
ZigBit Development Kits

ATZB-EVB-24-SMA, ATZB-EVB-900-SMA,

ATZB-EVB-A24-SMA, ATZB-EVB-24-PCB,

ATZB-EVB-24-A2

User Manual



**ZigBit
Development Kits**

User Manual

Rev. [0.5]-AVR-[09/11]





1 Introduction

Intended Audience and Purpose

This document is intended for engineers and software developers working with the ZigBit™ Development Kit (ZDK). The Kit is used to evaluate the performance and features of ZigBit modules and the BitCloud software, and to implement custom applications on top of BitCloud API.

Safety and Precautions

The product contains electronics, which are electrically sensitive. Please take necessary precautions when using such devices. Atmel does its best to protect the product components from electrostatic discharge phenomena, but we encourage our users to follow common guidelines to avoid electrostatics by using proper grounding etc.

The product complies with the FCC (Part 15), IC and ETSI (CE) rules applicable to the devices radiating in the uncontrolled environment. Please find out if the product complies with your local regulations.

Any modifications of the hardware, its components or improper use of the product can cause an uncontrolled violation of the in-band or out-band radiation levels. It can result in progressing violation of emission level limits, thus causing harmful interference.

Precautions

The product radiates power in the microwave band. Although the levels are considered to be low (less than 2 mW), it is reasonable to protect the operating personnel from possible harmful impact of the electromagnetic field. When the parts of the product are turned on, an operator should avoid touching the PCB antenna and the board itself. The recommended distance between an operator and antenna should be more than 20 centimeters.

AC/DC adapters which can be used with the product contain high voltage circuits. General precautions should be taken against electric shock before the product hardware is mains powered.

The ZigBit Development Kit contains fragile components. Please handle with care.

Related documents

- [1] ZigBit™ OEM Modules. Product Datasheet. Atmel Doc. M-251~01
- [2] BitCloud™ IEEE802.15.4/ZigBee Software. Product Datasheet. Atmel Doc. M-251~08
- [3] BitCloud™ Software 1.0. SerialNet. Reference Manual. AT-Command Set. Atmel Doc. P-ZBN-452~03
- [4] BitCloud™ Software 1.0. BitCloud Stack Documentation. Atmel Doc. P-ZBN-452~02
- [5] BitCloud Developer's Guide. Atmel Doc. P-ZBN-452~01



- [6] ZigBit™ OEM Module. Application Note. ZigBit Power Consumption Testing. Atmel Doc. AN-481~01
- [7] BitCloud™ Software 1.0. Serial Bootloader User's Guide. Atmel Doc. P-ZBN-451~02
- [8] ZigBit™ OEM Module. Application Note. Using ZigBit Module with Analog Sensors. Atmel Doc. AN-481~06
- [9] BitCloud™ Software 1.0. Range Measurement Tool User's Guide. Atmel Doc. P-ZBN-451~01
- [10] ZigBee Specification. ZigBee Document 053474r17, October 19, 2007
- [11] Serial asynchronous automatic dialing and control. ITU-T Recommendation V.250, 05/99
- [12] IEEE Std 802.15.4-2003 IEEE Standard for Information technology – Part 15.4 Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)
- [13] TSL2550 Ambient Light Sensor With Smbus Interface. TAOS Datasheet TAOS029E. February 2006
<http://www.taosinc.com/images/product/document/tsl2550-e67.pdf>
- [14] LM73 2.7V, SOT-23, 11-to-14 Bit Digital Temperature Sensor with 2-Wire Interface. National Semiconductor Corporation Datasheet DS201478. July 2006
<http://www.national.com/pf/LM/LM73.html#Datasheet>
- [15] CP2102, Single-Chip USB to UART Bridge, Rev. 1.1 9/05.
www.silabs.com
- [16] AVR Studio. User Guide. Available in HTML Help within the product.
- [17] JTAGICE mkII Quick Start Guide.
http://www.atmel.com/dyn/resources/prod_documents/doc2562.pdf
- [18] avr-libc Reference Manual 1.4.3
- [19] WinAVR User Manual – 20070525/ By Eric B. Weddington
- [20] Using the GNU Compiler Collection/ By Richard M. Stallman and the GCC Developer Community



Abbreviations and Acronyms

AC/DC	Alternating Current / Direct Current converter
ADC	Analog-to-Digital Converter
API	Application Programming Interface
Channel Mask	Channel mask is a number that defines the set of working channels
Coordinator	Within ZigBee networks, the ZigBee coordinator is responsible for starting the network and for choosing certain key network parameters. The network may be extended through the use of ZigBee router.
DIP	Dual In-line Package
EEPROM	Electrically Erasable Programmable Read-Only Memory
End device	In ZigBee networks, the ZigBee end device provides sensor data sent to a router. End device is often subject to power management restrictions, so it may be in sleeping mode most of the time.
ESD	Electrostatic Discharge
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
I ² C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IRQ	Interrupt Request
JTAG	Digital interface for debugging of embedded devices, also known as IEEE 1149.1 standard interface
LED	Light Emitting Diode
LQI	Link Quality Indicator
MAC	Medium Access Control layer
MCU	Microcontroller Unit. In this document, it also means the processor, which is the core of ZigBit Amp module
MIPS	Million Instructions per Second
NWK	Network layer



OEM	Original Equipment Manufacturer
OTA	Over-The-Air upgrade
PAN ID	Personal Area Network Identifier. In ZigBee, it is 16-bit number which must be unique for each one of multiple networks working on the same frequency channel
PCB	Printed Circuit Board
PHY	Physical layer
RAM	Random Access Memory
RF	Radio Frequency
RISC	Reduced Instruction Set Computing microprocessor
Router	In ZigBee networks, routers transfer data and control messages through the network using a hierarchical routing strategy. The ZigBee coordinator is also responsible for routing.
RP-SMA	Reversed Polarity Surface Mount Assembly
RS-232	Serial binary data interconnection interface, which is commonly used in computer serial ports (COM ports)
RSSI	Received Signal Strength Indicator
RTS/CTS	Request to Send / Clear to Send
RX	Receiver
SMA	Surface Mount Assembly
SPI	Serial Peripheral Interface bus
TTM	Time To Market
TX	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
UID	Unique Identifier
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VCP	Virtual Com Port
WSN	Wireless Sensor Network
ZDK	ZigBit Development Kit
ZigBee, ZigBee PRO	Wireless networking standards targeted at low-power sensor applications [10]



802.15.4

The IEEE 802.15.4-2003 standard applicable to low-rate wireless Personal Area Networks [12]

2 Development Kit Overview

ZigBit™ Development Kit (ZDK) is a simple, out-of-the-box solution designed for full range of WSN prototyping and development. It comes complete with ATZB-EVB (MeshBean) development boards containing ZigBit modules and a variety of tools to test the wireless network features and performance and to develop customized wireless solutions based on BitCloud Software.

ZigBit Development Kit includes:

1. Any one type of ATZB-EVB (MeshBean) board (1 item) – ATZB-EVB-24-SMA/ ATZB-EVB-900-SMA/ ATZB-EVB-24-A2/ ATZB-EVB-A24-SMA/ ATZB-EVB-24-PCB
2. USB 2.0 A/mini-B cable (3 items)
3. Swivel antenna (1/2 wave antenna)
4. Software & Documentation Distribution CD (1 item).

3 Hardware General Specifications

ATZB-EVB (MeshBean) board is intended to evaluate the performance of a ZigBit module. In turn, a ZigBit module with the embedded BitCloud software provides wireless connectivity for ATZB-EVB (MeshBean) board, enabling it as a node in a ZigBee network. MeshBeans also serve as a reference hardware platform for the customer's target devices utilizing ZigBit modules for wireless communication.

The ATZB-EVB (MeshBean) board can be configured to operate as a network coordinator, a router or an end device, by setting of DIP switches (see Section 0) and/or sending AT-commands. The node's role is defined by the embedded application.

The boards are delivered with ZigBit preprogrammed with Serial Bootloader and WSN Demo application firmware. For full list of demo applications see Section 3.2. Gerber files are available with Complete Support Package only. The ATZB-EVB (MeshBean) basic parameters are presented in Table 1.

Table 1. ATZB-EVB (MeshBean) Board Specifications

Parameter	Value
RF	
Compliance	2.4 GHz IEEE 802.15.4-2003 [12]
Operating Band	2400–2483.5 MHz
TX Output Power	from -17 dBm to +3 dBm
RX Sensitivity	-101 dBm
RF Transceiver	AT86RF230
Antenna versions	2.4 GHz (PCB on-board antenna, external 50 Ohm unbalanced antenna or dual chip antenna)
MCU	
Microcontroller	ATmega1281V
RAM	8K Bytes
Flash Memory	128K Bytes
EEPROM	4K Bytes
Performance	Up to 4 MIPS throughput at 4 MHz Clock
Power	
Power Supply	Dual AA type Battery, automatically switched to USB or AC/DC adapter
Over-Voltage Protection	Yes
Reverse Polarity Protection	Yes
Operating Voltage Range	1.8...3.6 V
Voltage Supervisor	Yes
Miscellaneous	
Sensors	Digital: Ambient Light/ Ambient Air Temperature



LED Indicators	3 programmable color status LEDs external power supply status LED
Switches	3 DIP switches
Buttons	2 programmable buttons
Size	60 x 63 x 24[1] mm
Operating Temperature Range	-40°C to 85°C. Minor degradation of clock stability may occur beyond the -20°C to +70°C range.

3.1 ATZB-EVB (MeshBean) Featured Components

ZigBit Module

ZigBit module is an ultra-compact, low-power, high sensitivity 2.4GHz 802.15.4/ZigBee OEM module from Atmel. ZigBit module is based on Atmel's Z-Link 2.4GHz platform. It includes ATmega1281V Microcontroller and AT86RF230 RF Transceiver.

In ZDK, every ZigBit module is delivered installed on a ATZB-EVB (MeshBean) board. Two different versions of ZigBit modules are available [1]: a version with balanced RF port for applications where the benefits of PCB or external antenna can be utilized, and a version with dual chip antenna satisfying the needs of size-sensitive applications.

Detailed specifications of the ZigBit module, including the module interfaces, voltage levels, power consumption, are available in the ZigBit datasheet [1].

Sensors

The board incorporates light sensor TSL2550T from TAOS and temperature sensor LM73CIMK from National Semiconductors. Both sensors are connected in parallel to the I²C bus. For more information on the sensors see their datasheets [13], [14] available from the corresponding manufacturers' websites.

NOTE:

In addition to the built-in, onboard sensors, external sensors selected by developer can be used. An external sensor can be connected to the terminals of External interface cable leading to the onboard Expansion slot. See the corresponding pinout in Table 2. As an example, connection of an external sensor is illustrated in Application Note [7].

USB to UART Bridge

CP2102, the USB to UART Bridge controller from Silicon Labs [15], is installed on the board. It provides seamless USB interface to any RS-232 legacy device. If the controller's driver has been installed on PC during the deployment of the whole Development Kit (see Section 4.3) the onboard USB port is visible on the PC as generic COM port with a particular number.

Silicon Serial for UID storage



UID (Unique Identifier) is HEX value, 8 bytes long. UID is used for setting unique MAC address of the node.

UID is hardware defined value. It is programmed into a chip (Silicon Serial Number DS2411R+ by Maxim/Dallas) at the factory.

UID is unique, and cannot be overwritten. In order to ensure the presence of UID on the board it is required to execute Hardware Test application (see Section 4.7 for details).

ATZB-EVB (MeshBean) Board Design

The ATZB-EVB (MeshBean) board contains the ZigBit module, which operates as ZigBee/802.15.4 transceiver. It also includes sensors, buttons, DIP switches, and a set of interfaces.

The board provides the following interfaces:

- USB 2.0 port
- Light and temperature sensors
- 2 push buttons controlling the software
- Reset button
- 3 DIP switches
- 3 software-controlled LEDs
- Symmetrical dipole PCB antenna (only for ATZB-EVB (MeshBean) with PCB antenna)
- SMA connector (only for ATZB-EVB (MeshBean) with external antenna)
- JTAG connector for software upload and debugging
- Power connector (3 V) to use an AC/DC adapter (not supplied with ZDK)
- 20-pin Expansion slot containing external ZigBit's interfaces (see Table), including:
 - Serial port interface (RS-232)
 - USART
 - Buffered I²C interface with ESD protection and voltage level translation
 - ADC/GPIO
- Battery compartment for AA-size batteries
- 3 configuration jumpers
- 3 clamps for power consumption measurements.

Also, the board contains an internal voltage regulator to supply most of the components with 3.6 V. This is needed if ZigBit's MCU is to be run at 8 MHz¹.

NOTE:

Normally ZigBit module is powered directly by the batteries, USB or AC/DC adapter (via protection circuitry); however, Jumper J2 (see Table) can switch ZigBit to 3.6 V supply.

See Figure 1 for the layout of ATZB-EVB (MeshBean) version with integrated PCB antenna and the board's functional diagram.

¹ 8MHz requires changes in the BitCloud Software that normally runs at 4 MHz in order to extend the voltage range and decrease power consumption.

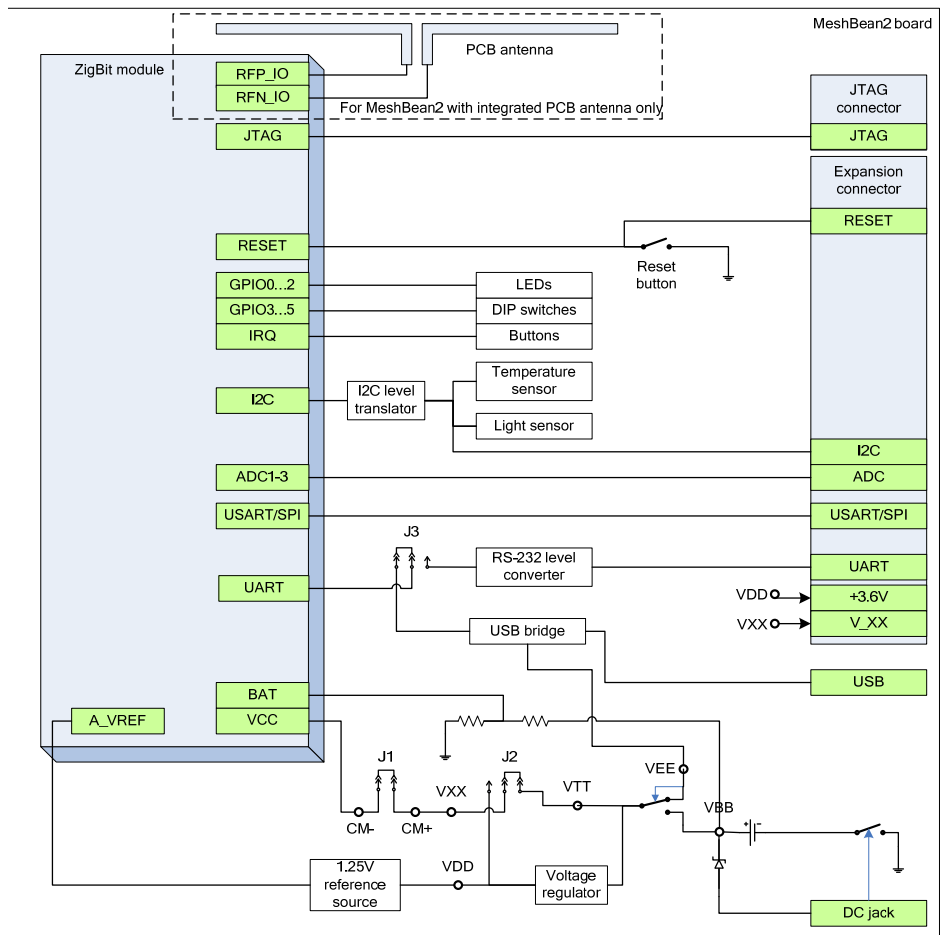
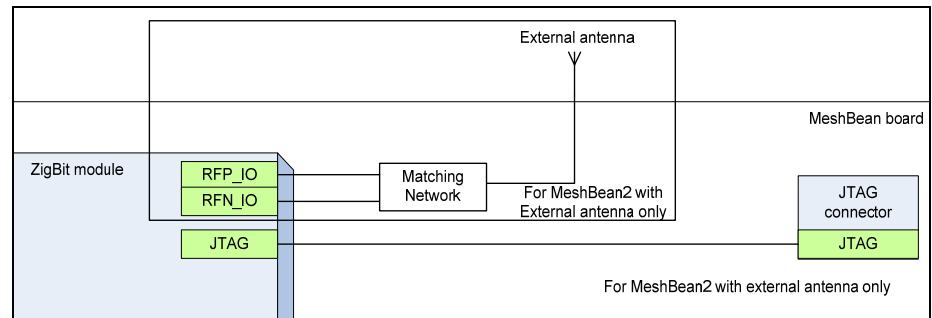


Figure 1. ATZB-EVB (MeshBean) functional diagram



Connectors and Jumpers

The board connector pinouts and jumper settings are presented in Table 2 through Table 7.

IMPORTANT NOTE:

All manipulations with connectors or jumpers should be done when the board is not powered!

Table 2. Expansion slot pinout

Pin	Name	I/O	Description
1	UART_RTS	Input	Request to Send Pin. RS-232 level.
2	UART_TXD	Input	Transmit Data Pin (meaning that the host device will transmit data to this line). RS-232 level.
3	UART_CTS	Output	Clear To Send signal from the module. Active low. RS-232 level.
4	UART_RXD	Output	Receive Data Pin (meaning that the host device will receive data from this line). RS-232 level.
5	GND		Digital/analog ground
6	GND		Digital/analog ground
7	I2C_CLK	Input	I ² C clock. It is connected to the I2C_CLK pin of the module via low-voltage level translators. For details, refer to ZigBit datasheet [1].
8	I2C_DATA	Bidirectional	I ² C data. It is connected to the I2C_DATA pin of the module via low-voltage level translators. For details, refer to ZigBit datasheet [1].
9	+3.6V	Output	Output of internal voltage regulator. Normally, the voltage is 3.6 V.
10	V_XX	Output	ZigBit supply voltage
11	RESET	Input	Reset Pin. Active low. This pin is connected in parallel to the RESET button on the board.
12	USART_TXD	Output	This is Transmit Data Pin for USART0 interface of the ZigBit module. It is connected directly to the USART0_TXD pin of the module. Digital logic level. For details, refer to ZigBit datasheet [1].[1]
13	USART_RXD	Input	This is Receive Data Pin for USART0 interface of the ZigBit module. It is connected directly to the USART0_RXD pin of the module. Digital logic level. For details, refer to ZigBit datasheet [1].



Pin	Name	I/O	Description
14	USART_CLK	Input	This is Clock Data Pin for USART0 interface of the ZigBit module. It is connected directly to the USART0_EXTCLK pin of the module. Digital logic level. For details, refer to ZigBit datasheet [1].
15	GND		Digital/analog ground
16	ADC_INPUT1	Input	ADC input. This pin is connected directly to the ADC_INPUT_1 pin of the module. For details, refer to ZigBit datasheet [1].
17	ADC_INPUT2	Input	ADC input. This pin is connected directly to the ADC_INPUT_2 pin of the module. For details, refer to ZigBit datasheet [1].
18	ADC_INPUT3	Input	ADC input. This pin is connected directly to the ADC_INPUT_3 pin of the module. For details, refer to ZigBit datasheet [1].
19	GND		Digital/analog ground
20	GND		Digital/analog ground

GENERAL NOTES:

Pins 12, 13, 14, 16, 17, 18 are not buffered and driven by the MCU pins directly. Thus this interface should be used with precautions at the low supply voltages to avoid damaging the module.

Pins 7 and 8 are connected via voltage level translators with ESD protection. Thus these pins can be used easily to connect extra I²C sensors without extra logic.

Voltage on the V_{XX} pin does not depend on the state of jumper J1 or ammeter connection between clamps CM+, CM-.

Table 3. JTAG connector pinout

Pin	Name	Description
1	JTAG_TCK	Scan clock
2	JTAG_GND	Digital ground
3	JTAG_TDO	Test data output
4	JTAG_VCC	Controller supply voltage
5	JTAG_TMS	Test mode select
6	JTAG_RST	Reset controller; active low
7	N_Cont	Not connected
8	N_Cont	Not connected
9	JTAG_TDI	Test data input
10	JTAG_GND	Digital ground



NOTE:

JTAG connector pinout is compatible with ATmega JTAGICE mkII in-circuit emulator connector.

Table 4. J1 jumper settings: current measurement

Jumper position	Description
J1 is mounted	This position is used for normal operation.
J1 is open	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 (see Section 3.8).4.8

Table 5. J2 jumper settings: ZigBit 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 6. J3 jumper settings: 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.

IMPORTANT NOTES:

Any other position of jumpers J2 and J3 or their omission may cause permanent damage of the hardware.

Powering the board without J1 jumper and ammeter connection between clamps CM+ and CM- may cause a permanent damage of the hardware.

When making connection to the PC's serial port through the Expansion slot consider the pinout as indicated below in Table .



Table 7. Serial interface pinout

Signal	Expansion slot pins	Serial port pins (PC side)
RXD	4 : UART_RXD	2
TXD	2 : UART_TXD	3
CTS	3 : UART_CTS	8
RTS	1 : UART_RTS	7
GND	5, 6, 15, 19, 20 : GND	5

Buttons, Switches and LEDs

The board includes 2 buttons, 3 DIP switches, one Reset button that generates a hardware reset signal, 3 software-defined LEDs (green, yellow and red) and a LED indicating powering the board from the USB. Any of onboard buttons, DIP switches and LEDs can be controlled by an embedded application running on a ZigBit.

For instance, the status of any DIP switch will be ignored when running SerialNet (see Section 6). DIP switches can be tested when running the Hardware Test application (see Section 4.7).

External Antenna

Among the ATZB-EVB (MeshBean) boards delivered with ZDK, one particular board is equipped with SMA connector to attach an external antenna. The specifications of external antenna supplied with the Development Kit are presented below in Table .

Table 8. External antenna specifications

Part Number	Manufacturer & Description	Gain, dBi	Impedance, Ohm	Min. Separation, cm
17010.1	WiMo, swivel antenna (1/2 wave antenna) with SMA connector, frequency range 2.35-2.5 GHz	2.1	50	20

IMPORTANT NOTE:

Only the Non-RP SMA Connector antennas can be used in combination with MeshBean boards equipped with an SMA connector!

Attach the external antenna through SMA connector before using the board. Match the antenna's male coaxial thread with the connector's female thread.

3.2 BitCloud Software

BitCloud is a full-featured, next generation embedded software stack from Atmel. The stack provides a software development platform for reliable, scalable, and secure wireless applications running on Atmel ZigBit modules. BitCloud is designed to support a broad ecosystem of user-designed applications addressing diverse requirements and enabling a full spectrum of software customization.

BitCloud is fully compliant with ZigBee PRO and ZigBee standards for wireless sensing and control. It provides an augmented set of APIs which, while maintaining 100% compliance with the standard, offer extended functionality designed with developer's convenience and ease-of-use in mind.

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 (start, reset, formation, join). ZDO also defines ZigBee Device Profile types, device and service discovery commands implemented by the stack.

This ZDK provides everything a developer would need to develop custom applications based on the BitCloud API [4]. The general guidelines to BitCloud programming are given in [5]. API-based demos are provided in source code which can be modified and extended, making it possible to develop WSN applications for a variety of networking scenarios. For example, an end device can be configured to communicate with a router between the periods of sleep thus saving power.

Another configuration of BitCloud Software, SerialNet, enables a user to implement customized WSN scenarios without developing any WSN application code. In this case WSN nodes are controlled via AT-commands (see Section 6).

The structure of BitCloud Software is presented in Figure 2. It is detailed in datasheet [2].

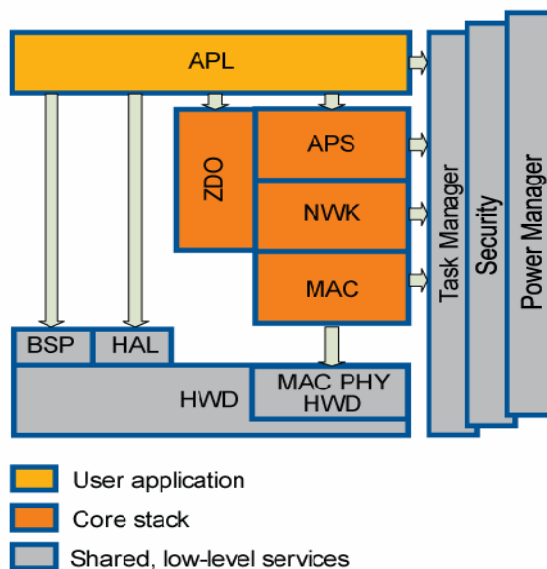


Figure 2. BitCloud Block Diagram

The Development Kit includes two kinds of applications (see 10.7). Evaluation tools are delivered in binary format. Sample applications are available in source code.

The following evaluation tools are delivered:



- **SerialNet** application lets the AT-commands be interpreted locally or forwarded for execution on remote nodes;
- **Hardware Test** (see Section 4.7) is a simple application which tests major ATZB-EVB (MeshBean) board components for correct operation;
- **Range Measurement Tool** is an application intended to measure radio performance of ZigBit-based devices and/or to make comparison with platforms of other manufacturers. Usage essential information and guiding instructions are given in [9];
- **WSNDemo** with WSN Monitor;
- **Serial Bootloader** is a software utility designed to program an application code into WSN nodes through USB or serial port, without using JTAG. See the description of Serial Bootloader in Section 7.

The following sample applications are delivered with source code (referenced as given in brackets):

- WSNdemo (WSNDemo)
- Low Power Networking (Lowpower)
- Ping-Pong (Pingpong)
- Peer-To-Peer Data Exchange (Peer2peer)
- Blink minimal sample application (Blink)
- Hardware Test (HardwareTest).

The WSNdemo application is a featured ZDK application demonstrating the WSN performance. It is presented in details in Section 5. The source code for WSNdemo is available with Complete Support Package only.

The rest of programs are sample implementations triggering common BitCloud APIs. Blink is a minimal application (see Section 8.3). Low Power Ping-Pong, and Peer-To-Peer applications are introduced in Section 8.4.

4 Getting Started

4.1 Overview

This section describes the system requirements and ZDK deployment. It also provides how-to instructions on handling the boards, testing WSN functionality and performing local hardware tests.

4.2 System Requirements

Before using the Kit, please ensure that the following system requirements are met (see Table 9).

Table 9. System requirements

Parameter	Value	Note
PC		
CPU	Intel Pentium III or higher, 800 MHz	
RAM	128 MB	



Hard disk free space	50 MB	
JTAG emulator	JTAGICE mkII emulator with cable	Necessary to upload and debug firmware onto the ATZB-EVB (MeshBean) board through JTAG (see Appendix B).
Software		
Operating system	Windows2000/XP	
USB driver	CP210x USB to UART Bridge VCP Driver	Necessary to connect ATZB-EVB (MeshBean) to PC via USB port (see Section 3.4)4.4
IDE	AVR Studio 4.14 + WinAVR	Necessary to upload firmware image through JTAG (see Appendix B), and to develop applications using API (see Section 7)
Serial Bootloader utility		Necessary to upload firmware image without using JTAG (see Section 6)
Java virtual machine	Java Runtime Environment (JRE) 5 Update 8, or more recent	Necessary to run the WSN Monitor application (see Section 4.5)5.6
Microsoft .NET framework	Version 2.0 Service Pack 1, or more recent	Necessary to run the GUI Bootloader (see Section 6)



4.3 Installing the Development Kit

In order to install the Development Kit, insert the ZDK Software and Documentation CD into your PC CDROM. The ZDK installation wizard should start automatically. Specify the installation path and follow the instructions.

As a result the ZDK file structure under the selected path will be generated on the PC, which is described in 10.7.

During the ZDK deployment the following auxiliary software can be optionally installed:

- USB to UART Bridge VCP driver for Windows platform
- Java Runtime Environment (JRE).
- Microsoft .NET Framework.

To complete the installation of VCP driver before use of the ZDK do the following:

- Connect ATZB-EVB (MeshBean) board to the USB port. Windows should detect the new hardware. Follow the instructions provided by the driver installation wizard.
- Make sure that the driver is installed successfully and the new COM port is present in the device list. Open the Device Manager window shown in Figure 3:
Start/Control Panel/System/Hardware/Device Manager.

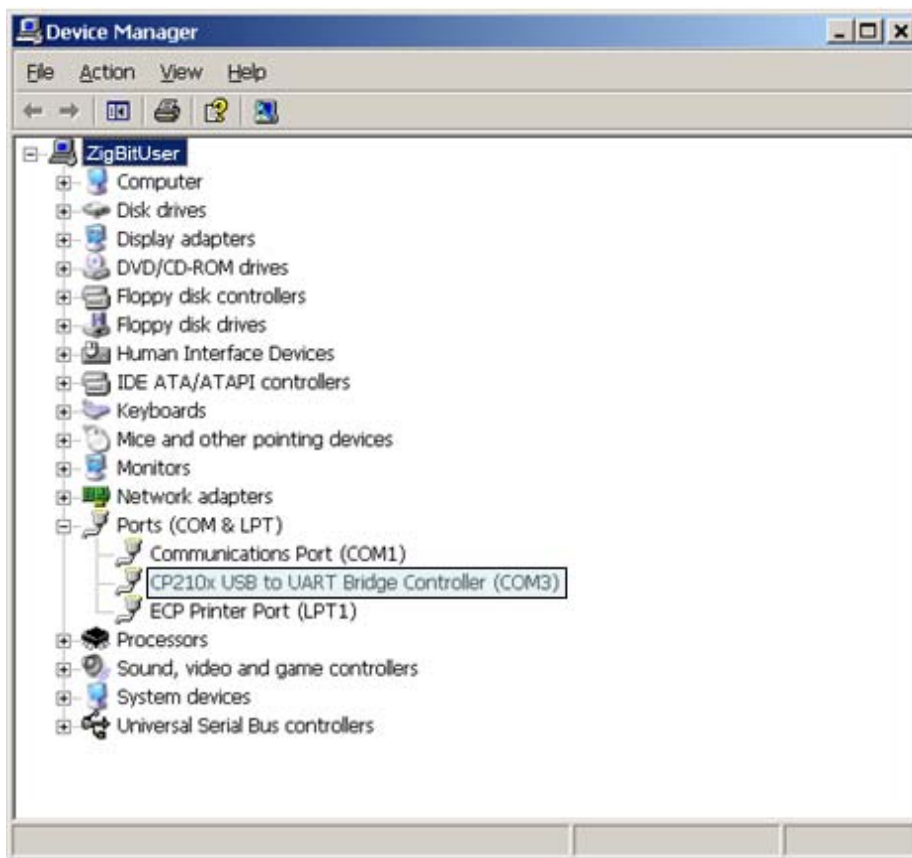


Figure 3. COM port drivers in the Windows Device Manager window

To resolve possible problems see Section 4.4.

NOTE:

USB to UART Bridge VCP driver for Windows platform is also available from the manufacturer's site:

http://www.silabs.com/tgwWebApp/public/web_content/products/Microcontrollers/USB/en/mcu_vcp.htm.

Java Runtime Environment (JRE) is also available from <http://java.sun.com/javase/downloads/index.jsp>.

There may be other Java instances already installed on your computer before the use of ZDK. To avoid confusion, edit `start.bat` file in `/Evaluation Tools/WSNDemo (WSN Monitor)` subdirectory containing the WSN Monitor. Make sure to provide full path to the Java executable file, specify its file name extension (`.exe`) explicitly.



Current version of the AVR Studio [16] with Service Pack can be freely downloaded from the Atmel's website (<http://www.atmel.com>). Simply launch the downloaded installer programs and follow the setup instructions.

The WinAVR suite of development tools can be downloaded from <http://sourceforge.net/projects/winavr>. To install WinAVR follow the setup instructions.

4.4 Connecting the Board to PC

The board can be connected to host PC via USB port, using USB 2.0 A/mini-B cable supplied within the Kit. USB is typical connection. Furthermore, it provides the convenient possibility to link multiple boards to a single PC. Besides, no battery is required once a board is powered via USB.

Because wireless applications usually employ host connection through COM port, linking the onboard USB to UART Bridge controller to PC requires installation of the USB to UART Bridge VCP driver (see details in Section 4.3, Section 0). As a result, generic COM port can be used to access a board via USB.

IMPORTANT NOTES:

When USB connection is used, the COM port number would be changed by the Windows operating system if the board has been reconnected. To avoid confusion use Windows Control Panel to check on the actual port number.

Under some circumstances, the boards can conflict with other USB devices recently installed. In such cases, the Windows Device Manager would show a problem occurred during the plug-and-play procedure or it would not detect the USB to UART Bridge controller at all. Possible solution is to change the USB ID for the board, using special utility available from the controller's manufacturer. See Section 9 for details.

Alternatively, the board can be connected to host PC via serial port, using a serial cable (not provided with ZDK). Serial port pinout is presented in Table .

IMPORTANT NOTE:

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 J3 (see Table).



4.5 Powering the Boards

The boards can be powered by a pair of AA-size batteries, via the USB port, once connected for data transfer, or via AC/DC adaptor. The nominal voltage is 3 V. Using AC/DC adaptor disconnects AA batteries automatically. Using USB port disconnects the AC/DC adaptor.

In order to make accurate measurements of sensor parameters, battery power is recommended. USB power is not stable enough, which can affect transmission of power level or RF parameters.

IMPORTANT NOTES:

It is strongly recommended to check up the power supply voltage before programming the boards by Serial Bootloader or by JTAG. Power drops happened during the programming process could result in an inoperable state of the ZigBit or its permanent damage.

Using the discharged batteries (when the voltage is below the specified limit) may cause damage of flash memory or EEPROM as well. If that happened, programming by means of Serial Bootloader would fail. In this case the only option becoming available would be using of JTAG emulator (see 10.8).

Using nickel-cadmium rechargeable batteries is allowed but with certain precautions. Nominally, their cell potential is 1.2 V. Although a pair gives 2.4 V thus fitting the operating voltage range (see Section 3), it is still lower than 3 V level, which a pair of the most popular alkaline cells give. Hence, nickel-cadmium rechargeable batteries could not be a proper alternative of the alkaline cells for all applications.

4.6 Testing WSN Functionality Using SerialNet

Program the boards with the SerialNet firmware (see Section 6).

Connect the board to PC (see Section 4.4).

Run standard Hyper Terminal utility which is a part of Windows 2000/XP:
Start/Programs/Accessories/Communications/HyperTerminal.

Select logical value for COM port provided by the system (see Section 4.4). COM port parameters should be set to the values given in Table 10.

Table 10. COM port settings for hardware testing

Option	Value
Data Rate	38 400 bps
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None, unless data transmission between the boards is planned; then, Hardware flow control option should be selected



Type the “AT” command and press `Enter` key.

The board responds to Hyper Terminal with “OK”.

Now, a user can play various networking scenarios by sending AT commands fully described in [3].

A simple networking scenario for building WSN, transmitting data between the WSN nodes and accessing the nodes’ interfaces is presented in the `Examples` Section of the document [3].

4.7 Testing the Board Controls and Sensors

To check the onboard controls and sensors the Hardware Test application can be used.

Connect the board to the PC.

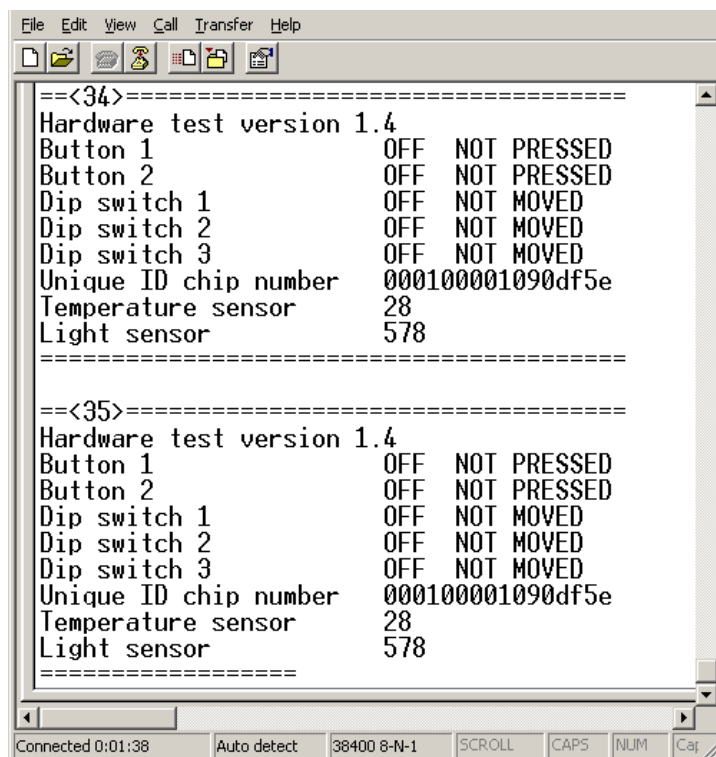
Upload Hardware Test image onto the boards. The Hardware Test image files are listed in 10.7.

Run Hyper Terminal utility in the same manner as described above (see Table).

While the Hardware Test is running, all the board LEDs are blinking. Reports are generated each second (see Figure), and include the status of buttons, DIP switches, the UID chip number and sensor readings. To test the hardware, you can perform simple manipulations with the board: press the buttons, move the DIP switches, manually hide the light sensor from light, finger the temperature sensor and so on. You should see the changes in parameters reported through Hyper Terminal (see Figure 4).

NOTE:

During the test, if you reconnect the board to USB or power off the board, the operating system would arbitrarily switch this particular USB connection to another COM port. Apparently, Hyper Terminal does not recognize such changes. If this happens, you have to reconnect Hyper Terminal to a proper port. Simply select `File/New Connection` menu item and repeat the connection procedure.



```

File Edit View Call Transfer Help
=====
==<34>====
Hardware test version 1.4
Button 1          OFF NOT PRESSED
Button 2          OFF NOT PRESSED
Dip switch 1     OFF NOT MOVED
Dip switch 2     OFF NOT MOVED
Dip switch 3     OFF NOT MOVED
Unique ID chip number 000100001090df5e
Temperature sensor 28
Light sensor      578
=====

==<35>====
Hardware test version 1.4
Button 1          OFF NOT PRESSED
Button 2          OFF NOT PRESSED
Dip switch 1     OFF NOT MOVED
Dip switch 2     OFF NOT MOVED
Dip switch 3     OFF NOT MOVED
Unique ID chip number 000100001090df5e
Temperature sensor 28
Light sensor      578
=====

Connected 0:01:38  Auto detect  38400 8-N-1  SCROLL  CAPS  NUM  Car

```

Figure 4. Hyper Terminal Hardware Test report

4.8 Measuring Power Consumption

The board allows measuring power consumption of the ZigBit module. To perform the measurements, simply connect ammeter to the clamps denoted as CM+ and CM- and remove jumper J1. Make sure that the board is powered by batteries only. However, such measurement would not be absolutely correct, because power is consumed by the interfaces and the peripherals connected to ZigBit. To measure power consumption correctly, all interfaces should be disconnected from ZigBit module, excluding RF ports. Refer to the Application Note [6] for details.

4.9 Antenna Precautions

Each type of antenna – namely, PCB antenna, dual chip antenna and external antenna – were matched and tuned, with taking into account all adjacent components, including the ZigBit module shield, battery compartment and plastic legs. Any object approached or put closely next to antenna affects its performance. Do not put the module into enclosure. Do not mount the board on metal surface. Do not use metal screws over 5 mm long to fasten the board legs. These factors would affect antenna performance.

Mount plastic legs from bottom side only, next to the battery compartment). Use plastic screws to fasten the legs. Do not use the legs made of different plastic composite. Omitting these plastic legs would significantly affect antenna performance.



The pattern of antenna is wide. The following facts should be considered. In far-field zone, it is a horizontal plane normal to the dipoles where electromagnetic radiation appears stronger. Contrarily, at distances of several centimeters the pattern is more complex. Approximate field patterns are given in the ZigBit datasheet [1].

Handle the external antenna with care to avoid mechanical damage.

5 WSN Demo Application

5.1 Overview

The network performance of ZigBit platform is demonstrated with the WSN Demo application which is based on the BitCloud software API. This application comprises the embedded firmware, which supports functions for coordinator, router and end device, and the GUI part – the WSN Monitor which is run on a PC.

Thanks to the WSN Demo application, the ATZB-EVB (MeshBean) boards are organized into a set of nodes constituting a wireless network. The LEDs of a board indicate the board current state and activities. End devices and routers read from the onboard sensors and send the readings to coordinator in packets. End devices also follow a duty cycle, waking up occasionally to transmit the sensor data. That data is displayed on WSN Monitor panes as temperature, light and battery level measurements.

End device is mostly sleeping, and it wakes up shortly each 10 seconds for activities. During the sleep period, you can force end device to wake up by pressing the SW1 button. Router sends data each 1 second. Using UART, the coordinator transmits the received packets, along with its own sensor data, to the GUI application (WSN Monitor).

The WSN Monitor visualizes the network topology in a form of tree. It also displays the node parameters like node addresses, node sensor information and node link quality data.

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

In regard to the WSN Demo, Section 5.4 describes how to use the boards. GUI is described in Section 5.6. Further instructions are given in Section 5.7.

The application is delivered with source code included (see 10.7). It is implemented on top of the BitCloud API and it can be modified as described in Section 8.

With WSN Demo, the number of routers and/or end devices used is limited only by the network parameters described in Section 8.4.



5.2 Programming the Boards

First, WSNDemo image should be loaded onto the board. The location of WSNDemo image file is listed in 10.7.

WSNDemo image file can be uploaded into the boards in one of two ways: by means of Serial Bootloader utility (see Section 5.3) or in AVR Studio, using JTAG emulator. JTAGICE mkII from Atmel [17]² (see Section 5.3.1) is recommended.

IMPORTANT NOTE:

Be careful selecting the method of the node programming. Each of ATZB-EVB (MeshBean) boards come with the bootstrap uploaded onto the ZigBit's MCU, which is needed to run Serial Bootloader. If JTAG had been used, this would make Serial Bootloader useless until bootstrap is reloaded to the board.

To be connected with WSN network each node should be identified with a unique MAC address. If MAC address is not defined by a UID hardware chip, the address of the node should be programmed manually. Programming a ATZB-EVB (MeshBean) board with MAC address can be performed in four ways.

MAC address can be uploaded to a board by means of Serial Bootloader using a command line flag (see [7]).

It can be specified in Configuration file when defining the compilation for an application (see details in Section 8.4). The resulting image file containing the unique MAC address can be uploaded to the board either by JTAG or using Serial Bootloader.

Otherwise, MAC address can be programmed sending SerialNet AT-commands, as described in [3].

Value stored in UID is used as MAC address.

MAC address is utilized for identification of the node within the network. Default value of MAC address is zero. The module would not join the network unless MAC address is set to any non-zero value which is not equal to 0xFFFFFFFFFFFFFFFF.

BitCloud software detects MAC address as follows. At startup, BitCloud Software tries to load MAC address from EEPROM. If there is 0 or 0xFFFFFFFFFFFFFFFF value in EEPROM, BitCloud attempts to load MAC address from UID chip. If there is no UID, the node will not be able to join the network.

² Another JTAG programmer may be also used but it should be compatible with the Atmel 1281 MCU.



5.3 Using Serial Bootloader

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

Connect ATZB-EVB (MeshBean) to the PC via USB or serial port, depending on the position of jumper J3 (see Table).

5. Run Serial Bootloader. In command line, specify the image file as `WSNDemo.srec` (see 10.7), the COM port and the optional command line parameters. See [7] for details.
6. Press reset button on the board.
7. 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 would stop.

NOTE:

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.

Make sure that J3 position corresponds to the actual connection of the board, namely Serial or USB.

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.

5.3.1 Using JTAG

Link JTAG emulator to the ATZB-EVB (MeshBean)'s on-board JTAG connector Start uploading process under AVR Studio, following the instructions from [16] and [17]. Select the image file as `WSNDemoApp.hex` (see 10.7) to upload.

Set the following options on the `Fuses` tab before uploading the image through JTAG:

Table 11. Fuse bits setting

Option	Value
BODLEVEL	Brown-out detection disabled
OCDEN	Disabled
JTAGEN	Enabled
SPIEN	Enabled
WDTON	Disabled



Option	Value
EESAVE	Disabled
BOOTSZ	Boot Flash size=1024 words start address=\$FC00
BOOTRST	Disabled
CKDIV8	Enabled
CKOUT	Disabled
SUT_CKSEL	Int. RC Osc.; Start-up time: 6 CK + 65 ms

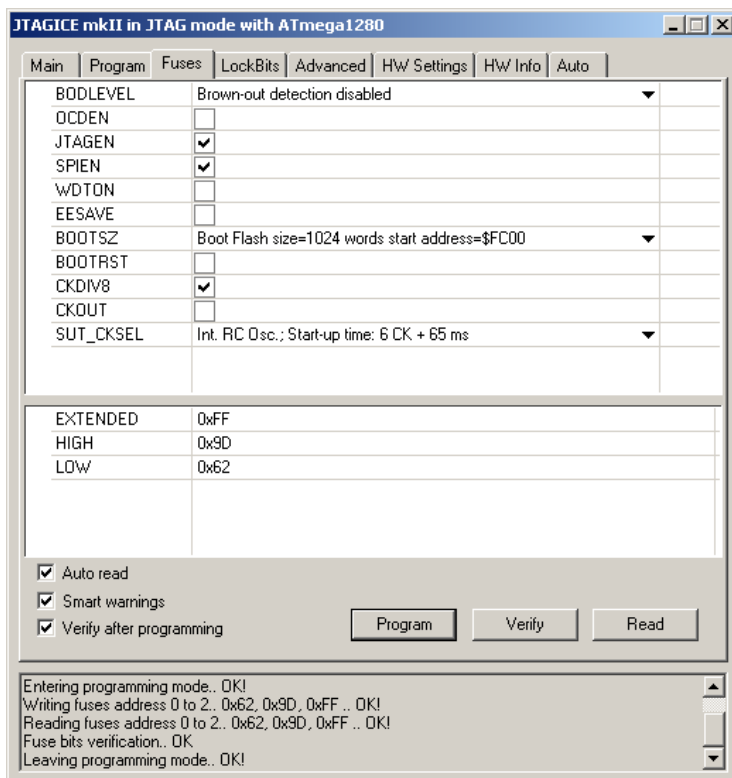


Figure 5. Fuse bits setting

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 (MCU) is preprogrammed with this fuse setting.

In addition, JTAG can be used to restore the device's ability to respond to Serial Bootloader commands. Serial Bootloader firmware can be reprogrammed with JTAG by selecting `bootloader.hex` image contained in your ZDK Distribution CD and uploading it to the device.

5.4 Using the Boards

At node startup, current channel mask is regularly read from EEPROM. If channel mask has been uploaded to EEPROM using Serial Bootloader, then no special action described below is needed before starting `WSNDemo`. Nevertheless, if you need to upload channel mask to EEPROM from flash (from an image file) then startup initialization of the node must be performed as follows.

Press and hold the on-board `SW1` button first (see **Error! Reference source not found.**). Power ON the board with holding the button pressed for at least 1 second. `LED2` will get flashing 3 times. Next, all LEDs will start flashing to indicate the node's role: they will flash once on router, twice on end device and three times on coordinator.

`LED1`, `LED2` and `LED3` will start blinking for 2 sec to indicate the acceptance of channel mask in EEPROM.

NOTE:

When the operation described above is completed, the channel mask preloaded to EEPROM is lost.

Starting the `WSNDemo`, do the following:

- Configure one single node as a coordinator, and make the others be routers and end devices (see Table 11). 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
- 8. Power on the coordinator node
- 9. Run `WSN Monitor` (see Section 0)
- 10. Power ON and reset the rest of the nodes.



NOTE:

While WSN Demo is running, channel mask can be changed anytime by sending the command through WSN Monitor (see Section 0). The channel mask which has been issued from WSN Monitor and received by a node is permanently stored in the node's EEPROM, regardless of power-offs. To restore the default channel mask in EEPROM repeat a node reinitializing procedure described above in this section or use Serial Bootloader.

Table 12. DIP switch configurations used in WSN Demo

DIP switches			Description
1	2	3	
ON	OFF	OFF	Board is configured to be a coordinator.
OFF	ON	OFF	Board is configured to be a router.
OFF	OFF	ON	Board is configured to be an end device.

Coordinator organizes the wireless network automatically. Upon starting, any node informs the network on its role. At that moment, LED1, LED2 and LED3 are flashing once on router, they are flashing twice on end device and they are flashing three times on coordinator.

After joining the network, a node starts sending data to the coordinator, which is indicated by LEDs.

WSN activity is observed in two ways:

- controlling the onboard LEDs (see LED indication described in Table 12)
- controlling the network information through the WSN Monitor installed on PC.

Table 13. LED indication used in WSN Demo

Node State	LED state		
	LED1 (Red)	LED2 (Yellow)	LED3 (Green)
Standby	blinking synchronously		
Searching for network	blinking	OFF	OFF



Joined to network	ON		
+ receiving data		blinking	
+ sending data (coordinator only)			blinking
Sleeping (end device only)	OFF	OFF	OFF

If you power ON the coordinator, it switches to an active state, even though no child node is present. This is normal, it means that the coordinator is ready and child nodes can join the network with coordinator's PAN ID.

By default, coordinator uses predefined PAN ID valued as D170, which is recognized by all routers.

NOTES:

If coordinator is absent or it has not been turned on, the routers are staying in the network search mode. In this mode, routers are scanning the selected frequency channels in search for a network with the selected PAN ID.

In rare cases, if radio channel is busy on the selected frequency the coordinator node is staying in the network searching mode. If this happens, you should switch it to other channel by changing the channel mask in WSN Monitor.

5.5 Sensors Data and Battery Level Indication

Each of the boards measures temperature, light and its own battery level. They send the data values to coordinator and, further to the PC. The WSN Monitor displays the readings from onboard sensor's next to a node icon along with visualization (see Section 5.6).

Temperature sensor measures ambient temperature. The sensor data is depicted in the WSN Monitor charts with resolution of 1 °C, but the actual sensor accuracy is better. Light sensor measures ambient illumination in Lux. The battery voltage is indicated with typical accuracy of about 0.1 V, which is enough for most applications and self-monitoring tasks.

NOTES:

In case the board is powered via USB port, the battery voltage sensor will steadily return 0. However, if batteries had been installed into the battery compartment, when the board is connected to the USB, the battery level indication is correct.

In case the board is powered via USB port, the heating voltage regulator, which is located next to the temperature sensor, can distort the sensor readings. Use battery-powered boards for more accurate measurements.

5.6 WSN Monitor

WSN Monitor is a PC-based GUI application for WSN Demo that is used to display WSN topology and other information about WSN network. See WSN Monitor screen in Figure . It contains the Topology Pane, Sensor Data Pane, Node Data Pane and Toolbars.

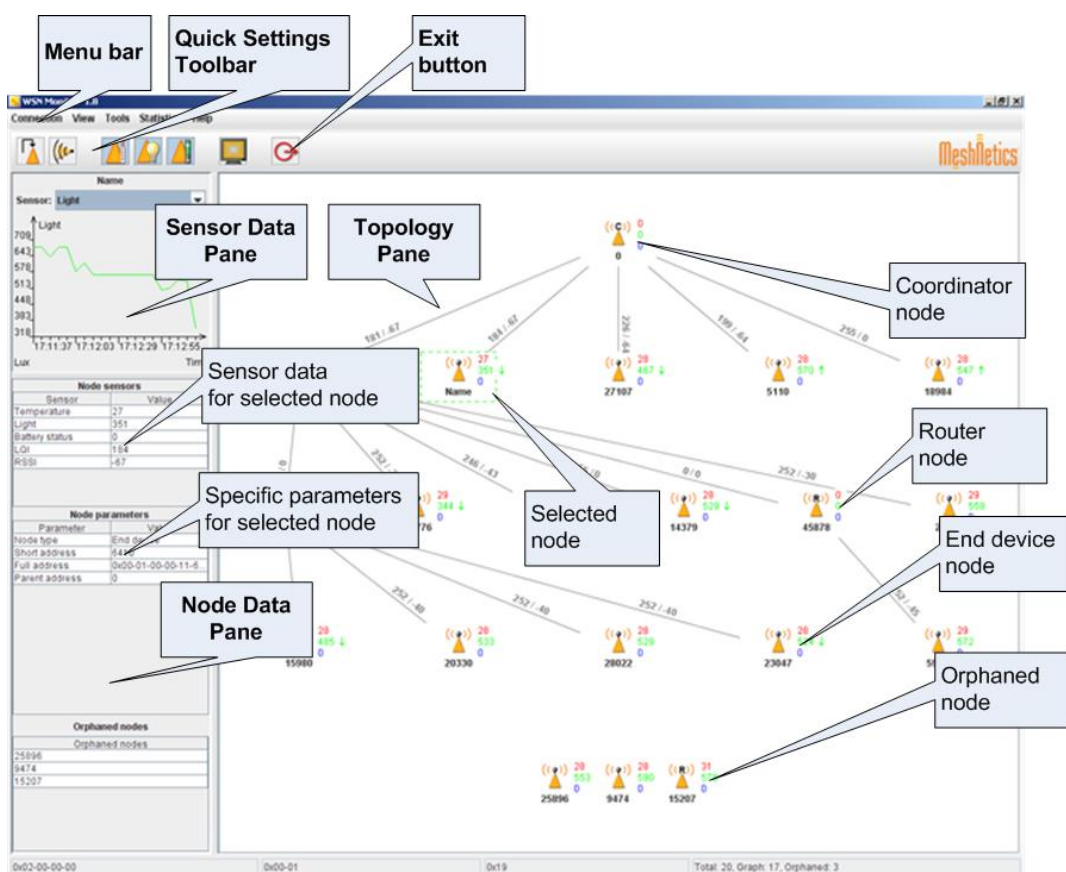


Figure 6. WSN Monitor GUI

Topology Pane displays the network topology in real time, which helps to the user monitor the formation and evolution of the network while the nodes join, send data or leave. The Topology Pane updates automatically while the nodes are discovered and while they join through coordinator. The networking tree is displayed in form of parent/child links which are tipped with RSSI and LQI values. Each of the nodes displayed is depicted by icon with the node's address below and sensor readings to the right.

Node Data Pane displays the data coming from onboard sensor's of each of the nodes (see Section 5.5). It is presented in graphs and in table form. Other parameters can be also observed for each node in table form. Node Data Pane includes a Sensor Selection combo-box used to switch between sensor types.



Node titles are defined in the `NodeNames.txt` file. By default, it is located in the following subdirectory:

```
"/Evaluation Tools/WSNDemo (WSN Monitor)/resources/  
configuration/".
```

Notice: the full path to the file depends on the root directory which has been specified during installation of the Development Kit (see Section 4.3).

`NodeNames.txt` contains one `-- NodeNames --` header string which is followed by a number of strings each of which contains 64-bit MAC address and the title of each node. For example, see Figure 6.

If the `NodeNames.txt` file is not found or its format is not recognized, the WSN Monitor designates the titles named by default.

```
-- NodeNames --  
000100000C29D8C8=Coordinator  
000100000E33F27F=Router_01  
000100000E3408C8=Router_02  
04DD03CC02BB01AA=TTT  
000100000e33c7EC=C7EC  
000100000b6bf889=F889  
000100000e33b85b=B85B  
000100000B6BD8D8=D8D8  
000100000B6BF81A=F81A  
000100000C29FD8C8=FD8C  
000100000E340C47=0C47  
000100000B6BF7E2=F7E2  
000100000E33C6F8=C6F8  
000100000B6BF861=F861  
000100000A3B7242=7242  
000100000B6BF852=F852
```

Figure 7. Example of file containing the node titles

5.7 Running WSNDemo

Starting WSNDemo on ATZB-EVB (MeshBean) nodes

First, connect the coordinator node to the USB or to serial port, in accordance with the J3 jumper setting (see Table). Next, run the WSN Monitor application on your PC. At startup, WSN Monitor will attempt using the default COM port to connect to coordinator. The WSN Monitor screen pops up. To observe the coordinator node icon on the Topology Pane (see Figure), you will have to set a proper COM port via `Connection/Settings` menu (see Figure) and probably restart the program.

Setting up node timeouts

The `Connection/Settings` menu contains a number of parameters. Timeouts are used to tune up visualization for coordinator, routers and end devices because they disappear from the network each time when link drop, power down, or reset occur. A node timeout means the waiting period, during which the WSN Monitor is expecting to receive data packet from that node, which would update the network Topology tree. To get smooth topology visualization, setting timeouts to 3 sec is recommended for

coordinator and router and 30 sec is recommended for end device. Those timeouts cover 3 periods between packet sending.

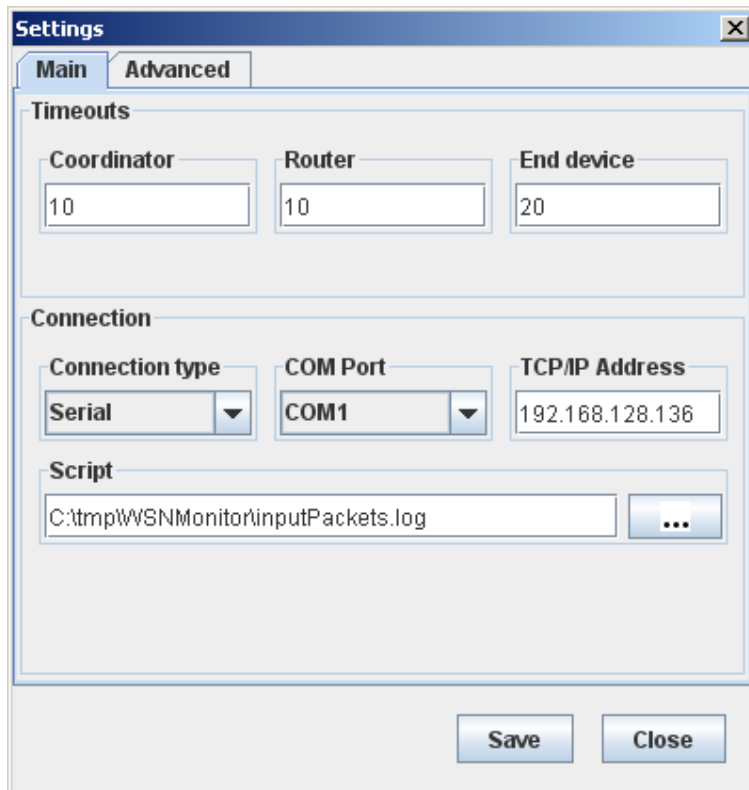


Figure 9. WSN Monitor Connection/Settings menu

Node Reset

A node can be reset by means of the WSN Monitor using the **Tools/Send Command** menu (see Figure 8). A node can be identified by its MAC address or it can be selected from the list of the nodes (using the combo-box) which are currently present in the Network Topology Pane.

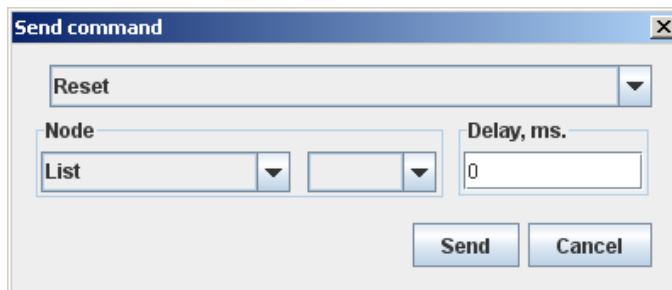


Figure 10. Resetting the node

Changing Frequency Channels



The network operation is supported on 16 upper channels in 2.4 GHz band, which are numbered from 11 (0x0B) through 26 (0x1A). Use Tools/Send Command window to set channel mask. By default, current channel mask is displayed there (see Figure 8).

NOTE:
Channel mask is a bit field which defines the channels available. The 5 most significant bits (b_{27}, \dots, b_{31}) of channel mask should be set to 0. The rest 27 least significant bits (b_0, b_1, \dots, b_{26}) indicate availability status for each of the 27 valid channels (1 = available, 0 = unavailable).

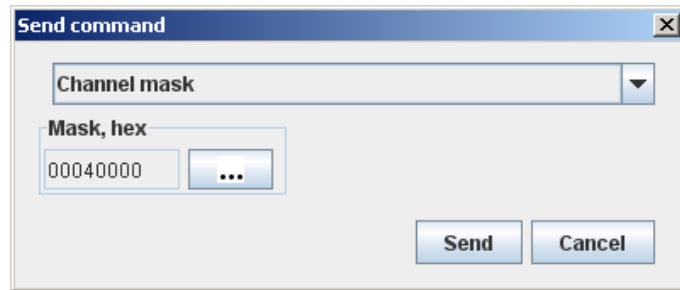


Figure 11. Setting channel mask dialog box

Open the Channel bit mask window by clicking the “...” button. Use checkboxes to check the allowed channels (see Figure 11).

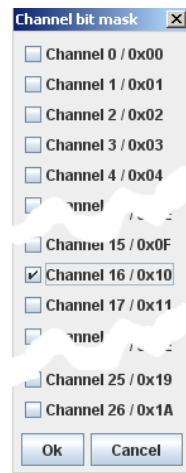


Figure 12 Setting the channel mask using checkboxes

When changing channel mask, coordinator sends the command to all of the nodes and waits for 1 minute more after having received the last packet using old channel mask. Next, coordinator forms the network on the new channel.

When channel mask command is being accepted by router or by end device the node stops sending packets for 1 minute, and the LED1, LED2 and LED3 start blinking. Next, it leaves the network and proceeds joining, using new channel mask.



When router is rejoining, the network indication LED, namely LED3, is blinking. Upon having router joined, LED3 is ON.

When end device is rejoining, the network indication LED, LED3, is blinking. Upon having end device joined, LED3 turns ON. LED1 flashes shortly to indicate sending a packet, LED1 flashes briefly to indicate having received acknowledgement. Next, all LEDs get turned OFF when end device is falling to sleep.

When channel mask is being changed, the Topology Pane might display an outdated topology tree. After changing channel mask, the network topology is updated.

Visualization of the Sensor Data

Observing the Topology tree and operating the GUI controls, user can select any node to monitor the node activity and see the node data in three different forms:

- Text table
- Chart
- The onboard sensor's data in the Topology Pane. These values are tipped with arrows indicating relative increase or decrease.

Topology Pane displays temperature and light readings as well as battery level for any selected node (which icon appears in dashed frame). Also, these data values are shown on the Sensor Data Graph Pane. You can easily check how they evolve over time.

The Sensor Data Graph Pane includes a Sensor Selection combo-box. Use the button on the Sensor Control Toolbar to display the desired types of sensor data.

6 SerialNet

SerialNet is a configuration of BitCloud software which allows control over the most of the ZigBit/BitCloud functionality through a serial communication interface using standardized Hayes-like AT command set.

The commands come from Serial/USB interface in simple text form. The command language principles are described in ITU-T V.250 recommendation (see [11]).

NOTES:

Strictly, the SerialNet is an application developed "on top" of BitCloud API.

Before running SerialNet application make sure that the corresponding image file (see 10.7) is uploaded to each board properly by means of Serial Bootloader or JTAG.

See the set of supported AT commands, their syntax and detailed description in the Reference Manual [3]. Chapter "Examples" of that document shows how to use the commands to do the following:

- to control LED and DIP switches
- to create a network (to set the node roles and addresses)
- to transmit data between the nodes
- to manage PAN ID and frequency channels
- to forward commands for remote execution
- to control power consumption for end device.



Due to flexibility of AT-commands, you can create other network scenarios addressing the specific needs of your application. The examples are recommended as a starting point in evaluation of SerialNet.

A variety of terminal programs provide capability to enter AT-command scripts and to analyze the responses from a board. In order to run the SerialNet application, follow step-by-step instructions from the `Examples` section of the document [3].

NOTE:

The `+IFC` and `+IPR` commands both change the rate and flow control parameters of Serial/USB port. If any of these commands is used, the COM port settings on the terminal program running on the PC should be changed accordingly.

7 Serial Bootloader

Serial Bootloader is software intended to burn firmware images in SREC format into WSN nodes without using JTAG (see 10.8). It also provides the capability to set up the network parameters for each node without altering its firmware manually.

Serial Bootloader consists of two parts: a PC application for Windows platforms (supplied in console and GUI versions) and bootstrap code residing in the MCU. In ZigBit Development Kit, each ATZB-EVB (MeshBean) board is delivered with fuse bits set up and the bootstrap preloaded to ZigBit MCU. Bootstrap itself can be recovered using JTAG when necessary. It is supplied in form of `bootloader.hex` image file (see 10.7).

Exhaustive information on using Serial Bootloader is contained in [7].

8 Programming with BitCloud API

8.1 API Overview

BitCloud internal architecture follows 802.15.4, ZigBee-defined separation of the networking stack into logical layers. Besides the core stack containing protocol implementation, BitCloud contains additional layers implementing shared services (e.g. task manager, security, 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 the set of APIs significantly help reduce application complexity and simplify integration. BitCloud Stack Documentation [4] provides detailed information on all public APIs and their 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 (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, security, 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 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 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.

Hardware Abstraction Layer (HAL) includes a complete set of APIs for using on-module hardware resources (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 (IRQ, I2C, SPI, UART, 1-wire). Board Support Package (BSP) includes a complete set of drivers for managing standard peripherals (sensors, UID chip, sliders, and buttons) placed on a MeshBean development board.

8.2 Using AVR Programming Tools

It is recommended that Atmel's AVR Studio [16] is to develop custom applications based on BitCloud API. This multiplatform Integrated Development Environment (IDE) provides the options for editing source code, compilation, linking object modules with libraries, debugging, making executable file automatically, and more. See Section 4.3 for the IDE installation instructions. Refer to the AVR Studio User's manual for further instructions.

AVR Studio can be integrated with WinAVR – a suite of software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform [19]. WinAVR contains a set of utilities including AVR GCC compiler, linker, automatic Makefile generator, system libraries, etc. GCC compiler is designed to be executed on the Windows platform, and is configured to compile C or C++ codes. For description of GCC compiler see WinAVR documentation. You can find command options for compilation and linking specified in [20].

In AVR Studio, the development of an application is organized under particular project. All the necessary information about a project is kept in project file. Such files assigned to the AVR Studio have an *.aps extension, so they open in AVR Studio automatically when double-clicked.

The easiest way to configure an AVR project is to use Makefile that is a plain text file which name has no extension. Makefile specifies compilation and linking flags. Makefile also specifies corresponding directories in order to include header files and to link the system object libraries.

The required BitCloud software is located in ZDK Distribution CD in the "BitCloud" structured subdirectory as presented in 10.7.

8.3 How to Build Minimum Application

For a quick start in programming, a user's sample application is designed to show the required application structure and coding conventions. This application (a local variety of the traditional "Hello World" demo) implements permanent blinking of the ATZB-EVB (MeshBean)'s LEDs, using the GPIO interface. The source code for minimum application is given in 10.9, along with the Makefile corresponding to the file structure specified in 10.7. Both are located in the ". /Sample Applications/Blink/" subdirectory. The resulting image files are also delivered. You can rebuild them any time as described below.

Open `blink.aps` file from the ". /Sample Applications/Blink/" subdirectory and just execute `Build/Rebuild All` item from the main menu of AVR Studio. The `blink.hex` and `blink.srec` image files will be generated. No *.eep image



file will be produced as EEPROM is not needed for Blink. To test the minimum application, upload any of the image files into a ATZB-EVB (MeshBean) board, following the instructions which are given in Section 7 or in 10.8, correspondingly.

You can modify the code to extend the application's functionality by using other BitCloud API functions. Make sure that your application code satisfies the programming conventions specified in [5].

Play with the other API demos (see Section 8.4), building them similarly to enhance the application with new functionalities. Make sure your applications are uploaded into the boards before use.

8.4 Sample Applications

ZDK is supplied with the set of BitCloud API sample applications in source code. These are named as given in brackets:

- WSNDemo application (`WSNDemo`)
- Low-Power Networking application (`Lowpower`)
- Peer-to-Peer Data Exchange application (`Peer2peer`)
- Ping-Pong application (`Pingpong`)
- Hardware Test (`HardwareTest`).

WSNDemo is a ZDK featured application demonstrating the formation of network based on BitCloud software and ATZB-EVB (MeshBean) hardware. In WSNDemo, the nodes communicate based on a proprietary messaging protocol. WSNDemo is presented in details in Section 5.

The source codes for WSNDemo application (available with Complete Support Package only) can be found inside the `./Sample Applications/WSNDemo` subdirectory (see 10.7), once the Development Kit is installed to user's PC (see Section 4.3).

Network parameters (including security settings) and their default values are defined in `Configuration` file as below:



```
# Stack parameters being set to Config Server
-----
CS_AUTONETWORK = 1
# If CS_AUTONETWORK is 1 CS_CHANNEL_MASK
should be declared
CS_CHANNEL_MASK = "(11<<0x15)"
# Parameter is used only for RF212
CS_CHANNEL_PAGE = 0
CS_RF_TX_POWER = 3
CS_END_DEVICE_SLEEP_PERIOD = 5000
CS_NEIB_TABLE_SIZE = 8
CS_MAX_CHILDREN_AMOUNT = 7
CS_MAX_CHILDREN_ROUTER_AMOUNT = 2
CS_ROUTE_TABLE_SIZE = 25
CS_ADDRESS_MAP_TABLE_SIZE = 25
CS_ROUTE_DISCOVERY_TABLE_SIZE = 10
CS_APS_DATA_REQ_BUFFER_SIZE = 4
CS_APS_ACK_FRAME_BUFFER_SIZE = 3
CS_DUPLICATE_REJECTION_TABLE_SIZE = 7
CS_NWK_DATA_REQ_BUFFER_SIZE = 4
CS_NWK_DATA_IND_BUFFER_SIZE = 4

USE_STATIC_ADDRESSING = 0
# Used only for static addressing
CS_NWK_ADDR = 0x7001

USE_NETWORK_KEY = 0
# Pre-configured key which is used by NWK
# If it is not TRUST CENTRE &&
CS_ZDO_SECURITY_STATUS=3 the key is cleared by
ZDO
CS_NETWORK_KEY =
"{0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,
,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC,0xCC}"

#0 - preconfigured network key
#1 - preconfigured trust centre link key
#2 - preconfigured trust centre master key
#3 - not preconfigured
CS_ZDO_SECURITY_STATUS = 3
```

To compile WSNdemo application use make utility. Otherwise, open the WSNdemo.aps file from the ". /Sample Applications/WSNdemo/" subdirectory



with AVR Studio and just execute Build/Rebuild All item from the main menu. The WSNDemoApp.hex and WSNDemoApp.srec image files will be then generated. Low-Power, Peer-to-peer and Ping-Pong applications are described in details in [4].

9 Troubleshooting

In case of any operational problem with your system please check the power first, and make sure that all of your equipment is properly connected.

Check if your PC conforms to the minimum system requirements (see Section 4.2). Check if the PC interfaces (COM, USB) are present and drivers are installed.

Check on LED indication of a node if it is not responding or behaving unusually. Make sure the DIP switches are set according to the application running on the board.

You can retest the particular node as described in Section 4.7, if needed.

You may be required to reset the node.

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

Table 14. Typical problems and solutions

Problem	Solution
The board does not indicate its activity with LEDs.	Make sure that either WSNDemo image or Hardware Test image is loaded. For SerialNet, the LED status is controlled by AT-commands.
The board does not respond to outer commands (in case of external antenna)	Make sure the external antenna is not broken and it is properly connected to the board.
In effort to connect several boards to the same PC their detection fails due to ID recognition conflict.	Detect ID for any single connected board using the USBView.exe utility from Silicon Laboratories. It can be downloaded from http://www.silabs.com/tgwWebApp/public/web_content/products/Microcontrollers/USB/en/USBXpress.htm You can use the CP210xSetIDs.exe utility from Silicon Laboratories which is included in AN144SW. It is described at http://www.silabs.com/public/documents/tpub_doc/anote/Microcontrollers/Interface/en/an144.pdf and it can be downloaded from http://www.silabs.com/public/documents/software_doc/othersoftware/Microcontrollers/Interface/en/an144sw.zip .
WSN Monitor fails to start.	Make sure Java machine is properly installed on your PC. Java Runtime Environment installation program can be found in <code>./Third Party Software/</code> directory as <code>jre-6u6-windows-i586-p.exe</code> file (see Appendix Section 10.7)
No node is shown on the Topology Pane in the WSN Monitor	Check if the WSN Monitor uses 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 user anytime. These names are stored in the node title file (see Section 4.5) along with MAC addresses mapped to the nodes.
At WSN Monitor startup, all node's LEDs are blinking, or none of them is flashing.	The WSNDemo application was not uploaded into the node. Upload this application to the node.
Neither Serial Bootloader nor other application work with a node, except for the Hardware Test.	Make sure that J3 is set on the board properly to correspond to the actual connection type (either Serial or USB).
	Make sure the microcontroller flash memory was not erased before, and the bootstrap was not lost there after having the node programmed through JTAG.

10 Appendix

10.1 FCC Statements

10.2 Equipment usage

This equipment is for use by developers for evaluation purposes only and must not be incorporated into any other device or system.

10.3 Compliance Statement (Part 15.19)

These devices comply with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. These devices may not cause harmful interference, and
2. These devices must accept any interference received, including interference that may cause undesired operation.

10.4 Warning (Part 15.21)

Changes or modifications not expressly approved by Atmel Norway could void the user's authority to operate the equipment.

10.5 Compliance Statement (Part 15.105(b))

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation.



This equipment generates uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.

10.6 FCC IDs

- ATZB-EVB-24-A2- This equipment carries a certified Radio module with FCC ID VW4A090664
- ATZB-EVB-24-SMA- This equipment carries a certified Radio module with FCC ID VW4A090665
- ATZB-EVB-900-SMA- This equipment carries a certified Radio module with FCC ID VW4A090666
- ATZB-EVB-A24-SMA- **VW4A090969**
- ATZB-EVB-24-PCB- **VW4A090971**

10.7 ZDK File Structure

The installation of ZDK to the user's PC is performed from the ZDK Software and Documentation CD (see Section 4.3). As the result the following file structure will be generated under the user defined destination (see Table).

Table 15. The ZDK file structure

Directory/File	Description
Readme.html	Introductory document containing the links to the documentation files
ZigBit Development Kit Release Notes.txt	The ZDK release notes
EULA.txt	End User License Agreement
./Documentation	Documentation on hardware and software, datasheets, application notes
./Product Information	Getting Started document, product briefs and case study documents
./Bootloader/Bootloader.exe	Console Serial Bootloader executable file GUI Serial Bootloader installer



<code>./Bootloader/GuiBootloaderSetup.msi</code> <code>./Bootloader/bootloader.hex</code>	Binary image file containing bootstrap code
<code>./Evaluation Tools/Hardware Test/HardwareTest.srec</code> <code>./Evaluation Tools/Hardware Test/HardwareTest.hex</code>	Hardware Test image files
<code>./Evaluation Tools/WSNDemo (Embedded)/WSNDemoApp.srec</code> <code>./Evaluation Tools/WSNDemo (Embedded)/WSNDemoApp.hex</code>	WSNDemo image files
<code>./Evaluation Tools/WSNDemo (WSN Monitor)/WSNMonitorSetup.exe</code>	WSN Monitor installer
<code>./Evaluation Tools/SerialNet/serialnet.srec</code> <code>./Evaluation Tools/SerialNet/serialnet.hex</code>	SerialNet image files
<code>./BitCloud/Components</code>	Header files for BitCloud Stack
<code>./BitCloud/Components/BSP/</code>	Source, header and library files for BitCloud BSP
<code>./BitCloud/lib</code>	Library files for BitCloud Stack
<code>./Sample Applications/WSNDemo</code>	Source and image files for WSNDemo application. Source code is available with Complete Support Package only.
<code>./Sample Applications/Blink</code>	Source and image files for Blink application



./Sample Applications/Lowpower	Source and image files for Low Power sample application
./Sample Applications/Peer2peer	Source and image files for Peer-to-Peer sample application
./Sample Applications/SerialNet/SNE_Demo	Source and image files for SerialNet Extension Demo sample application. Source code is available with Complete Support Package only.
./Sample Applications/Pingpong	Source and image files for PingPong sample application
./Evaluation Tools/RangeTest/range_tool.vi	Range Measurement Tool application's GUI
./Evaluation Tools/RangeTest/RangeTestRf230.hex	Range Measurement Tool image files
./Evaluation Tools/Range Test/RangeTestRf230.srec	
.Third Party Software/ CP210x_VCP_Win2K_XP_S2K3.exe	USB to UART Bridge VCP driver installation program
.Third Party Software/ jre-6u6-windows-i586-p.exe	Java Runtime Environment installation program
.Third Party Software/ NetFx20SP1_x86.exe	Microsoft .NET Framework 2.0 installation program

10.8 Using JTAG Emulator

Programming with JTAG gives more flexibility in managing the loading process, but requires special hardware. For Windows environment it's recommended using the AVR Studio 4.14. AVaRICE 2.40 may be used for Linux. In both cases, the recommended JTAG emulator is JTAGICE mkII from Atmel. Other programming devices can be utilized as well, but make sure before use that the particular model supports programming an Atmega1281 MCU.



Using AVR Studio both flash memory and EEPROM of a board can be separately programmed with images having Intel HEX format. EEPROM image has `.eep` extension while flash image has `.hex` extension. To upload firmware, follow the instructions from the device manufacturer's manuals [16], [17], [18]. A sample pop-up window is shown in Figure .

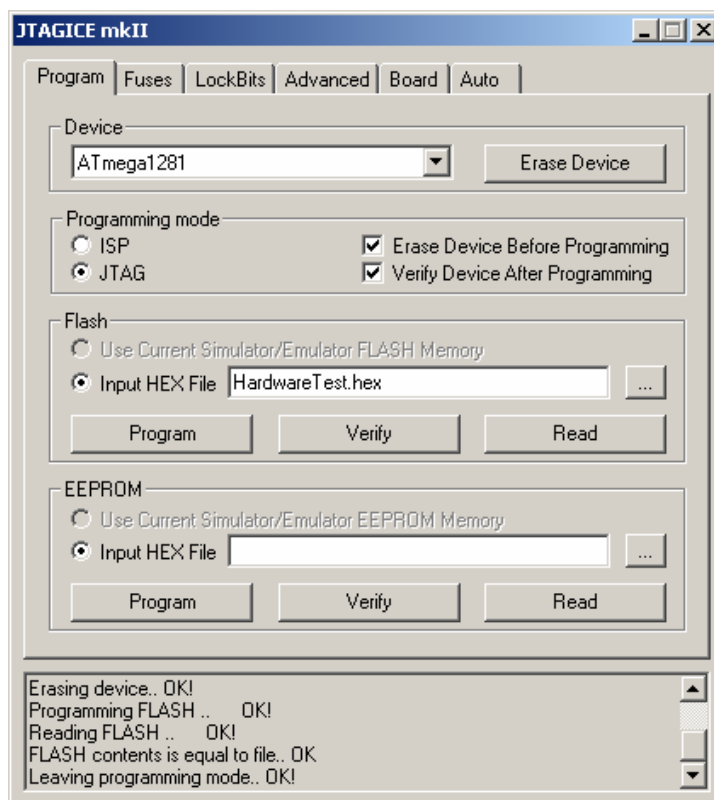


Figure 13. AVR Studio dialog box for firmware upload using JTAG

The well-known command line utility, `avrdude`, which is a part of WinAVR environment (<http://sourceforge.net/projects/winavr>) can be used for upload as well. This utility recognizes both Intel HEX and Motorola SREC formats.

IMPORTANT NOTES:

To avoid corruption of the bootstrap code required for serial booting, do not erase device when using JTAG.

For JTAG programming, the `Boot Reset` vector fuse bit should be disabled. To enable serial booting this fuse bit should be enabled.



```
#define BLINK_INTERVAL
(BLINK_PERIOD / 2)      // Blink interval.
#define MIN_BLINK_INTERVAL
(MIN_BLINK_PERIOD / 2)  // Minimum blink
interval.
#define MAX_BLINK_INTERVAL
(MAX_BLINK_PERIOD / 2)  // Maximum blink
interval.

#ifndef _BUTTONS_
#define BSP_KEY0          0
#define BSP_KEY1          1
#endif //ifndef _BUTTONS_

#define HALF_PERIOD_BUTTON      BSP_KEY0
// Button that reduces blink interval to a
half.
#define DOUBLE_PERIOD_BUTTON    BSP_KEY1
// Button that doubles blink interval.

static HAL_AppTimer_t blinkTimer;
// Blink timer.

#ifndef _BUTTONS_
static HAL_AppTimer_t changeBlinkTimer;
// Buttons emulation timer.
#endif //ifndef _BUTTONS_

static void buttonsReleased(uint8_t
buttonNumber);          // Button release
event handler.
static void blinkTimerFired(void);
// blinkTimer handler.

#ifndef _BUTTONS_
static void changeTimerFired(void);
//Buttons emulation timer handler.
#endif //ifndef _BUTTONS_

/*****
*****
Description: application task handler.

Parameters: none.

Returns: nothing.
```

```
*****
*****/
void APL_TaskHandler(void)
{
#ifdef _LEDS_
    BSP_OpenLeds(); // Enable LEDs
#endif // #ifdef _LEDS_

#ifdef _BUTTONS_
    BSP_OpenButtons(NULL, buttonsReleased);
    // Register button event handlers
#else
    // Configure blink timer
    changeBlinkTimer.interval = 10000;
    // Timer interval
    changeBlinkTimer.mode =
TIMER_REPEAT_MODE; // Repeating mode
(TIMER_REPEAT_MODE or TIMER_ONE_SHOT_MODE)
    changeBlinkTimer.callback =
changeTimerFired; // Callback function for
timer fire event
    HAL_StartAppTimer(&changeBlinkTimer);
    // Start blink timer
#endif // #ifdef _BUTTONS_

    // Configure blink timer
    blinkTimer.interval = BLINK_INTERVAL;
    // Timer interval
    blinkTimer.mode = TIMER_REPEAT_MODE;
    // Repeating mode (TIMER_REPEAT_MODE or
TIMER_ONE_SHOT_MODE)
    blinkTimer.callback = blinkTimerFired;
    // Callback function for timer fire event
    HAL_StartAppTimer(&blinkTimer);
    // Start blink timer
}

#endif #ifndef _BUTTONS_
void changeTimerFired(void)
{
    static uint8_t button = HALF_PERIOD_BUTTON;
    //Buttons emulation
    buttonsReleased(button);
    if (HALF_PERIOD_BUTTON == button)
        button = DOUBLE_PERIOD_BUTTON;
    else

```



```
        button = HALF_PERIOD_BUTTON;
    }
#endif // #ifndef _BUTTONS_

/*****
***** Description: blinking timer
fire event handler.

Parameters: none.

Returns: nothing.
*****
*****/
static void blinkTimerFired()
{
    BSP_ToggleLed(LED_RED);
    BSP_ToggleLed(LED_YELLOW);
    BSP_ToggleLed(LED_GREEN);
}

/*****
*****
***** Description: button release event handler.

Parameters: buttonNumber - released button
number.

Returns: nothing.
*****
*****/
static void buttonsReleased(uint8_t
buttonNumber)
{
    HAL_StopAppTimer(&blinkTimer); // Stop blink
timer

    // Dependent on button being released,
update blink interval
    if (HALF_PERIOD_BUTTON == buttonNumber)
    {
        blinkTimer.interval /= 2;
        if (blinkTimer.interval <
MIN_BLINK_INTERVAL)
            blinkTimer.interval =
MIN_BLINK_INTERVAL;
    }
}
```

```
    }
    else if (DOUBLE_PERIOD_BUTTON ==
buttonNumber)
    {
        blinkTimer.interval *= 2;
        if (blinkTimer.interval >
MAX_BLINK_INTERVAL)
            blinkTimer.interval =
MAX_BLINK_INTERVAL;
    }

    blinkTimerFired();           // Update
LED status immediately.
    HAL_StartAppTimer(&blinkTimer); // Start
updated blink timer.
}

/*****
***** Description: just a stub.

Parameters: are not used.

Returns: nothing.
*****
*****/
void
ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t
*nwkParams)
{
    nwkParams = nwkParams; // Unused parameter
warning prevention
}

/*****
*****
***** Description: just a stub.

Parameters: none.

Returns: nothing.
*****
*****/
void ZDO_WakeUpInd(void)
{
}
```



```
/* *****  
*****  
Description: just a stub.  
  
Parameters: none.  
  
Returns: nothing.  
*****  
*****/  
void ZDO_SleepInd(void)  
{  
}  
  
//eof blink.c
```

Makefile

```
# Components path definition -----  
-----  
COMPONENTS_PATH = ../../Components  
  
# Application makerules including -----  
-----  
include  
$(COMPONENTS_PATH)/../../lib/MakerulesBcAll  
  
# Project name -----  
-----  
PRJ_NAME = blink  
  
# Application parameters -----  
-----  
CFLAGS += -DBLINK_PERIOD=1000  
CFLAGS += -DMIN_BLINK_PERIOD=100  
CFLAGS += -DMAX_BLINK_PERIOD=10000  
  
# Stack parameters being set to Config Server  
-----  
  
# Output debug port for ARM platforms only ---  
-----  
ifeq ($(HAL), AT91SAM7X256)  
CFLAGS += -D_DBG_  
endif
```

```

# Stack libraries paths -----
LIB_PATH =
-L$(COMPONENTS_PATH)/../lib \
-L$(PDS_PATH)/lib \
-L$(CS_PATH)/lib \
-L$(BSP_PATH)/lib

## Stack include paths -----
INCLUDES =
-I$(SE_PATH)/include \
-I$(APS_PATH)/include \
-I$(NWK_PATH)/include \
-I$(ZDO_PATH)/include \
-I$(MAC_PHY_PATH)/include \
-I$(HAL_PATH)/include \
-I$(HAL_HWD_PATH)/include \
-I$(BSP_PATH)/include \
-I$(CS_PATH)/include \
-I$(PDS_PATH)/include \
-I$(TC_PATH)/include \
-I$(SSP_PATH)/include

# Linking -----
ifeq ($(HAL), ATMEGA1281)
LINK_OBJECTS =
$(COMPONENTS_PATH)/../lib/WdtInitatmega1281.o
LINKER_FLAGS = -o$(PRJ_NAME).elf -
Map=$(PRJ_NAME).map
endif
ifeq ($(HAL), AT91SAM7X256)
LINK_OBJECTS =
$(COMPONENTS_PATH)/../lib/FirmwareBoot.o
LDSCRIPT = -Tatmel-rom.ld
LINKER_FLAGS = -Xlinker -o$(PRJ_NAME).elf -
Xlinker -M -Xlinker -Map=$(PRJ_NAME).map -
nostartfiles
endif

# Build -----

```



```
all: pds cs
\
    $(PRJ_NAME).elf $(PRJ_NAME).srec
$(PRJ_NAME).hex $(PRJ_NAME).bin \
    size

pds:
    @echo
    @echo -----PDS library creation-----
    -----
    make all -C $(PDS_PATH)

cs:
    @echo
    @echo -----Configuration Server library
creation-----
    make all -C $(CS_PATH)

$(PRJ_NAME).o: %.o: %.c
    @echo
    @echo -----Application executable
creation-----
    $(CC) -c $(CFLAGS) $(INCLUDES) $^ -o $@

$(PRJ_NAME).elf: $(PRJ_NAME).o
    $(CC) $(CFLAGS) $(INCLUDES)
$(PRJ_NAME).o $(LIB_PATH) $(LINK_OBJECTS) -
-l$(STACK_LIB) -l$(CS_LIB) -l$(PDS_LIB) -
-l$(BSP_LIB) -l$(STACK_LIB) $(LDSCRIPT)
$(LINKER_FLAGS)
    rm -f *.o

%.srec: %.elf
    $(OBJCOPY) -O srec --srec-len 128 $< $@

%.hex: %.elf
    $(OBJCOPY) -O ihex $(HEX_FLASH_FLAGS) $< $@

%.bin: %.elf
    $(OBJCOPY) --strip-debug --strip-unnneeded -
O binary $< $@

size:
    $(SIZE) -td $(PRJ_NAME).elf

flash:
    jtagiceii -d ATmega1281 -f 0x1F62 -e -pf -if
$(PRJ_NAME).hex
```



```
# Cleaning... -----  
-----  
clean:  
    @echo  
    @echo -----Application cleaning-----  
-----  
    rm -rf $(PRJ_NAME).elf $(PRJ_NAME).hex  
$(PRJ_NAME).srec $(PRJ_NAME).o $(PRJ_NAME).map  
$(PRJ_NAME).bin  
    @echo  
    @echo -----PDS library cleaning-----  
-----  
    make clean -C $(PDS_PATH)  
    @echo  
    @echo -----Configuration Server library  
cleaning-----  
    make clean -C $(CS_PATH)  
    @echo  
  
# eof Makefile
```



11 EVALUATION BOARD/KIT IMPORTANT NOTICE

This evaluation board/kit is intended for use for **FURTHER ENGINEERING, DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY**. It is not a finished product and may not (yet) comply with some or any technical or legal requirements that are applicable to finished products, including, without limitation, directives regarding electromagnetic compatibility, recycling (WEEE), FCC, CE or UL (except as may be otherwise noted on the board/kit). Atmel supplied this board/kit "AS IS," without any warranties, with all faults, at the buyer's and further users' sole risk. The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies Atmel from all claims arising from the handling or use of the goods. Due to the open construction of the product, it is the user's responsibility to take any and all appropriate precautions with regard to electrostatic discharge and any other technical or legal concerns.

EXCEPT TO THE EXTENT OF THE INDEMNITY SET FORTH ABOVE, NEITHER USER NOR ATMEL SHALL BE LIABLE TO EACH OTHER FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

No license is granted under any patent right or other intellectual property right of Atmel covering or relating to any machine, process, or combination in which such Atmel products or services might be or are used.

Mailing Address: Atmel Corporation, 2325 Orchard Parkway, San Jose, CA 95131



Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
U.S.A.
Tel: (+1) (408) 441-0311
Fax: (+1) (408) 487-2600

International

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun, Tong, Kowloon
HONG KONG
Tel: (852) 2245-6100
Fax: (852) 27232-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D- 85748 Garching b.
MUNICH
Tel.: (+49) 89-31970-111
Fax: (+49) 89-3194621

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-003
JAPAN
Tel: (+81) 3-3523-3551
Fax: (+81) 3-3523-7581

Product Contact

Web Site
<http://www.atmel.com/>

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2010 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, AVR® and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.