APPLICATION NOTE

AT03755: Android Demo Application to Control IEEE 802.15.14 Devices

Atmel MCU Wireless

Description

The purpose of this application note is to introduce users to the Low-Cost Gateway demo applications. Low-Cost Gateway is a reference design featuring Atmel® ATmega256RFR2 microcontroller with embedded IEEE 802.15.4® transceiver and WizNet W5200 TCP/IP embedded Ethernet controller.

This document describes how to start quickly with the provided sample applications and how to modify them for a specific need.

Features

- Low-Cost Gateway reference design and evaluation board
- Device and Gateway sample applications
- NetworkDemo Android™ application

Table of Contents

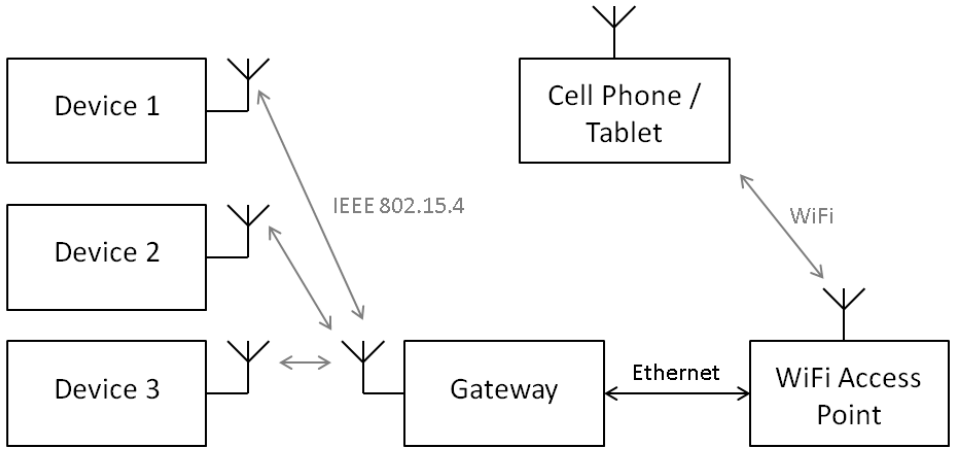
1. Introduction

A block diagram of the system and all the components described in this document is presented on the Figure 1-1.

The idea of the application is to provide an interface from the Wi-Fi enabled cell phone or tablet running Android operating system to the remote devices via IEEE 802.15.4 protocol. The Low-Cost Gateway [3] is used as a bridge between IEEE 802.15.4 and Ethernet. A standard Wi-Fi Access Point is used to provide Wi-Fi and Ethernet connectivity.

Demo applications presented in this document are created using Atmel Lightweight Mesh stack [2], but there are no limitations on the stack from hardware point of view and any stack supported by the hardware can be used.

Figure 1-1. System Diagram



Demo application for the system allows monitoring a state of one button and controlling the state of one LED on the “Device N” nodes via Android application running on a cell phone. There can be up to APP_DEVICE_TABLE_SIZE devices in the networks at the same time and up to APP_UDP_CLIENT_TABLE_SIZE clients controlling them at the same time (see Section 2.2).

2. Gateway Application

2.1 Overview

Gateway application is designed to run on the Low-Cost Gateway board. This application is architected to be a simple pass-though device between the Ethernet port and IEEE 802.15.4 network. In this way application interface is truly defined between the device and the client and gateway acts as a transparent connecting device without interpreting transferred data. This allows for quicker modifications to the protocol since only two sides that are interested in the modifications have to be changed.

The current gateway application only supports UDP clients, but provisions are made to support other types of clients. Multiple clients connected at once. Every message received from the network is distributed to all clients and any client can send a message to the network. This way all clients are always synchronized, so action initiated from one client will be reflected on other clients in a real time.

Application uses simple text-based protocol for communication with the clients. This is done to accommodate for different types of interfaces, some of which might not be able to carry raw binary data (for example HTTP). Protocol is described in details in Section 2.4.

2.2 Configuration Parameters

In addition to the core stack configuration parameters described in [1], the user can control application parameters described below. Application parameters are located in the *config.h* file.

- APP_ADDR – network address. This parameter should always be set to *0* on the gateway. It is located in the configuration file only for uniformity and information reasons.
- APP_CHANNEL – frequency channel for the network operation. This parameter must be set to the same value on all devices in the network.
- APP_PANID – network PAN ID. This parameter must be set to the same value on all devices in the network.
- APP_ENDPOINT – endpoint ID that application uses for communication
- APP_ETH_HOST_NAME – name of the device that will be presented to DHCP server
- APP_ETH_MAC_ADDRESS – Ethernet MAC address. This parameter should be unique for each device if multiple devices operate on the same network. Most Wi-Fi routers use MAC address for IP address reservation in DHCP server settings.
- APP_DEVICE_TABLE_SIZE – the size of the table that holds information about currently active devices. This table is used to send “removed from a network” command on behalf of a device.
- APP_DEFAULT_TTL – specifies the default expiration time (Time To Live) for an entry in the Device Table. To ensure correct network operation, a value of this parameter should be consistent with the report interval of the devices (see the description of APP_UPDATE_INTERVAL in Section 3.2).
- APP_UDP_CLIENT_TABLE_SIZE – the size of the table that holds information about currently active clients (cell phones or tablets). This table is used to notify clients about messages received from the network.
- APP_UDP_CLIENT_TTL – specifies the default expiration time (Time To Live) for an entry in the Client Table. Clients must send a command or an update request to stay in the table.

2.3 Building and Programming the Application

To build the application use provided Atmel Studio solution (*LcGw_ATmega256rfr2.atsln*) or provided Makefile (*Makefile_LcGw_ATmega256rfr2*). Refer to the respective development environment documentation for the information on how to compile, program and debug projects.

Before programming a compiled binary image into the chip make sure that MCU Fuses are set to the correct values:

- Extended: 0xFE
- High: 0x9D
- Low: 0xC2

2.4 Making Modifications

Normally Gateway application will not need any customization since it only transfers messages without interpreting them.

An entry point to the Ethernet message processing is located in the *APP_ClientProcessUdpMessage()* function. This function receives a buffer of text data formatted in as described below. Application can send messages back to the Ethernet port using function *APP_ClientSend()*. It uses the same text-based format for the data.

Text messages between the Gateway and the Client have following structure: *“<command>|<addr>:<data>”*., where:

- <*command*> is one character in length and represents a command code. Currently defined commands are:
  - ‘+’ – update client information. Clients must send this command at least every APP_UDP_CLIENT_TTL seconds.
  - ‘|’ – data message. This is the main command used to pass messages.
- <*addr*> is a hexadecimal representation of the destination (in case of the Client message) or the source (in case of the Gateway message) address. Leading zeros must be present so that entire field is four characters long. The value of this field can be set to *“ffff”* to indicate broadcast command.
- <*data*> is a hexadecimal representation of the data stream, each byte is encoded as two hexadecimal numbers and all bytes are concatenated into one string.

It is recommended to define new commands only for the purpose of communication between the Gateway and the Client.

3. Device Application

3.1 Overview

Device application is designed to run on ATmega256RFR2 Xplained Pro board (ordering code ATMEGA256RFR2-XPRO). Application allows remote control and monitoring of the state of the button and LED located on the board.

Device application is structured in a way that it can be easily modified and adopted for other applications. Commands processed and generated by the device are defined by the *AppCommand_t* structure located in the *appCommand.c* file.

Currently defined commands are:

- APP_COMMAND_SCAN – sent from the client to the device to collect information about the network by the new client and it is usually sent as a broadcast. In response device generates APP_COMMAND_REPORT command.
- APP_COMMAND_REPORT – sent from the device to the client in response to APP_COMMAND_SCAN and periodically with APP_UPDATE_INTERVAL interval (see Section 3.2).
- APP_COMMAND_NAME_SET – sent from the client to the device to set a name for the device. In response device generates APP_COMMAND_REPORT command with a new name in the payload.
- APP_COMMAND_LED_SET – sent from the client to the device to set LED state. In response device generates APP_COMMAND_LED_STATE command with updated LED state.
- APP_COMMAND_LED_GET – sent from the client to the device to request LED state information. In response device generates APP_COMMAND_LED_STATE command with the LED state information.
- APP_COMMAND_LED_STATE – sent from the device to the client in response to APP_COMMAND_LED_SET and APP_COMMAND_LED_GET commands
- APP_COMMAND_BUTTON_GET – sent from the client to the device to request button state information. In response device generates APP_COMMAND_BUTTON_STATE command.
- APP_COMMAND_BUTTON_STATE – sent from the device to the client in response to APP_COMMAND_BUTTON_GET command and when the physical button state is changed

Commands for handling specific GPIOs (LED and Button) can be generalized to handle a number of similar GPIOs with one command and additional index argument. For the information on how to extend command set see Section 3.4.

3.2 Configuration Parameters

In addition to the core stack configuration parameters described in [1] user can control application parameters described below. Application parameters are located in the *config.h* file.

- APP_ADDR – network address. This parameter should always be set to a unique value for each device in the network.
- APP_CHANNEL – frequency channel for the network operation. This parameter must be set to the same value on all devices in the network.
- APP_PANID – network PAN ID. This parameter must be set to the same value on all devices in the network.
- APP_ENDPOINT – endpoint ID that application uses for communication
- APP_UPDATE_INTERVAL – defines how often device will report its state even if there are no events to report. The value of this parameter must be a balance between the amount of network traffic generated by the devices and the update rate. See the description of APP_DEFAULT_TTL parameter in Section 2.2 to learn about gateway configuration to accommodate changes to this parameter.
- APP_NWK_BUFFERS_SIZE – defines number of buffers that can be allocated at the same time for the commands. Frequently application may need more than one buffer to store information about quick events. For example pressing and releasing a button will generate two separate events and second event can happen faster than the first one is transmitted. Note that even if one of the events is lost, clients still will be updated of the new state of the device due to the periodic state reporting.
- APP_NWK_RETRIES – number of retries that will be performed to deliver a message

3.3 Building and Programming the Application

To build the application use provided Atmel Studio solution (*XplainedPro_ATmega256rfr2.atsln*) or provided Makefile (*Makefile_XplainedPro_ATmega256rfr2*). Refer to the respective development environment documentation for the information on how to compile, program and debug projects.

Note that each device image must have a unique value for the APP_ADDR parameter (see Section 3.2).

Before programming compiled application into the chip make sure that MCU Fuses are set to the correct values:

- Extended: 0xFE
- High: 0x9D
- Low: 0xC2

3.4 Making Modifications

The customizable part of the Device application is located in the *appCommand.c* file. In particular, function APP_CommandReceived() is called when a command is received from the gateway. This function receives a pointer to the raw data and it is up to the application to interpret the data. Current implementation assumes that the first byte is a Command ID and the rest are command-specific arguments.

appCommand.c also provides functions for other functional modules such as *APP_CommandGpioChange()* that GPIO module can use to indicate change in the state of the button, or *APP_CommandReport()* that is used to provide periodic status reports. It is recommended that all extensions to the protocol are made in the similar way.

Function *APP_NwkSendBuffer()* is used to send messages to the Gateway. This function operates on the *AppNwkBuffer_t* structure, which can be allocated using *APP_NwkGetBuffer()* function.

Typical use of *APP_NwkGetBuffer()* and *APP_NwkSendBuffer()* functions

```
static void appCommandSendStateCommand(uint8_t id, uint8_t state)
{
  AppNwkBuffer_t *buf;

  if (NULL == (buf = APP_NwkGetBuffer()))
    return;

  buf->size = 0;
  buf->data[buf->size++] = id;
  buf->data[buf->size++] = state;

  APP_NwkSendBuffer(buf);
}
```

4. Android Application

4.1 Overview

NetworkDemo is an Android application that implements a Client protocol and communicates to the Low-Cost Gateway. Application interface is shown on the Figure 4-1. The main window of the application contains a list of device that is populated dynamically based on the information collected from the network. Each item in the list contains:

- A device name that can be changed by a long press and hold
- A short summary of the device state, including a device address
- A button image representing a state of the button on the device. This image can show one of two possible states – pressed or released.
- A lamp button representing a state of the LED on the device. A lamp button can be also used to control the state of the LED. The image on the button can be in one of three possible states – on, off and unconfirmed. An unconfirmed state is set after a button is pressed to change the state and command is sent to the device, but confirmation has not been received yet.

Figure 4-1. Screenshot of the Android Application Interface

The settings dialog can be called by pressing a Menu button on the phone and selecting “Settings” from the menu. Settings dialog allows user to configure network parameters of the Gateway. For description of the various configuration options see Section 5.

Note that if no devices are connected application interface will look like a blank screen with a grey background.

4.2 Building the Application

Currently the only way to build the application is to use Apache Ant [6] – a command line tool. Latest version of the Android SDK [7] must be installed as well.

To build the application run *"ant debug"* in a root directory of the application. Sometimes and especially after making significant changes to the application or resource files, Ant builder incorrectly calculates dependencies and produces broken applications as a result. In this case run *"ant clean"* and then *"ant debug"* again.

A successful run of a build process will create a file named *NetworkDemo-debug.apk* in the *bin* directory. There are a few ways to install the application:

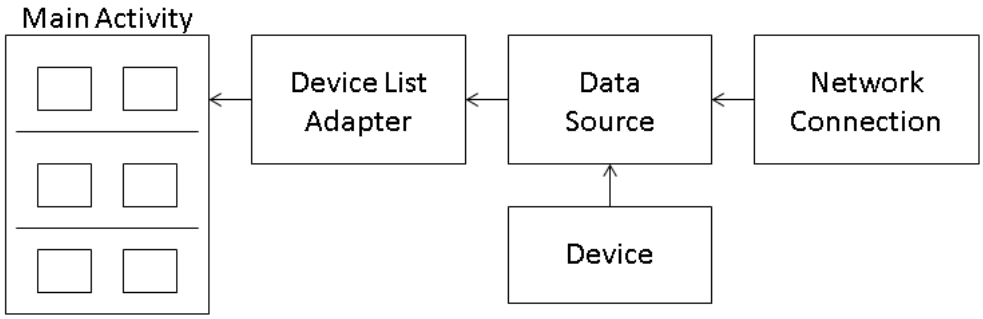
1. If phone is connected to the PC via USB port then it is possible to use *adb* tool from the Android SDK. Run *"adb install -r bin/NetworkDemo-debug.apk"* to install the application. Make sure that USB Debugging is enabled and installation of Third-Party Applications is allowed in the phone settings.
2. Or, download *NetworkDemo-debug.apk* to the phone using any other method (SD Card, Bluetooth, from the Web, etc) and run it from the phone using file explorer. Installation of Third-Party Applications must be allowed in the phone settings.

4.3 Making Modifications

NetworkDemo application logically consists of two activities – Main device list and Preferences. Code for the Main activity is located in the *MainActivity.java* file and layout is defined by the *main.xml* file. Code for the Preferences activity is located in the *Preferences.java* file and layout is defined by the *preferences.xml* file.

Overall application structure is presented on the Figure 4-2. Main activity consists of a single ListView widget displaying formatted information about the devices and providing buttons to control them. According to Android design principles List View displays data from a Device List Adapter, which is responsible for updating and maintaining displayed data. Due to dynamic nature of the device state Data Source storage is necessary. Data Source acts as a source of the information for Device List Adapter on one side and as a way to update the information based on the commands received from the network by a Network Connection on the other side. As a unit of storage Data Source uses Device class, which provides simple interface to the device properties and state variables (name, LED sate, button state).

Figure 4-2. Application Structure



To extend an application with new parameters the following changes have to be made:

1. Add new widgets that will represent visual appearance of the new parameter to the List Item layout description file (*list_item.xml*).
2. Add corresponding state update code to the Device List Adapter.
3. Add new commands handlers to the Data Source.
4. Extend Device definition to handle new parameters.

5. Configuring the Demo Setup

There are a few steps to be performed to configure the demo system.

First, Wi-Fi access point needs to be configured to provide fixed or known IP address to the Gateway. If a dedicated Wi-Fi access point is used and it is possible to gain access to the DHCP server configuration, then a reservation should be made to provide a fixed IP address for a MAC address of the Gateway. Some routers might use a host name for reservation as well. See the description for the parameter APP_ETH_MAC_ADDRESS and APP_ETH_HOST_NAME in the Section 2.2.

If device has to be used in a network with unknown parameters and access to the DHCP settings is not possible, then a debug output of the Gateway can be used to pick up the assigned IP address. This method requires external hardware, so it is not recommended for general use.

Low-Cost Gateway board exposes debug UART (38400 8N1) interface on the connector J2. Pinout of this connector is following: 1 – GND, 2 – Tx, 3 – Rx. After reset application will print debug messages, which will contain assigned IP address:

Debug output during Gateway application startup sequence

```
--- start ---
ETH: link up
DHCP Discover
DHCP Notification: 0
IP = 192.168.0.110
GW = 192.168.0.1
SN = 255.255.255.0
DNS server = 192.168.0.1
Lease time = 0x00015180
```

Next, NetworkDemo application must be configured to connect to the Gateway IP address. In the Settings dialog, set “Host name or IP address” parameter equal to the known IP address of the gateway. Unless it has been changed in the Gateway firmware, “Port number” parameter must remain the same (34567).

And finally, make sure that Android phone running NetworkDemo application is connected to the same Wi-Fi network as one that is connected to the Gateway.

Now NetworkDemo application should recognize remote devices in the network and display them in the list.

6. References

[1] Atmel AVR®2130: Lightweight Mesh Developer guide
[2] Atmel Lightweight Mesh
[3] AT01030 Low-Cost Ethernet to Wireless Gateway with ATmega256RFR2
[4] Studio Archive (AVR Studio® installer downloads)
[5] Atmel Studio 6
[6] Apache Ant
[7] Android SDK

7. Revision History

Doc. Rev.	Date	Comments
42163A	04/2014	Added a shortcut for [3] and made some small changes according to the template
42163A	07/2013	Initial revision

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