APPLICATION NOTE

Atmel AT02744: Lightweight Mesh to Ethernet Gateway with SAM3X - Software User's Guide

Atmel 32-bit Microcontroller

Features

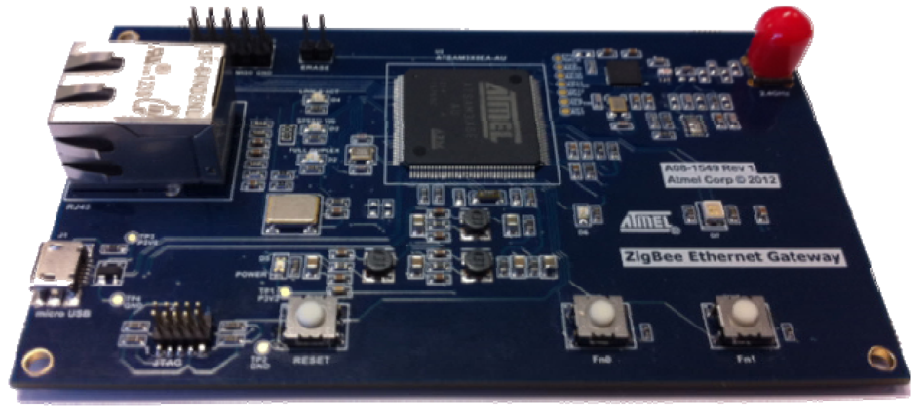
- Atmel® ATSAM3X8E microcontroller
- Atmel AT86RF231 2.4GHz radio transceiver
- Atmel proprietary Lightweight Mesh software stack
- 10/100Mbps Ethernet
  - LwIP TCP/IP stack support
  - TCP/IP client
- Single MCU gateway solution
- Preprogrammed firmware of wireless lighting control via PC software

Introduction

This application note mainly describes the software architecture and the application programming interfaces (API) of Lightweight Mesh to Ethernet Gateway reference design (hereafter the Gateway).

A getting started guide at the end of this document gives details about the setup and operation of preprogrammed firmware.

Figure 1. Lightweight Mesh to Ethernet Gateway.



The Gateway is based on Atmel ATSAM3X8E microcontroller and Atmel AT86RF231 2.4GHz radio transceiver. For gateway hardware design details, refer to Atmel AT2200: ZigBee® to Ethernet and Wi-Fi Gateway with SAM3X - Hardware User's Guide.

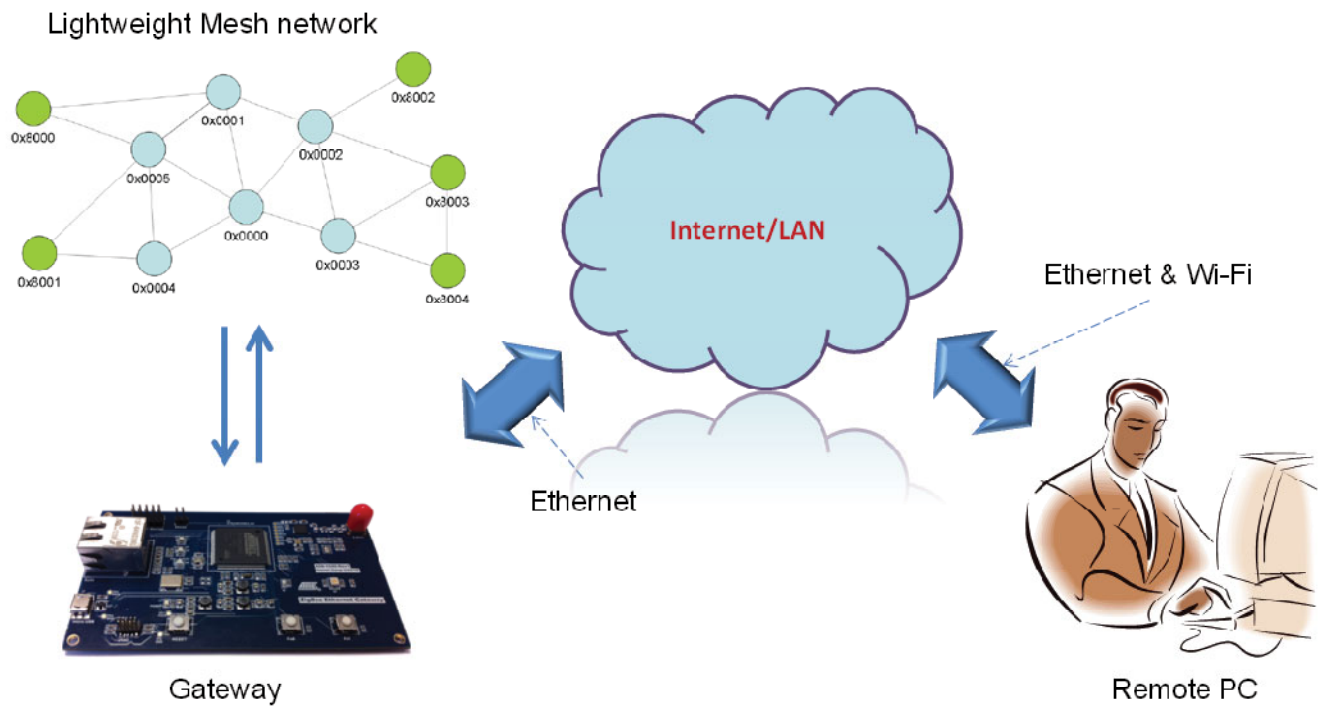
For this reference design, the hardware design files (schematic, BOM and PCB Gerber) and software source code can be downloaded from Atmel website. The provided hardware documentation can be used with no limitations to manufacture the reference hardware solution for the design.

Table of Contents

1. Overview

The Lightweight Mesh to Ethernet Gateway is designed to interface Lightweight Mesh network to Ethernet network. Through the Gateway, the user can control and monitor any nodes in Lightweight Mesh network remotely via Ethernet. A typical application scenario is shown in Figure 1-1.

Figure 1-1. Typical Gateway Application Scenario.



In the preprogrammed firmware, a wireless lighting control and monitor network is established. The lights in Lightweight Mesh network can be controlled and monitored via PC software remotely.

2. Development Tools

To download or debug the preprogrammed firmware, the following development toolchain is needed:

- Atmel Studio 6. Version: 6.1.2674 with Service Pack 1 or above
- Atmel ARM® GNU Toolchain. Version: 4.7.3.158 - GCC 4.7.3 or above
- Atmel Software Framework. Version: 3.9.1 or above
- Programming and debugging device: Atmel SAM-ICE™
- SAM-ICE Adaptor: a minimized (1.27mm pitch 10-pin header) adaptor for Atmel SAM-ICE. For more details refer to Atmel AVR2033: SAM-ICE Adapter - Hardware User Manual

3. Software Architecture

The software for the Gateway is composed of two main parts:

- Atmel Lightweight Mesh Software Stack
- LwIP TCP/IP Stack

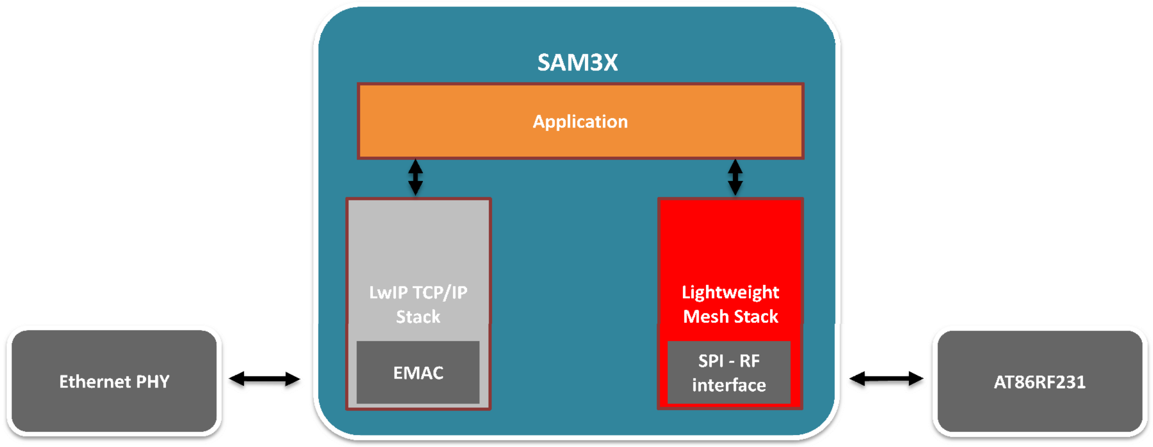
Besides the Gateway itself, PC software named “TCP server” is also provided as a simple graphic user interface (GUI).

The Gateway application is designed based on Atmel Software Framework (ASF). In fact, except the Lightweight Mesh Software Stack, other modules are from ASF.

The Gateway exchanges data between Lightweight Mesh network and Ethernet network. The Lightweight Mesh network data is collected by the Gateway and transferred into Ethernet network, and finally displayed in PC software. The user input in PC software is also sent back to Lightweight Mesh network via the Gateway.

The software block diagram of the Gateway is given in Figure 3-1.

Figure 3-1. The Gateway Software Block Diagram.



3.1 Atmel Lightweight Mesh Software Stack

Atmel Lightweight Mesh is the easy-to-use proprietary low power wireless mesh network protocol from Atmel. It is designed to work with all Atmel IEEE® 802.15.4 transceivers and SoCs. To find more detailed information about the Lightweight Mesh architecture and application development process, refer to Atmel AVR2130: Lightweight Mesh Developer Guide.

Atmel Lightweight Mesh software stack features:

- Simplicity of configuration and use
- Up to 65535 nodes in one network (theoretical limit)
- Up to 65535 separate PANs on one channel
- Up to 15 independent application endpoints
- No dedicated node is required to start a network
- No periodic service traffic occupying bandwidth
- Two distinct types of nodes:
  - Routing (network address < 0x8000)
  - Non-routing (network address >= 0x8000)
- Once powered on, the node is ready to send and receive data; no special joining procedure is required
- No child-parent relationship between the nodes
- Non-routing nodes can send and receive data to/from any other node (including non-routing nodes), but they will never be used for routing purposes
- Route discovery happens automatically if route to the destination is not known
- Routing table is updated automatically based on the data from the received and transmitted frames
- Duplicate frames (broadcast or multipath unicast) are rejected
- Small footprint (less than 8kB of Flash and 4kB of RAM for a typical application)

Currently the public release version of Lightweight Mesh software stack works with AVR®-based MCUs, but given its extreme portability and low resource requirements, it can be run on almost any Atmel MCU.

In the Gateway, it runs on ATSAM3X8E MCU. The Lightweight Mesh software stack version is v1.0.0.

Note that at time of writing this application note, the Atmel Lightweight Mesh Software Stack is not integrated into ASF. In this reference design, the files in hal folder of Lightweight Mesh Software Stack are modified to reuse the low level drivers from ASF.

3.2 LwIP TCP/IP Stack

The Lightweight TCP/IP stack is designed for embedded systems. The focus of the LwIP TCP/IP implementation is to reduce resource usage while still having a full scale TCP.

LwIP features:

- IP (Internet Protocol) including packet forwarding over multiple network interfaces
- ICMP (Internet Control Message Protocol) for network maintenance and debugging
- UDP (User Datagram Protocol) including experimental UDP-lite extensions
- TCP (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery/fast retransmit
- Specialized raw API for enhanced performance
- Optional Berkeley-alike socket API
- DHCP (Dynamic Host Configuration Protocol)
- PPP (Point-to-Point Protocol)
- ARP (Address Resolution Protocol) for Ethernet

For more detailed information about the LwIP, refer to LwIP Wiki: http://lwip.wikia.com/wiki/LwIP_Wiki or the Atmel AVR32817: Getting Started with the 32-bit AVR® UC3 Software Framework LwIP TCP/IP Stack application note.

In the Gateway, only TCP/IP client is implemented. The LwIP version is 1.4.0.

3.3 PC software

PC software named “TCPServer” is provided to control and monitor devices in Lightweight Mesh network.

The following information is displayed in the “TCPServer” main window.

- Device: End device or Router
- Address: Device short address in Lightweight Mesh network
- Status: Device in network or not
- Channel: Working channel
- PAN ID: Lightweight Mesh network PAN ID
- LQI: Link quality Indicator of the last data transfer
- RSSI: Received Signal Strength Indication of the last data transfer
- TSensor: Dummy sensor data sent from devices in Lightweight Mesh network
- LED status: LED on / off status

When clicking on a specific device in main window, the selected device can be controlled and monitored in the “TCPServer” control window.

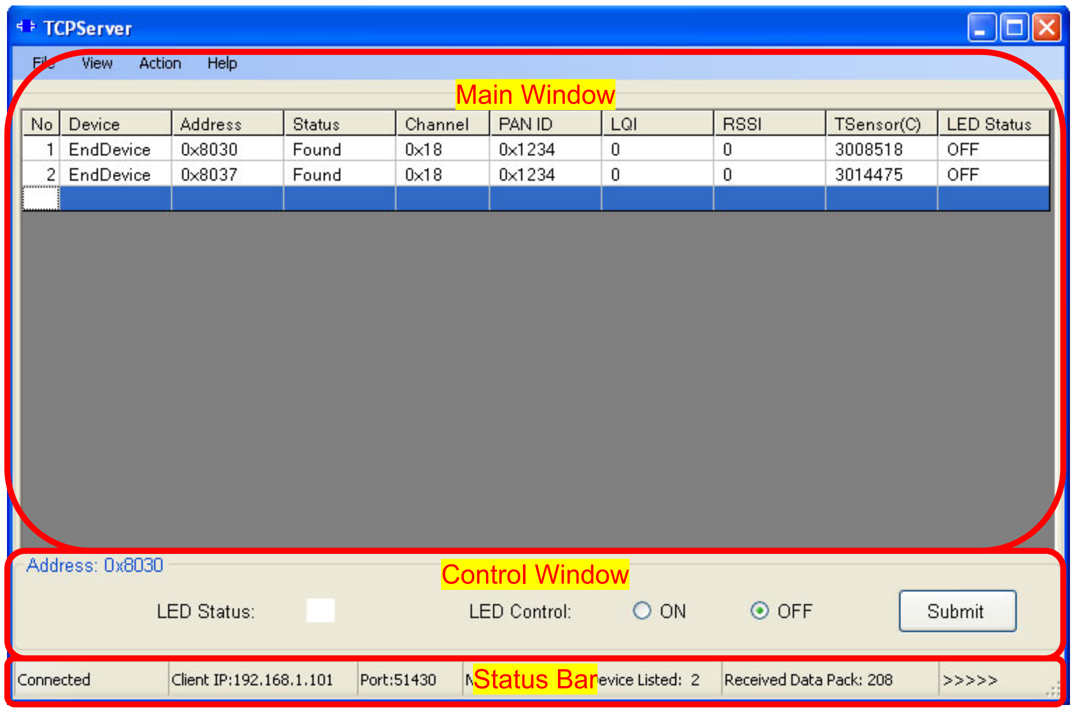
- The LED status will be displayed. Red stands for “ON”, and blank indicates “OFF”
- The LED can be turned on or off by selecting “ON” or “OFF” and clicking “Submit”

The following information is displayed in the status bar:

- The server status
- Client IP address
- Port number
- Number of devices in Lightweight Mesh network
- Received data pack

A screenshot of the PC software is shown in Figure 3-2. For details of the PC software operation, check Chapter 8.

Figure 3-2. The Gateway PC Software "TCPServer".



No	Device	Address	Status	Channel	PAN ID	LQI	RSSI	TSensor(C)	LED Status
1	EndDevice	0x8030	Found	0x18	0x1234	0	0	3008518	OFF
2	EndDevice	0x8037	Found	0x18	0x1234	0	0	3014475	OFF

Address: 0x8030

LED Status:

LED Control: ON [✓ OFF]

Connected | Client IP:192.168.1.101 | Port:51430 | Nevice Listed: 2 | Received Data Pack: 208 | >>>>>

4. Inside the Gateway Application

In this chapter, the overall Gateway application layer structure is explained. And by different functions, the main application tasks in the Gateway are introduced in two parts: Lightweight Mesh, LwIP task.

4.1 Gateway Application Layer Structure

As the Gateway is based on WSNDemo example from Lightweight Mesh Software Stack, it has similar structure in main(). The main() function of the Gateway is shown below:

```
int main(void)
{
        sysclk_init();
        board_init();
…
        init_ethernet();
…
        /* This is the main polling loop */
        while (1) {
                /* LwMesh task handler */
                SYS_TaskHandler();
                HAL_UartTaskHandler();
                …
                APP_TaskHandler();
                ...
                ethernet_task();
                …
        }
}
```

`sysclk_init()` and `board_init()` are two functions to initialize clock and board from ASF. The Lightweight Mesh Software Stack hardware initializations are also put in `board_init()`. `init_ethernet()` is to initialize Ethernet.

The task handlers in the while loop handles application tasks. They are introduced in the following sections. By following this structure, more functions can be added into main() to enrich the Gateway features.

Several macro switches are defined in conf_board.h to give different application options. Here are some examples.

- `#define LWMESH_USED                      1 // Enable LWMESH PHY`
- `#define ETH_USED                         1 // Enable Ethernet`

…

To run the Gateway with default features, don't not change the macro switches unless you know what it changes clearly.

4.2 Lightweight Mesh Task

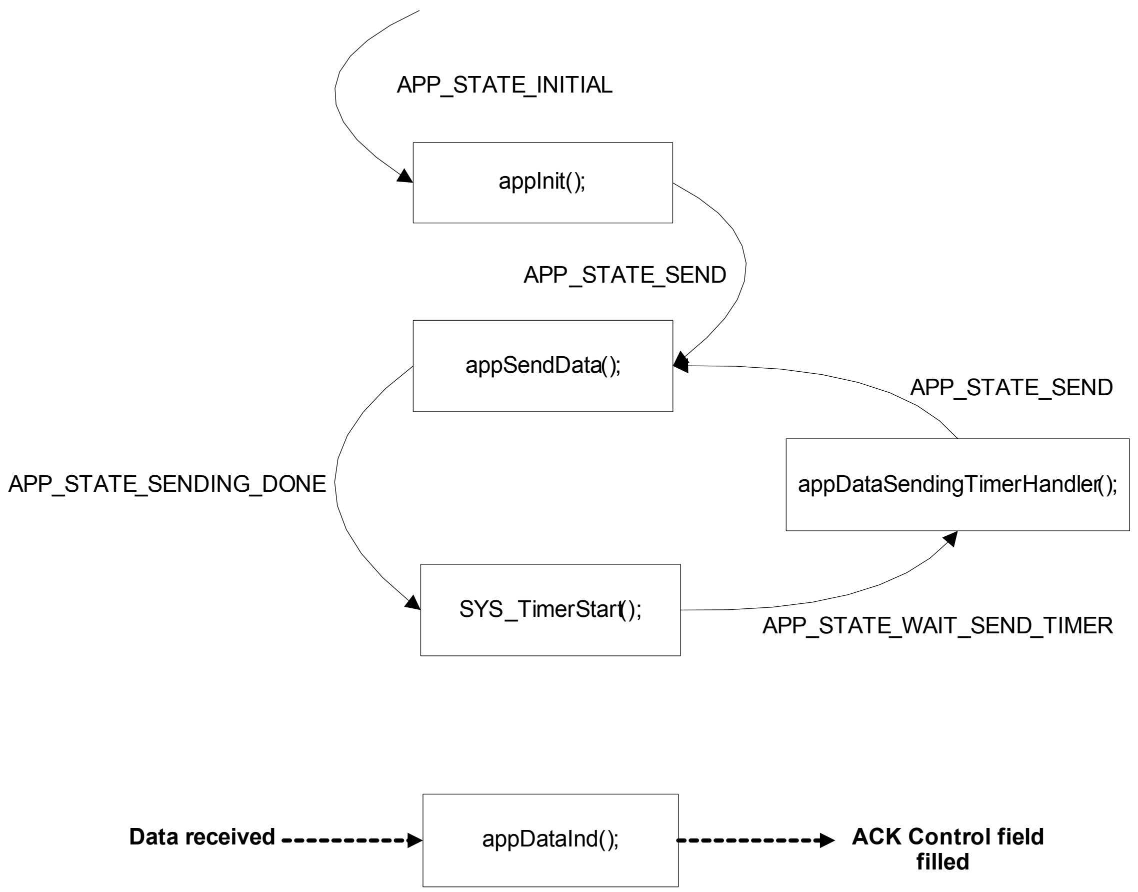
In the Gateway, Lightweight Mesh task is based on WSNDemo example from Lightweight Mesh Software Stack.

The `SYS_TaskHandler()` and `APP_TaskHandler()` are the two APIs of Lightweight Mesh task. They should be called as frequently as possible. The Lightweight Mesh stack low layer tasks are handled by calling `SYS_TaskHandler()`. The application layer is handled in `APP_TaskHandler()`.

The Gateway is a node in Ethernet network. It sends data to Ethernet by calling `appSendData()` from `APP_TaskHandler()` at a predefined interval.

The Gateway acts as Coordinator in Lightweight Mesh network. It receives data from other Lightweight Mesh devices from callback function `appDataInd()` and sends data back to Lightweight Mesh network by filling the Acknowledgment command frame in this function.

Figure 4-1. Lightweight Mesh Task Flow.

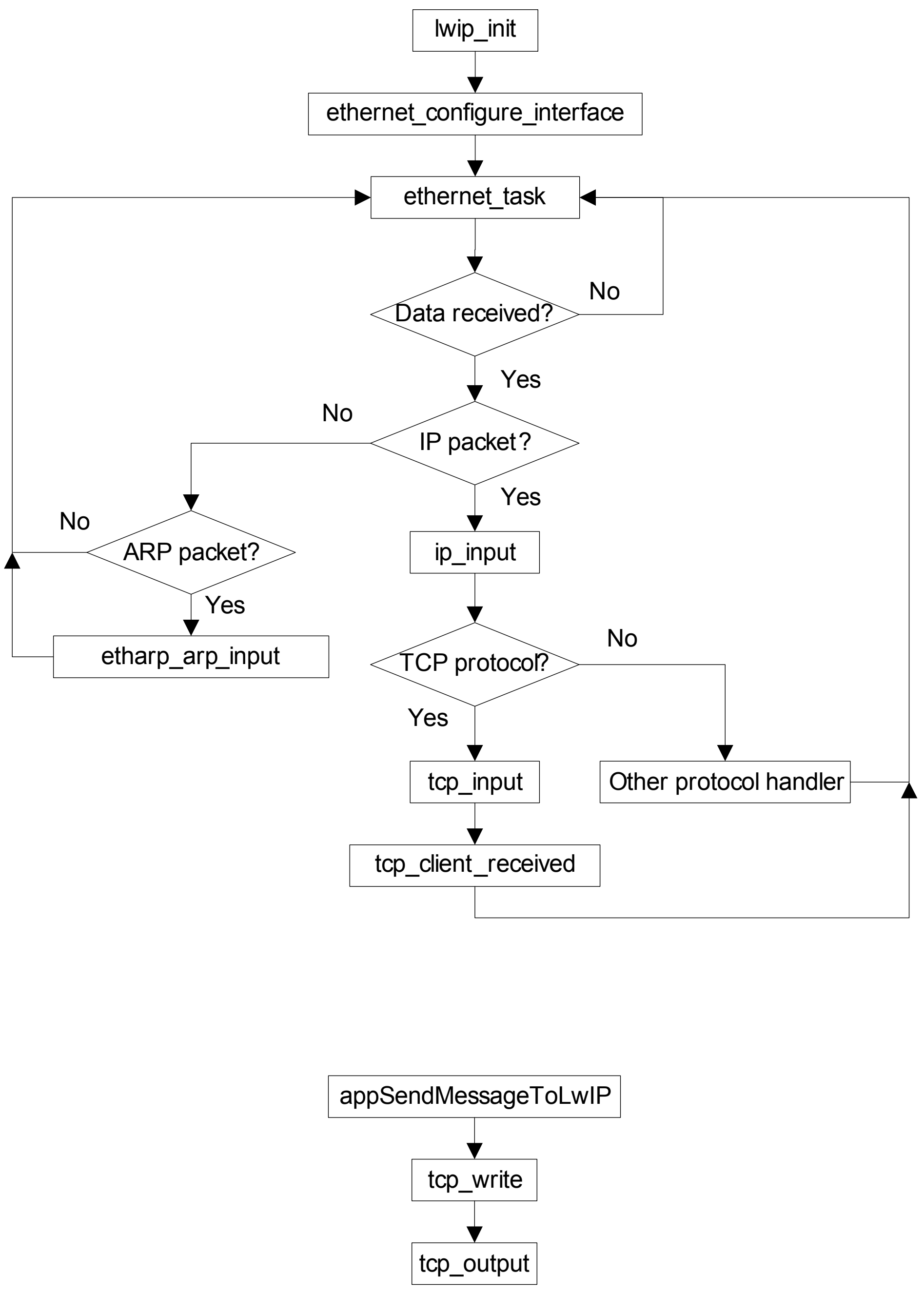


4.3 LwIP Task

The Gateway is configured as TCP client when LwIP network connection is initialized.

The LwIP task uses the `ethernet_task()` function to read data packet and run periodical tasks. This function should be called periodically. If Ethernet is used, the Gateway tries to connect to TCP server on a port specified in firmware. After successful connection, the Gateway is informed of incoming data through callback function `tcp_client_received()`. The data is sent to LwIP by calling `appSendMessageToLwIP()`.

Figure 4-2. LwIP Task Flow.



5. Main API Introduction

The main API's introduction will be divided into two parts: Lightweight Mesh Software Stack API, LwIP Stack API driver API. Each part will be introduced respectively in the following sections. Note the APIs described here are focusing on application layer. For complete API descriptions, refer to the stacks and their documents mentioned in Chapter 3.

5.1 Lightweight Mesh Software Stack API

As the Lightweight Mesh task is based on WSNDemo example, similar APIs in WSNDemo are used in the Gateway. Most of the Lightweight Mesh APIs used in Gateway are in WSNDemo.c.

The main APIs are as below.

- `SYS_Init()`

  It initializes Lightweight Mesh HAL, PHY, NWK layer and system timer. It's called from `board_init()`.

- `SYS_TaskHandler()`

  It's the core API of Lightweight Mesh. The PHY, NWK and system timer task handlers are called in this API.

- `APP_TaskHandler()`

  It's the application layer task handler of Lightweight Mesh.

The main purpose of using Lightweight Mesh on Gateway is to exchange data. Here are some APIs for sending and receiving data.

- `appDataInd()`

  The call back function registered by `NWK_OpenEndpoint()` in `appInit()`. It's called when valid data received from Lightweight Mesh low level layer.

  It also fills the Acknowledgment command frame control field by calling `NWK_SetAckControl()`. Thus the data sent back to Lightweight Mesh network is implemented in a very simple manner. The limit here is the control field is only one byte. If large amount of data needs to be transmitted, a dedicated API and structure should be used. Refer to the none-coordinator code in WSNDemo.c for example.

- `appSendData()`

  It's called by a predefined interval from `APP_TaskHandler()`. `appSendMessageToLwIP()` and `appSendMessage()` are called in this function to send data to LwIP Tx buffer respectively.

The timer API is used to generate fix interval through call back function. Here is the timer call back function in application layer.

- `appDataSendingTimerHandler()`

  It is registered by `SYS_TimerStart()` and called at predefined interval.

As the Gateway acts as Coordinator in Lightweight Mesh network, some APIs from the stack are not used. For more details about other APIs in Lightweight Mesh, refer to the software package and documents inside. The latest Lightweight Mesh Software Stack package can be downloaded from http://www.atmel.com/tools/LIGHTWEIGHT_MESH.aspx.

5.2 LwIP API Introduction

As no operating system is running, LwIP raw API has been used in the Gateway. And the Gateway is set as TCP client. Most of the LwIP APIs are in ethernet_sam.c.

The main APIs of LwIP are as below:

- `init_ethernet()`

  It initializes LwIP Ethernet interface and related hardware.

- `ethernet_task()`

  The LwIP Ethernet task handler. It polls the Ethernet tasks periodically. The specific application codes are implemented in several call back functions.

- `tcp_client_init()`

  It initializes the Gateway as TCP client. By default, static IP is assigned to Gateway and a port number is bound. In this function, it tries to connect to TCP server with the default parameters.

- `appSendMessageToLwIP()`

  This function sends data from Lightweight Mesh to LwIP. It fills the data buffer to be transferred. `tcp_write()` and `tcp_ouput()` are the actual functions sends data in LwIP.

Here is the list of callback functions used in the Gateway.

- `tcp_client_received()`

  It's the callback function invoked whenever a data packet is received from LwIP. For Gateway, it stores data received from TCP server in a buffer.

- `tcp_client_connected()`

  It's the callback function invoked when a TCP connection is established. It sends a string to TCP server after successful connection and set TCP client in receive state by registering a callback function `tcp_client_received()` in `tcp_recv()`.

- `tcp_err_handler()`

  It's the callback function for TCP error handler. It re-initializes Gateway to TCP client if connection aborted or connection reset occurs in LwIP.

- `status_callback()`

  It's the callback function for a status change in default network interface. It initializes the Gateway as TCP client by calling `tcp_client_init()`.

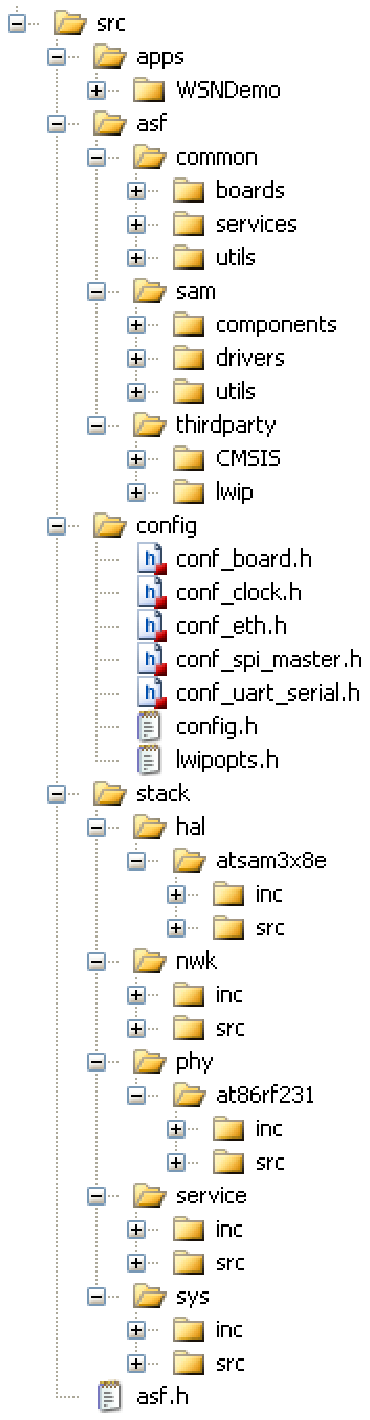
For more details about LwIP APIs, refer to LwIP stack.

6. Software Package Content

As mentioned before, the Gateway is developed based on ASF. The directory structure of the software package integrates ASF structure and Lightweight Mesh Software Stack structure. For details of the structure of ASF, refer to Atmel AVR4029: Atmel Software Framework - Getting Started. For the structure of Lightweight Mesh, refer to Atmel AVR2130: Lightweight Mesh Developer Guide.

The Gateway directory structure is shown as below:

Figure 6-1. The Gateway Directory Structure.



```
src
    apps
        WSNDemo
    asf
        common
            boards
            services
            utils
        sam
            components
            drivers
            utils
        thirdparty
            CMSIS
            lwip
    config
        conf_board.h
        conf_clock.h
        conf_eth.h
        conf_spi_master.h
        conf_uart_serial.h
        config.h
        lwipopts.h
    stack
        hal
            atsam3x8e
                inc
                src
        nwk
            inc
            src
        phy
            at86rf231
                inc
                src
        service
            inc
            src
        sys
            inc
            src
    asf.h
```

The directory details are described below:

- apps:
  - WSNDemo

    The Gateway application layer code. The main() is in WSNDemo.c.
- asf:
  - common
    - boards

      This directory contains the various board definitions shared between multiple architectures. As the Gateway is not a standard Atmel Kit, it's defined as USER_BOARD. And the board details are defined user_board.h and conf_board.h.
    - services

      ASF common services.
    - utils

      ASF common utilities.
  - sam
    - components

      Components supported by sam. Here the Gateway Ethernet PHY chip is supported by ASF.
    - drivers

      ASF sam drivers. It contains the low level drivers of sam peripherals.
    - utils

      ASF sam utilities.
  - thirdparty
    - CMSIS

      ARM Cortex® Microcontroller Software Interface Standard folder.
    - lwip

      LwIP stack folder.
- config:
  - conf_board.h

    The ASF config file of board. The Gateway board settings and macros are placed in this file.
  - conf_clock.h

    The ASF config file of clock. The Gateway clock settings can be configured here.
  - conf_eth.h

    The ASF config file of EMAC. The Gateway Ethernet hardware, MAC address and IP address (if static IP if used) etc. are defined in this file.
  - conf_spi_master.h

    The ASF config file of SPI in master mode.
  - conf_uart_serial.h

    The ASF config file of UART port. It's for debug purpose in the Gateway.
  - config.h

    The config file of Lightweight Mesh Software Stack. The device type and working channel are defined in this file.
  - lwipopts.h

    The config file of LwIP stack.

- stack:
  - hal
    - atsam3x8e

      Hardware abstraction layer of Lightweight Mesh Software Stack. In the Gateway, it reuses the low level drivers from ASF.
  - nwk

    Network layer of Lightweight Mesh Software Stack.
  - phy
    - at86rf231

      The radio PHY chip supported by the Gateway.
  - Services

    The Lightweight Mesh application services. OTA service is provided, but it’s not used in the Gateway. It can be removed from the project folder with no harm. It’s kept there for not breaking Lightweight Mesh Software Stack original structure.
  - Sys

    The Lightweight Mesh system services.

7. Footprint

Figure 7-1 and Figure 7-2 illustrate the CODE and RAM spaces that each module used in the software of the Gateway.

Figure 7-1. Lightweight Mesh to Ethernet Gateway CODE Footprint [KB].

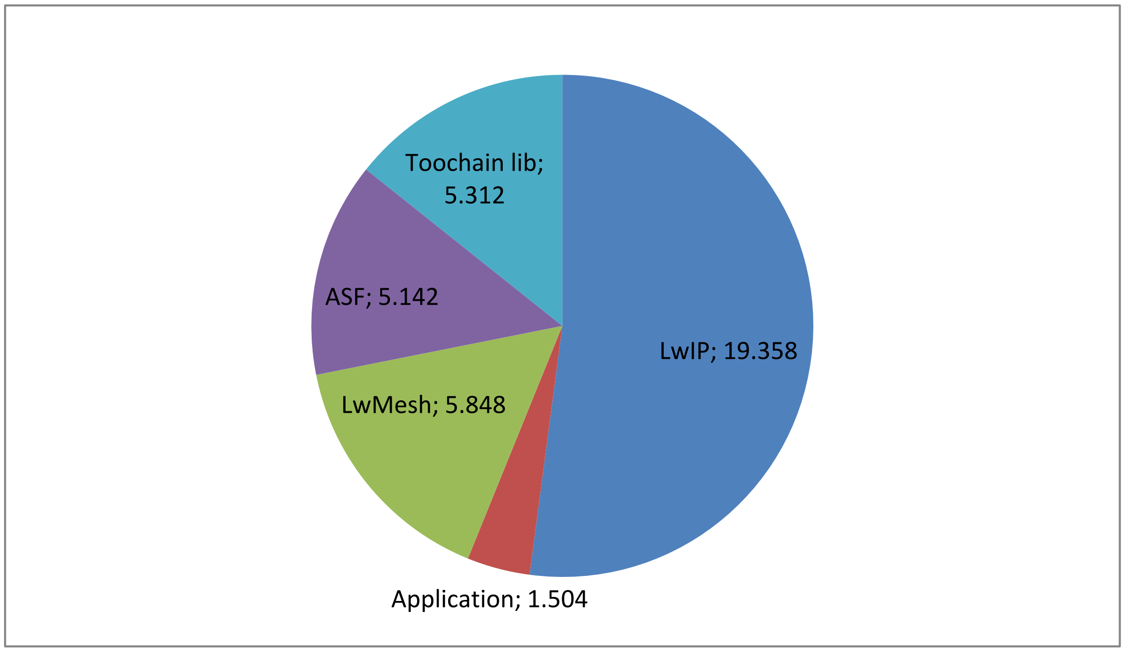
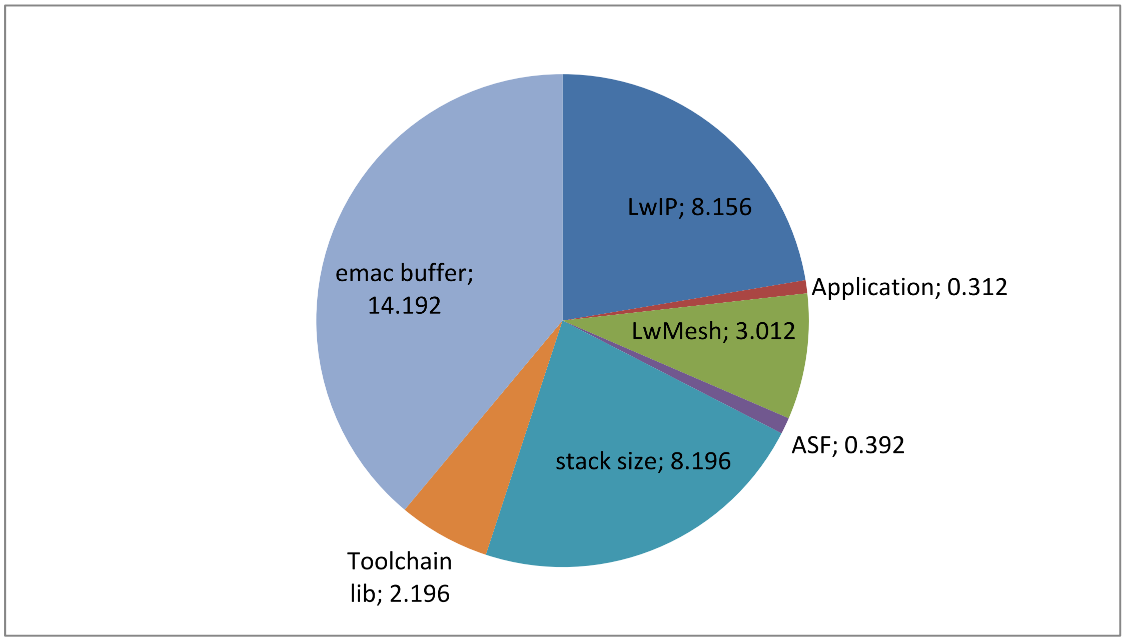


Figure 7-2. Lightweight Mesh to Ethernet Gateway RAM Footprint [KB].



The Lightweight Mesh and LwIP stack RAM usage should be configured in their corresponding configuration file according to real application.

8. Getting Started Guide

In this chapter, it gives step-by-step guide to setup the Gateway and run the preprogrammed firmware.

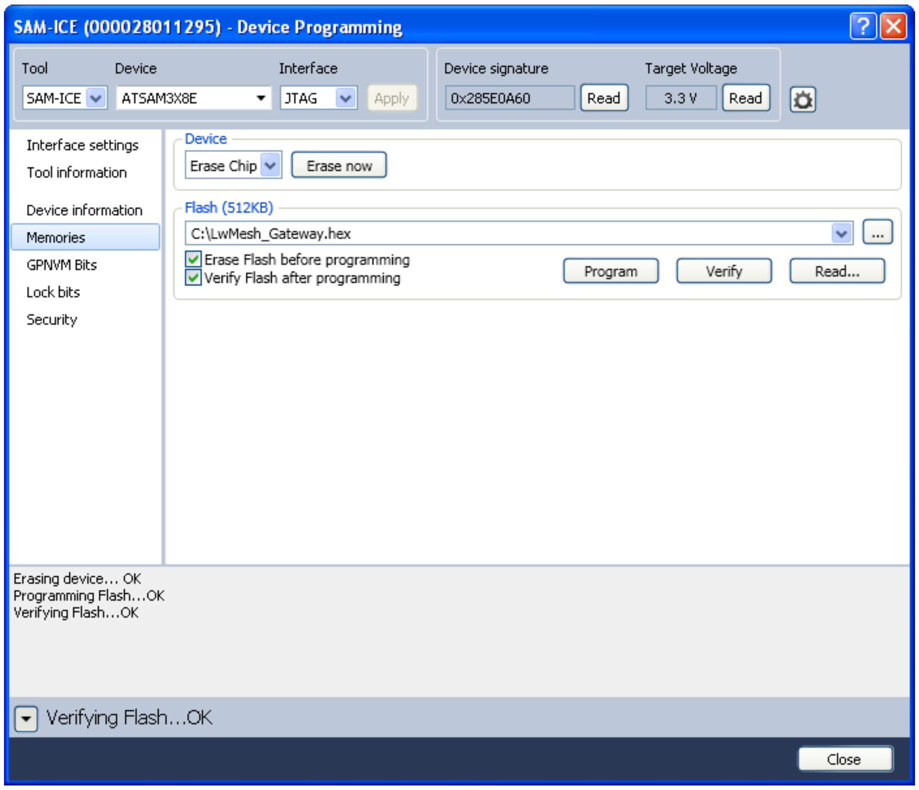
8.1 Programming the Gateway

Along with this document, two hex files are provided. One is for the Gateway (LwMesh_Gateway.hex) and the other is for Lightweight Mesh devices connecting to the Gateway (LwMesh_RCB128RFA1.hex).

To program the Gateway hex file, SAM-ICE adaptor mentioned in Chapter 2 is needed. The steps are as below.

1. Connect SAM-ICE to the SAM-ICE adapter.
2. Connect SAM-ICE adapter to the Gateway programming header J2.
3. Power the Gateway via the USB cable.
4. Open Atmel Studio and select menu “Tools -> Device Programming”.
5. Choose SAM-ICE for Tool, ATSAM3X8E for Device and JTAG for Interface, and then click “Apply” button.
6. Click the Device signature “Read” button to check if the connection is correct.
7. Select the Memories tab and then select the pre-built image for the Gateway from “…” in Flash section.
8. Click Program. If the pre-built image is downloaded to the board, message “Verifying Flash…OK” appears.

Figure 8-2. Programming the Gateway.


SAM-ICE (000028011295) - Device Programming

Tool: SAM-ICE
Device: ATSAM3X8E
Interface: JTAG
Device signature: 0x285E0A60
Target Voltage: 3.3 V

Flash (512KB): C:\LwMesh_Gateway.hex
☑ Erase Flash before programming
☑ Verify Flash after programming

```
Erasing device... OK
Programming Flash...OK
Verifying Flash...OK
```

Verifying Flash...OK

The hex file for Lightweight Mesh devices is running on RCB128RFA1. It acts as End Device in the network. LED (D4) on RCB128RFA1 can be monitored and controlled through the Gateway. Refer to Atmel AVR2131: Lightweight Mesh Getting Started Guide for the setup and programming the RCB128RFA1 Radio Control Board.

8.2 Connecting to Ethernet

In the preprogrammed firmware, the Gateway Ethernet is configured as below.

- TCP client
- Server IP: 192.168.1.105. Port: 8840
- Gateway static IP: 192.168.1.**101**. Sub net mask: 255.255.255.0. Default Gateway: 192.168.1.1

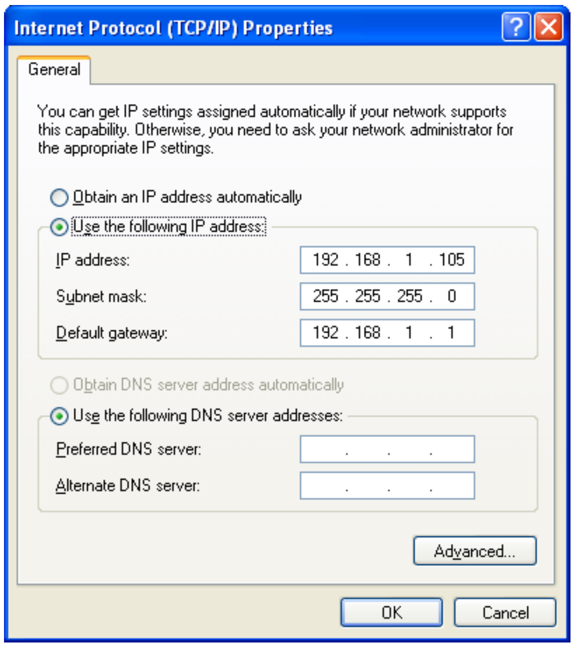
The parameters can be changed in the Gateway firmware, but the PC software "TCPServer" provided with this document is designed to work with the settings above.

8.2.1 Step-by-step guide

To directly connect the Gateway to PC via Ethernet, follow the steps below.

1. Configure PC IP address to 192.168.1.105, Sub net mask: 255.255.255.0, Default Gateway: 192.168.1.1 as shown in Figure 8-1.

Figure 8-1. PC IP address configuration.



Internet Protocol (TCP/IP) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 1 . 105

Subnet mask: 255 . 255 . 255 . 0

Default gateway: 192 . 168 . 1 . 1

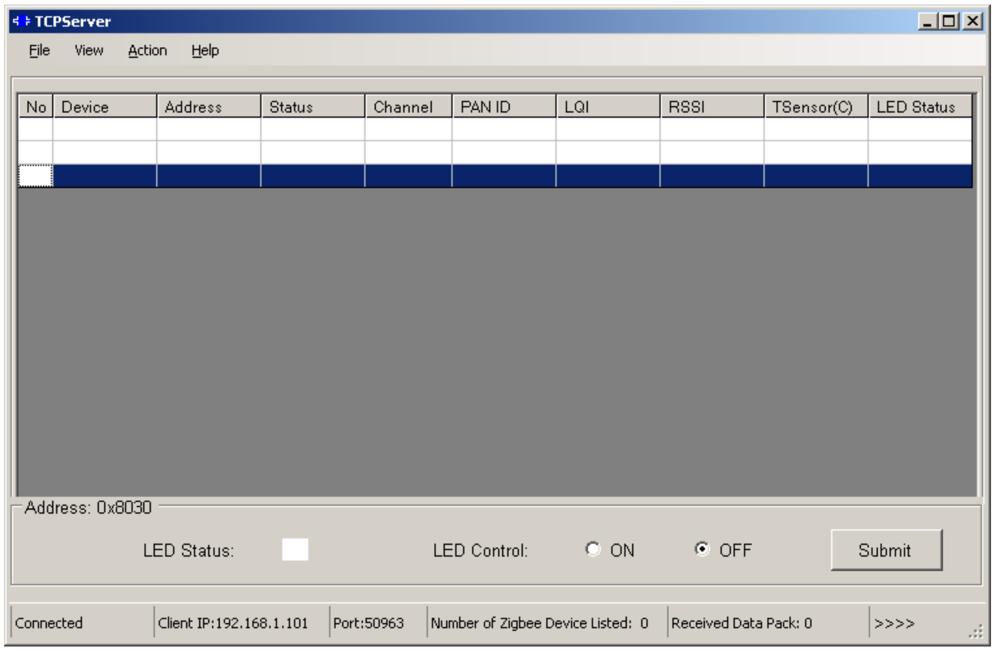
Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server: . . .

Alternate DNS server: . . .

2. Connect Ethernet cable between Gateway and PC.
3. Power on Gateway by connecting USB cable. Ethernet connection status is indicated by LED D2, D3 and D4 on Gateway. The red color of bi-color LED D7 on Gateway will be on when Lightweight Mesh network ready.
4. Open "TCPServer" in PC. If the Gateway and PC are setup correctly, "TCPServer" will report "Connected", "Client IP" and "Port" in the status bar as shown in Figure 8-2.

Figure 8-2. The Gateway connected to “TCPServer”.



No	Device	Address	Status	Channel	PAN ID	LQI	RSSI	TSensor(C)	LED Status

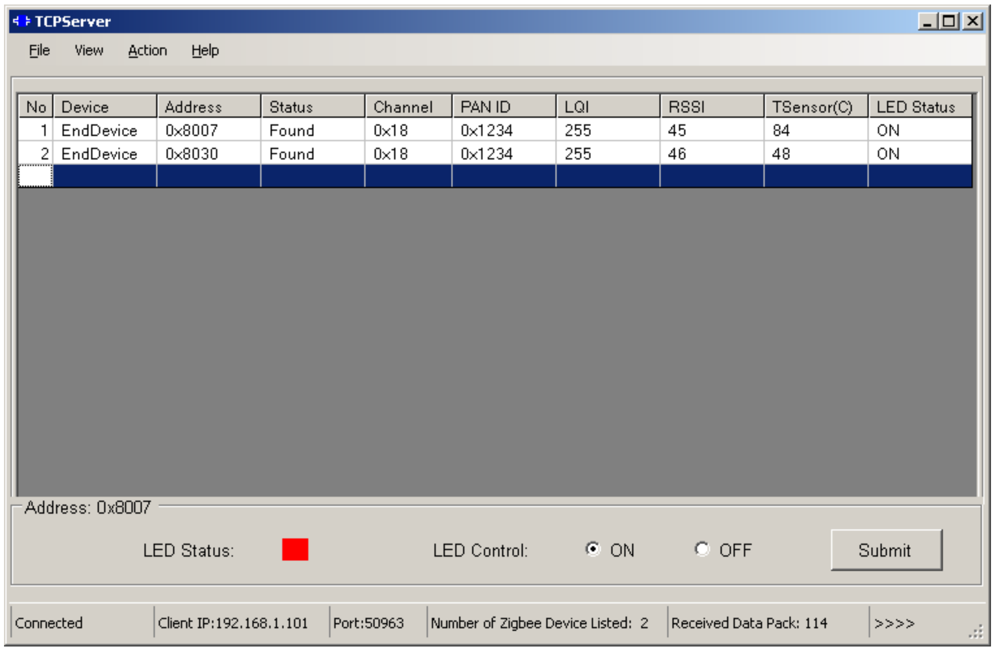
Address: 0x8030

LED Status: ⬜ LED Control: ( ) ON (● OFF) Submit

Connected | Client IP:192.168.1.101 | Port:50963 | Number of Zigbee Device Listed: 0 | Received Data Pack: 0 | >>>>

5. Power on the other Lightweight Mesh devices. If the Lightweight Mesh devices are setup correctly, the device information will be displayed in the main window of “TCPServer” as shown in Figure 8-3. The green color of bi-color LED D7 on Gateway will toggle each time it receives data from connected devices.

Figure 8-3. Lightweight Mesh devices connected via Ethernet.



No	Device	Address	Status	Channel	PAN ID	LQI	RSSI	TSensor(C)	LED Status
1	EndDevice	0x8007	Found	0x18	0x1234	255	45	84	ON
2	EndDevice	0x8030	Found	0x18	0x1234	255	46	48	ON

Address: 0x8007

LED Status: 🟥 LED Control: (● ON) ( ) OFF Submit

Connected | Client IP:192.168.1.101 | Port:50963 | Number of Zigbee Device Listed: 2 | Received Data Pack: 114 | >>>>

6. Click on one device in main window. The LED status on selected device is displayed in control window. Select “ON” or “OFF” and then click “Submit”, the LED on the device can be controlled accordingly.

8.3 PC software Menu

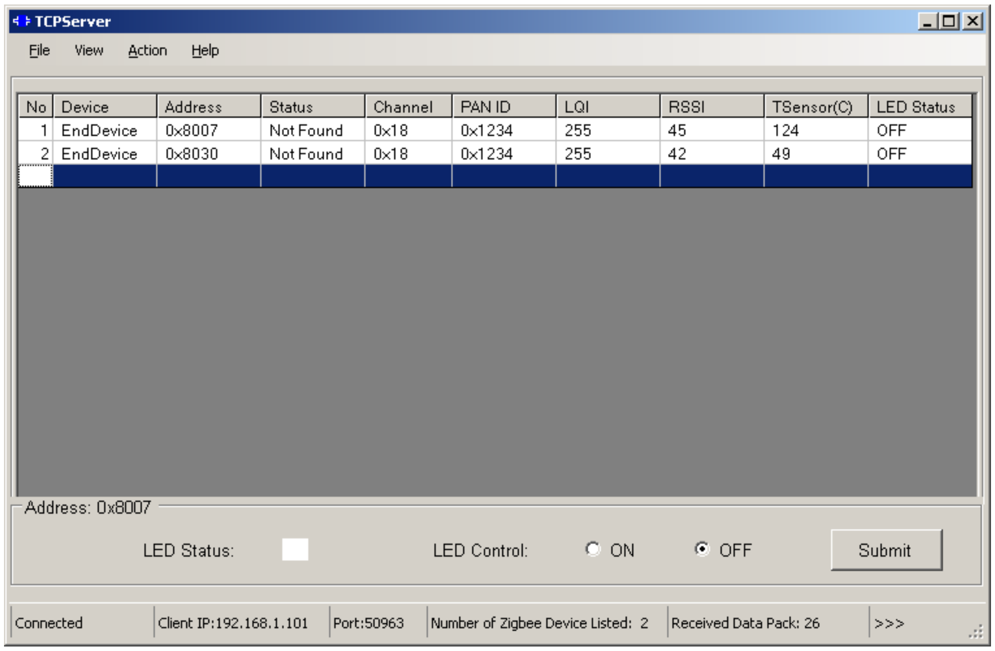
The TCPServer provided with this document is a simple GUI to demonstrate the Gateway reference design. In a real application, a more complicated GUI may be used.

For an overview of the PC software features refer to Section 3.3. The menu of TCPServer is simple. Only the menu items under "Action" are implemented. They are described as below.

- Refresh List: Refresh the device list manually
- Start Listening: Start listening on the default server port 8840
- Stop Listening: Stop listening on the default server port 8840

Whenever the device list is not updated or showing "Not Found" in main window as in Figure 8-4, the TCPServer can restart working by selecting "Stop Listening" and "Start Listening".

Figure 8-4. Device Not Found in TCPSerever.



No	Device	Address	Status	Channel	PAN ID	LQI	RSSI	TSensor(C)	LED Status
1	EndDevice	0x8007	Not Found	0x18	0x1234	255	45	124	OFF
2	EndDevice	0x8030	Not Found	0x18	0x1234	255	42	49	OFF

Address: 0x8007

LED Status: LED Control: ON [✓ OFF]

Connected | Client IP:192.168.1.101 | Port:50963 | Number of Zigbee Device Listed: 2 | Received Data Pack: 26

Appendix A. Additional Information

A.1 Lightweight Mesh Configuration

Table A-1 lists the Lightweight Mesh Software Stack configuration used in this reference design and this configuration can be modified in src/config/config.h.

Table A-1. Lightweight Mesh Options.

Option	Value	Description
APP_ADDR	0 - Coordinator	Node network address. It should be 0 for the Gateway
APP_CHANNEL	0x18	Radio transceiver channel. Valid range for 2.4GHz radios is 11 – 26 (0x0b – 0x1a)
APP_PAN_ID	0x1234	Network identifier
APP_ENDPOINT	1	Application main data communication endpoint
NWK_BUFFERS_AMOUNT	10	Number of buffers reserved for stack operation
HAL_UART_RX_FIFO_SIZE	4	UART RX buffer size

A.2 LwIP Configuration

Table A-2 lists the LwIP stack configuration in this reference design, and these configurations can be modified in src/config/lwipopts.h.

Table A-2. LwIP Options.

Option	Value	Description
MEM_SIZE	3*1024	The size of the heap memory
MEMP_NUM_PBUF	12	The number of memp struct pbufs
MEMP_NUM_TCP_PCB	2	The number of simultaneously active TCP connections
MEMP_NUM_TCP_PCB_LISTEN	1	The number of listening TCP connections
MEMP_NUM_TCP_SEG	9	The number of simultaneously queued TCP segments
PBUF_POOL_SIZE	6	The number of buffers in the pbuf pool
PBUF_POOL_BUFSIZE	512	The size of each pbuf in the pbuf pool
LWIP_TCP	1	Turn on TCP
TCP_WND	1500	The size of a TCP window
TCP_MSS	1500	TCP Maximum segment size
TCP_SND_BUF	2150	TCP sender buffer space
TCP_SND_QUEUELEN	(6 * (TCP_SND_BUF) + (TCP_MSS - 1))/(TCP_MSS)	TCP sender buffer space

Appendix B. Revision History

Doc. Rev.	Date	Comments
42165A	11/2013	Initial document release

Atmel | Enabling Unlimited Possibilities®