# AVR2052: BitCloud Quick Start Guide

## Features

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

## 1 Introduction

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

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 wireless sensor network. The SDK provides a complete software and documentation toolkit for prototyping, developing and debugging custom applications on top of BitCloud's application programming interface (API).

8-bit AVR® Microcontrollers

Application Note

Rev. 8200H-AVR-02/10

## 2 References


[1] AVR2051: BitCloud Stack Documentation

[2] AVR2050: BitCloud User Guide

[3] AVR Studio. User Guide. Available in HTML Help within the product.

[4] WinAVR User Manual – 20090313

[5] Using the GNU Compiler Collection

[6] RZRAVEN Firmware Documentation. (AVR2017: RZRAVEN Firmware)

[7] AVR2015: RZRAVEN Quick Start Guide

[8] AT91 USB CDC Driver Implementation. 6269A–ATARM–10-Oct-06
http://www.atmel.com/dyn/resources/prod_documents/doc6269.pdf

[9] ZigBit Development Kit User's Guide
http://www.meshnetics.com/downloads/docs/

[10] AVR205X: SerialNet User Guide. Available in BitCloud SDK
http://www.atmel.com/bitcloud/

[11] AVR2054: Serial Bootloader User Guide. Available in BitCloud SDK
http://www.atmel.com/bitcloud/

[12] Java Runtime Environment
http://java.sun.com/javase/downloads/index.jsp

[13] IAR Embedded Workbench for Atmel AVR
http://www.iar.com/website1/1.0.1.0/107/1/

[14] ATmega128RFA1 α-package Quick Start Guide

[15] ATSTK600 description
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4254

[16] Radio Extender Board REB231 V4.0.2
http://www.dresden-elektronik.de/shop/prod72.html

[17] ATEVK1105 description
http://www.atmel.com/evk1105

[18] AVR32 UC3 Software Framework 1.5.0
http://www.atmel.com/dyn/resources/prod_documents/AVR32-SoftwareFramework-AT32UC3-1.5.0.zip

[19] AVR32 GNU Toolchain
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4118

[20] IAR Embedded Workbench for Atmel AVR32
http://www.iar.com/website1/1.0.1.0/124/1/

# 3 Overview

BitCloud is a full-featured, professional grade embedded software ZigBee stack from Atmel®. The stack provides a software development platform for reliable, scalable, and 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 following hardware platforms are supported by BitCloud SDK.

**Table 3-1. Supported hardware platforms**

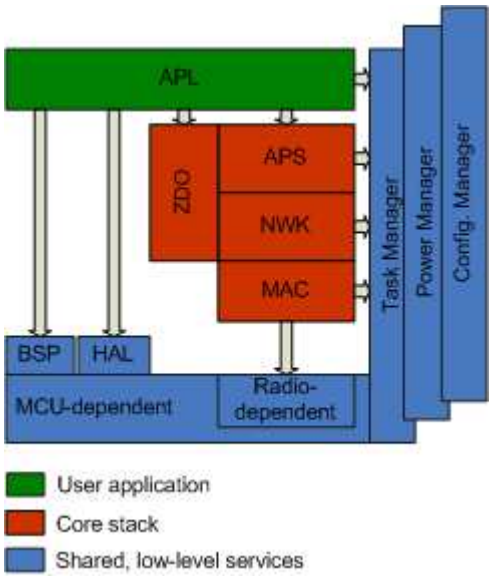
| Name in This Document | Platform (MCU + RF) | Supported Modules | Supported Evaluation Kit | Appropriate SDK |
|---|---|---|---|---|
| ATAVRRZRAVEN | AT90USB1287 + AT86RF230<br>ATmega1284P + ATmega3290P + AT86RF230 | N/A | ATAVRRZRAVEN (consists of ATAVRRZUSBSTICK and ATAVRRAVEN devices) | BitCloud for ATAVRRZRAVEN |
| ZigBit | ATmega1281 + AT86RF230 | ATZB-24-B0 (ZigBit B0);<br>ATZB-24-A2 (ZigBit A2) | ATZB-DK-24 (ZDK) | BitCloud for ZDK |
| ZigBit Amp | ATmega1281 + AT86RF230 | ATZB-A24-UFL (ZigBit Amp) | ATZB-DK-A24<br>(ZDK Amp) | BitCloud for ZDK Amp |
| ZigBit 900 | ATmega1281 + AT86RF212 | ATZB-900-B0 (ZigBit 900) | ATZB-DK-900 (ZDK 900) | BitCloud for ZDK 900 |
| megaRF | ATmega128RFA1 | N/A | α-package and ATmega128RFA1 card hosted on ATSTK600 | BitCloud for megaRF |
| UC3 | AT32UC3A0512 + AT86RF231 | N/A | EVK1105 | BitCloud for AVR32 UC3 |

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

BitCloud stack is fully compliant with 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 developer's convenience and ease-of-use in mind.

The main structure of the BitCloud stack is presented in Figure 3-1.

**Figure 3-1.** BitCloud Block Diagram




The topmost of the core stack layers, APS, provides the highest level of networking-related API visible to the application. ZDO provides a set of fully compliant ZigBee Device Object API which enable main network management functionality (e.g. start, reset, formation, join). It also defines ZigBee Device Profile types, device and service discovery commands implemented by the stack.

The general guidelines to developing applications with BitCloud are presented in [2]. The SDK also includes WSN Monitor PC application in binary format and WSNDemo embedded application available in binary format and source code.

The source code for WSNDemo application can be modified and extended, making it possible to develop WSN applications for a variety of application scenarios. WSN Monitor and WSNDemo applications are described in detail in Section 5.

For ZigBit/ZigBit Amp/ZigBit 900 and ATAVRRZRAVEN platforms, the SDK also includes other reference applications as described in Section 6.2.1.

# 4 Getting Started

This chapter describes how to quickly get BitCloud running on the selected hardware platform. BitCloud SDK is available for several platforms as described in Section 2. Before proceeding, select the SDK version that matches your target platform.

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

**Table 4-1.** Hardware-Specific Getting Started Sections

| For Platform | Refer to Section |
|---|---|
| ATAVRRZRAVEN | 8.1 |
| ZigBit / ZigBit Amp / ZigBit 900 | 9.1 |

| For Platform | Refer to Section |
|---|---|
| megaRF | 10.1 |
| UC3 | 11.1 |

After completing the installation, try running WSNDemo application by programming the devices with ready-to-use images as described in Section 5.2.

Finally, Section 5.5 describes how to get started creating new or modifying existing applications based on BitCloud C API.

# 5 WSNDemo Application

## 5.1 Overview

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

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

For ZigBit /ZigBit Amp / ZigBit 900 platforms, end devices and routers and the coordinator read the sensor data from on-board light and temperature sensors, and forward collected data to WSN Monitor application for visualization. On ATAVRRZRAVEN, megaRF and UC3 platforms, zero values are sent to the network coordinator to emulate sensor data and demonstrate data transmission.

End devices follow a duty cycle (i.e. microcontroller and radio transceiver are put to sleep periodically), waking up to transmit the data to the coordinator device. Using the serial connection, the coordinator transmits the received packets, along with its own sensor data (also 0s for ATAVRRZRAVEN, megaRF and UC3), to WSNMonitor application. Those transmitted values are displayed on WSNMonitor panes as temperature, light and battery level measurements (0s for ATAVRRZRAVEN, megaRF and UC3).

WSN Monitor also visualizes the network topology by drawing a tree of nodes which have joined the network. For each of the nodes, the parameters like the node's address, its node sensor information and link quality data are displayed.

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

In reference to WSNDemo application, Section 5.3 describes how to setup and use the boards. The user interface is described in Section 5.5.

The application is delivered with source code which demonstrates how to develop a wireless network application using BitCloud API and provides a number of useful programming templates for common application tasks. Development of custom applications is described in Section 5.5.

In the WSNDemo, the number of routers and end devices is limited only by the network parameter settings described in Section 6.2.1. However, for ATAVRRZRAVEN kit additional restrictions apply. These are outlined in Table 5-1.

**Table 5-1.** Allowed board role / device type combinations for WSNDemo on ATAVRRZRAVEN

| Device Type | Allowed Board Role | Comments |
|---|---|---|
| Coordinator | RZUSBSTICK | Coordinator needs USB interface to send data to PC-side WSNMonitor application |
| Router | AVRRAVEN or RZUSBSTICK | |
| End-Node | AVRRAVEN | End-nodes also demonstrate sleep capabilities (MCU and RF only) |

## 5.2 Programming the Boards

As a first step, WSNDemo images should be loaded onto the boards. The locations of WSNDemo image files are platform specific and are provided in sections specified in Table 4-1.

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

**Table 5-2.** Platform-Specific Programming Sections

| For Platform | Refer to Section |
|---|---|
| ATAVRRZRAVEN | 8.2 |
| ZigBit / ZigBit Amp / ZigBit 900 | 9.2 |
| megaRF | 10.2 |
| UC3 | 11.2 |

Running any ZigBee or ZigBee PRO application, WSNDemo included, requires that every device in the network has a 64-bit unique *MAC address*. See the appropriate sections in Table 5-2 for how MAC addresses are assigned for each type of supported boards. Typically, there is a number of pre-compiled images provided with the SDK that can be used right away without any modification.

ZigBit, ZigBit Amp, ZigBit 900, ATAVRZRAVEN platforms do not require manual assignment of MAC addresses as the evaluation boards are equipped with a dedicated unique ID chip, which BitCloud stack uses automatically on start up. ATAVRRZRAVEN boards include an external EEPROM chip with an embedded unique ID, which is also used by BitCloud automatically on start up.

Also, note that the default images are configured to use a particular *extended PAN ID* and *channel mask*. To change those parameters you must also modify the Configuration file and rebuild the application. All and all, 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.

## 5.3 Running WSNDemo

The details of running WSNDemo differ for each target platform. Please refer to an appropriate section listed in the table below for the platform-specific instructions.

**Table 5-3.** Platform-Specific WSNDemo Sections

| For Platform | Refer to Section |
|---|---|
| ATAVRRZRAVEN | 8.4 |
| ZigBit / ZigBit Amp / ZigBit 900 | 9.4 |
| megaRF | 10.4 |
| UC3 | 0 |

## 5.4 Network Organization

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

If you power on the coordinator, 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 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 Configuration file as described in Section 6.2.1.

**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.

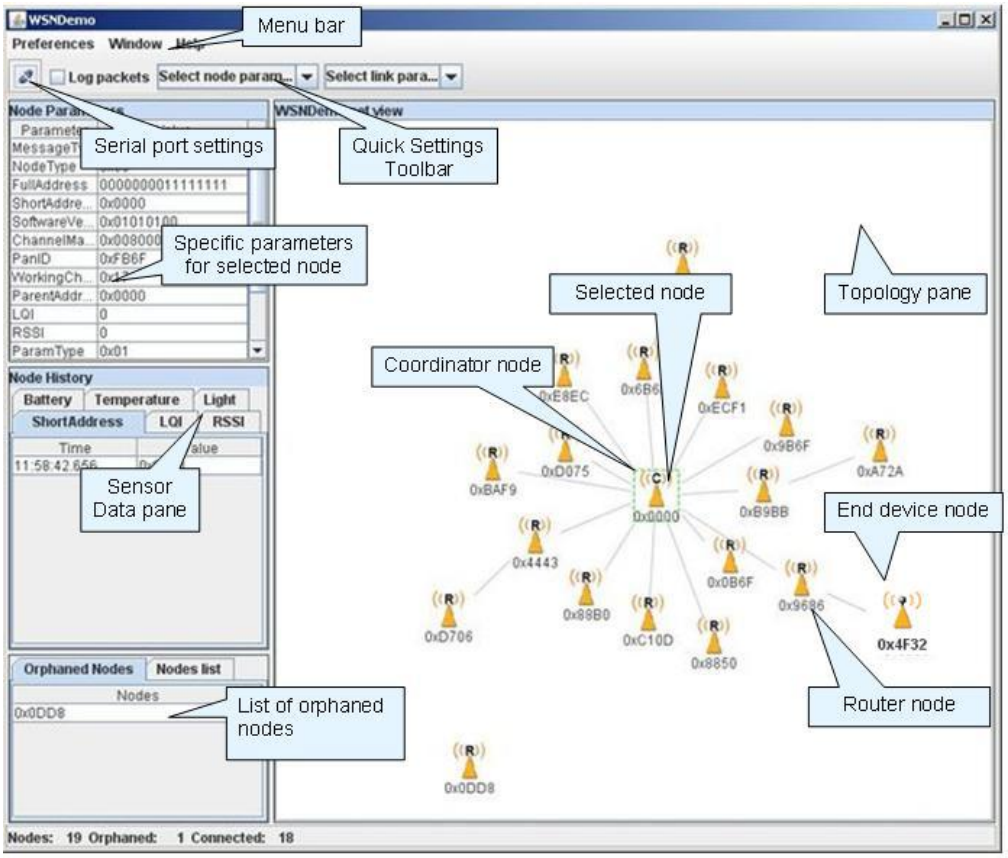
By default, the channel mask for all application images provided with SDK contains a single channel. In rare cases, if the frequency corresponding to the radio channel is busy, the coordinator node may stay in the network search mode. If this happens, you should change the application's channel mask to select another channel by changing the application's Configuration file, and recompiling the application as described in Section 6.2.1.

## 5.5 WSN Monitor

WSN Monitor is a PC GUI application for WSNDemo that is used to display a ZigBee network topology and other information about a wireless sensor network. A typical WSN Monitor screen is shown in the figure below. It contains *topology pane*, *sensor data pane*, *node data pane* and application toolbars.

**Figure 5-1. WSN Monitor GUI**

*Topology pane* displays the network topology in real time, which helps the user monitor the formation and dynamic changes in the network while the nodes join,




send data or leave. 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.

*Sensor data pane* displays data coming from onboard sensors of the selected node (see Section 5.5.2). It is presented in graph and table form. Other parameters can be observed for each node in table form. Node data pane includes a *sensor selection combo-box* used to switch between sensor types.

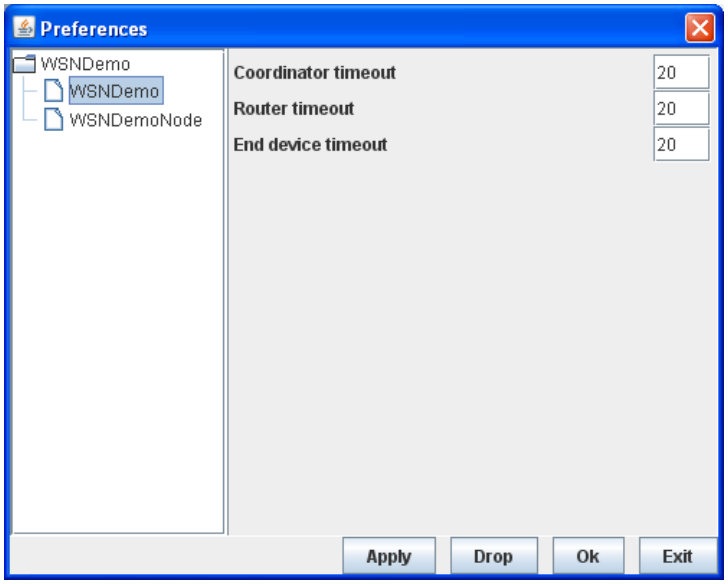
By default in topology pane nodes are labeled with their short addresses. However by a double click another title can be assigned to any desired node. If "Cancel" is pressed in opened window short address is set back as node's title.

### 5.5.1 Setting up node timeouts

The `Window/Preferences` menu of WSN Monitor 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 WSN Monitor application waits for a packet from a node before assuming that a node is no longer part of the network. Note that this value does not correspond to the frequency with which data is transmitted by

each type of device. To get smooth topology visualization, setting timeouts to 20 sec is recommended for coordinator and router and 30 sec is recommended for end device. Assuming default application configuration, these timeouts cover 3 periods between sending a packet so at least 3 packets would need to be lost before a node is removed from WSN Monitor's Topology Pane.

**Figure 5-2.** WSNMonitor Preferences menu




### 5.5.2 Sensor data visualization

Each of the boards sends temperature/light/battery sensors readings (or emulated values) to the coordinator, which in turn sends it to the PC. The WSN Monitor allows to displays the readings from onboard sensors next to a node icon inside Topology Pane (see Section 5.5). For this corresponding option shall be selected in the node/link parameters from the Quick Settings Toolbar.

The user can select any node in Topology Pane to monitor the node's activity and see the node data in three different forms:

- Text table
- Chart
- The onboard sensor's data displayed next to each node in 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 clicked on and a dashed frame is drawn around it.

The same values are shown on Sensor Data Pane, so the user can observe how the values change over a period of time.

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.

# 6 Programming with BitCloud API

## 6.1 API Overview

The BitCloud internal architecture follows IEEE 802.15.4 and ZigBee-defined convention for splitting the networking stack into its logical layers. Besides the core stack containing the protocol implementation, BitCloud contains additional layers implementing shared services (e.g. task manager, configuration manager, and power manager) and hardware abstractions (e.g. 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 BitCloud API significantly reduce application complexity and simplify the development effort. BitCloud Stack Documentation [1] provides detailed information on the stack's C API and its use.

The topmost of the core stack layers, APS, provides the highest level of networking-related APIs visible to the application. ZDO provides a set of fully compliant ZigBee Device Object APIs which enable main network management functionality (e.g. start, reset, formation, join). ZDO also defines ZigBee Device Profile types, device and service discovery commands implemented by the stack.

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. Task manager is the stack scheduler which all multiple internal stack components and the user application to run on the same microcontroller. The task manager utilizes a proprietary priority queue-based algorithm specifically tuned for multi-layer stack environment and demands of time-critical network protocols. Power management routines are responsible for gracefully shutting down all stack components and saving system state when preparing to sleep and restoring system state when waking up. Configuration manager is used by both the internal stack components and the user application alike to provide a common way to store and retrieve network parameters like Extended PAN ID and channel mask.

The Hardware Abstraction Layer (HAL) includes a complete set of APIs for using on-module hardware resources (e.g. 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 (e.g. IRQ, TWI, SPI, UART, 1-wire), where hardware interface is supported by the platform. Board support package (BSP) includes a complete set of drivers for managing standard peripherals (e.g. sensors, UID chip, sliders, and buttons) placed on development boards such as those provided with ZigBit, ZigBit Amp and ZigBit 900 evaluation kits.

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

## 6.2 Development Tools

Development tools consist of (1) a integrated development environment (e.g. AVR Studio or IAR Embedded Workbench) where sample applications may be modified, compiled and debugged, (2) a corresponding compiler tool chain (e.g. WinAVR, IAR) which provides everything necessary to compile application source code into binary images, and (3) a programming device (e.g. JTAG), which may be used to program and debug the application on a target platform.

Atmel's AVR Studio [3] and/or IAR Embedded Workbench for Atmel AVR [13] may be used to develop and debug applications based on BitCloud API on AVR-based platforms including ZigBit, ZigBit Amp, ZigBit 900 and ATAVRRZRAVEN. This IDE supports editing of application source code, compilation, linking object modules with libraries, and application debugging. AVR Studio is integrated with WinAVR – a Windows port of GNU compiler tool chain for the Atmel AVR microprocessors. More information about WinAVR and GNU compiler tools is available in [4] and [5].

In AVR Studio, each application has a corresponding project file identified by the `.aps` extension. All the necessary information about a project is contained in the project file, which can be double-clicked to open the application's project in AVR Studio. Likewise, In IAR Embedded Workbench, each application has a corresponding `.eww` file which can be double-clicked to open the application's project. For detailed instructions on how to compile and debug applications using the supported tools, refer to Section 6.2.2.

Platform-specific sections which describe development tools installation and setup instructions are listed in Table 4-1.

### 6.2.1 Sample Applications

For all platforms, the SDK is supplied with WSNDemo sample application provided in source code. WSNDemo is presented in detail in Section 5. To better understand the communication between the network nodes and between the coordinator and the PC, the user can refer to Section 12 and Section 13.

For some platforms additional sample applications are available as indicated in the table below.

**Table 6-1.** Sample Applications Availability

| Application | Brief Description | ATAVRRZR AVEN | ZigBit, ZigBit Amp ZigBit 900 | megaRF | UC3 |
|---|---|---|---|---|---|
| WSNDemo | Featured SDK application demonstrating network functionality of software and additional network visualization with WSN Monitor. See section 5. | 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 | | |
| Lowpower | Show how to collect data from low-power, sleeping devices employing the simplest power management strategy. | X | 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 | | |

| Application | Brief Description | ATAVRRZR AVEN | ZigBit, ZigBit Amp ZigBit 900 | megaRF | UC3 |
|---|---|---|---|---|---|
| PingPong | Shows how process multiple simultaneous data transmissions. Each node is waiting for a wireless message, and then passes it to the next node. | | X | | |
| ThroughputTest | Measures wireless UART bandwith 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 inside the "`./Sample Applications/WSNDemo`" directory. For other sample applications (where available), the source code can be found in "`./Sample Applications/<application-name>`" directories.

Network parameters and their default values are defined in Configuration file. However, when the application is compiled using IAR Embedded Workbench IDE, the network parameters are defined in "`iarConfiguration.h`" file located in "`iar/AVR`" subdirectory of the appropriate application directory. For the WSNDemo application, this file is located in "`./BitCloud/Sample Applications/WSNDemo/iar/AVR`". In all other cases, including compiling from the command line using IAR compiler, Configuration file will be used.

### 6.2.2 Compiling Applications

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

**Table 6-2.** Platform-Specific Compilation Options

| For Platform | AVR Studio + WinAVR | IAR Embedded Workbench |
|---|---|---|
| ATAVRRZRAVEN | X | X |
| ZigBit / ZigBit Amp / ZigBit 900 | X | X |
| megaRF | X | X |
| UC3 | | X |

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

#### 6.2.2.1 AVR Studio + WinAVR

- *Command line:* Compile application by running `make` utility. Before running `make`, be sure that Configuration file has `COMPILER_TYPE` variable set to `GCC`.
- *IDE:* Open the `.aps` file from the appropriate directory with AVR Studio and execute "`Build/Rebuild All`" from the main menu.

As a result, .hex, .srec, .bin and .elf application images will be generated.

6.2.2.2 IAR Embedded Workbench

- *Command line:* Compile application by running make utility. Before running make, be sure that Configuration file has COMPILER_TYPE variable set to IAR. The .hex, .srec, .bin and .elf image files will then be generated.
- *IDE:* Open the .eww file in the "iar/AVR" for ZigBit, RZRAVEN platforms, subdirectory of the appropriate application directory (for WSNDemo WSNDemo.eww file from the "./Sample Applications/WSNDemo/iar/AVR" subdirectory) with IAR Embedded Workbench and execute "Rebuild All" item from the Project menu. By default the .a90 (for WSNDemo: WSNDemo.a90) file will be generated in the "iar/AVR/Debug/exe" subdirectory (for WSNDemo: in "./Sample Applications/WSNDemo/iar/ AVR/Debug/exe" directory) with format as specified in Linker Output Options of the IAR project.

## 6.3 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 hardware abstraction layer (HAL) of the BitCloud stack. When building custom applications on top of the BitCloud API, the user is encouraged to use the high-level APIs instead of the low-level register interfaces to ensure that the resource use does not overlap with that of the stack.

The hardware resources reserved for the internal use by the stack in BitCloud are listed in platform-specific sections specified in the table below. 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 6-3.** Platform-Specific Reserved Resources

| For Platform | Refer to Section |
|---|---|
| ATAVRRZRAVEN | 8.5 |
| ZigBit / ZigBit Amp / ZigBit 900 | 9.4 |
| ATmega128RFA1 | 10.5 |
| UC3 | 11.5 |

# 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. Check if your PC conforms to the minimum system requirements (see Section 4).
3. Check if the PC USB or UART interface is working and the correct drivers are installed (see Section 4).
4. Check hardware kit documentation if you have setup and are using the hardware in the right way. See Section 4 for specific hardware setup requirements.
5. For ATAVRRZRAVEN, check LCD indication of AVRRAVEN nodes to detect the cases when they are not responding or behaving unusually.
6. Make sure you have programmed the right images and set the correct Fuses values (see Section 5.2).

Resetting the node may be required.

The table below represents some typical problems that you may encounter while working with the Development Kit and possible solutions.

**Table 7-1.** Typical problems and solutions

| Problem | Solution |
|---|---|
| For ATAVRRZRAVEN:<br>The AVRRAVEN board does not indicate its activity on LCD. | Make sure that WSNDemo image is loaded. The LCD controlling logic depends on the application, and may work differently for the images built by you. |
| WSN Monitor fails to start. | Make sure Java machine is properly installed on your PC. See Section 4. |
| No node is shown on the Topology Pane in the WSN Monitor | Check if the WSN Monitor uses the proper COM port and if not, change it and restart the program. |
| WSN Monitor shows NO DATA in the Sensor Data Graph Pane. | No node is selected. Select the required node by mouse-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 5.5) along with MAC addresses mapped to the nodes. |

# 8 Appendix A-1. ATAVRRZRAVEN Specifics

## 8.1 Getting Started

### 8.1.1 Required Hardware

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

1. One ATAVRRZUSBSTICK
2. One or more ATAVRRAVEN boards
3. 100-mil to 5o mil JTAG adapter
4. JTAGICE mkII

### 8.1.2 Hardware Setup

1. Solder the JTAG headers onto the boards as described in [7].
2. Make sure that the boards have fresh batteries.

### 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 8-1.** System requirements for ATAVRRZRAVEN

| Parameter | Value | Note |
|---|---|---|
| CPU | Intel Pentium III or higher, 800MHz | |
| RAM | 128MB | |
| Free space on hard disk | 50MB | |
| JTAG emulator | JTAGICE mkII emulator with cable | Required to upload and debug firmware onto the boards through JTAG (see Section 5.2). |
| Operating system | Windows 2000/XP | |
| IDE | AVR Studio 4.17 and WinAVR 20090313 (1)<br>OR<br>IAR Embedded Workbench AVR 5.3<br>(with IAR C/C++ Compiler for AVR 5.30.6 (2)) | Required to upload firmware images through JTAG (see Section 5.2), and to develop applications using API (see Section 6.2) |
| Java Virtual Machine | Java Runtime Environment (JRE) 5 Update 8, or later | Required to run WSNMonitor application |

Notes: 1,2 Users are strongly recommended to use specified versions of WinAVR 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. Make sure that path to this folder contains no blank spaces. As a result, the following SDK folders and files will be created.

**Table 8-2.** The SDK file structure

| Directory/File | Description |
|---|---|
| ./Documentation | Documentation on BitCloud software |
| ./Evaluation Tools/WSNDemo (Embedded) | Ready-to-use image files for evaluating WSNDemo. Refer to section 8.3 5.2for the description of the images |
| ./Evaluation Tools/WSNDemo (WSN Monitor)/WSNMonitorSetup.exe | WSN Monitor installer |
| ./Evaluation Tools/SerialNet | Ready-to-use image file for SerialNet application. Firmware can be used on ATAVRRZUSBSTICK only. Refer to [10] for more information on SerialNet. |
| ./BitCloud/Components | Header files for BitCloud Stack |
| ./BitCloud/Components/BSP/ | Source, header and library files for BitCloud BSP |
| ./ BitCloud/Components/BSP/ RAVEN/AT3290P | Source and header files for LCD controller firmware |
| ./BitCloud/lib | Library files for BitCloud Stack |
| ./Sample Applications/ | Source files for sample applications. |
| ./Third Party Software/6119.inf | USB to Serial Converter driver |

2. Install desired IDE:

   2.1. For AVR Studio and WinAVR:

      2.1.1. Install AVR Studio [3], if not already installed on your PC.

      2.1.2. Install WinAVR development suite [4], if not already installed on your PC. Be sure to install only the supported version of WinAVR as specified in Table 8-1.

   2.2. For IAR Embedded Workbench AVR:

      2.2.1. Install IAR Embedded Workbench for AVR [13], if not already installed on your PC.

      2.2.2. In SDK directory "./BitCloud/lib/" make sure that in files "Makerules_AtmlUsbDongle_At90usb1287_8Mhz_Iar" and "Makerules_Raven_Atmega1284_4Mhz_Iar" the IAR_PATH variable points to the correct installation directory of IAR Embedded Workbench. Update, if needed.

      2.2.3. Add IAR Embedded Workbench "bin" directory (for default installation located in "C:\Program Files\IAR Systems\Embedded Workbench 5.3\avr\bin") to the system PATH environment variable. To update the PATH variable go to Control Panel > System > Advanced > Environment Variables, select "Path" variable from the "System variables" list, press "Edit", and append ";" followed by the actual "bin" directory name to the end of the Variable

value, then press “OK”. This step is required if you plan to build embedded images using IAR Embedded Workbench from command line.

3. Install USB to Serial Converter driver. To install the driver, please attach the RZUSBSTICK device to your PC and wait for Windows to request for a specific driver for the device. If the RZUSBSTICK already has an assigned driver, or Windows assigned driver to it automatically, go to Start/Control Panel/System/Hardware/Device Manager, double-click the RZUSBSTICK device and select “`Update Driver…`”. Choose the “`Install from a list or specific location`” option and point to 6119.inf provided with this SDK. Please refer to section 4.9.1 of [8] for further details and basic troubleshooting options.
4. Download and install Java Runtime Environment [12], if not already installed on your PC.

## 8.2 Programming the Boards

### 8.2.1 Setting Parameters

At startup, the software assigns the 64-bit MAC address to the device as follows. If at compile time `CS_UID` parameter is set to 0 BitCloud attempts to load MAC address from an external EEPROM chip available on RZRAVEN and RZUSBSTICK boards. If there is no such UID then 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 address cannot be obtained automatically from external source, separate firmware images shall be created for each device with unique `CS_UID` parameter specified in application configuration (see Section 116.2.1) every time an image is compiled.

### 8.2.2 Programming

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

Set the following options in the Fuses tab before uploading the image through JTAG. Note the values differ for different types of boards.

**Table 8-3.** Fuse bits setting for AT90USB1287 (RZUSBSTICK)

| Option | Value |
|---|---|
| BODLEVEL | `Brown-out detection at VCC=2.4 V` |
| HWBE | Disabled |
| OCDEN | Disabled |
| JTAGEN | Enabled |
| SPIEN | Enabled |
| WDTON | Disabled |
| EESAVE | Disabled |
| BOOTSZ | `Boot Flash size=4096 words start address=$F000` |
| BOOTRST | Disabled |
| CKDIV8 | Disabled |
| CKOUT | Disabled |
| SUT_CKSEL | `Ext. Crystal Osc. 3.0-8.0 MHz; Start-up time: 16K CK + 65 ms` |
| EXTENDED | `0xFC` |

| Option | Value |
|---|---|
| HIGH | 0x99 |
| LOW | 0xFD |

**Table 8-4.** Fuse bits setting for ATmega1284p (AVRRAVEN)

| Option | Value |
|---|---|
| BODLEVEL | Brown-out detection at VCC=1.8V |
| OCDEN | Disabled |
| JTAGEN | Enabled |
| SPIEN | Enabled |
| WDTON | Disabled |
| EESAVE | Disabled |
| BOOTSZ | Boot Flash size=512 words start address=$FE00 |
| BOOTRST | Disabled |
| CKDIV8 | Enabled |
| CKOUT | Disabled |
| SUT_CKSEL | Int. RC Osc.; Start-up time: 6 CK + 65 ms |
| EXTENDED | 0xFE |
| HIGH | 0x9F |
| LOW | 0x62 |

**Table 8-5.** Fuse bits setting for ATmega3290p (LCD on AVRRAVEN)

| Option | Value |
|---|---|
| BODLEVEL | Brown-out detection at VCC=1.8V |
| RSTDISBL | Disabled |
| OCDEN | Disabled |
| JTAGEN | Enabled |
| SPIEN | Enabled |
| WDTON | Disabled |
| EESAVE | Disabled |
| BOOTSZ | Boot Flash size=512 words start address=$3E00 |
| BOOTRST | Disabled |
| CKDIV8 | Enabled |
| CKOUT | Disabled |
| SUT_CKSEL | Int. RC Osc.; Start-up time: 6 CK + 65 ms |
| EXTENDED | 0xFD |
| HIGH | 0x9D |
| LOW | 0x62 |

For additional details, please refer to "readme.html > RZRAVEN: RZRAVEN Firmware Documentation > Miscellaneous information > Programming the RZRAVEN Firmware with Programmer/Debugger" section of [6].

## 8.3 Pre-Built Images

The SDK comes with ready-to-use binary images of WSNDemo application. There is a set of images for different device types:

1. `WSNDemoApp_USB_Coord.hex` – for RZUSBSTICK (AT90USB1287 controller) acting as Coordinator
2. `WSNDemoApp_USB_Router.hex` – for RZUSBSTICK (AT90USB1287 controller) acting as Router
3. `WSNDemoApp_Raven_Router.hex` – for AVRRAVEN (ATmega1284p microcontroller) acting as Router
4. `WSNDemoApp_Raven_EndDev.hex` – for AVRRAVEN (ATmega1284p microcontroller) acting as End Device
5. `Raven_3290P_LCD.hex` – for AVRRAVEN's LCD controller (ATmega3290p)

And SerialNet application [10] for RZUSBSTICK:

6. `SerialNet_USB.hex`.

In all ready-to-use binary images MAC address (`CS_UID`) is loaded automatically from a dedicated external EEPROM chip, ensuring that unique MAC addresses are assigned to all network nodes. Note that the default WSNDemo images are configured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and channel mask with only channel `0x0F` enabled for operation.

## 8.4 Running WSNDemo

### 8.4.1 Starting WSNDemo

To start WSNDemo, do the following:

1. Setup the hardware as described in Section 8.1.2.
2. Install BitCloud SDK as described in Section 8.1.4.
3. Load precompiled WSNDemo firmware images to devices:
    a. On RZUSBSTICK (Coordinator): `WSNDemoApp_USB_Coord.hex`;
    b. On AVRRAVEN (Router): `WSNDemoApp_Raven_Router.hex` for ATmega1284p microcontroller and `Raven_3290P_LCD.hex` for ATmega3290p LCD controller (the board contains two JTAG headers – refer to [6]);
    c. On AVRRAVEN (End-Device): `WSNDemoApp_Raven_EndDev.hex` for ATmega1284p microcontroller and `Raven_3290P_LCD.hex` for ATmega3290p LCD controller (the board contains two JTAG headers – refer to [6]);
4. Plug-in the coordinator USB stick into PC.
5. Run WSN Monitor (see Section 5.5).
6. Power ON the rest of the nodes.

### 8.4.2 Monitoring WSNDemo Activity

Network activity can be monitored in two ways:

- observing the LCD screens of AVRRAVEN devices and color LEDs of RZUSBSTICK devices (see meaning of LCD information and LEDs described in the tables below);
- viewing the network information through the WSN Monitor installed on PC.

**Table 8-6.** LCD indication for AVRRAVEN boards used in WSNDemo

| Node State | Visual Information on LCD Screen |
|---|---|
| Searching for network | “JOINING” string displayed; red LED blinking; “sun” symbol displayed |
| Joined to network | “ROUTER” or “ENDDEV” string displayed, depending on the node role; red LED is on; “sun” symbol displayed |
| + receiving data | “RX” indicator visible (please note the limitations due to LCD refresh rate) |
| + sending data | “TX” indicator visible (please note the limitations due to LCD refresh rate) |
| Sleeping (end device only) | Red LED is off; “moon” symbol displayed |

**Table 8-7.** LED indication for the RZUSBSTICK devices used in WSNDemo

| Node State | LEDs indication |
|---|---|
| Powered on | Blue LED is on |
| Searching for network | Red LED blinking |
| Joined to network | Red LED is on |
| + receiving data | Yellow LED |
| + sending data to ZigBee network (routers and end devices only) | Green LED |
| + sending data to USB (coordinator only) | Green LED |
| Sleeping | Not supported for RZUSBSTICK |

## 8.5 Reserved Hardware Resources

**Table 8-8.** Hardware resources reserved by the stack on RZUSBSTICK devices

| Resource | Description |
|---|---|
| Processor main clock | 8MHz oscillator with external quartz |
| SPI | Radio interface |
| AT90USB1287 ports: PB0, PB1, PB2, PB3, PB4, PB5, PB7, PD4 | Radio interface |
| Timer/Counter3 | Radio interface |
| Timer/Counter1 capture input | Radio interface |
| Timer/Counter1 | System timer |

**Table 8-9.** Hardware resources reserved by the stack on AVRRAVEN devices

| Resource | Description |
|---|---|
| Processor main clock | 4 or 8MHz from internal RC-oscillator or external radio frequency |
| SPI | Radio interface |
| ATmega ports PB0, PB1, PB3, PB4, PB5, PB6, PB7, PD6 | Radio interface |
| ATmega ports PC6, PC7 | Asynchronous timer interface |
| Timer/Counter2 | Asynchronous timer |
| Timer/Counter3 | Radio interface |
| Timer/Counter1 | System timer |
| Timer1 ICP IRQ | Radio interface |
| EEPROM | Storage for user settings accessible via Persistent Data Server |

# 9 Appendix A-3: ZigBit, ZigBit Amp and ZigBit 900 Specifics

## 9.1 Getting Started

### 9.1.1 Required Hardware

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

- ATZB-DK-24, ATZB-DK-A24, or ATZB-DK-900:
    - ATZB-DK-24 contains ATZB-EVB-24-B0, ATZB-EVB-24-SMA, ATZB-EVB-24-A2 (MeshBean evaluation board) with mounted ZigBit modules;
    - ATZB-DK-A24 contains ATZB-EVB-A24-UFL (MeshBean Amp evaluation board) with ZigBit Amp modules;
    - ATZB-DK-900 contains ATZB-EVB-900-B0 (MeshBean 900 evaluation board) with ZigBit 900 modules.
- JTAGICE mkII

### 9.1.2 Hardware Setup

No special pre-usage assembly is required for MeshBean boards supplied with ZigBit Development Kits.

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 9.1.4);
- via an AC/DC adaptor.

The nominal voltage is 3V for MeshBean and MeshBean 900 boards, 3.3V for MeshBean Amp. Using AC/DC adaptor automatically disconnects AA batteries. Using USB port disconnects the AC/DC adaptor.

### 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 9-1.** 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 | JTAGICE mkII emulator with cable | Required to upload and debug firmware onto the boards through JTAG (see Section 5.2). |
| Operating system | Windows 2000/XP | |

| Parameter | Value | Note |
|---|---|---|
| IDE | AVR Studio 4.17 and WinAVR 20090313 (1)<br>OR<br>IAR Embedded Workbench AVR 5.3<br>(with IAR C/C++ Compiler for AVR 5.30.6 (2)) | Required to upload firmware images through JTAG (see Section 5.2), and to develop applications using API (see Section 6.2) |
| Java Virtual Machine | Java Runtime Environment (JRE) 5 Update 8, or later | Required to run WSNMonitor application |

Notes: 1,2 Users are strongly recommended to use the specified version of WinAVR and IAR C/C++ Compiler for AVR. 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. As a result, the following SDK folders and files will be created.

**Table 9-2.** The SDK file structure

| Directory/File | Description |
|---|---|
| `./Documentation` | Documentation on BitCloud software |
| `./Bootloader` | Contains Serial bootloader image file and installer for PC |
| `./Evaluation Tools/WSNDemo (Embedded)` | Ready-to-use image files for evaluating WSNDemo. Refer to section 9.3 for the description of the images. |
| `./Evaluation Tools/WSNDemo (WSN Monitor)/WSNMonitorSetup.exe` | WSN Monitor installer |
| `./Evaluation Tools/SerialNet` | Ready-to-use image files for SerialNet application. Refer to [10] for more information on SerialNet. |
| `./BitCloud/Components` | Header files for BitCloud Stack |
| `./BitCloud/Components/BSP` | Source, header and library files for BitCloud BSP |
| `./BitCloud/lib` | Library files for BitCloud Stack |
| `./Sample Applications` | Source files for sample applications. |

2. Install desired IDE:

   2.1. For AVR Studio and WinAVR:

      2.1.1. Install AVR Studio [3], if not already installed on your PC.

      2.1.2. Install WinAVR development suite [4], if not already installed on your PC. Be sure to install only the supported version of WinAVR as specified in Table 8-1.

   2.2. For IAR Embedded Workbench AVR:

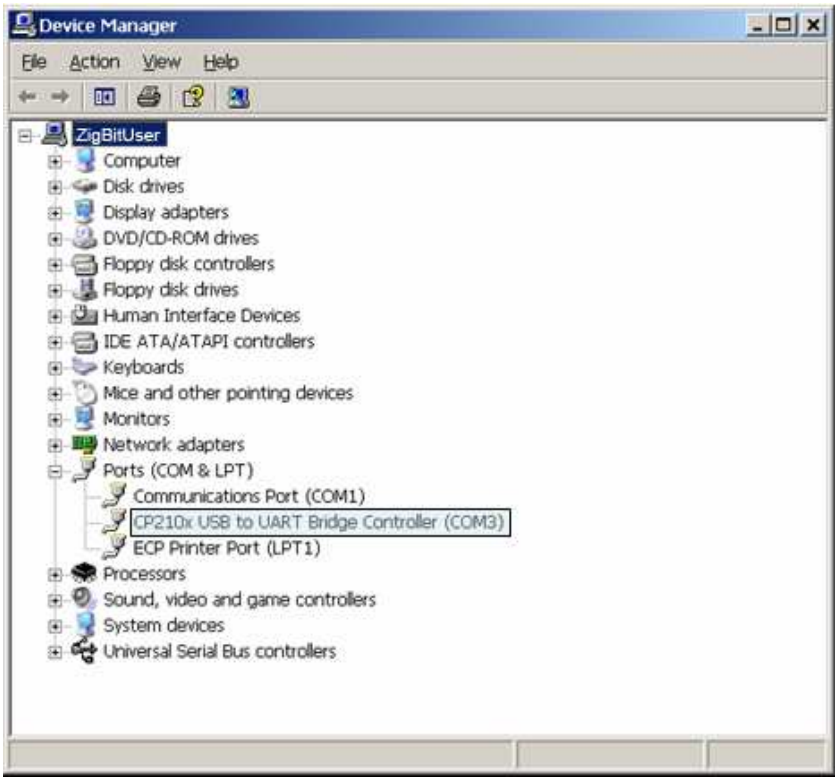
2.2.1. Install IAR Embedded Workbench for AVR [13], if not already installed on your PC.

2.2.2. In SDK directory "`./BitCloud/lib/`" make sure that in file "`Makerules_ZigBit_Atmega1281_8Mhz_Iar`" and the IAR_PATH parameter points to the correct installation directory of IAR Embedded Workbench. Modify it, if needed.

2.2.3. Add IAR Embedded Workbench "`bin`" directory (for default installation located in "`C:\Program Files\IAR Systems\Embedded Workbench 5.3\avr\bin`") to the system PATH environment variable. To update the PATH variable go to `Control Panel > System > Advanced > Environment Variables`, select "Path" variable from the "System variables" list, press "Edit", and append ";" followed by the actual "bin" directory name to the end of the Variable value, then press "OK". This step is required if you plan to build embedded images using IAR Embedded Workbench from command line.

3. The board can be connected to host PC via USB port, using USB 2.0 A/mini-B cable. USB is a familiar connection option. Furthermore, it provides the 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 host PC via serial port, using a serial cable. Please note that USB and serial port (RS-232) share the same physical port on the board. They cannot be used at the same time. Keep in mind that the connection mode is controlled by setting of jumper on a MeshBean. Refer to Section 9.3 for the description of connectors and jumpers on MeshBean boards.
5. If you plan to use USB connection, install USB to UART Bridge VCP driver. To install the driver, please do the following:
    1. Download the driver from https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx
    2. 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.
    3. 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 on the figure below:

**Figure 9-1.** Correctly installed COM port for MeshBean device




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

### 9.1.5 Selected 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 pinouts refer to [9]. Note that J2 settings differ for ZigBit, ZigBit 900, and ZigBit Amp.

**Table 9-3.** J2 jumper settings for ZigBit and ZigBit 900: power source

| Jumper position | Description |
|---|---|
| J2 bridges POWER pin and BAT pin | ZigBit is powered by primary source (battery, USB or AC/DC adapter). |
| J2 bridges `POWER` pin and `DC/DC` pin | ZigBit is powered by 3.6 V internal voltage regulator. |

**Table 9-4.** J2 jumper settings for ZigBit Amp: power source

| Jumper position | Description |
|---|---|
| J2 bridges pin 2 and pin 3 | ZigBit Amp is powered by USB |
| J2 bridges `pin 2 and pin 1` | ZigBit Amp is powered by external DC source or by batteries if external DC source is disconnected. |

**Table 9-5.** J3 jumper settings for all MeshBean types: Serial/USB selection

| Jumper position | Description |
|---|---|
| J3 bridges central pin and RS-232 pin | The board will use serial port (available in the Expansion slot) for connection to the host. |
| J3 bridges central pin and USB pin | The board will use USB for connection to the host. |

**Warning:** Any other position of jumpers J2 and J3 or their omission may permanently damage the MeshBean boards.

## 9.2 Programming the Boards

### 9.2.1 Setting Parameters

At startup, the software assigns the 64-bit MAC address to the device as follows. If at compile time `CS_UID` parameter is set to 0 BitCloud attempts to load MAC address from a dedicated UID chip available on MeshBean board via 1-wire interface. If there is no such UID then 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 address cannot be obtained automatically from external source, separate firmware images shall be created for each device with unique `CS_UID` parameter specified in application configuration (see Section 116.2.1) every time an image is compiled.

### 9.2.2 Programming

An image file can be uploaded into the boards in one of two ways: using Serial Bootloader utility, or, in AVR Studio, using JTAG emulator.

Be careful selecting the method of the node programming. Each of MeshBean boards provided as a part of ZDK come with the bootstrap uploaded onto the ZigBit's microcontroller, which is needed to run Serial Bootloader. Using a JTAG to program the microcontroller will erase the bootstrap, making the loading of application images with Serial Bootloader inoperable until the bootstrap is restored.

To program a board using Serial Bootloader perform the following steps:

1. Connect MeshBean to the PC via USB or serial port, depending on the position of jumper J3 (see Section 9.1.5).
2. Run Serial Bootloader. In command line or in GUI, specify the image file as WSNDemo.srec and the COM port. See [11]
3. Press reset button on the board. If a node has been configured as end device and it is currently controlled by an application, the node should be powered off before reprogramming.
4. Release reset button on the board. Serial Bootloader expects that the button will be released within approximately 30 seconds. If this does not happen, the booting process will terminate.
5. Serial Bootloader indicates the operation progress. Once an upload is successfully completed, the board would restart automatically. If an upload fails, Serial Bootloader would indicate the reason. In rare cases, booting process can fail due to the communication errors between the board and the PC. If this happened, attempt booting again or try using conventional serial port, instead of

USB. If booting fails, the program written to the board recently would be corrupted, but the board can be reprogrammed again as the bootstrap should remain intact.

Refer to AVR Studio documentation for the description of how the images can be programmed to the boards using JTAG.

Set the following options in the Fuses tab before uploading the image through JTAG.

**Table 9-6.** Fuse bits setting for ZigBit, ZigBit Amp, ZigBit 900

| Option | Value |
|---|---|
| BODLEVEL | Brown-out detection disabled |
| OCDEN | Disabled |
| JTAGEN | Enabled |
| SPIEN | Enabled |
| WDTON | Disabled |
| EESAVE | Disabled |
| BOOTSZ | Boot Flash size=1024 words start address=$FC00 |
| BOOTRST | Disabled*<br>If the node is to be programmed with the use of Serial Bootloader, enable the BOOTRST option. |
| CKDIV8 | Enabled |
| CKOUT | Disabled |
| SUT_CKSEL | Int. RC Osc.; Start-up time: 6 CK + 65 ms |

Make sure the following hex values appear in the bottom part of `Fuses` tab:

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

If the node is to be programmed with the use of Serial Bootloader, enable additionally the BOOTRST option. Make sure the following hex value string appears at the bottom of Fuses tab:

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

By default, each of the boards coming in ZDKs is preprogrammed with this fuse setting.

## 9.3 Pre-Built Images

The SDK comes with the ready-to-use binary images in `.hex`, and `.elf` formats for programming using JTAG and in `.srec` format if using Serial Bootloader [11]):

- ZigBit/ZigBit Amp
  - WSNDemo application: `WSNDemoApp.*`.
  - SerialNet application [10] : `SerialNet.*`
- For ZigBit 900:
  - WSNDemo application: `WSNDemoApp_US.*`, `WSNDemoApp_EU.*` and `WSNDemo_China.*`, with specific settings for indicated regional regulatory requirements.
  - SerialNet application [10]: `SerialNet.*`

The ready-to-use binary images retrieve MAC address automatically from UID 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 ZigBit Amp (`WSNDemoApp.*`),
- channel `0x00` and channel page 0 (`WSNDemoApp_EU.*`) or channel `0x01` and channel page 0 (`WSNDemoApp_US.*`) for ZigBit 900,
- channel `0x01` and channel page 5 (`WSNDemoApp_China.*`) for ZigBit 900.

### 9.3.1 Starting WSNDemo

To start WSNDemo, do the following:

- Setup the hardware as described in Section 9.1.2.
- Install BitCloud SDK as described in Section 9.1.4.
- Program devices as described in section 9.2.
- Configure one single node as a coordinator, and make the others be routers and end devices (see Section 9.3.2). Any of the boards provided can be configured with any role.
- Connect the coordinator node to the PC, using USB port on the coordinator board
- Power on the coordinator node
- Run WSN Monitor (see Section 5.5)
- Power ON and reset the rest of the nodes.

### 9.3.2 Node Role Configuration

The role of the node – coordinator, router, or end-device is configured using DIP switches on MeshBean board.

**Table 9-7.** DIP switches configurations on MeshBean boards used in WSNDemo

| DIP Switches | | | Selected Role |
|---|---|---|---|
| **1** | **2** | **3** | |
| ON | OFF | OFF | Coordinator |
| OFF | ON | OFF | Router |
| OFF | OFF | ON | End-Device |

### 9.3.3 Monitoring WSNDemo Activity

Network activity can be monitored in two ways:

- observing color LEDs of MeshBean boards (see the table below);
- viewing the network information through WSN Monitor installed on PC.

**Table 9-8.** 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 | |

| Node State | LED1 (Red) | LED2 (Yellow) | LED3 (Green) |
|---|---|---|---|
| + sending data to UART (coordinator only) | | | Blinking |
| Sleeping (end device only) | OFF | OFF | OFF |

## 9.4 Reserved Hardware Resources

**Table 9-9.** Hardware resources reserved by the stack on ZigBit, ZigBit Amp, and ZigBit 900 modules

| Resource | Description |
|---|---|
| Processor main clock | 8 MHz 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 |

# 10 Appendix A-4: ATmega128RFA1 Specifics

## 10.1 Getting Started

BitCloud supports two different development platforms with ATmega128RFA1: α-package development kit [14] and STK600 boards [15]. If in the text below difference for these platforms is not stated explicitly then it is valid for both of them.

### 10.1.1 Required Hardware

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

1. For α-package:
    - Two or more RCB128RFA1 boards with 2.4GHz antennas and AAA batteries
    - one or more RCB Breakout Boards
    - one RS232 interface cable for RCB Breakout Board
    - JTAGICE mkII.
2. For STK600 boards:
    - Two or more STK600 boards, each with Atmega128RFA1 top card and 2.4GHz antenna
    - JTAGICE mkII

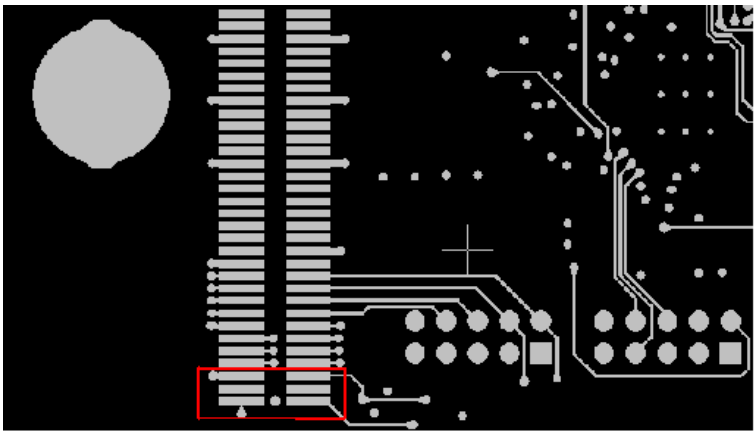
### 10.1.2 Hardware Setup

For α-package please refer to [14] for hardware setup instructions.

STK600-ATmega128RFA1 top cards require following modifications:

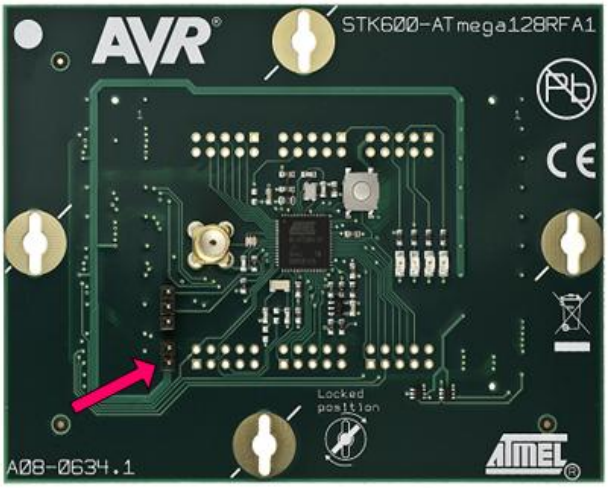
1. Cover three bottom-left contacts on the rear side of the top card with non-conducting material (e.g. paper sticker) as shown by red rectangular on Figure 10-1.

**Figure 10-1.** Contacts to be insulated on STK600-ATmega128RFA1 top card



2. Make sure jumper is present at pins as indicated by red arrow on Figure 10-2.

**Figure 10-2.** Jumper setting for STK600-ATmega128RFA1 top card.




3. Assemble top card with STK600 board.
4. Default output for USART interface is performed via PD2, PD3 pins. In order to communicate over RS232 port they shall be connected to RXD and TXD pins of RS232 SPARE port respectively.

Before continuing any further operations, perform the steps required to get started with ATSTK6000 [15]. Refer to AVR Studio Help [3] for details on that subject. At least, make sure that STK600 firmware is up-to-date, and configure the voltage provided by STK600 for ATmega128RFA1 top card. For that, perform the following steps:

a. Attach STK600 to PC using USB cable.
b. In AVR Studio open `Tools > Program AVR > Connect...` dialogue
c. Choose the right Platform = STK600 and press Connect.
d. Update the STK600 firmware, if suggested.
e. Go to HW Settings tab.
f. Specify the 3.3V in the VTarget field and press Write.

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

### 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 10-1.** System requirements for Atmega128RFA1

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

| Parameter | Value | Note |
|---|---|---|
| JTAG emulator | JTAGICE mkII emulator with cable | Required to upload and debug firmware onto the boards through JTAG (see Section 5.2). |
| Operating system | Windows 2000/XP | |
| IDE | AVR Studio 4.17 and WinAVR 20090313(1)<br>OR<br>IAR Embedded Workbench AVR 5.3 (with IAR C/C++ Compiler for AVR 5.30.6 (2)) | Required to upload firmware images through JTAG (see Section 5.2), and to develop applications using API (see Section 6.2) |
| Java Virtual Machine | Java Runtime Environment (JRE) 5 Update 8, or later | Required to run WSNMonitor application |

Notes: 1. Users are strongly recommended to use the specified version of WinAVR. 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. As a result, the following SDK folders and files will be created.

**Table 10-2.** The SDK file structure

| Directory/File | Description |
|---|---|
| `./Documentation` | Documentation on BitCloud software |
| `./Bootloader` | Contains serial bootloader image file and installer for PC |
| `./Evaluation Tools/WSNDemo (Embedded)` | Ready-to-use image files for evaluating WSNDemo. Refer to section 0 for the description of the images. |
| `./Evaluation Tools/WSNDemo (WSN Monitor)/WSNMonitorSetup.exe` | WSN Monitor installer |
| `./BitCloud/Components/` | Header files for BitCloud Stack |
| `./BitCloud/Components/BSP/` | Source, header and library files for BitCloud BSP |
| `./BitCloud/lib` | Library files for BitCloud Stack |
| `./Sample Applications/` | Source files for sample applications. |

2. Install AVR Studio [3], if not already installed on your PC. Be sure to install only the supported version of AVR Studio as specified in Table 10-1.
3. Install WinAVR development suite [4], if not already installed on your PC. Be sure to install only the supported version of WinAVR as specified in Table 10-1.
4. Download and install Java Runtime Environment [12], if not already installed on your PC.

## 10.2 Programming the Boards

### 10.2.1 Setting Parameters

At startup, the software assigns the 64-bit MAC address to the device as follows. If at compile time `CS_UID` parameter is set to 0 BitCloud attempts to load MAC address from a dedicated external EEPROM chip available on RCB128RFA1 as well as on Atmega128RFA1 top card via SPI interface. If there is no such chip then 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 address cannot be obtained automatically from external source, separate firmware images shall be created for each device with unique `CS_UID` parameter specified in application configuration (see Section 116.2.1) every time an image is compiled.

### 10.2.2 Programming

An image file can be uploaded into the boards in one of two ways: using Serial Bootloader utility or using JTAG emulator.

Programming a board using Serial Bootloader requires that bootstrap is loaded to the device via JTAG. For RCB128RFA1 with RCB breakout board Bootloader_ATmega128RFA1_RCB_BB.hex image file shall be flashed via JTAG. For STK600 Bootloader.hex file shall be loaded to ATmega128RFA1. In both cases in the fuse bit configuration provided in Table 10-3 BOOTRST should be enabled. If bootstrap is loaded following steps should be executed to upload the application image file to the board:

1. Assemble board and connect it to PC:
    a. For RCB128RFA1:
        i. Assemble RCB128RFA1 and RCB Breakout boards (RCB_BB) together.
        ii. Connect RS232 interface cable to J1 extender on RCB_BB and COM1 port on the PC.
    b. For STK600:
        i. Assemble STK600 board and ATmega128RFA1 top card as described in Section 10.1.2.
        ii. Connect PC COM1 port to RS232 SPARE port.
2. Run Serial Bootloader application on the PC. In command line or in GUI, specify the .srec image file and the COM port. See [11].
3. Perform HW reset on the board if requested. Serial Bootloader expects that the reset will be done within 30 seconds. If this does not happen, the booting process will terminate.
4. Serial Bootloader indicates the operation progress. Once upload is successfully completed, the board would restart automatically. If an upload fails, Serial Bootloader would indicate the reason. In rare cases, booting process can fail due to the communication errors between the board and the PC. If this happened, attempt booting again. If booting fails, the program written to the board recently would be corrupted, but the board can be reprogrammed again as the bootstrap should remain intact.

Refer to [14] and [15] for the description of how the images can be programmed to corresponding development boards using JTAG.

Note that using a JTAG to program the microcontroller will erase the bootstrap if present, thus loading of application images with Serial Bootloader will become inoperable until the bootstrap is loaded to ATmega128RFA1 again.

Set the following options in the Fuses tab before uploading the image through JTAG.

**Table 10-3.** Fuse bits setting for ATmega128RFA1

| Option | Value |
|---|---|
| BODLEVEL | Brown-out detection at VCC=1.8 V |
| OCDEN | Disabled |
| JTAGEN | Enabled |
| SPIEN | Enabled |
| WDTON | Disabled |
| EESAVE | Disabled |
| BOOTSZ | Boot Flash size=1024 words start address=$FC00 |
| BOOTRST | Disabled*<br>If the node is to be programmed with the use of Serial Bootloader, enable the BOOTRST option. |
| CKDIV8 | Enabled |
| CKOUT | Disabled |
| SUT_CKSEL | Int. RC Osc.; Start-up time: 6 CK + 65 ms |

Make sure the following hex values appear in the bottom part of `Fuses` tab:

```
0xFE, 0x9D, 0x62.
```

If the node is to be programmed with the use of Serial Bootloader, enable additionally the BOOTRST option. Make sure the following hex value string appears at the bottom of Fuses tab:

```
0xFE, 0x9C, 0x62.
```

## 10.3 Pre-Built Images

The SDK comes with the following ready-to-use binary images that can be used on STK600 or RCB128RFA1 boards:

- `WSNDemoApp_Coord.hex`, `WSNDemoApp_Coord.srec` – for Coordinator node
- `WSNDemoApp_Router.hex`, `WSNDemoApp_Router.srec` – for Router nodes
- `WSNDemoApp_EndDev.hex`, `WSNDemoApp_EndDev.srec` – for End Device nodes

These `.hex` files should be loaded using JTAG, while `.srec` files can be flashed using Serial Bootloader. In all ready-to-use binary images MAC address (`CS_UID`) is loaded automatically from a dedicated external EEPROM chip, ensuring that unique values are assigned to all network nodes. Note that the default WSNDemo images are configured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and channel mask with only channel `0x0F` enabled for operation.

## 10.4 Running WSNDemo

### 10.4.1 Starting WSNDemo

To start WSNDemo, do the following:

- Setup the hardware as described in Section 10.1.2
- Install BitCloud SDK as described in Section 10.1.4
- Program one device with coordinator image file and other with either router or end device images as described in Section 10.2
- Connect the coordinator node to the PC, using serial interface
- Power on the coordinator node
- Run WSN Monitor (see Section 5.5)
- 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 of the development boards as described in Table 10-4. LED Dx label corresponds to RCB128RFA1 board, and color label to STK600 board.
- viewing the network information through WSN Monitor installed on PC (see Section 5.5).

**Table 10-4. LED indication for RCB128A1 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 |

## 10.5 Reserved Hardware Resources

**Table 10-5.** Hardware resources reserved by the stack on Atmega128RFA1

| Resource | Description |
|---|---|
| Processor main clock | 8 MHz 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 |

# 11 Appendix A-4: UC3 Specifics

## 11.1 Getting Started

### 11.1.1 Required Hardware

Before installing and using the BitCloud SDK for AVR32 UC3 make sure that all necessary hardware is available:

1. Two or more EVK1105 boards [17]
2. Two or more radio extender boards REB231 [16]
3. All necessary connectors
4. JTAGICE mkII

### 11.1.2 Hardware Setup

To prepare the hardware:

1. Install J12 and J16 extension headers on the board (if not already installed)
2. Install JTAG pin header on the board (if not already installed)
3. Use several 2-wire cables to connect J16 pins to corresponding pins on REB231 boards as indicated in Table 11-1.

**Table 11-1.** 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 |

### 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 UC3

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

| Parameter | Value | Note |
|---|---|---|
| JTAG emulator | JTAGICE mkII emulator with cable | Required to upload and debug firmware onto the boards through JTAG (see Section 5.2). |
| Operating system | Windows 2000/XP | |
| IDE | IAR Embedded Workbench AVR32 (with IAR C/C++ Compiler for AVR32 5.20.1[1] ) and AVR32 GNU Toolchain v2.3 | Required to upload firmware images through JTAG (see Section 5.2), and to develop applications using API (see Section 6.2). AVR32 GNU Toolchain is only needed to install USB VCP driver. |
| Java Virtual Machine | Java Runtime Environment (JRE) 5 Update 8, or later | Required to run WSNMonitor application |

Notes: 1. Users are strongly recommended to use specified versions of 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 with no blank spaces present in the directory path. As a result, the following SDK folders and files will be created.

**Table 11-3.** The SDK file structure

| Directory/File | Description |
|---|---|
| `./Documentation` | Documentation on BitCloud software |
| `./Evaluation Tools/WSNDemo (Embedded)` | Ready-to-use image files for evaluating WSNDemo. Refer to section 0 for the description of the images. |
| `./Evaluation Tools/WSNDemo (WSN Monitor)/` | Contains WSN Monitor installer |
| `./BitCloud/Components/` | Header files for BitCloud Stack |
| `./BitCloud/Components/BSP/` | Source, header and library files for BitCloud BSP |
| `./BitCloud/lib` | Library files for BitCloud Stack |
| `./Sample Applications/` | Source files for sample applications. |

2. Install IAR Embedded Workbench for AVR32 [20], if not already installed on your PC. Be sure to install only the supported version of IAR Embedded Workbench as specified in Table 11-2.
3. Install AVR32 GNU Toolchain [19], if not already installed on your PC.
4. Download and install Java Runtime Environment [12], if not already installed on your PC.
5. Install USB VCP driver on EVK1105 to allow it to communicate with your PC.
    a. Connect JTAG to UC3B JTAG header, and power on the board.

b. From `Third Party Software\EVK1105_UC3B_VCP` run `program_evk1105_at32uc3b-isp-cdc-1.0.1.cmd` Windows command script.

c. VCP driver should now be installed on the board.

6. Attach EVK1105 board to the USB port of your PC using 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 `Third Party Software` folder of the SDK.

## 11.2 Programming the Boards

### 11.2.1 Setting Parameters

For proper operation all nodes in ZigBee network shall have unique MAC address values. For UC3 a unique `CS_UID` parameter must be specified for each node in the application configuration (see details in Section 6.2.1), and then the application image must be built separately for each board.

### 11.2.2 Programming

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

7. Assemble board and connect it to PC:
8. Connect JTAG to UC3A JTAG header. Power on the board and JTAG ICE mkII.
9. Start IAR Embedded Workbench for AVR32
10. From `File -> Open -> Workspace` navigate to and open desired IAR project (e.g. `WSNDemoApp.eww` file in `Sample Applications\WSNDemo\iar\avr32` folder).
11. Select `Project -> Download and Debug`
12. Once the firmware is loaded, select `Debug -> Stop Debugging`
13. Unplug JTAG from UC3A JTAG header.
14. Reset EVK1105.

Alternatively it is possible to load ready image file in .elf format using AVR32 GNU Toolchain [19] by running following command in console:

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

Make sure the following fuse options in `JTAGICE mkII -> Fuse` handler menu of IAR Embedded Workbench for AVR32 are set.

**Table 11-4.** Fuse bits setting for AT32UC3A0512

| Option | Value |
|---|---|
| BODLEVEL | Brown-out detection at VCC=1.92 V (63) |
| BODHYST | Enabled (1) |
| BODEN | Disabled (3) |
| LOCK0 – LOCK15 | Unlocked (1) |
| EPFL | External instruction fetch enabled (1) |
| BOOTPROT | No bootloader (7) |

| Option | Value |
|---|---|
| GF29 | 1 |
| GF30 | 1 |
| GF31 | 1 |

## 11.3 Pre-Built Images

The SDK comes with ready-to-use binary images of WSNDemo application (`WSNDemoApp_*.hex`). There is a set of images for different roles and with different MAC addresses that can be used for creating a small network.

- `WSNDemoApp_Coord_0x01.elf` – for node acting as Coordinator
- `WSNDemoApp_Router_0x02.elf` – for node acting as Router
- `WSNDemoApp_EndDev_0x03.elf` – for node acting as End Device

The table below specifies MAC addresses pre-programmed in ready-to-use images:

**Table 11-5.** AT32UC3A0512 hosted on EVK1105

| Image Name | MAC address |
|---|---|
| WSNDemoApp_Coord_0x01.hex | 0x0000000000000001 (Coordinator) |
| WSNDemoApp_Router_0x02.hex | 0x0000000000000002 (Router) |
| WSNDemoApp_EndDev_0x03.hex | 0x0000000000000003 (End device) |

Note that the default images are configured to use Extended PAN ID `0xAAAAAAAAAAAAAAAA` and channel mask with only channel 0x0F enabled for operation.

## 11.4 Running WSNDemo

### 11.4.1 Starting WSNDemo

To start WSNDemo, do the following:

15. Setup the hardware as described in Section 11.1.2
16. Install BitCloud SDK as described in Section 11.1.4
17. Program one device with coordinator image file and other with either router or end device images as described in Section 11.2
18. Connect the coordinator node to the PC, using serial interface
19. Power on the coordinator node
20. Run WSN Monitor (see Section 5.5)
21. Power ON and reset the rest of the nodes.

### 11.4.2 Monitoring WSNDemo Activity

Network activity can be monitored in two ways:

- observing LEDs of the development boards as described in Table 11-6. LEDx label corresponds to EVK1105 board.
- viewing the network information through WSN Monitor installed on PC (see Section 5.5).

**Table 11-6. LED indication for WSNDemo application on 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 |

## 11.5 Reserved Hardware Resources

**Table 11-7.** Hardware resources reserved by the stack on AT32UC3A0512

| Resource | Description |
|---|---|
| Processor main clock | 48 MHz from external quartz |
| AVR32 ports A9, A20, A11, A12, A13 | Radio interface |
| Timer Channel 0 | Timer |
| AVR32 port B30, B31 | Sleep / reset |

# 12 Appendix B-1: 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 (e.g. those using 3rd party sensors not available on the standard evaluation boards and kits) to transfer sensor readings and have them visualized in the same WSN Monitor application.

## 12.1 Message Format

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

**Table 12-1.** 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:<br>0 – coordinator<br>1 – router<br>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 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 below in section 12.2 |

## 12.2 Additional fields

The message may contain zero, one, or more additional fields that follow the mandatory fixed-width fields described in the table above. 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 12-2.** Additional field format

| Sub-Field Name | Length | Description |
|---|---|---|

| 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 12-3.** Additional field types

| Field Type | Description |
|---|---|
| 0x01 | Sensors data for board type 1. Used for ATAVRRZRAVEN kit boards and MeshBean boards. |
| 0x20 | Node name. |

Please note that in the current version of WSNDemo devices send additional fields of type 0x01 (sensors readings for boards of type 1) only. Unrecognized additional fields are discarded by WSN Monitor application. The Field Data format for different field types are described in the following tables.

**Table 12-4.** Field Data for type 0x01: Sensors data for board 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 12-5.** Field Data for type 0x20: Node name

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

# 13 Appendix B-2: Serial Protocol

This appendix describes the protocol and message format used over the serial connection between the network coordinator and the WSN Monitor application running on the PC. The messages sent on the serial connection are basically the messages defined in section 8.1 wrapped as defined below:

**Table 13-1.** Serial message format

| Offset | Length | Description |
|---|---|---|
| 0 | 2 bytes | Start sequence: 0x10 0x02 |
| 2 | N bytes | Variable-length payload: the message received from end-node or router or generated by the coordinator, in the format described in section 8<br>All 0x10 bytes in this payload are duplicated to avoid confusion with 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 |

## Headquarters


***Atmel Corporation***
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600


## International


***Atmel Asia***
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

***Atmel Europe***
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

***Atmel Japan***
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581


## Product Contact

| ***Web Site*** | ***Technical Support*** | ***Sales Contact*** |
|---|---|---|
| http://www.atmel.com/ | avr@atmel.com | www.atmel.com/contacts |

***Literature Request***
www.atmel.com/literature