Bluetooth Mesh Hands-On Training Lab

KEY FEATURES

Learning the basics how light control(LC) models work

Adding sensor server model and sensor setup model to the light example

Developing the smart light application

Using smartphone to provision and configure a mesh node

Using the Bluetooth Mesh App to change the LC server parameters, which reflects the behaviors on controlling the lightness

This training lab shows how to use Silicon Labs Bluetooth Mesh SDK to develop the smart lighting application, then use the smartphone app – Bluetooth Mesh from Silicon Labs to provision and configure it. You will also have a chance to tune the parameters to change the behaviors how the lightness should be controlled. You will need the following items prepared to finish the lab:

* [Simplicity Studio v4](https://www.silabs.com/products/development-tools/software/simplicity-studio) (If already installed then make sure it is updated)
* The latest Bluetooth Mesh SDK which you can install via Simplicity Studio
* An iOS or Android smartphone with Bluetooth Mesh app installed
* ThunderBoard Sense 2

# Introduction

## Light Lightness Linear State

The Light Lightness Linear state represents the lightness of a light on a linear scale. The values for the state are defined in the following table.

|  |  |
| --- | --- |
| **Value** | **Description** |
| 0x0000 | Light is not emitted by the element. |
| 0x0001–0xFFFE | The lightness of a light emitted by the element. |
| 0xFFFF | The highest lightness of a light emitted by the element. |

Table 1: Light Lightness Linear state

## Light Control Model

Automated lighting control is handled by light controllers that are defined as state machines and feedback regulators.

Light controllers have inputs for collecting data from sensors, usually by receiving sensor messages.

Outputs from light controllers are represented as states that are bound to other states within an element. In this lab, the controller that controls light level has its output state bound with the Light Lightness Linear state.

Figure 1 illustrates the principles of operation of a Light Lightness controller.

地图的截图

描述已自动生成

Figure 1. Operation of a Light Lightness controller

The controller has eight phases of operation:

1. Off – the controller is turned off and does not control lighting

2. Standby – the controller is turned on and awaits an event from an occupancy sensor or a manual switch

3. Fade On – the controller has been triggered and gradually transitions to the Run phase, gradually dimming the lights up.

4. Run – the lights are on and the timer counts down (but may be retriggered by a sensor or a switch event)

5. Fade – the Run timer has expired and the controller gradually transitions to the Prolong state

6. Prolong – the lights are at a lower level and the timer counts down (but may be retriggered by a sensor or a switch event)

7. Fade Standby Auto – the controller gradually returns to the Standby state

8. Fade Standby Manual – the controller gradually returns to the Standby state after external event

**Placeholder for LC states and properties introduction**

# Lab Basics

As introduced in section 1.2, the LC server model takes sensor messages as input and output the lightness level. Below is the detailed information about the inputs and output.

**Inputs:**

* **Ambient light** – there is an ambient light sensor on the Thunderboard Sense 2 board, we need to sample the sensor data and send it to the LC server. The LC server regulates the light lightness output against the reported ambient lightness value to make sure that the ambient lightness equals the target level. The lightness regulating follows the formula in chapter 6.2.6 - Light LC PI Feedback Regulator of Mesh Model v1.0.1 (Hyper link).
  + If the ambient lightness level is higher than the expected level, then the LC server will try to dim down the light lightness until the real ambient level meets the expected level or 0%.
  + If the ambient lightness level is lower than the expected level, then the LC server will try to dim up the light lightness until the real ambient level meets the expected level or 100%.
* **Occupancy state** – In this lab, we will use the People Count to represent the occupancy state. People count being 0 represents non-occupancy, while equals or greater than 1 represents occupancy. Given that there isn’t a people count sensor on Thunderboard Sense 2, we will use the buttons instead.
  + People count increments when pressing the button 0.
  + People count decrements when pressing the button 1.
  + If occupancy is reported, the LC server will start the state machine to change the light from Off or Standby state to Run state if it’s not yet in run state.
  + If no occupancy is reported for a certain time, the LC server will start transitioning the light from Run state to Standby state if it’s not in Standby state.
  + If non-occupancy is reported, the LC server will transition the light from Run state to Standby state if it’s not in Standby state.

**Output:**

* **Light Lightness** – There are 4 lights on the Thunderboard Sense 2, the LC server will regulate the lightness of these 4 lights according to the sensor inputs.

Figure 2 shows the diagram of the project.

手机屏幕截图

描述已自动生成

Figure 2. Diagram of the Project

# Finish the Lab

## Import the Project

You will get a customized light example for Thunderboard Sense 2 as the starting point, which is created from Simplicity Studio based on the light example in the Bluetooth Mesh SDK. The project contains the skeleton of the LC server and the sensor models. Follow below steps to import it to your Simplicity Studio.

1. Placeholder for importing guidance.

## Add Sensor Models

Like mentioned in the lab basics, the LC server needs the ambient light level and the occupancy states as input, while the light example doesn’t include them natively. So, we need to add the sensor models to the project. Follow below steps to add the sensor server model and sensor setup server model to your project.

1. Placeholder for importing guidance.

## Complete the Project

### LC server

All LC Server state resides in and is own by the Bluetooth Mesh stack. The state update notification events to the application are informational - the application is not required to react to them. The light example saves the LC Server state in persistent storage and set the states in the LC Server following a restart. You can check the implementation in the light\_controller.c file to get all the information how to initialize and integrate the LC server into other projects.

### Sensor Server

We already add the sensor server model and the sensor server setup model to the DCD, to make it works, we still need to do more work on it, they are:

* Initialize and configure sensor server model - You can find the implementation from the btmesh-soc-sensor-server example. The skeleton project has already implemented it so you don’t need to work on it.
* Data sampling – This is not the target for the lab so all the necessary functions for sampling the sensor data including the button handling are already provided.
* Send sensor messages to the LC server – you need to complete the code for sending sensor messages to the LC server, you can find the places which are marked by “TODO” in the lab/models/sensor\_server.c and add your code there.

# Provision and Configure the Node

We will use the Bluetooth Mesh App to provision and configure the node.

Follow below steps.

1. Placeholder

# Testing the Project

## Lightness vs Ambient Level

Placeholder for setting the target by lightness or ambient level

## Occupancy vs Non-Occupancy

Placeholder for how to report both and check the light.

## LC Properties

Placeholder for how the change of LC properties results in controlling the light.