data		Blinking	
+ sending data to UART (coordinator only)			Blinking

13.5 Reserved hardware resources

Table 13-4. Hardware resources reserved by the stack on Atmel ATSAM3S4C.

Resource	Description
SPI	Radio interface
MCU ports PB3, PA15, PA17, PA18, PA12, PA13, PA14	Radio interface
Timer0, channel0	System timer
Port A interrupt	Radio interface

14 Appendix B-1: WSNDemo over-the-air protocol

This appendix describes the protocol used by the WSNDemo sample application. The description includes the format of the messages exchanged over the air between the connected nodes. The protocol description allows non-standard nodes (for example, those using third-party sensors not available on the standard evaluation boards and kits) to transfer sensor readings and have them visualized in the same WSNMonitor application.

14.1 Message format

End devices and routers send messages to the coordinator using the following format.

Table 14-1. WSNDemo message format.

Field name	Length	Description
Message type	1 byte	Type of the messages. Must be 0x01 (0x01 is the only supported message type for the current revision of WSNDemo)
Node type	1 byte	Type of the sending node: 0 – coordinator 1 – router 2 – end device
IEEE address	8 bytes	IEEE address of the sending node
Short address	2 bytes	Short address of the sending node
Version	4 bytes	Version of the WSNDemo application protocol used by the sending node. Currently set to 0x01010100
Channel mask	4 bytes	Channel mask set on the sending node
PANID	2 bytes	PAN ID of the network to which the sending node is attached
Channel	1 byte	The channel on which the sending node operates
Parent address	2 bytes	Short address of the parent node
LQI	1 byte	LQI observed by the node that sends this message
RSSI	1 byte	RSSI observed by the node that sends this message
<Additional fields>	<Variable>	Optional additional fields; see description in Section 14.2, below

14.2 Additional fields

The message may contain zero, one, or more additional fields that follow the mandatory fixed-width fields described in Table 14-1. The order of the additional fields is not fixed. The size of the additional fields may vary – each field contains a sub-field defining its size. Below is the description of the general format of an additional field.

Table 14-2. Additional field format.

Sub-field name	Length	Description
Field type	1 byte	Type of the additional field. The possible values are listed below
Field size	1 byte	Size of the field data in bytes. Note: this size does not include the field type and field size sub-fields
Field data	<Variable>	The data depend on the field type, the size of the data is provided by the field size

The following types of additional fields are defined:

Table 14-3. Additional field types.

Field type	Description
0x01	Sensor data for board type 1. Used for MeshBean boards
0x20	Node name

Please note that in the current version of WSNDemo, devices send additional fields of type 0x01 (sensor readings for boards of type 1) only. Unrecognized additional fields are discarded by the WSNMonitor application. The field data format for different field types are described in Table 14-4 and Table 14-5.

Table 14-4. Field data for type 0x01: Sensor data for boards of type 1.

Offset	Length	Data type	Description
0	4 bytes	Unsigned int	Battery status reading
4	4 bytes	Unsigned int	Temperature sensor reading
8	4 bytes	Unsigned int	Light sensor reading

Table 14-5. Field data for type 0x20: Node name.

Offset	Length	Description
0	<Variable>	Zero-terminated ASCII string

15 Appendix B-2: WSNMonitor serial protocol

This appendix describes the protocol and message format used over the serial connection between the network coordinator and the WSNMonitor application running on the PC. The messages sent on the serial connection are basically the messages wrapped as defined below:

Table 15-1. Serial message format.

Offset	Length	Description
0	2 bytes	Start sequence: 0x10 0x02
2	N bytes	Variable-length payload: the message received from the end device or router, or generated by the coordinator, in the format described in Section 14.1. All 0x10 bytes in this payload are duplicated to avoid confusion with a start sequence or end sequence
N+2	2 bytes	End sequence: 0x10 0x03
N+4	1 byte	Checksum: Sum of the bytes [0..N+3] mod 256

16 References

[1] BitCloud Stack API Reference

[2] AVR2050: BitCloud Developer's Guide; User Guide

[3] Studio Archive (AVR Studio installer downloads)

[4] WinAVR User Manual – 20100110

[5] Using the GNU Compiler Collection

[8] AT91 USB CDC Driver Implementation

[9] ZigBit Development Kit User's Guide

[10] AVR2051: SerialNet User Guide

[11] AVR2054: Serial Bootloader User Guide

[12] Java Runtime Environment

[13] IAR Embedded Workbench for Atmel AVR

[14] AVR2044: RCB128RFA1 – Hardware User Manual

[15] ATAVR128RFA1-EK1 description

[16] ATSTK600 board description

[17] Radio Extender Board REB231 V4.0.2

[18] ATEVK1105 description

[19] 32-bit AVR UC3 Software Framework 1.5.0

[20] AVR32 GNU Toolchain

[21] IAR Embedded Workbench for Atmel 32-bit AVR

[22] IAR Embedded Workbench for Atmel ARM

[23] Eclipse IDE for C/C++ Developers

[24] YAGARTO compiler tool chain

[25] AT91SAM7X-EK Evaluation Board for AT91SAM7X and AT91SAM7XC.

[26] AT91 In-system Programmer (ISP)

[27] SAM Boot Assistant (SAM-BA) User Guide

[28] AVR600: RZ600 HW Manual

[29] SAM3S-EK Development Board User Guide

[30] 095264r12 ZigBee Over-the-Air Upgrading Cluster Specification

[31] ATmega1281 description

[32] ATmega128RFA1 description

[33] ATxmega256A3 and ATxmega256D3 descriptions

[34] AVR2042: REB Controller Base Board - Hardware User Guide

[35] AVR2058: BitCloud OTAU User Guide

[36] RF231USB – User Guide

[37] AVR2059: BitCloud Porting Guide

17 Table of contents

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chou-ku, Tokyo 104-0033
JAPAN
Tel: (+81) 3523-3551
Fax: (+81) 3523-7581