APPLICATION NOTE

Atmel AVR2131: Lightweight Mesh Getting Started Guide

8-bit Atmel Microcontrollers

Features

- Atmel® Lightweight Mesh Software Development Kit (SDK)
- WSNDemo sample application
- Custom applications

Description

The purpose of this application note is to introduce users to the Lightweight Mesh network protocol stack and typical application development process from Atmel. This document describes how to start quickly with the Lightweight Mesh SDK, by setting up the development environment and programming devices with sample applications.

To find more detailed information about the Lightweight Mesh architecture and application development process, refer to [1].

Table of Contents

1. Introduction

Atmel Lightweight Mesh is an easy to use proprietary low power wireless mesh network protocol. Lightweight Mesh was designed to address the needs of a wide range of wireless connectivity applications. Some of these applications include:

- Remote control
- Alarms and security
- Automatic Meter Reading (AMR)
- Home and commercial building automation
- Toys and educational equipment

Lightweight Mesh is designed to work with all Atmel IEEE® 802.15.4 transceivers and SoCs. Currently the stack works with AVR®-based MCUs, but given its extreme portability and low resource requirements, it can be run on almost any Atmel MCU. Table 1-1 gives a summary of the currently supported hardware platforms.

Table 1-1. Supported hardware platforms.

Board or module	MCU	Radio Transceiver
ZigBit® (ATZB-24-B0, ATZB-24-A2)	ATmega1281	AT86RF230B
STK®600-mega128rfa1 board (ATAVR128RFA1-EK1)	ATmega128RFA1	ATmega128RFA1
XMEGA®-B1 Xplained and Rz600 radio modules (ATXMEGAB1-XPLD and ATAVRRZ600)	ATxmega128B1	AT86RF212, AT86RF231
RCB128RFA1	ATmega128RFA1	ATmega128RFA1
RCB231	ATmega1281	AT86RF231

All demonstrations in this document will use the RCB128RFA1 board [3] and the WSNDemo sample application as an example, but the same techniques can be applied to any other development kit, or a custom board and application.

2. Development tools

A development toolchain consists of:

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

IAR Embedded Workbench for Atmel AVR [4] can be used to develop and debug applications for AVR-based platforms. The IAR IDE support's editing of application source code, compilation, linking object modules with libraries, and application debugging.

Atmel AVR Studio 5.1 [5] or Atmel Studio 6 [6] can be used to develop and debug applications for AVR-based platforms. Atmel Studio is equipped with the GCC toolchain and does not require external tools to compile Lightweight Mesh applications.

3. WSNDemo sample application

The WSNDemo application implements a typical wireless sensor network scenario, in which one central node collects the data from a network of sensors and passes this data over a serial connection for further processing. In the case of the WSNDemo this processing is performed by the WSNMonitor PC application. The BitCloud® Quick Start Guide [2] provides a detailed description of the WSNDemo application scenario, and instructions on how to use WSNMonitor.

The majority of the information in [2] applies to the WSNDemo application running on top of Lightweight Mesh stack. However since BitCloud is a ZigBee® PRO stack, there are a few differences in the protocol:

- Device types (Coordinator, Router and End Device) are simulated on the application level; there is no such separation in Lightweight Mesh on the stack level
- The value of the extended address field is set equal to the value of the short address field
- For all frames, the LQI and RSSI fields are filled in by the coordinator with the values of LQI and RSSI from the received frame. This means that nodes that are not connected to the coordinator directly will have the same values as the last node on the route to the coordinator
- Sensor data values are generated randomly on all platforms
- Sending data to the nodes on the network is not implemented and not supported in this demo application

4. Using precompiled binaries

4.1 Overview

The SDK comes with a set of ready-to-use binary images of the WSNDemo application. It includes a set of images for different roles, which are preconfigured with distinct network addresses so they can be used for creating a small sensor network right away. Precompiled binaries have the following naming convention:

WSNDemo_ <Board> _<DeviceType>_<Address>.hex

Here *<Board>* is a shortened name of the board or the module for which this image is compiled, *<DeviceType>* specifies a logical device type (“Coord”, “Router” or “EndDev”) and *<Address>* is a preconfigured network address of the node.

4.2 Programming the boards

To program the precompiled binaries provided with the SDK using Atmel Studio, follow the steps below:

- Start Atmel Studio
- Open “AVR Programming” dialog (Tools -> AVR Programming)
- Select correct tool, device, interface and press “Apply”
- Connect programming tool to the board and power on the board
- Press “Read” button located near the “Device ID” field. Make sure that Device ID is correct
- On the “Fuses” tab set the fuse values and then press “Program” to write them to the device. Refer to Table 4-1 for correct fuse settings
- On the “Memories” tab, provide image file name in the “Flash” field and press “Program”
- Disconnect the programming tool and power cycle (or reset) the board. Device should be working now

Table 4-1. Fuse settings for the precompiled binaries.

Board or module	MCU	Extended	High	Low
RCB128RFA1	ATmega128RFA1	0xFE	0x9D	0xC2
RCB231	ATmega1281	0xFE	0x9D	0xC2
ZigBit	ATmega1281	0xFE	0x9D	0xC2

4.3 Running the application

After all boards are programmed connect coordinator board to the PC and run the WSNMonitor application. Observe the coordinator and other node icons appearing on the screen. Refer to [2] for details on how to use the hardware and PC software.

5. Using provided projects

5.1 Overview

Applications are located in the *apps* directory in the SDK. All sample applications in the Lightweight Mesh SDK come with the project files for Atmel Studio, IAR Embedded Workbench and GNU make utility.

All Lightweight Mesh applications include a configuration file *config.h*. This file contains settings for the application and the stack. WSNDemo application settings are listed in Table 5-1. For system settings mentioned in the configuration file see [1].

Table 5-1. WSNDemo application settings.

Parameter	Description
APP_ADDR	Node network address. This parameter also determines emulated device type: • 0x0000 – Coordinator • 0x0001-0x7fff – Router • 0x8000-0xfffe – End Device
APP_CHANNEL	Radio transceiver channel. Valid range for 2.4GHz radios is 11 – 26 (0x0b – 0x1a)
APP_PANID	Network identifier
APP_SENDING_INTERVAL	This parameter has a different meaning for different device types: • Coordinator: interval between sending sensor values to the UART • Router: Interval between reporting sensor values to the coordinator • End Device: Sleep interval
APP_ENDPOINT	Application main data communication endpoint
APP_OTA_ENDPOINT	Over-the-Air upgrade service endpoint
APP_SECURITY_KEY	Security encryption key

Note: For normal network operation all devices should have different network addresses. There is no automatic address assignment mechanism, so it is the developer's responsibility to ensure that addresses are unique.

Refer to the respective development environment documentation for the information on how to compile and debug projects.

5.2 Over-the-Air upgrade

WSNDemo sample application includes a limited demo of the Over-the-Air (OTA) upgrade feature. Lightweight Mesh SDK comes with a set of tools required to perform OTA upgrades. For further assistance with using this feature, please contact technical support (avr@atmel.com).

6. Creating a new application

6.1 Starting from a template application

The best way to start a new standalone application is to use the provided *Template* application as a base, and make custom modifications. Using template project files will ensure that all necessary components are included in the build, and that all required definitions are present. The template application can be found in the *<SDK Root>/apps/Template* directory.

6.2 Starting from scratch

If Lightweight Mesh has to be integrated into a larger existing project, it is recommended to include all required files and definitions into the existing project. Table 6-1, Table 6-2 and Table 6-3 present a lists of files, include paths and definitions that are required for normal Lightweight Mesh operation.

Table 6-1. Required files.

Platform	Files
Common for all platforms	<SDK Root>\nwk\src\nwk.c <SDK Root>\nwk\src\nwkDataReq.c <SDK Root>\nwk\src\nwkSecurity.c <SDK Root>\nwk\src\nwkFrame.c <SDK Root>\nwk\src\nwkRoute.c <SDK Root>\nwk\src\nwkRx.c <SDK Root>\nwk\src\nwkTx.c <SDK Root>\sys\src\sys.c <SDK Root>\sys\src\sysTimer.c <SDK Root>\sys\src\sysEncrypt.c (only if security is enabled)
ZigBit	<SDK Root>\hal\atmega1281\src\hal.c <SDK Root>\hal\atmega1281\src\halPhy.c <SDK Root>\hal\atmega1281\src\halTimer.c <SDK Root>\phy\at86rf230\src\phy.c
STK600-mega128rfa1, RCB128RFA1	<SDK Root>\hal\atmega128rfa1\src\hal.c <SDK Root>\hal\atmega128rfa1\src\halTimer.c <SDK Root>\phy\atmega128rfa1\src\phy.c
RCB231	<SDK Root>\hal\atmega1281\src\hal.c <SDK Root>\hal\atmega1281\src\halPhy.c <SDK Root>\hal\atmega1281\src\halTimer.c <SDK Root>\phy\at86rf231\src\phy.c

Table 6-2. Required include paths.

Platform	Include Paths
Common for all platforms	<SDK Root>\nwk\inc <SDK Root>\sys\inc <Application Root> (required to locate *config.h* file)
ZigBit	<SDK Root>\hal\atmega1281\inc <SDK Root>\phy\at86rf230\inc
STK600-mega128rfa1, RCB128RFA1	<SDK Root>\hal\atmega128rfa1\inc <SDK Root>\phy\atmega128rfa1\inc
RCB231	<SDK Root>\hal\atmega1281\inc <SDK Root>\phy\at86rf231\inc

Table 6-3. Required definitions.

Platform	Definitions
Common for all platforms	F_CPU=<MCU Operating Frequency> Note that if MCU frequency is different from the supported by default then you may need to change frequency depended code in the *<SDK Root>\hal* directory.
ZigBit	PHY_AT86RF230 HAL_ATMEGA1281 PLATFORM_ZIGBIT
STK600-mega128rfa1, RCB128RFA1	PHY_ATMEGA128RFA1 HAL_ATMEGA128RFA1 PLATFORM_RCB128RFA1
RCB231	PHY_AT86RF231 HAL_ATMEGA1281 PLATFORM_RCB231

The execution environment should ensure that:

- SYS_Init() function is called before any other Lightweight Mesh API call
- SYS_TaskHandler() function is called as often as possible

Note that the *HAL_Init()* function (called from *SYS_Init()* function) will perform low level hardware initialization. If such initialization already is performed by the existing project environment, then it should be removed from the *HAL_Init()* function.

7. References

[1] Atmel AVR2130: Lightweight Mesh Developer guide
[2] Atmel AVR2052: Atmel BitCloud Quick Start Guide
[3] Atmel AVR2044: RCB128RFA1 – Hardware User Manual
[4] IAR Embedded Workbench for Atmel AVR
[5] Studio Archive (AVR Studio installer downloads)
[6] Atmel Studio 6

8. Revision History

Doc. Rev.	Date	Comments
42029A	09/2012	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

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) 2722-1369

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

Atmel Japan G.K.
16F Shin-Osaki Kangyo Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370