

Huawei IoT Certification Training

# HCIP-IoT Developer Huawei Certified ICT Professional-Internet of Things Developer Lab Guide

ISSUE: 2.5



Huawei Technologies Co., Ltd.

Copyright © Huawei Technologies Co., Ltd. 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## Trademarks and Permissions



and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

# Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129  
People's Republic of China

Website: <http://e.huawei.com>

## Huawei Certification System

Huawei Certification follows the "platform + ecosystem" development strategy, which is a new collaborative architecture of ICT infrastructure based on "Cloud-Pipe-Terminal". Huawei has set up a complete certification system consisting of three categories: ICT infrastructure certification, Platform and Service certification and ICT vertical certification. Huawei certification is the only all-range technical certification in the industry. Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE). The certification covers all ICT fields and complies with the industry trend of ICT convergence. With its leading talent development system and certification standards, Huawei is committed to fostering new ICT talent in the digital era and building a sound ICT talent ecosystem.

Huawei Certified ICT Professional-Internet of Things Developer (HCIP-IoT Developer) is designed for field engineers from Huawei and representative offices or anyone who wants to learn how to use Huawei IoT product technologies (university students or IoT practitioners). The HCIP-IoT Developer certification covers the HUAWEI CLOUD IoT platform, Huawei LiteOS, and IoT communication technologies (wireless communication and IoT gateway).

Stay on the cutting edge with Huawei certification.

# Huawei Certification System



## Huawei Certification



# About This Document

---

## Overview

This document is intended for candidates who are preparing for the HCIP-IoT Developer exam or anyone who wants to understand Huawei's full-stack IoT solution.

## Description

This document consists of 8 parts, including a basic Huawei LiteOS exercise, case exercises for reporting data to MQTT Broker, and a comprehensive exercise.

- Exercise 1 is an exercise for operating system kernel implementation based on Huawei LiteOS. It helps you master the use of Huawei LiteOS functional modules.
- Exercise 2 is a BearPi exercise about how to control the LCD and blink the LED. It helps you understand the working principles of the development board.
- Exercise 3 is a device-cloud synergy exercise using Wi-Fi for a smart agriculture case. It helps you master smart agriculture development.
- Exercise 4 is a device-cloud synergy exercise using Wi-Fi for smart smoke detectors. It helps you master smart smoke detector development.
- Exercise 5 is a device-cloud synergy exercise using Wi-Fi for smart manhole covers. It helps you master smart manhole cover development.
- Exercise 6 is a device-cloud synergy exercise using Wi-Fi for human body sensors. It helps you master human body sensor development.
- Exercise 7 is a device-cloud synergy exercise using Wi-Fi for vending machines. It helps you master vending machine development.
- The comprehensive exercise aims to implement device-cloud synergy using smart logistics and smart street lamps. It helps you master device-cloud synergy development based on LiteOS.

## Background Knowledge Required

This course is specifically designed for the HCIP-IoT Developer certification. To complete this course, you need to:

- Have basic C language programming ability

## Lab Environment Overview

### Device Introduction

To meet exercise requirements, the following environment configurations are recommended:

The following table lists the devices required.

Device Name	Model
IoT development board suite	BearPi development board suite: 1. BearPi-IoT motherboard 2. Communications expansion board: Wi-Fi 3. Case expansion boards: smart agriculture, smart smoke detector, smart logistics, and smart street lamp
MQTT Broker	EMQ X Broker 4.2.0

## Exercise Environment Preparation

### Device Check

Before the exercise, each group of candidates should check whether the devices are ready. The following table lists the exercise devices.

Device Name	Quantity	Remarks
Laptop	One for each candidate	The device can access the public network.
IoT development board suite	One set for each group	BearPi development suite

# 1 Huawei LiteOS Kernel Exercise

---

## 1.1 Introduction

### 1.1.1 About This Exercise

In this exercise, you will use LiteOS Studio to develop IoT devices and use LiteOS to control the IoT development board.

### 1.1.2 Objectives

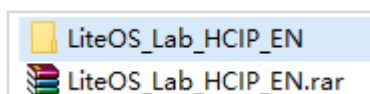
- Master how to use LiteOS Studio.
- Master how to execute LiteOS tasks.
- Become familiar with the LCD.
- Become familiar with the LED and buttons of the development board.

## 1.2 Tasks

### 1.2.1 Opening a LiteOS Project

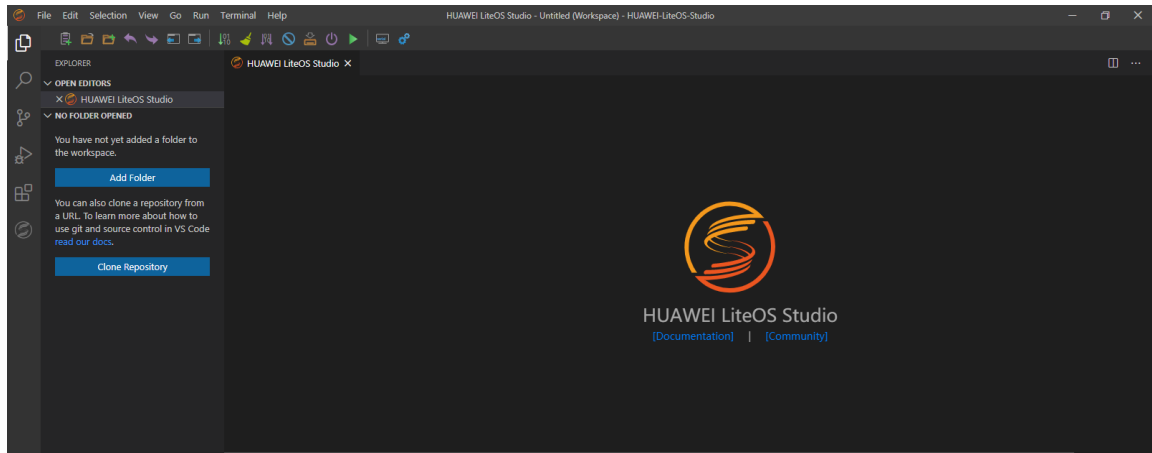
Step 1 Decompress the downloaded project.

Decompress the **LiteOS\_Lab\_HCIP\_EN.rar** file to the root directory of any disk. Ensure that the path does not contain Chinese characters or spaces.

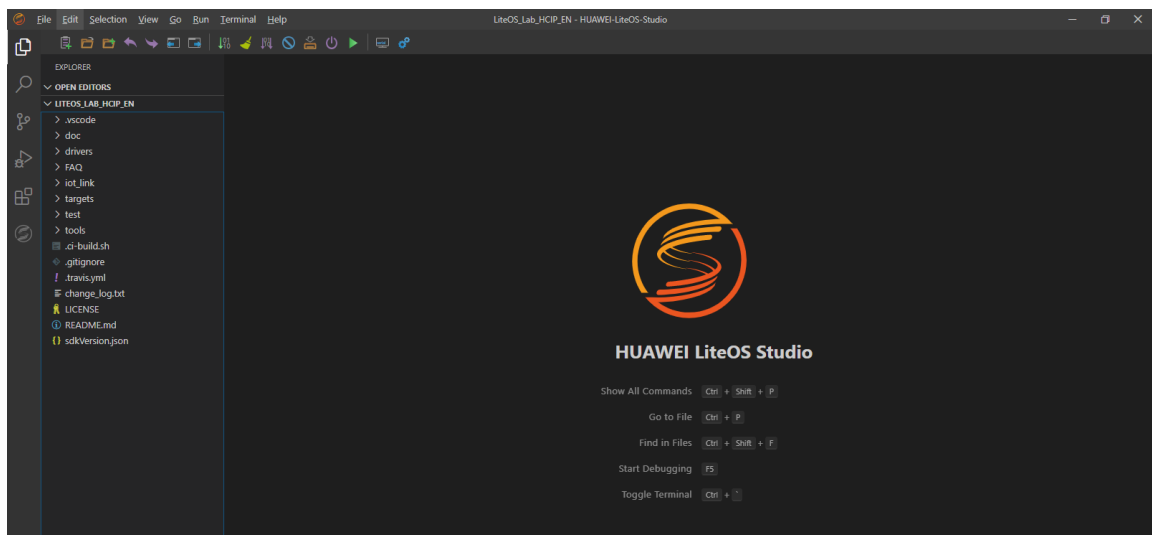


Step 2 Open the project.

Open the installed LiteOS Studio.



Choose **File > Open Folder** in the upper left corner and select the **LiteOS\_Lab\_HCIP\_EN** folder.

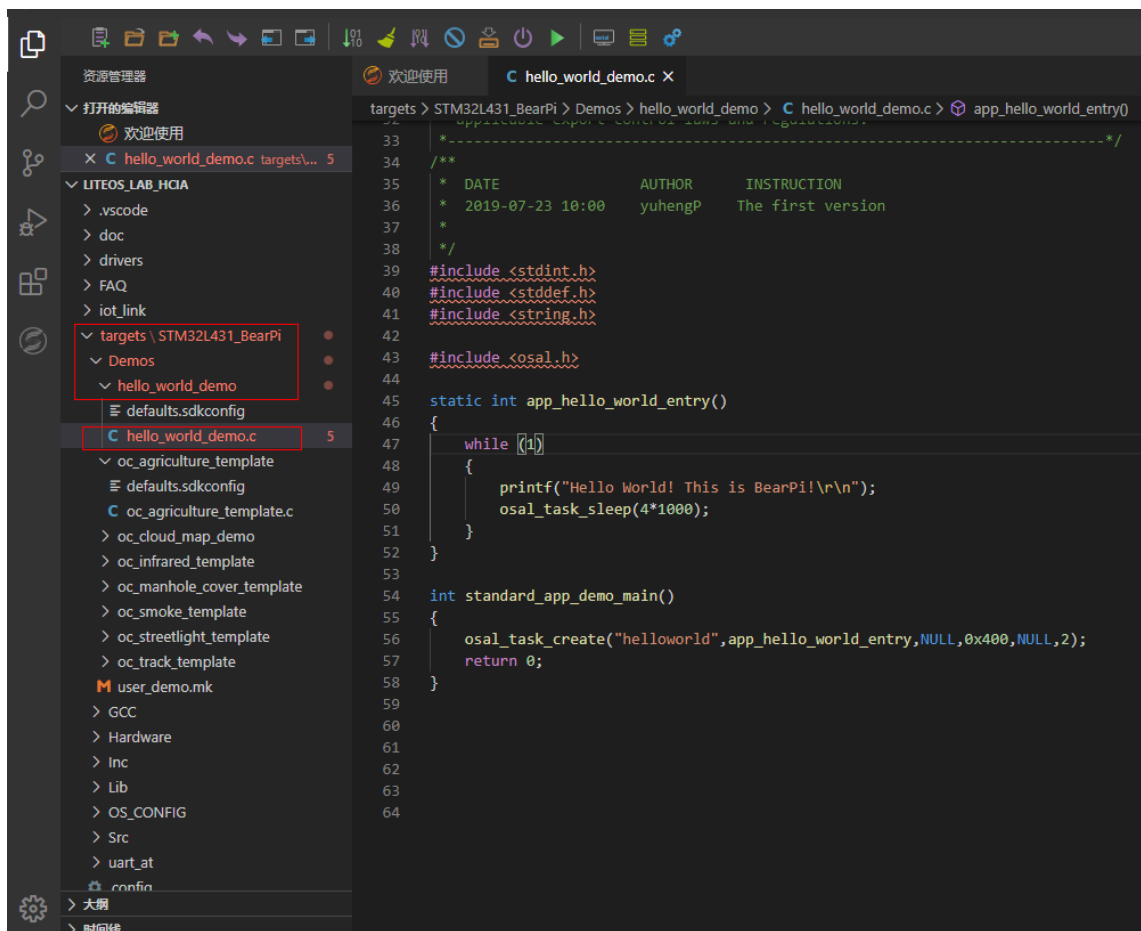


## 1.2.2 Running the HelloWorld Task

Step 1 Open **hello\_world\_demo.c**.

Choose **LiteOS\_Lab\_HCIP\_EN > targets > STM32L431\_BearPi > Demos > hello\_world\_demo > hello\_world\_demo.c**.





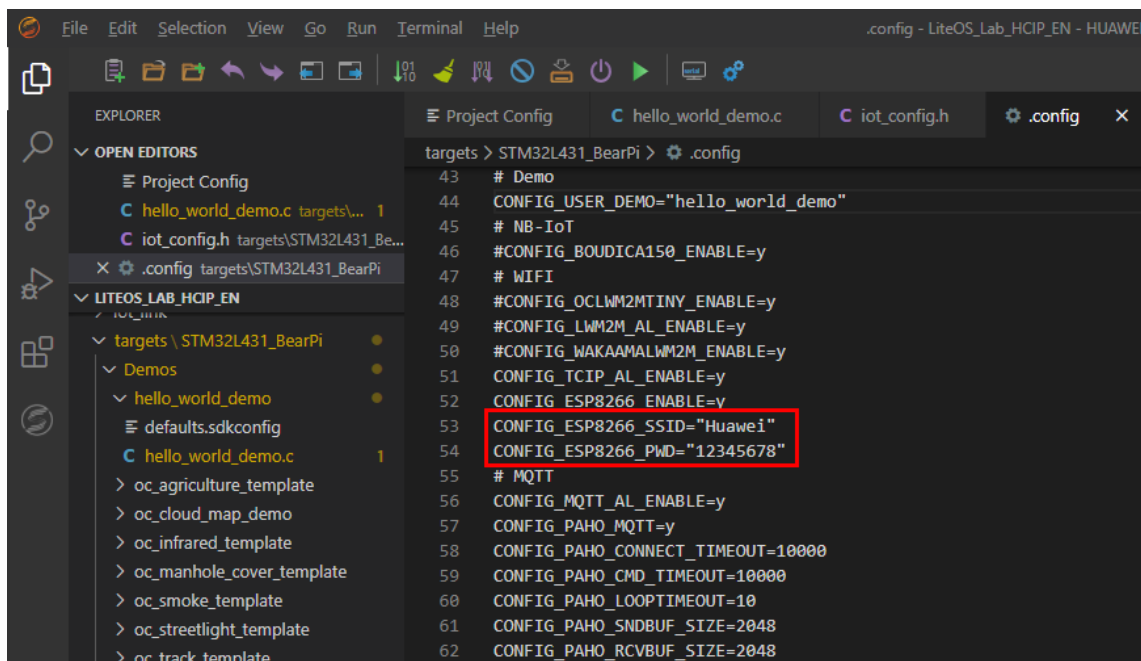
## Step 2 Modify the .config file.

Choose **targets > STM32L431\_BearPi > .config**.

Set **CONFIG\_ESP8266\_SSID** to the Wi-Fi username.

Set **CONFIG\_ESP8266\_PWD** to the Wi-Fi password.

Press **Ctrl+S** to save the .config file.



```

43 # Demo
44 CONFIG_USER_DEMO="hello_world_demo"
45 # NB-IoT
46 #CONFIG_BOUDICA150_ENABLE=y
47 # WIFI
48 #CONFIG_OCLWM2MTINY_ENABLE=y
49 #CONFIG_LWM2M_AL_ENABLE=y
50 #CONFIG_WAKAAMALWM2M_ENABLE=y
51 CONFIG_TCIP_AL_ENABLE=y
52 CONFIG_ESP8266_ENABLE=y
53 CONFIG_ESP8266_SSID="Huawei"
54 CONFIG_ESP8266_PWD="12345678"
55 # MQTT
56 CONFIG_MQTT_AL_ENABLE=y
57 CONFIG_PAHO_MQTT=y
58 CONFIG_PAHO_CONNECT_TIMEOUT=10000
59 CONFIG_PAHO_CMD_TIMEOUT=10000
60 CONFIG_PAHO_LOOPTIMEOUT=10
61 CONFIG_PAHO_SNDBUF_SIZE=2048
62 CONFIG_PAHO_RCVBUF_SIZE=2048

```

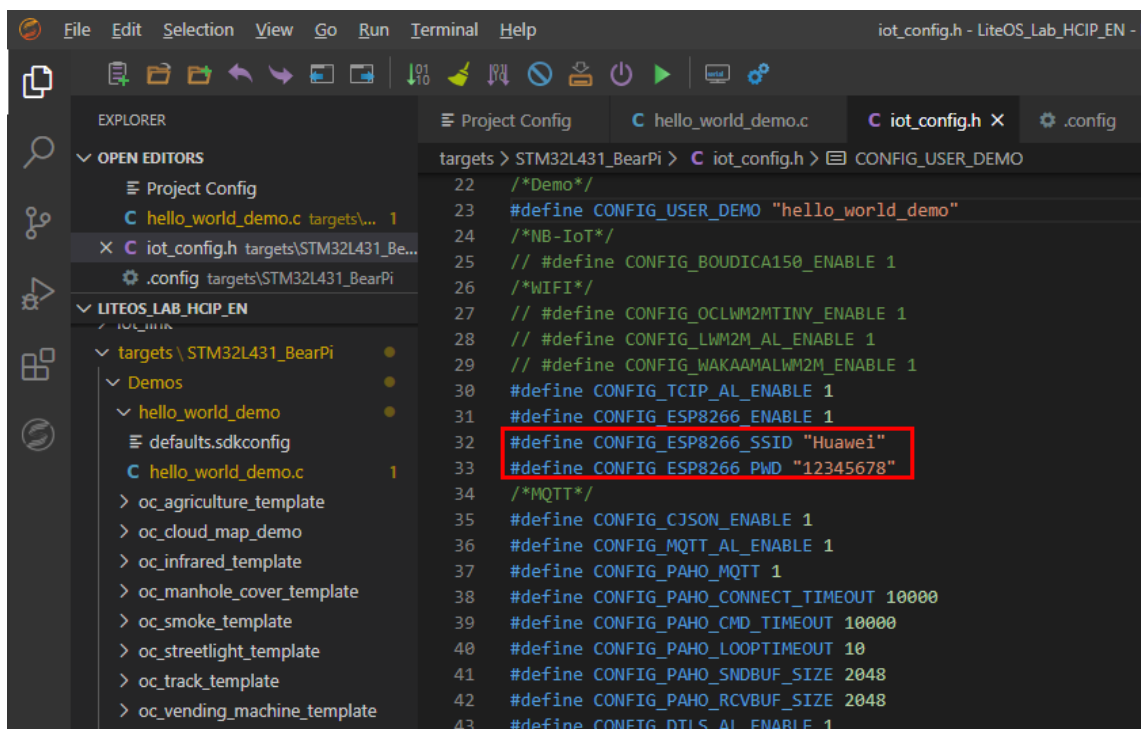
Step 3 Modify the `iot_config.h` file.

Choose `targets > STM32L431_BearPi > iot_config.h`.

Set `CONFIG_ESP8266_SSID` to the Wi-Fi username.

Set `CONFIG_ESP8266_PWD` to the Wi-Fi password.

Press `Ctrl+S` to save the `iot_config.h` file.



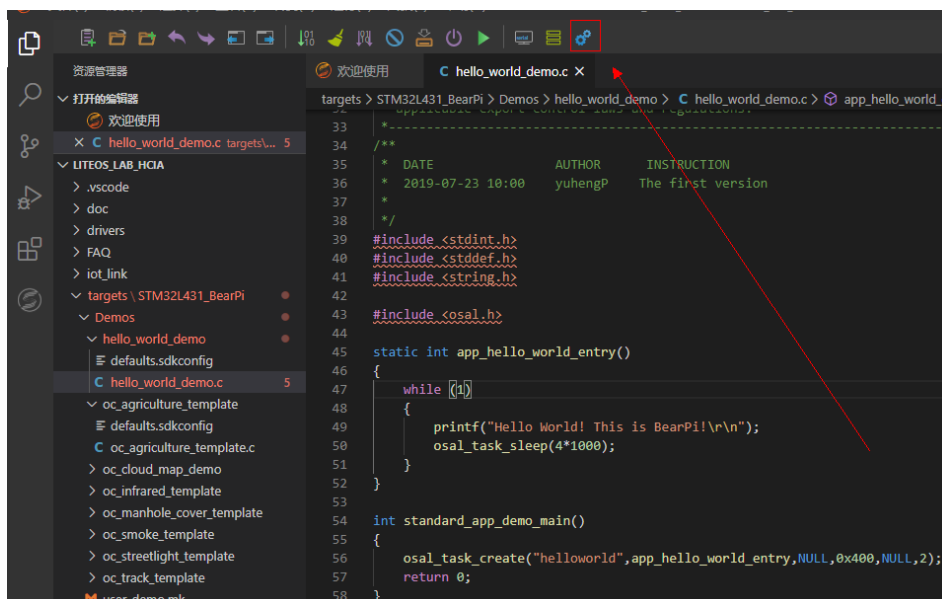
```

22 /*Demo*/
23 #define CONFIG_USER_DEMO "hello_world_demo"
24 /*NB-IoT*/
25 // #define CONFIG_BOUDICA150_ENABLE 1
26 /*WIFI*/
27 // #define CONFIG_OCLWM2MTINY_ENABLE 1
28 // #define CONFIG_LWM2M_AL_ENABLE 1
29 // #define CONFIG_WAKAAMALWM2M_ENABLE 1
30 #define CONFIG_TCIP_AL_ENABLE 1
31 #define CONFIG_ESP8266_ENABLE 1
32 #define CONFIG_ESP8266_SSID "Huawei"
33 #define CONFIG_ESP8266_PWD "12345678"
34 /*MQTT*/
35 #define CONFIG_CJSON_ENABLE 1
36 #define CONFIG_MQTT_AL_ENABLE 1
37 #define CONFIG_PAHO_MQTT 1
38 #define CONFIG_PAHO_CONNECT_TIMEOUT 10000
39 #define CONFIG_PAHO_CMD_TIMEOUT 10000
40 #define CONFIG_PAHO_LOOPTIMEOUT 10
41 #define CONFIG_PAHO_SNDBUF_SIZE 2048
42 #define CONFIG_PAHO_RCVBUF_SIZE 2048
43 #define CONFIG_DTLS_AL_ENABLE 1

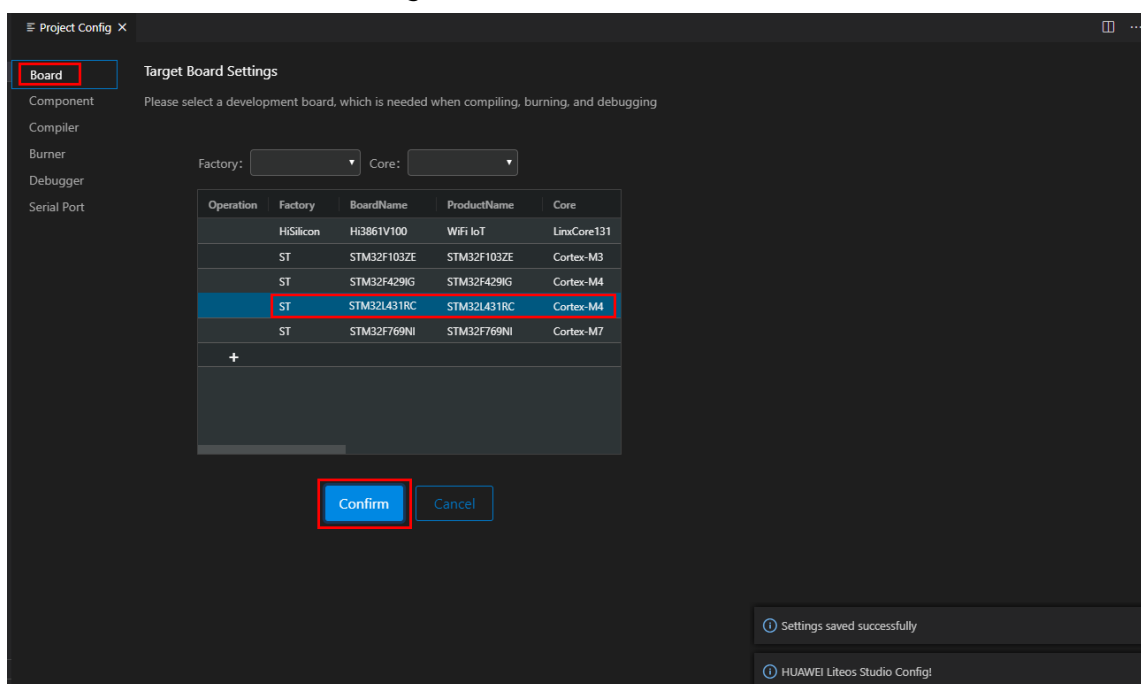
```



Step 4 Configure the project.

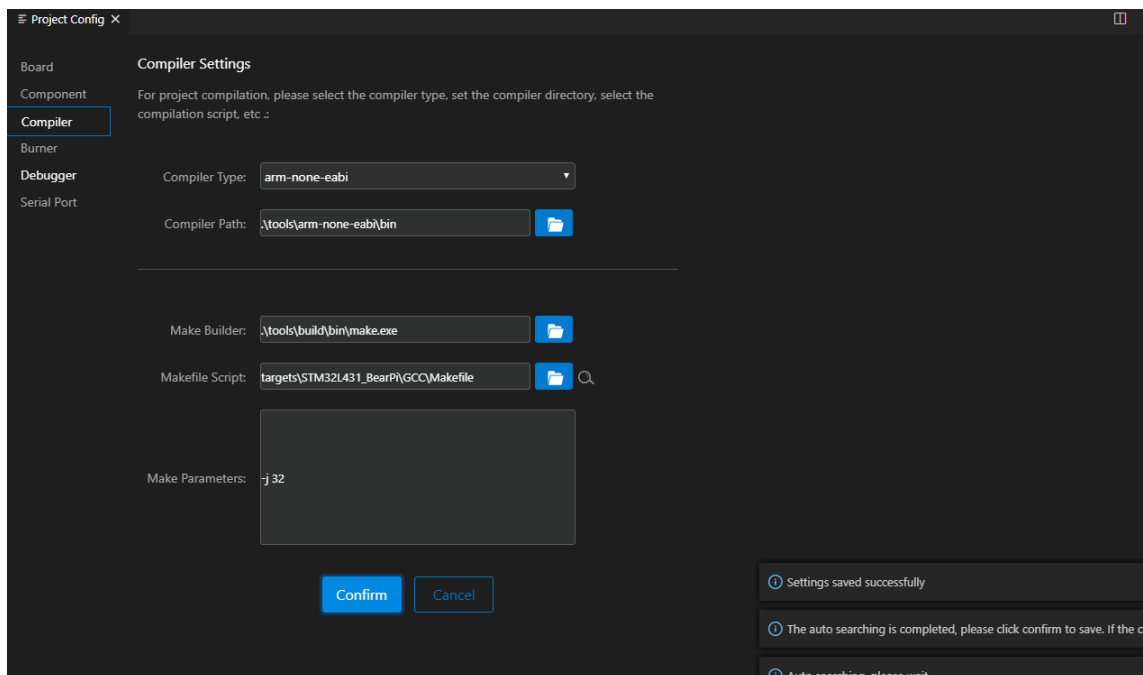
Click  on the toolbar.



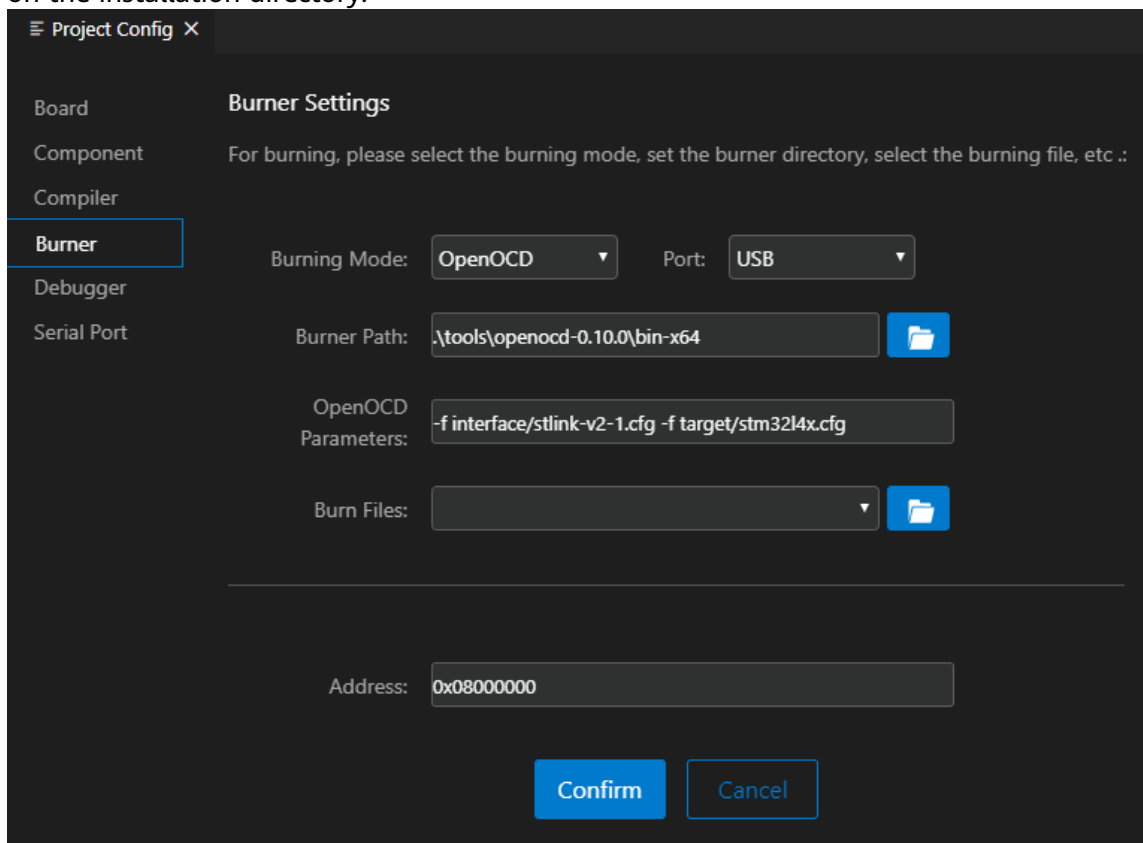
Select **STM32L431RC** as the target board and click **Confirm**.




Choose **Compiler**, click  on the right of **Makefile Script**, and click **Confirm**. If the Makefile script cannot be found, click  and select the Makefile script in the **targets\STM32L431\_BearPi\GCC** directory of the project. Set **Make Builder** based on the installation directory and click **Confirm**.

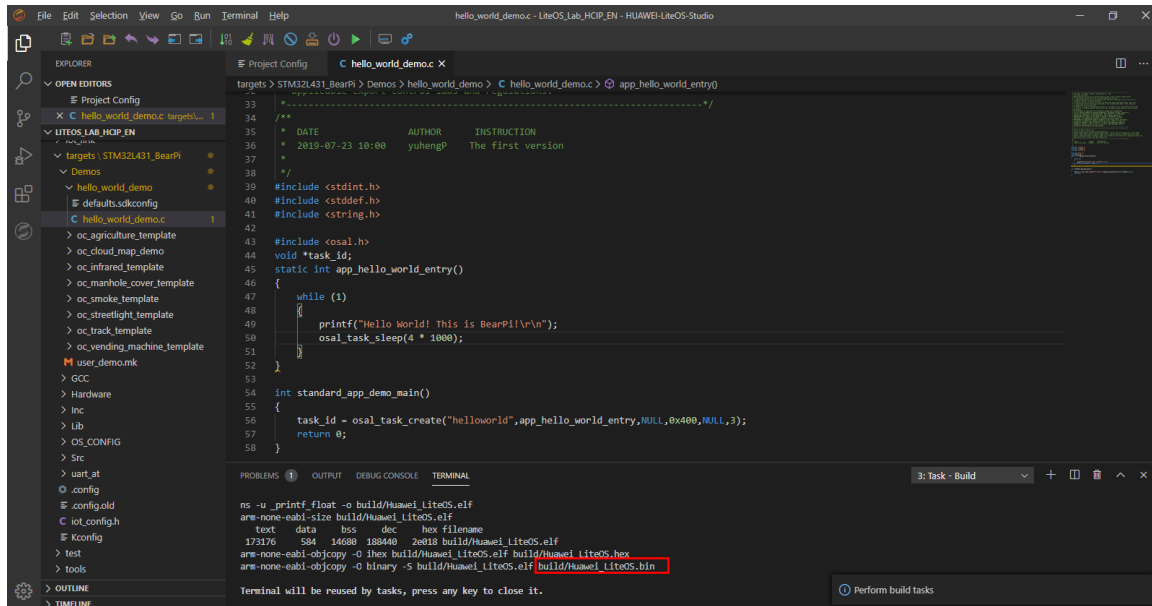


Choose **Burner**, set **Burning Mode** to **OpenOCD**, and click **Confirm**. Set **Burner Path** based on the installation directory.



**Step 5** Compile the program.

Click  on the toolbar and wait until the compilation is complete. A message is displayed indicating that the **Huawei\_LiteOS.bin** file is generated.

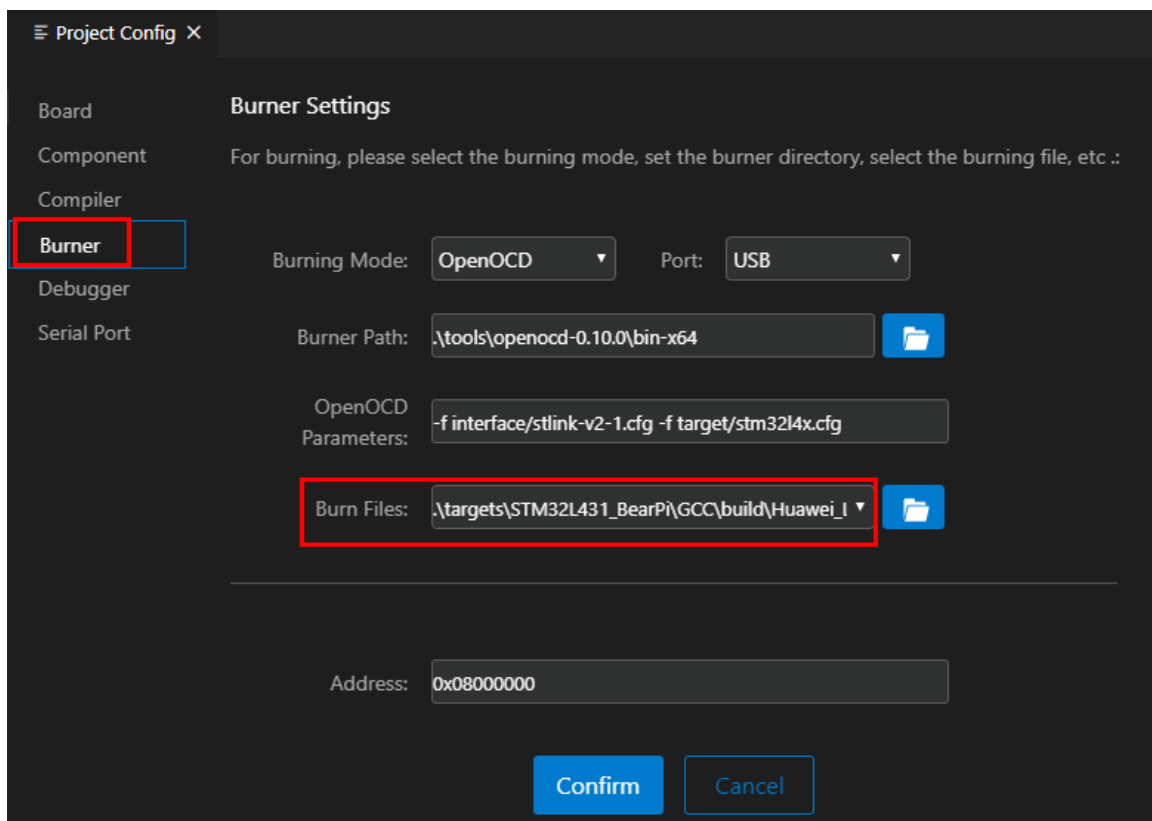



### Step 6 Configure the development board.

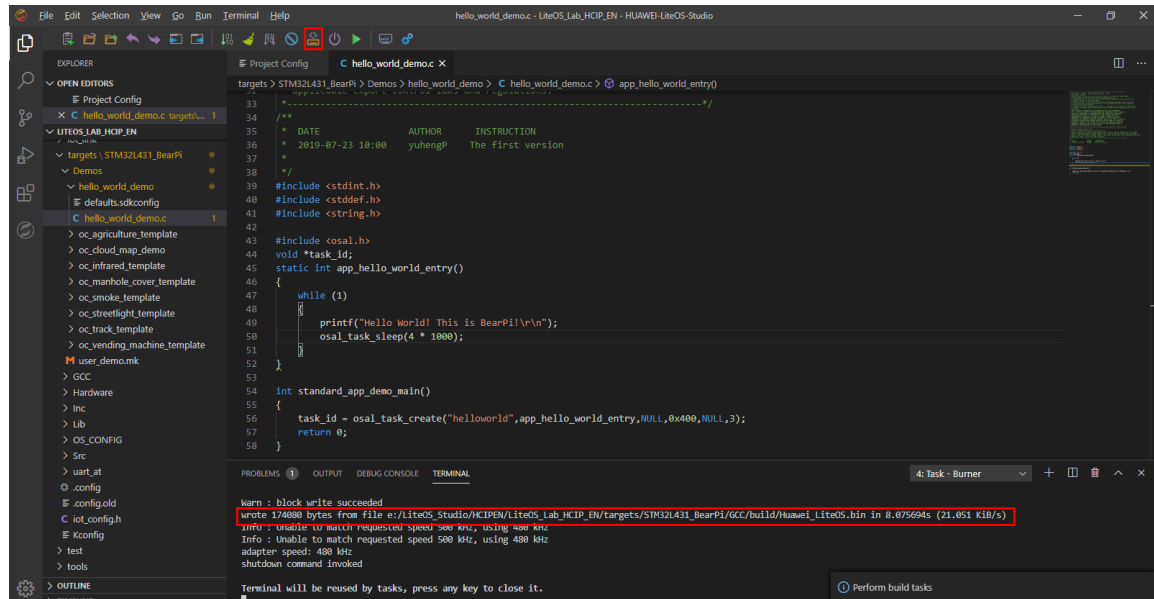
Set the switch of the serial port mode to AT<->MCU, and connect the development board to the PC using a USB cable.

### Step 7 Burn the program.


Access **Project Config**, choose **Burner**, set **Burn Files** to **Huawei\_LiteOS.bin**, and click **Confirm**.

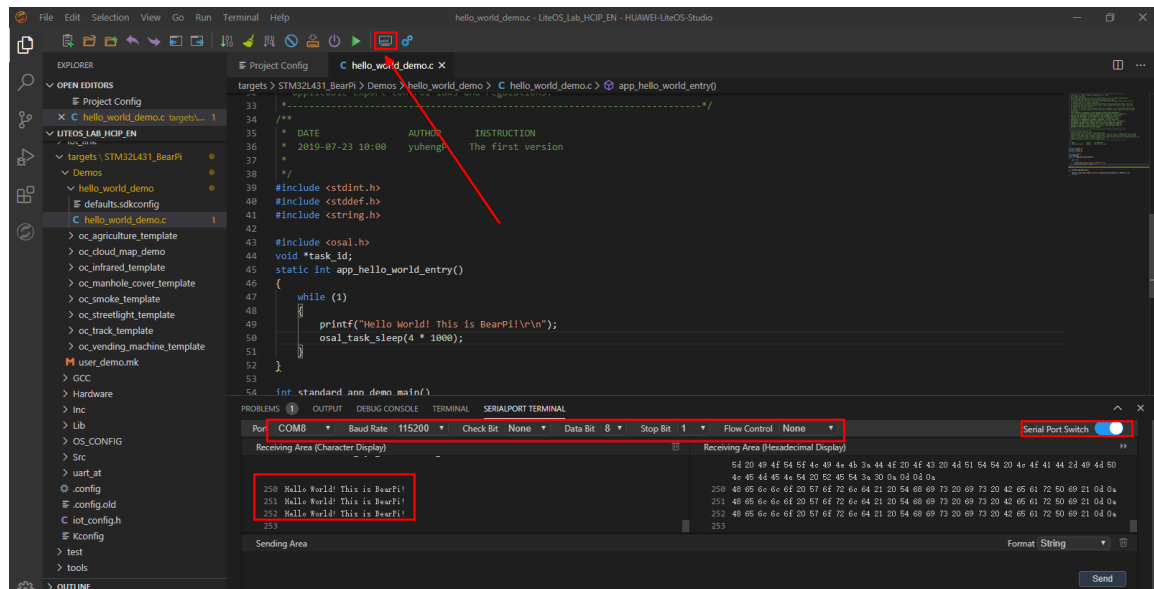


Open the hotspot on the phone, click  on the toolbar, and wait for the burning to complete.



Step 8 View the result.

Click  on the toolbar to open the serial port terminal, set the baud rate to 115200, and enable the serial port. The following information is displayed in the receiving area:  
Hello World! This is BearPi!



## 1.2.3 Managing Tasks

Step 1 Add the code of task2.

In `hello_world_demo.c`, add the code for executing `task2`.

```
static int task2()
```

```
{
    while (1)
    {
        printf("This is Task2!\r\n");
        osal_task_sleep(4*1000);
    }
}
```



The screenshot shows a code editor with a dark theme. The file name is 'hello\_world\_demo.c'. The editor shows the following code:

```
44
45 static int app_hello_world_entry()
46 {
47     while (1)
48     {
49         printf("Hello World! This is BearPi!\r\n");
50         osal_task_sleep(4*1000);
51     }
52 }
53
54 static int task2()
55 {
56     while (1)
57     {
58         printf("This is Task2!\r\n");
59         osal_task_sleep(4*1000);
60     }
61 }
62
```

The code defines two functions: `app_hello_world_entry` and `task2`. Both functions contain a `while (1)` loop that prints a message and calls `osal_task_sleep(4*1000)`. The `task2` function is highlighted with a red box.

Step 2 Create task2.

```
osal_task_create("task2",task2,NULL,0x400,NULL,2);
```

```
63 int standard_app_demo_main()
64 {
65     osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,2);
66     osal_task_create("task2",task2,NULL,0x400,NULL,2);
67     return 0;
68 }
```

Step 3 Compile and burn the program and view the result.

Compile and burn the program, open the serial port terminal, and check the printed information.

Hello World! This is BearPi! and This is Task2! are printed alternately.

```

13
14
15 Hello World! This is BearPi!
16 This is Task2!
17 Hello World! This is BearPi!
18 This is Task2!
19 Hello World! This is BearPi!

```

Step 4 Delete the **Helloworld** task from **task2**.

Add a task ID.

```
void* task_id;
```

```

39 #include <stdint.h>
40 #include <stddef.h>
41 #include <string.h>
42
43 #include <osal.h>
44 void* task_id;
45 static int app_hello_world_entry()
46 {
47     while (1)
48     {
49         printf("Hello World! This is BearPi!\r\n");
50         osal_task_sleep(4*1000);
51     }
52 }

```

Get the task ID.

```
task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,2);
```

Add the code for deleting the task.

```

int num = 0;
while (1)
{
    printf("This is Task2!\r\n");
    num++;
    if(num == 5){
        osal_task_kill(task_id);
    }
    osal_task_sleep(4*1000);
}

```



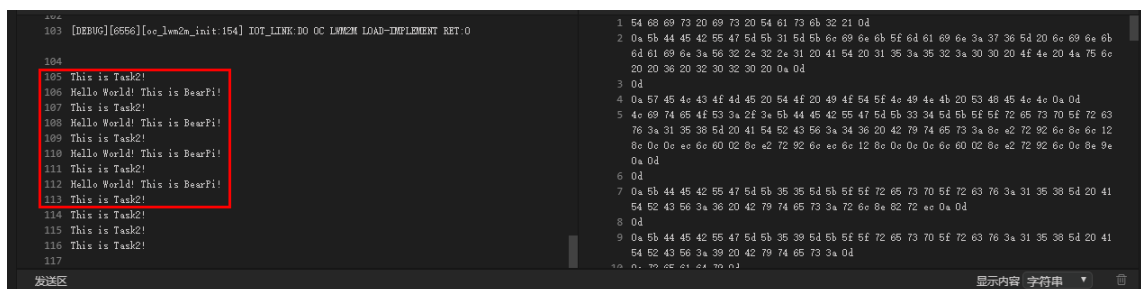
```

54 static int task2()
55 {
56     int num = 0;
57     while (1)
58     {
59         printf("This is Task2!\r\n");
60         num++;
61         if(num == 5){
62             osal_task_kill(task_id);
63         }
64         osal_task_sleep(4*1000);
65     }
66 }
67

```

Step 5 Compile and burn the program and view the result.

Compile and burn the program, open the serial port terminal, and view the print information. After **This is Task2!** is printed five times, **Hello World! This is BearPi!** is no longer printed.



```

103 [DEBUG][6556][oe_lwm2m_init:154] IoT_LINK:DO OC LWM2M LOAD-IMPLEMENT RET 0
104
105 This is Task2!
106 Hello World! This is BearPi!
107 This is Task2!
108 Hello World! This is BearPi!
109 This is Task2!
110 Hello World! This is BearPi!
111 This is Task2!
112 Hello World! This is BearPi!
113 This is Task2!
114 This is Task2!
115 This is Task2!
116 This is Task2!
117

```

## 1.2.4 Creating a Mutex

Step 1 Add the code of task1.

```

uint32_t public_value = 0;
osal_mutex_t public_value_mutex;
static int mutex_task1_entry()
{
    while(1)
    {
        if(true == osal_mutex_lock(public_value_mutex))
        {
            printf("\r\ntask1: lock a mutex.\r\n");
            public_value += 10;
            printf("task1: public_value = %ld.\r\n", public_value);
            printf("task1: sleep...\r\n");

```

```

osal_task_sleep(10);
printf("task1: continue...\r\n");
printf("task1: unlock a mutex.\r\n\r\n");
osal_mutex_unlock(public_value_mutex);
if(public_value > 100)
    break;
}
}
return 0;
}

```

```

69  uint32_t public_value = 0;
70  osal_mutex_t public_value_mutex;
71  static int mutex_task1_entry()
72  {
73      while(1)
74      {
75          if(true == osal_mutex_lock(public_value_mutex))
76          {
77              printf("\r\ntask1: lock a mutex.\r\n");
78              public_value += 10;
79              printf("task1: public_value = %ld.\r\n", public_value);
80              printf("task1: sleep...\r\n");
81              osal_task_sleep(10);
82              printf("task1: continue...\r\n");
83              printf("task1: unlock a mutex.\r\n\r\n");
84              osal_mutex_unlock(public_value_mutex);
85              if(public_value > 100)
86                  break;
87          }
88      }
89      return 0;
90  }
91

```

Step 2 Add the code of **task2**.

```

static int mutex_task2_entry()
{
    while (1)
    {
        if(true == osal_mutex_lock(public_value_mutex))
        {

```

```

printf("\r\n task2: lock a mutex.\r\n");
public_value += 5;
printf("task2: public_value = %ld.\r\n", public_value);
printf("task2: unlock a mutex.\r\n\r\n");
osal_mutex_unlock(public_value_mutex);
if(public_value > 90)
    break;
osal_task_sleep(10);
}
}
return 0;
}

```

```

91
92 static int mutex_task2_entry()
93 {
94     while (1)
95     {
96         if(true == osal_mutex_lock(public_value_mutex))
97         {
98             printf("\r\n task2: lock a mutex.\r\n");
99             public_value += 5;
100             printf("task2: public_value = %ld.\r\n", public_value);
101             printf("task2: unlock a mutex.\r\n\r\n");
102             osal_mutex_unlock(public_value_mutex);
103             if(public_value > 90)
104                 break;
105             osal_task_sleep(10);
106         }
107     }
108     return 0;
109 }
110
111 int standard_app_demo_main()
112 {

```

Step 3 Comment out the code for task management

```

111 int standard_app_demo_main()
112 {
113     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
114     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
115     return 0;
116 }

```

Step 4 Create a mutex.

```
osal_mutex_create(&public_value_mutex);
```

```
111 int standard_app_demo_main()
112 {
113     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
114     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
115
116     osal_mutex_create(&public_value_mutex);
117     return 0;
118 }
```

Step 5 Create task1 and task2.

```
osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
```

```
111 int standard_app_demo_main()
112 {
113     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
114     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
115
116     osal_mutex_create(&public_value_mutex);
117     osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
118     osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
119
120     return 0;
121 }
```

Step 6 Compile and burn the program and view the result.

Compile and burn the program, open the serial port terminal, and view the print information. **task 1** is created first, but its priority is low. Therefore, **task 2** is executed in preemption mode. **task 2** obtains the mutex and performs operations on the shared resources. When the operations are complete, **task 2** is unlocked and then suspended. **task 1** obtains the mutex and performs operations on the shared resources. When the operations are complete, **task 2** is suspended. **task 2** is running but cannot obtain the mutex. Therefore, **task 2** is blocked and waits to be executed. After **task 1** is unlocked, task 2 is woken up and executed.

```

173 task2: lock a mutex.
174 task2: public_value = 5.
175 task2: unlock a mutex.
176
177
178 task1: lock a mutex.
179 task1: public_value = 15.
180 task1: sleep...
181 task1: continue...
182 task1: unlock a mutex.
183
184
185 task2: lock a mutex.
186 task2: public_value = 20.
187 task2: unlock a mutex.
188
189
190 task1: lock a mutex.
191 task1: public_value = 30.
192 task1: sleep...
193 task1: continue...
194 task1: unlock a mutex.

```

## 1.2.5 Managing Memory

Step 1 Add the code of **task**.

```

static int mem_access_task_entry()
{
    uint32_t i = 0;
    size_t mem_size;
    uint8_t* mem_ptr = NULL;
    while (1)
    {
        mem_size = 1 << i++;
        mem_ptr = osal_malloc(mem_size);
        if(mem_ptr != NULL)
        {
            printf("access %d bytes memory success!\r\n", mem_size);
            osal_free(mem_ptr);
            mem_ptr = NULL;
            printf("free memory success!\r\n");
        }
        else

```

```

{

    printf("access %d bytes memory failed!\r\n", mem_size);
    return 0;
}
}

```

```

111 static int mem_access_task_entry()
112 {
113     uint32_t i = 0;
114     size_t mem_size;
115     uint8_t* mem_ptr = NULL;
116     while (1)
117     {
118         mem_size = 1 << i++;
119         mem_ptr = osal_malloc(mem_size);
120         if(mem_ptr != NULL)
121         {
122             printf("access %d bytes memory success!\r\n", mem_size);
123             osal_free(mem_ptr);
124             mem_ptr = NULL;
125             printf("free memory success!\r\n");
126         }
127         else
128         {
129
130             printf("access %d bytes memory failed!\r\n", mem_size);
131             return 0;
132         }
133     }
134 }
135
136 int standard_app_demo_main()
137 {

```

Step 2 Comment out the code for creating a mutex.

```

136 int standard_app_demo_main()
137 {
138     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
139     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
140
141     // osal_mutex_create(&public_value_mutex);
142     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
143     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
144
145     return 0;
146 }

```

Step 3 Create a memory task.

```
osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
```

```
136 int standard_app_demo_main()
137 {
138     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
139     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
140
141     // osal_mutex_create(&public_value_mutex);
142     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
143     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
144     osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
145
146     return 0;
147 }
```

Step 4 Compile and burn the program and view the result.

Compile and burn the program, open the serial port terminal, and check the printed information.

```
241 access 1 bytes memory success!
242 free memory success!
243 access 2 bytes memory success!
244 free memory success!
245 access 4 bytes memory success!
246 free memory success!
247 access 8 bytes memory success!
248 free memory success!
249 access 16 bytes memory success!
250 free memory success!
251 access 32 bytes memory success!
252 free memory success!
253 access 64 bytes memory success!
254 free memory success!
```

## 1.2.6 Creating a Semaphore

Step 1 Add the code of task1.

```
osal_semp_t sync_semp;
```

```
static int semp_task1_entry()
{
    printf("task 1 post a semp!\r\n");
    osal_semp_post(sync_semp);
    printf("task 1 end!\r\n");
}
```

```
136  osal_semp_t sync_semp;
137  static int semp_task1_entry()
138  {
139      printf("task 1 post a semp!\r\n");
140      osal_semp_post(sync_semp);
141      printf("task 1 end!\r\n");
142  }
```

Step 2 Add the code of task2.

```
static int semp_task2_entry()
{
    printf("task2 is waiting for a semp...\r\n");
    osal_semp_pend(sync_semp, cn_osal_timeout_forever);
    printf("task 2 access a semp!\r\n");
}
```

```
144  static int semp_task2_entry()
145  {
146      printf("task2 is waiting for a semp...\r\n");
147      osal_semp_pend(sync_semp, cn_osal_timeout_forever);
148      printf("task 2 access a semp!\r\n");
149  }
150
```

Step 3 Comment out the code for memory management.

```
151  int standard_app_demo_main()
152  {
153      // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
154      // osal_task_create("task2",task2,NULL,0x400,NULL,2);
155
156      // osal_mutex_create(&public_value_mutex);
157      // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
158      // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
159      // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
160
161      return 0;
162  }
```

Step 4 Create a semaphore.



```
osal_semp_create(&sync_semp, 1, 0);
printf("sync_semp semp create success.\r\n");
```

```
151 int standard_app_demo_main()
152 {
153     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
154     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
155
156     // osal_mutex_create(&public_value_mutex);
157     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
158     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
159     // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
160     osal_semp_create(&sync_semp, 1, 0);
161     printf("sync_semp semp create success.\r\n");
162
163     return 0;
164 }
```

Step 5 Create task1 and task2.

```
osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
```

```
151 int standard_app_demo_main()
152 {
153     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
154     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
155
156     // osal_mutex_create(&public_value_mutex);
157     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
158     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
159     // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
160     osal_semp_create(&sync_semp, 1, 0);
161     printf("sync_semp semp create success.\r\n");
162     osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
163     osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
164
165     return 0;
166 }
```

Step 6 Compile and burn the program and view the result.

Compile and burn the program, open the serial port terminal, and check the printed information.

```
128 task2 is waiting for a semp...
129 task 1 post a semp!
130 task 2 access a semp!
131 task 1 end!
```

Step 7 Comment out the code for creating a semaphore.

```
152 int standard_app_demo_main()
153 {
154     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
155     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
156     // osal_mutex_create(&public_value_mutex);
157     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
158     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
159     // osal_task_create("mem access task",mem access task entry,NULL,0x400,NULL,11);
160     // osal_semp_create(&sync_semp, 1, 0);
161     // printf("sync_semp semp create success.\r\n");
162     // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
163     // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
164
165     return 0;
166 }
167
```

## 1.3 Exercise

### 1.3.1 Changing the Task Priority to Print "This is Task2!" Prior to "Hello World! This is BearPi!"

# 2 Basic BearPi Exercise

---

## 2.1 Introduction

### 2.1.1 About This Exercise

In this exercise, you will control the LCD and blink the LED, which will help you understand the working principles of the development board.

### 2.1.2 Objectives

- Enable the onboard LCD.
- Blink the onboard LED.
- Use GPIO to scan and detect the LED controlled by the onboard buttons.
- Use EXIT to detect the LED controlled by the onboard buttons.

## 2.2 Tasks

### 2.2.1 Displaying a String on the Onboard LCD

Step 1 Import the header file of the LCD.

```
#include "lcd.h"
```

```

39 #include <stdint.h>
40 #include <stddef.h>
41 #include <string.h>
42
43 #include <osal.h>
44 #include "lcd.h"
45 void *task_id;
46 static int app_hello_world_entry()
47 {
48     while (1)
49     {
50         printf("Hello World! This is BearPi!\r\n");
51         osal_task_sleep(4 * 1000);
52     }
53 }

```

Step 2 Add the code for clearing the LCD.

```
LCD_Clear(BLACK);
```

```

152 int standard_app_demo_main()
153 {
154     LCD_Clear(BLACK);
155     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
156     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
157
158     // osal_mutex_create(&public_value_mutex);
159     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
160     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
161     // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
162     // osal_semp_create(&sync_semp, 1, 0);
163     // printf("sync_semp semp create success.\r\n");
164     // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
165     // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
166
167     return 0;
168 }

```

Step 3 Add the code of the content to be shown on the LCD.

```
POINT_COLOR = GREEN;
LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");
```

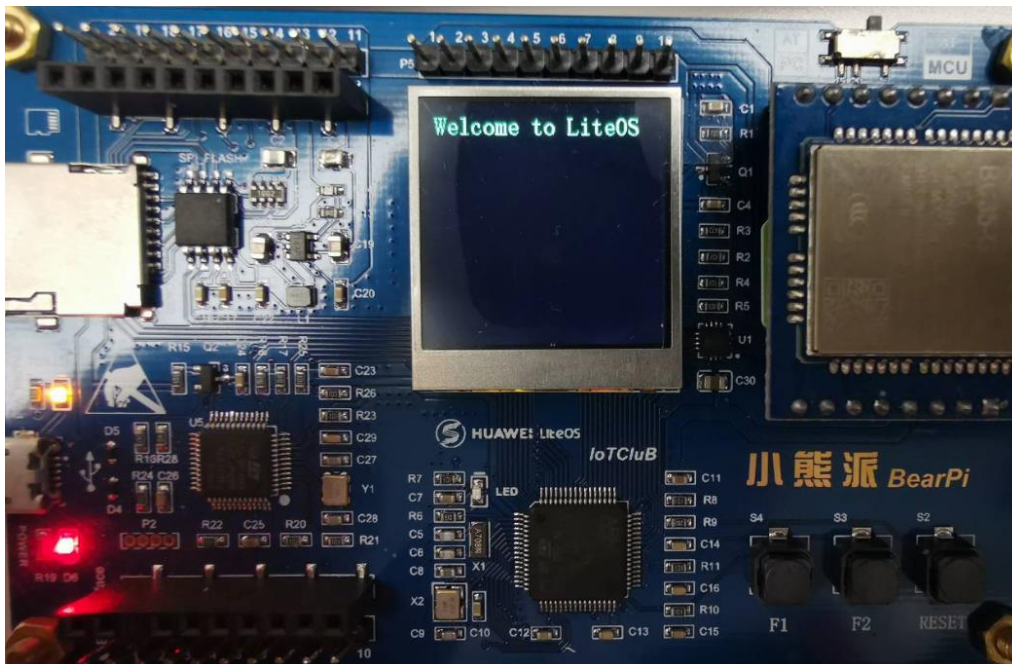
```

152 int standard_app_demo_main()
153 {
154     LCD_Clear(BLACK);
155     POINT_COLOR = GREEN;
156     LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");
157
158     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
159     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
160

```

Step 4 Compile and burn the program and view the result.

Compile and burn the program, and check whether the LCD displays **Welcome to LiteOS**.



## 2.2.2 Blinking the Onboard LED

Step 1 Add the LED blinking code.

```
static int led_task()
{
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.Pin = GPIO_PIN_13;
    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStructure.Pull = GPIO_NOPULL;
    GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
    while (1)
    {
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
        osal_task_sleep(1*1000);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
        osal_task_sleep(1*1000);
    }
}
```

```

151
152 static int led_task()
153 {
154     GPIO_InitTypeDef GPIO_InitStructure;
155     GPIO_InitStructure.Pin = GPIO_PIN_13;
156     GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
157     GPIO_InitStructure.Pull = GPIO_NOPULL;
158     GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
159     HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
160     while (1)
161     {
162         HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
163         osal_task_sleep(1*1000);
164         HAL_GPIO_WritePin(GPIOC,GPIO_PIN_13,GPIO_PIN_RESET);
165         osal_task_sleep(1*1000);
166     }
167 }
168
169 int standard_app_demo_main()

```

Step 2 Create an LED blinking task.

```
osal_task_create("led_task",led_task,NULL,0x400,NULL,2);
```

```

169 int standard_app_demo_main()
170 {
171     LCD_Clear(BLACK);
172     POINT_COLOR = GREEN;
173     LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");
174
175     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
176     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
177
178     // osal_mutex_create(&public_value_mutex);
179     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
180     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
181     // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
182     // osal_semp_create(&sync_semp, 1, 0);
183     // printf("sync_semp semp create success.\r\n");
184     // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
185     // osal task create("semp task2",semp task2 entry,NULL,0x400,NULL,11);
186     osal_task_create("led_task",led_task,NULL,0x400,NULL,2);
187
188     return 0;
189 }

```

Step 3 Compile and burn the program and view the result.

Compile and burn the program, and check whether the LED blinks.

## 2.2.3 Using GPIO to Scan and Detect the LED Controlled by the Onboard Buttons

Step 1 Comment out the LED blinking loop code.

```
152 static int led_task()  
153 {  
154     GPIO_InitTypeDef GPIO_InitStructure;  
155     GPIO_InitStructure.Pin = GPIO_PIN_13;  
156     GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;  
157     GPIO_InitStructure.Pull = GPIO_NOPULL;  
158     GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;  
159     HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);  
160     while (1)  
161     {  
162         // HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);  
163         // osal_task_sleep(1*1000);  
164         // HAL_GPIO_WritePin(GPIOC,GPIO_PIN_13,GPIO_PIN_RESET);  
165         // osal_task_sleep(1*1000);  
166     }  
167 }
```

Step 2 Add the button judgment code.

```
if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2)==GPIO_PIN_RESET)  
{  
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);  
}else if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_3)==GPIO_PIN_RESET)  
{  
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);  
}
```

```

152 static int led_task()
153 {
154     GPIO_InitTypeDef GPIO_InitStructure;
155     GPIO_InitStructure.Pin = GPIO_PIN_13;
156     GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
157     GPIO_InitStructure.Pull = GPIO_NOPULL;
158     GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
159     HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
160     while (1)
161     {
162         // HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
163         // osal_task_sleep(1*1000);
164         // HAL_GPIO_WritePin(GPIOC,GPIO_PIN_13,GPIO_PIN_RESET);
165         // osal_task_sleep(1*1000);
166         if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2)==GPIO_PIN_RESET)//查询按键KEY1低电平
167         {
168             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
169         }else if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_3)==GPIO_PIN_RESET)//查询按键KEY2低电平
170         {
171             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
172         }
173     }
174 }
175

```

Step 3 Compile and burn the program and view the result.

Compile and burn the program. Press the **F1** button in the lower right corner of the development board to turn on the LED, and press the **F2** button to turn it off.

## 2.2.4 Using EXIT to Detect the LED Controlled by the Onboard Buttons

Step 1 Add the **key1** interrupt processing function.

```

static Key1_interrupt_entry()
{
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
    __HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_2);
}

```

```

176 static Key1_interrupt_entry()
177 {
178     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
179     HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_2);
180 }
181

```

Step 2 Add the **key2** interrupt processing function.

```

static Key2_interrupt_entry()
{

```



```

HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
__HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_3);
}

```

```

182 static Key2_interrupt_entry()
183 {
184     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
185     __HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_3);
186 }
187

```

Step 3 Comment out the code for creating an LED blinking task.

```

188 int standard_app_demo_main()
189 {
190     LCD_Clear(BLACK);
191     POINT_COLOR = GREEN;
192     LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");
193
194     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
195     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
196
197     // osal_mutex_create(&public_value_mutex);
198     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
199     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
200     // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
201     // osal_semp_create(&sync_semp, 1, 0);
202     // printf("sync_semp semp create success.\r\n");
203     // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
204     // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
205     // osal_task_create("led_task",led_task,NULL,0x400,NULL,2);
206
207     return 0;
208 }

```

Step 4 Create the interrupt functions.

```

GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.Pin = GPIO_PIN_13;
GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
osal_int_connect( EXTI2_IRQn, 3, NULL, Key1_interrupt_entry, NULL);
osal_int_connect( EXTI3_IRQn, 4, NULL, Key2_interrupt_entry, NULL);

```

```

188 int standard_app_demo_main()
189 {
190     LCD_Clear(BLACK);
191     POINT_COLOR = GREEN;
192     LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");
193
194     // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
195     // osal_task_create("task2",task2,NULL,0x400,NULL,2);
196
197     // osal_mutex_create(&public_value_mutex);
198     // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
199     // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
200     // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
201     // osal_semp_create(&sync_semp, 1, 0);
202     // printf("sync_semp semp create success.\r\n");
203     // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
204     // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
205     // osal_task_create("led_task",led_task,NULL,0x400,NULL,2);
206     GPIO_InitTypeDef GPIO_InitStructure;
207     GPIO_InitStructure.Pin = GPIO_PIN_13;
208     GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
209     GPIO_InitStructure.Pull = GPIO_NOPULL;
210     GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
211     HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
212     osal_int_connect( EXTI2_IRQn, 3, NULL, Key1_interrupt_entry, NULL);
213     osal_int_connect( EXTI3_IRQn, 4, NULL, Key2_interrupt_entry, NULL);
214
215     return 0;
216 }

```

Step 5 Compile and burn the program and view the result.

Compile and burn the program. Press the **F1** button in the lower right corner of the development board to turn on the LED, and press the **F2** button to turn it off.

## 2.3 Exercise

### 2.3.1 Creating a Multi-Line LCD Display

### 2.3.2 Creating a Multi-Color LCD Display

### 2.3.3 Printing the Status of the LED on the Console

### 2.3.4 Displaying the LED Status on the LCD

# 3 Wi-Fi-based Smart Agriculture Exercise

## 3.1 Introduction

### 3.1.1 About This Exercise

In this exercise, you will use Wi-Fi to implement a smart agriculture case, which involves collecting real-time data, responding to command delivery, and implementing device-cloud synergy.

### 3.1.2 Objectives

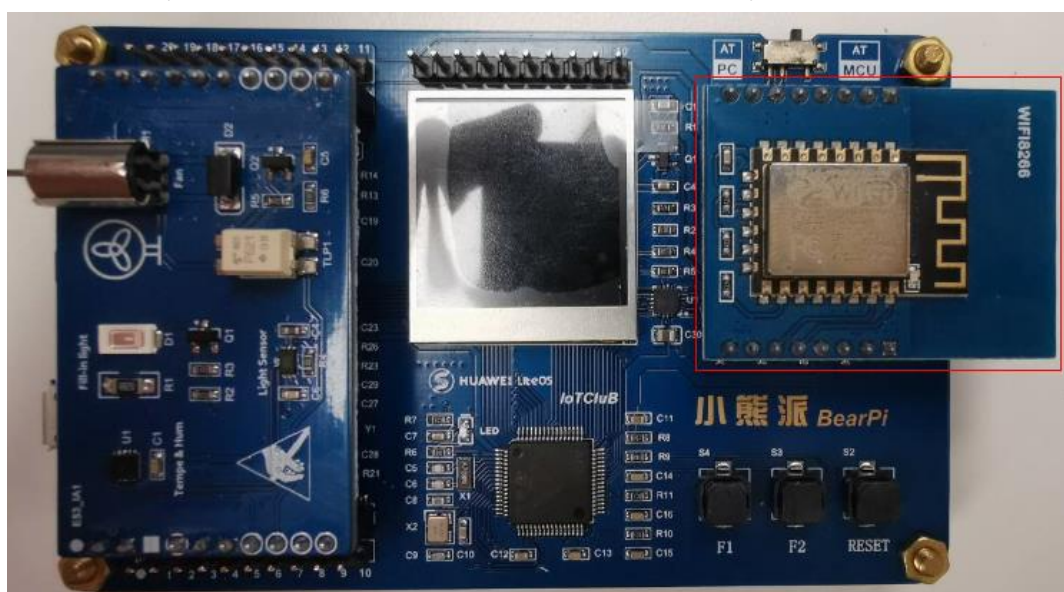
- Master how to configure the Wi-Fi communication mode.
- Master how to develop smart agriculture cases.

## 3.2 Tasks

### 3.2.1 Configuring a Smart Agriculture Case

Step 1 Install the smart agriculture expansion board E53\_IA1.

Insert the expansion board E53\_IA1 into the BearPi development board.



Step 2 Modify the .config file.

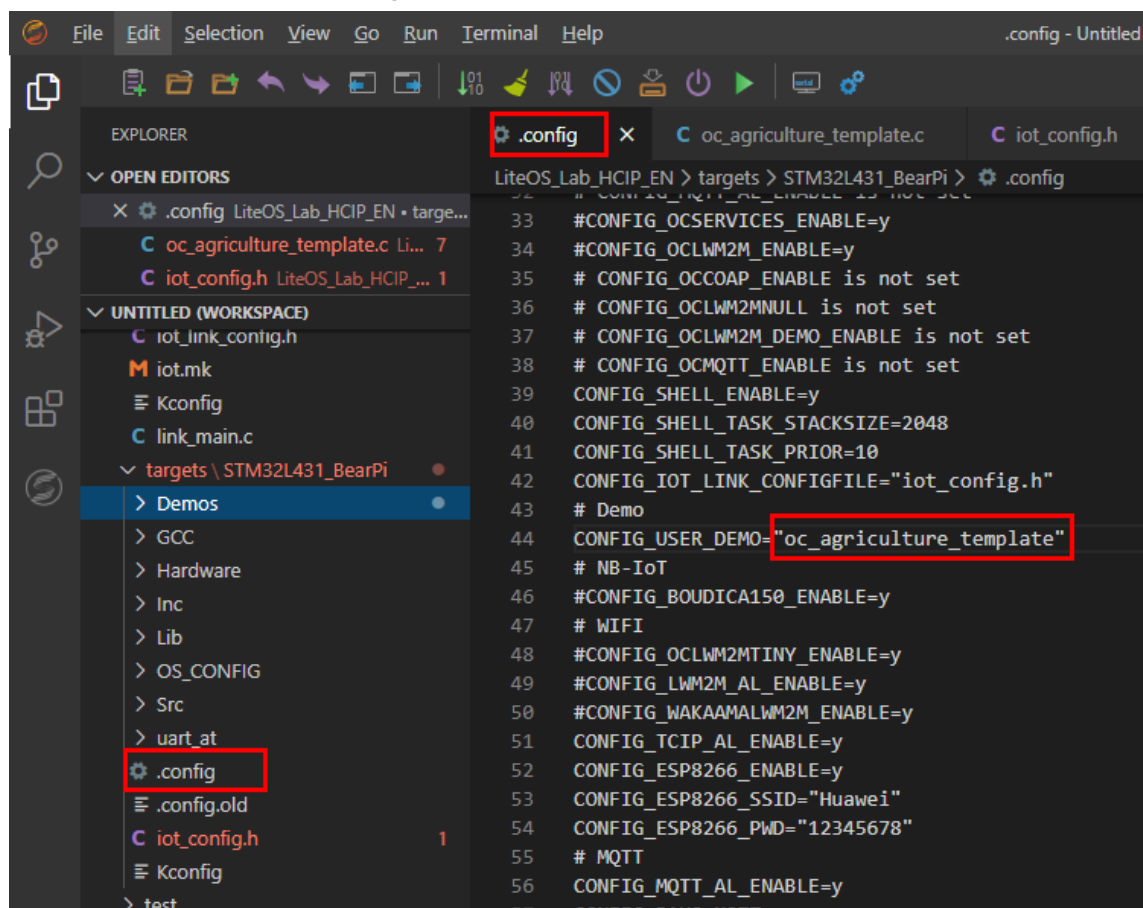
Choose **targets > STM32L431\_BearPi > .config**.

Set **CONFIG\_USER\_DEMO** to **oc\_agriculture\_template**.

Set **CONFIG\_ESP8266\_SSID** to the Wi-Fi username.

Set **CONFIG\_ESP8266\_PWD** to the Wi-Fi password.

Press **Ctrl+S** to save the **.config** file.



**Step 3** Modify the **iot\_config.h** file.

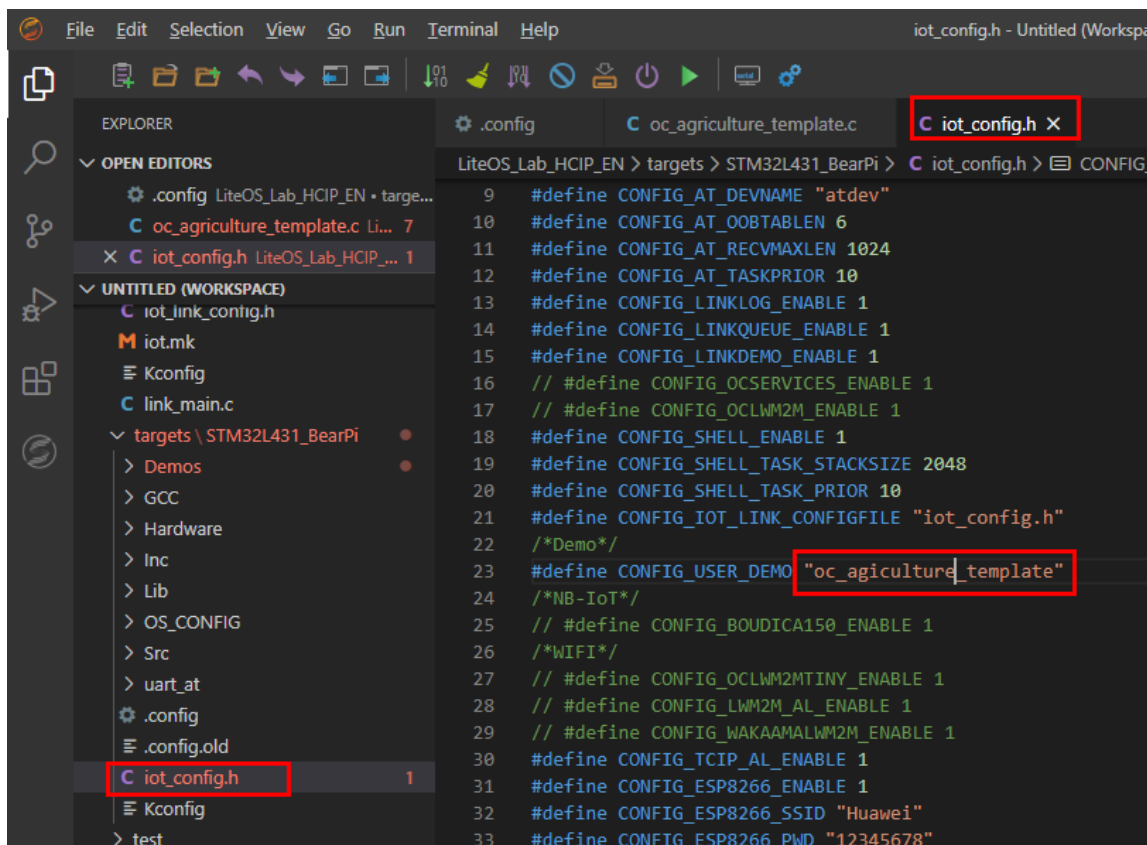
Choose **targets > STM32L431\_BearPi > iot\_config.h**.

Set **CONFIG\_USER\_DEMO** to **oc\_agriculture\_template**.

Set **CONFIG\_ESP8266\_SSID** to the Wi-Fi username.

Set **CONFIG\_ESP8266\_PWD** to the Wi-Fi password.

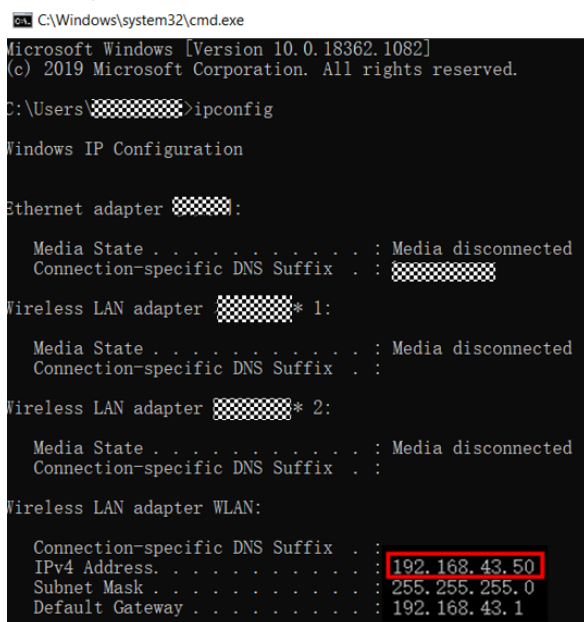
Press **Ctrl+S** to save the **iot\_config.h** file.



### 3.2.2 Configuring the IP Address of the Broker

Step 1 Query the IP address of the Broker.

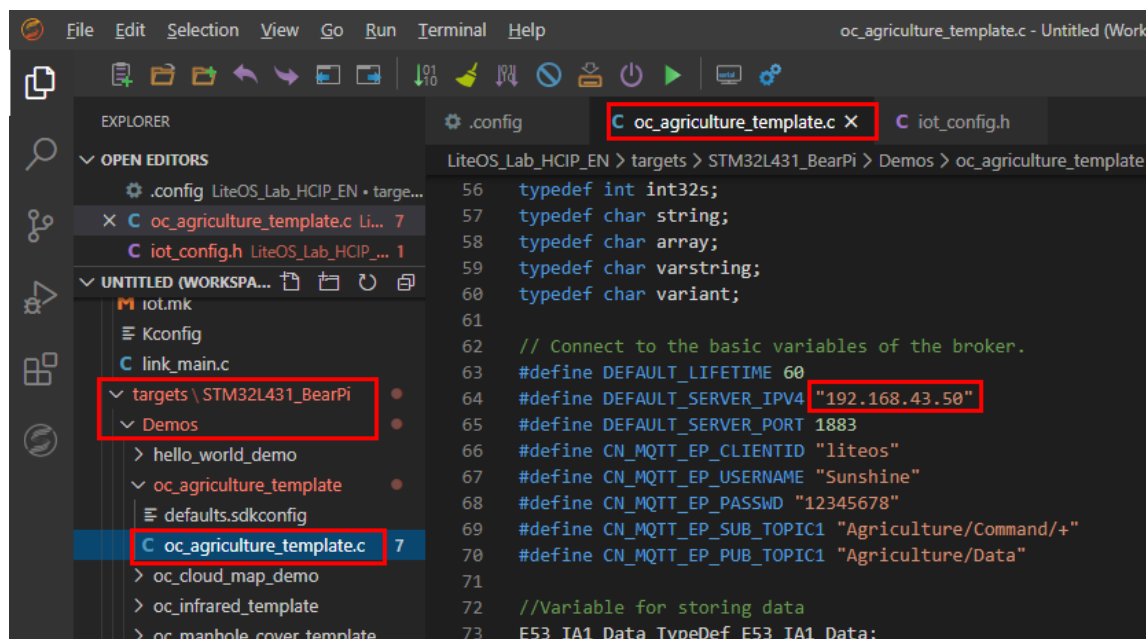
The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.



Step 2 Change the IP address in the code.


Choose **STM32L431\_BearPi** > **Demos** > **oc\_agriculture\_template** > **oc\_agriculture\_template.c**.

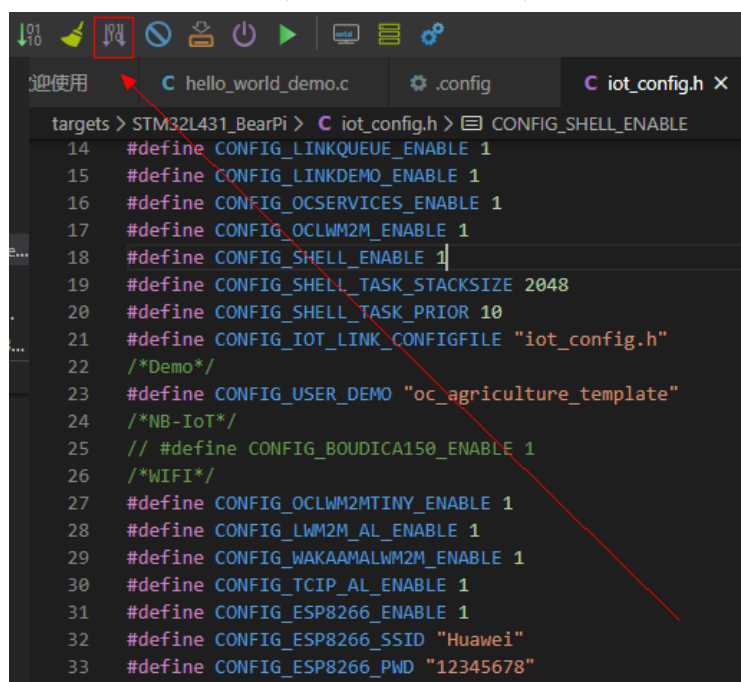
Set **DEFAULT\_SERVER\_IPV4** to the queried IP address.



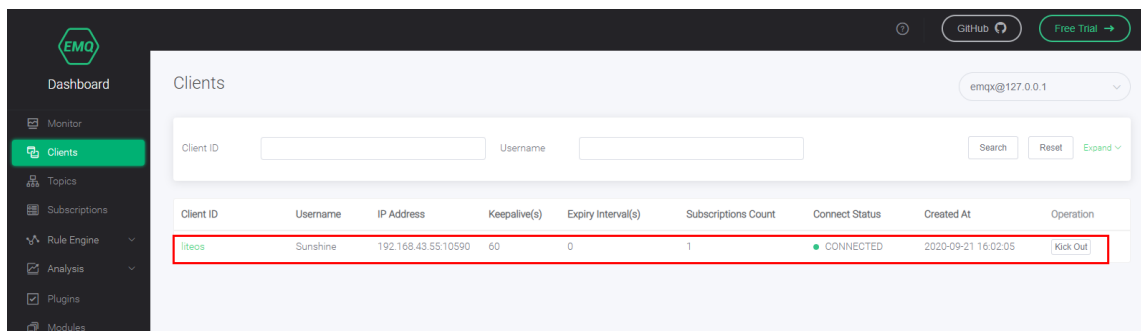
### 3.2.3 Compiling and Burning the Program

Step 1 Compile and burn the program and view the result.

Enable the Wi-Fi hotspot on the mobile phone and click  to recompile the program.



Burn the program and check whether the device is online in EMQ Broker.

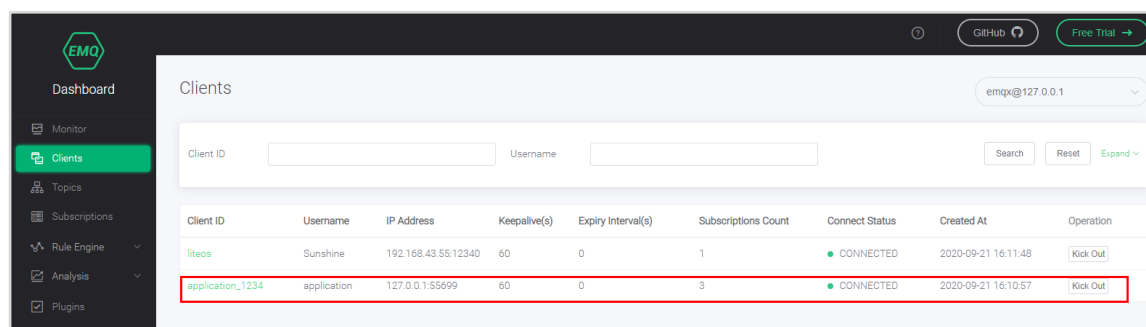


## 3.2.4 Running Applications

Step 1 Run the Java program to view smart agriculture data.

Right-click **HCIP-IoTEN** and choose **Run As > Java Application** from the shortcut menu.

On the Broker page, the device is online.



View the real-time data on the console.

```
Content of the received message:Temperature:28,Humidity:63,Luminance:480
Receive Message Subject:Agriculture/Data
Content of the received message:Temperature:28,Humidity:63,Luminance:487
Receive Message Subject:Agriculture/Data
Content of the received message:Temperature:28,Humidity:63,Luminance:480
Receive Message Subject:Agriculture/Data
Content of the received message:Temperature:28,Humidity:63,Luminance:480
```

Step 2 Run the Java program to deliver the command for turning on the LED.

Choose **HCIP-IoTEN > src > app > Application.java**.

Uncomment the command for turning on the LED and run the program again.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    MqttClient client = connect();
    subscribe(client);
    Command(client, pubTopicAgricultureLight, "ON");
    // Command(client, pubTopicAgricultureLight, "OFF");
    // Command(client, pubTopicAgricultureMotor, "ON");
    // Command(client, pubTopicAgricultureMotor, "OFF");
    // Command(client, pubTopicSmokeBeep, "ON");
    // Command(client, pubTopicSmokeBeep, "OFF");
    // Command(client, pubTopicTrackBeep, "ON");
    // Command(client, pubTopicTrackBeep, "OFF");
}
```

The LED on the development board is on.



## 3.3 Exercise

### 3.3.1 Delivering All Light and Motor Commands



# 4

## Wi-Fi-based Smart Smoke Detector Exercise

---

### 4.1 Introduction

#### 4.1.1 About This Exercise

In this exercise, you will use Wi-Fi to implement a smart smoke detector case, which involves collecting real-time data, responding to command delivery, and implementing device-cloud synergy.

#### 4.1.2 Objectives

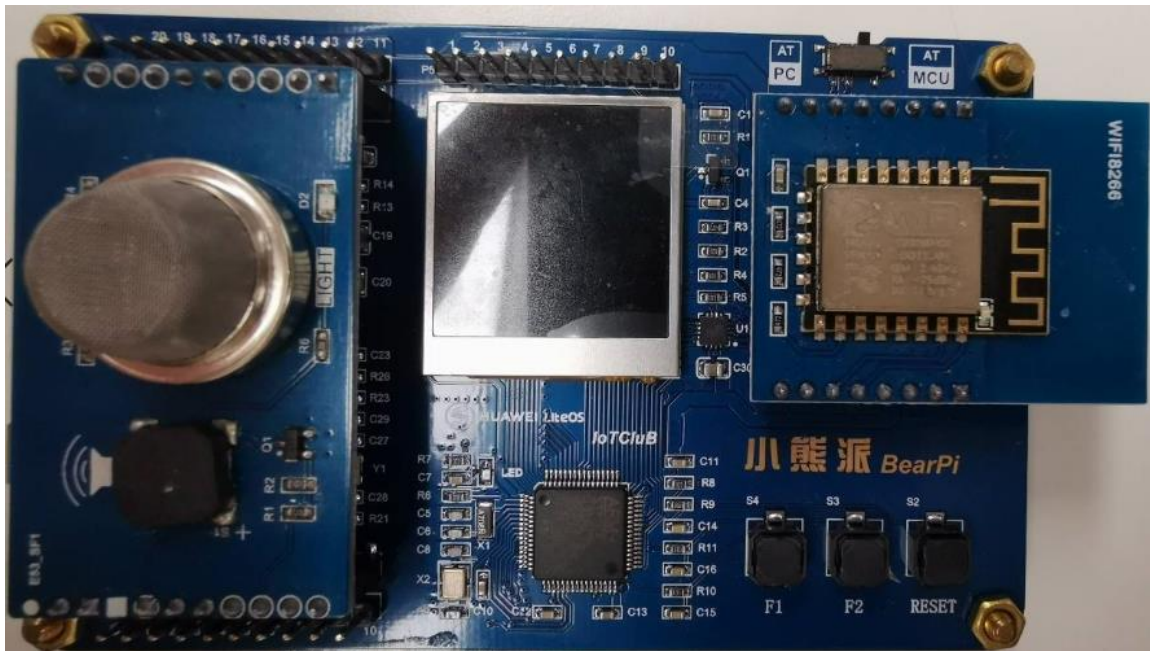
- Master how to configure the Wi-Fi communication mode.
- Master how to develop smart smoke detector cases.

### 4.2 Tasks

#### 4.2.1 Configuring a Smart Smoke Detector Case

Step 1 Install the smart smoke detector expansion board E53\_SF1.

Insert the expansion board E53\_SF1 into the BearPi development board.



Step 2 Modify the .config file.

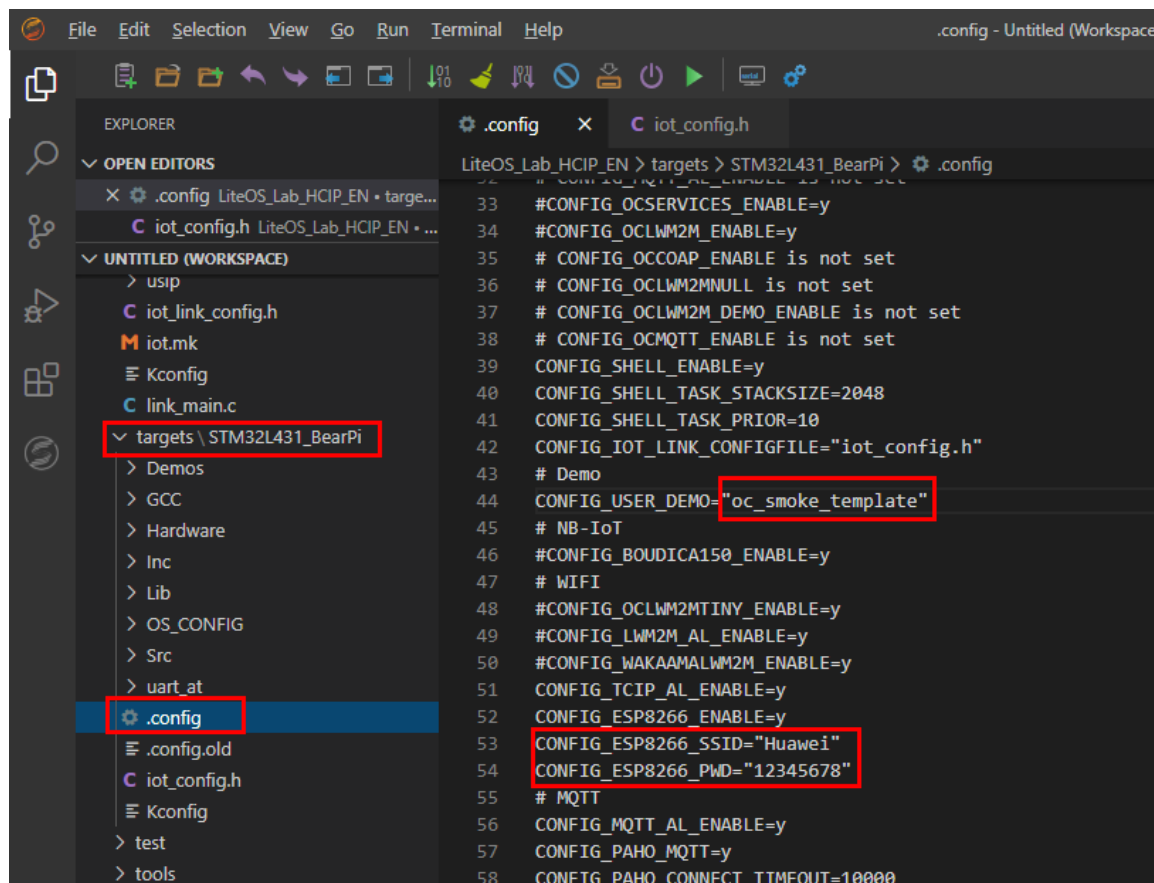
Choose **targets > STM32L431\_BearPi > .config**.

Set **CONFIG\_USER\_DEMO** to **oc\_smoke\_template**.

Set **CONFIG\_ESP8266\_SSID** to the Wi-Fi username.

Set **CONFIG\_ESP8266\_PWD** to the Wi-Fi password.

Press **Ctrl+S** to save the .config file.



Step 3 Modify the `iot_config.h` file.

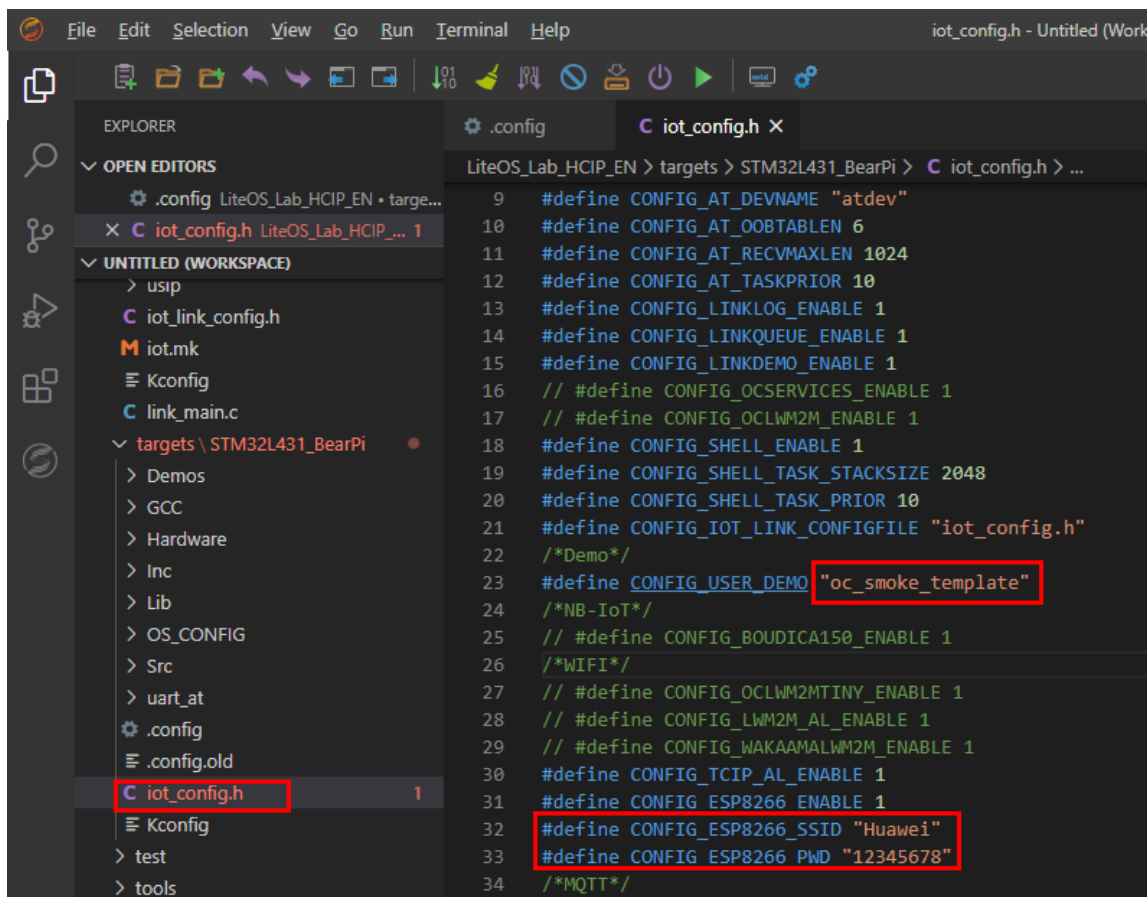
Choose `targets > STM32L431_BearPi > iot_config.h`.

Set `CONFIG_USER_DEMO` to `oc_smoke_template`.

Set `CONFIG_ESP8266_SSID` to the Wi-Fi username.

Set `CONFIG_ESP8266_PWD` to the Wi-Fi password.

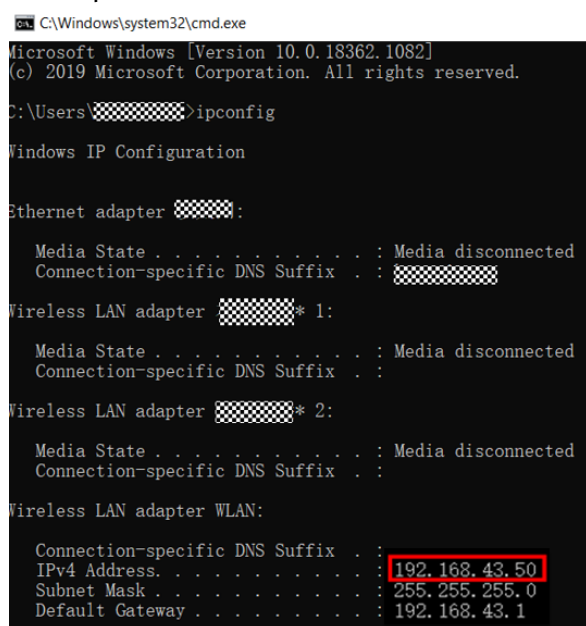
Press `Ctrl+S` to save the `iot_config.h` file.



## 4.2.2 Configuring the IP Address of the Broker

Step 1 Query the IP address of the Broker.

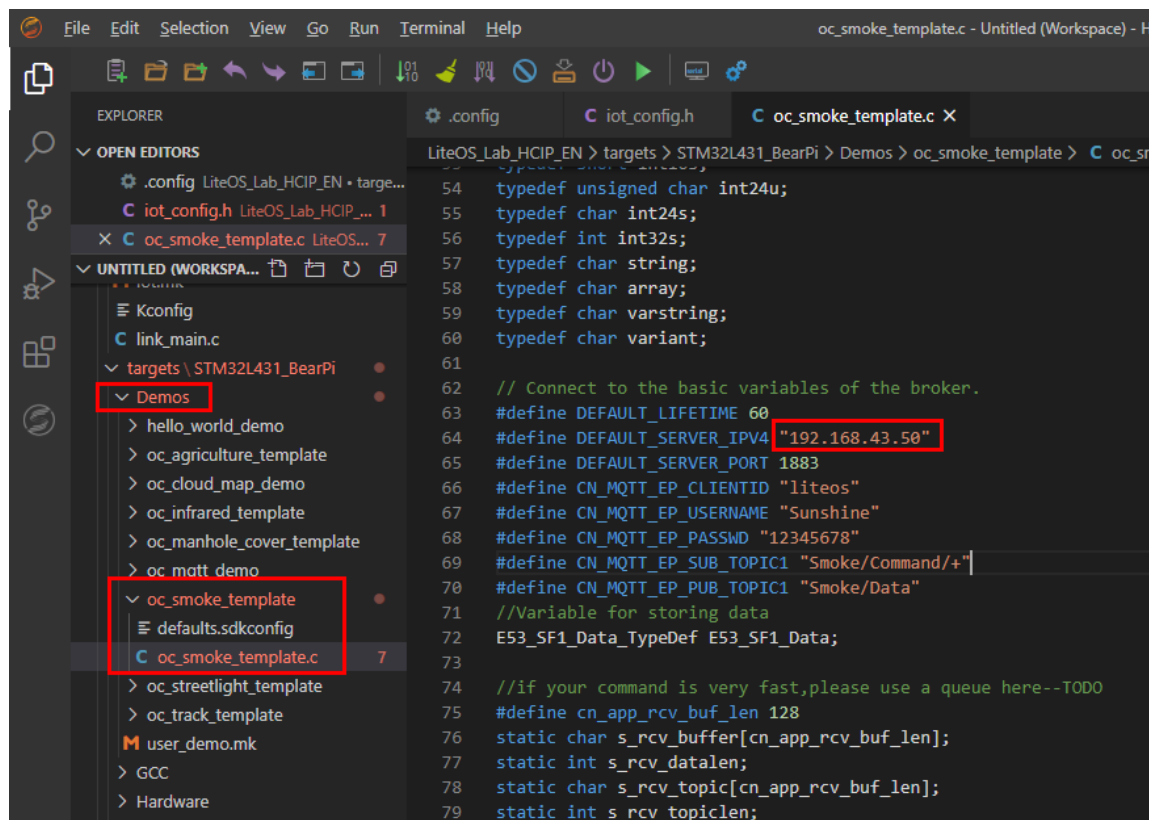
The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.



Step 2 Change the IP address in the code.

Choose **STM32L431\_BearPi > Demos > oc\_smoke\_template > oc\_smoke\_template.c**.

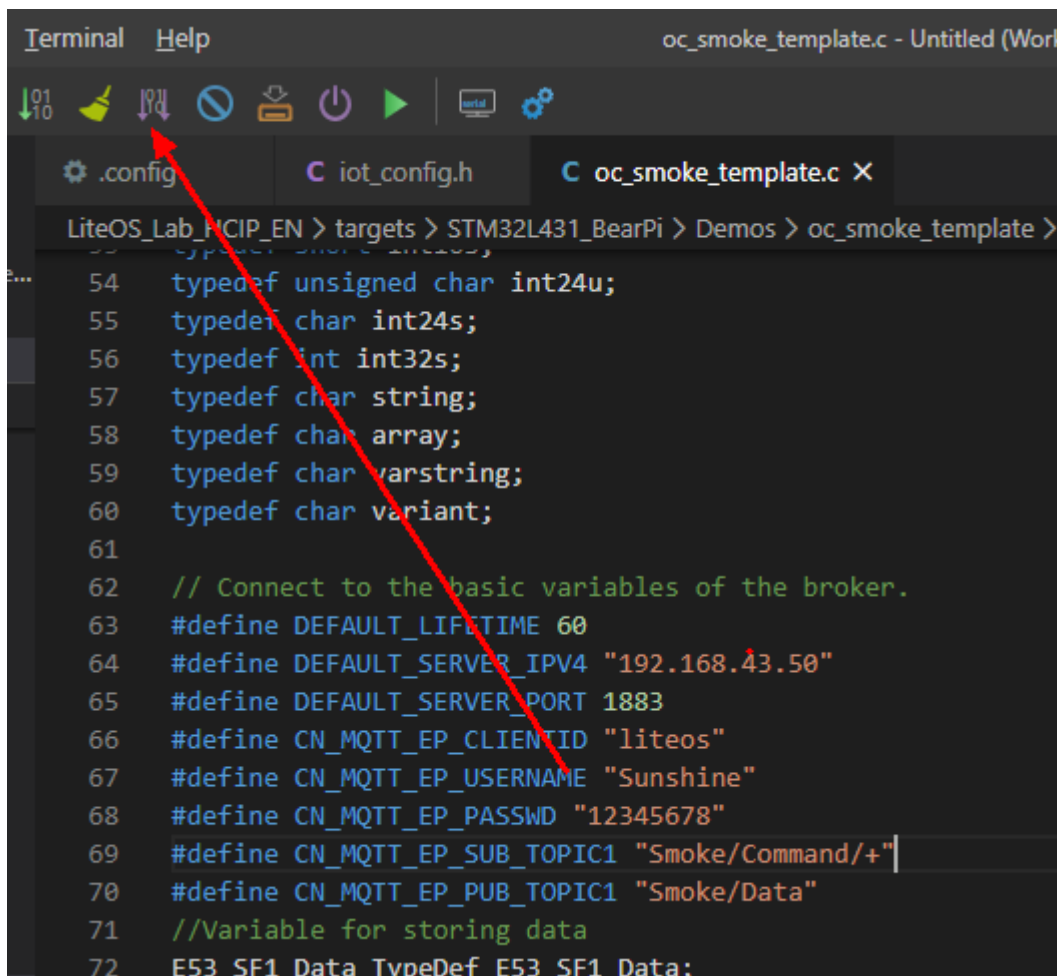
Set **DEFAULT\_SERVER\_IPV4** to the queried IP address.



## 4.2.3 Compiling and Burning the Program

Step 1 Compile and burn the program and view the result.

Enable the Wi-Fi hotspot on the mobile phone and click  to recompile the program.

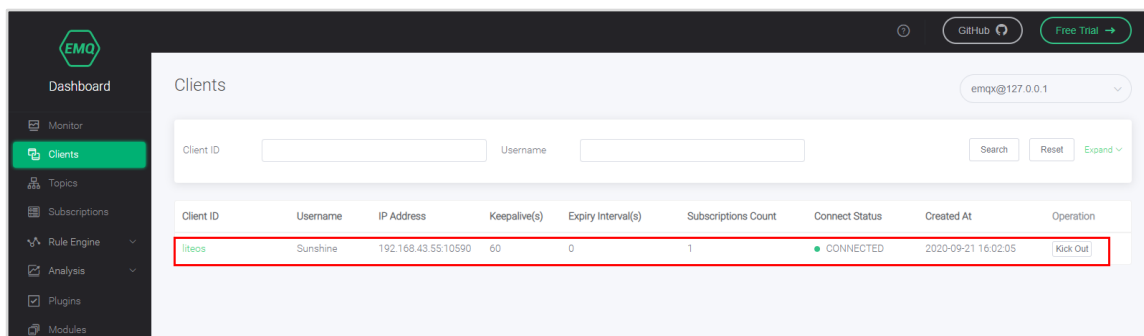


```

Terminal Help oc_smoke_template.c - Untitled (Work
...
54 typedef unsigned char int24u;
55 typedef char int24s;
56 typedef int int32s;
57 typedef char string;
58 typedef char array;
59 typedef char varstring;
60 typedef char variant;
61
62 // Connect to the basic variables of the broker.
63 #define DEFAULT_LIFETIME 60
64 #define DEFAULT_SERVER_IPV4 "192.168.43.50"
65 #define DEFAULT_SERVER_PORT 1883
66 #define CN_MQTT_EP_CLIENTID "liteos"
67 #define CN_MQTT_EP_USERNAME "Sunshine"
68 #define CN_MQTT_EP_PASSWD "12345678"
69 #define CN_MQTT_EP_SUB_TOPIC1 "Smoke/Command/+"
70 #define CN_MQTT_EP_PUB_TOPIC1 "Smoke/Data"
71 //Variable for storing data
72 E53 SF1 Data TypeDef E53 SF1 Data;

```

Burn the program and check whether the device is online in EMQ Broker.

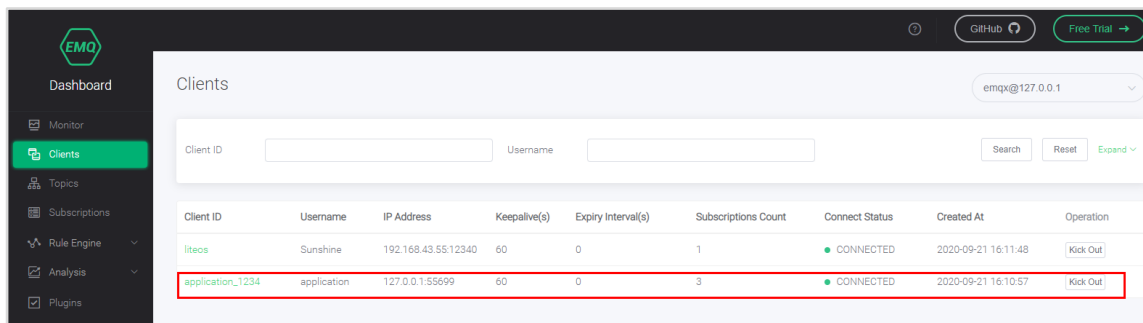


Client ID	Username	IP Address	Keepalive(s)	Expiry Interval(s)	Subscriptions Count	Connect Status	Created At	Operation
liteos	Sunshine	192.168.43.55:10590	60	0	1	CONNECTED	2020-09-21 16:02:05	Kick Out

## 4.2.4 Running Applications

Step 1 Run the Java program to view smart smoke detector data.

Right-click **HCIP-IoTEN** and choose **Run As > Java Application** from the shortcut menu.  
On the Broker page, the device is online.



Client ID	Username	IP Address	Keepalive(s)	Expiry Interval(s)	Subscriptions Count	Connect Status	Created At	Operation
liteos	Sunshine	192.168.43.55:12340	60	0	1	CONNECTED	2020-09-21 16:11:48	Kick Out
application_1234	application	127.0.0.1:55699	60	0	3	CONNECTED	2020-09-21 16:10:57	Kick Out

View the real-time data on the console.

```
Content of the received message:Smoke:5
Receive Message Subject:Smoke/Data
Content of the received message:Smoke:2
```

Step 2 Run the Java program to deliver the command for turning on the LED.

Choose **HCIP-IoT DeveloperEN > src > app > Application.java**.

Uncomment the command for enabling the buzzer and run the program again.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    MqttClient client = connect();
    subscribe(client);
    // Command(client, pubTopicAgricultureLight, "ON");
    // Command(client, pubTopicAgricultureLight, "OFF");
    // Command(client, pubTopicAgricultureMotor, "ON");
    // Command(client, pubTopicAgricultureMotor, "OFF");
    Command(client, pubTopicSmokeBeep, "ON");
    // Command(client, pubTopicSmokeBeep, "OFF");
    // Command(client, pubTopicTrackBeep, "ON");
    // Command(client, pubTopicTrackBeep, "OFF");
    // Command(client, pubTopicVendingMachine, "9123456780");
}
```

The buzzer on the development board rings.

## 4.3 Exercise

### 4.3.1 Delivering the Command for Disabling the Buzzer

# 5

## Wi-Fi-based Smart Manhole Cover Exercise

---

### 5.1 Introduction

#### 5.1.1 About This Exercise

In this exercise, you will use Wi-Fi to implement a smart manhole cover case, which involves collecting real-time data, responding to command delivery, and implementing device-cloud synergy.

#### 5.1.2 Objectives

- Master how to configure the Wi-Fi communication mode.
- Master how to develop smart manhole cover cases.

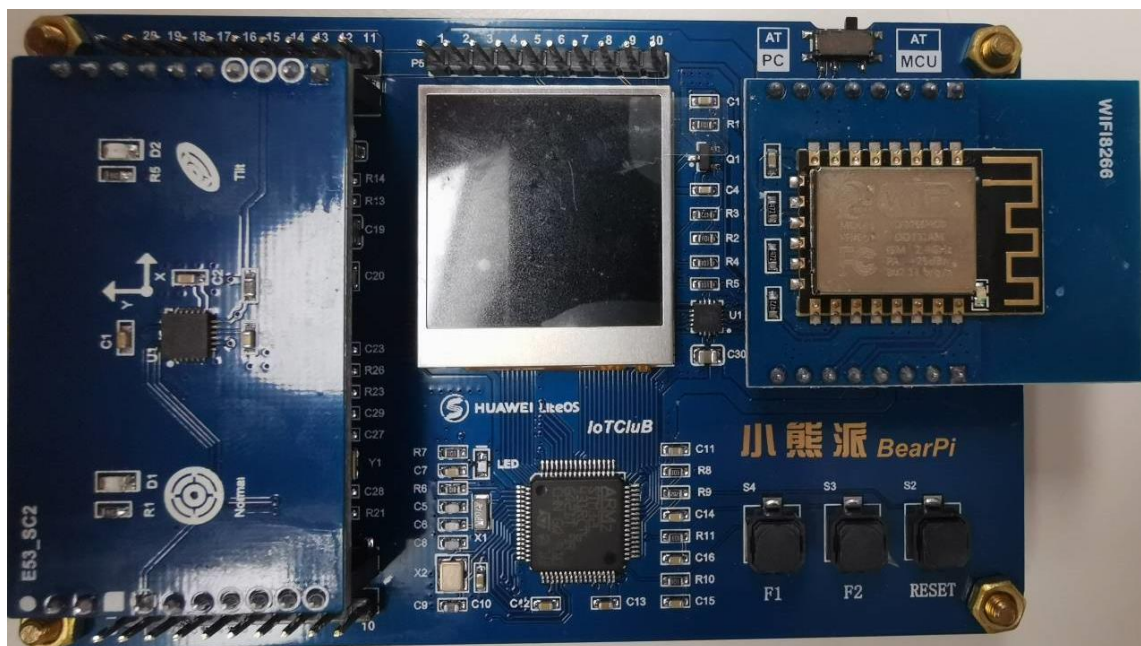
### 5.2 Tasks

#### 5.2.1 Configuring a Smart Manhole Cover Case

Step 1 Install the smart manhole cover expansion board E53\_SC2.

Insert the expansion board E53\_SC2 into the BearPi development board.





Step 2 Modify the `.config` file.

Choose **targets > STM32L431\_BearPi > .config**.

Set `CONFIG_USER_DEMO` to `oc_manhole_cover_template`.

Set `CONFIG_ESP8266_SSID` to the Wi-Fi username.

Set `CONFIG_ESP8266_PWD` to the Wi-Fi password.

Press **Ctrl+S** to save the `.config` file.

Step 3 Modify the `iot_config.h` file.

Choose **targets > STM32L431\_BearPi > iot\_config.h**.

Set `CONFIG_USER_DEMO` to `oc_manhole_cover_template`.

Set `CONFIG_ESP8266_SSID` to the Wi-Fi username.

Set `CONFIG_ESP8266_PWD` to the Wi-Fi password.

Press **Ctrl+S** to save the `iot_config.h` file.

## 5.2.2 Configuring the IP Address of the Broker

Step 1 Query the IP address of the Broker.

The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.

```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\>ipconfig

Windows IP Configuration

Ethernet adapter :

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter * 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter * 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

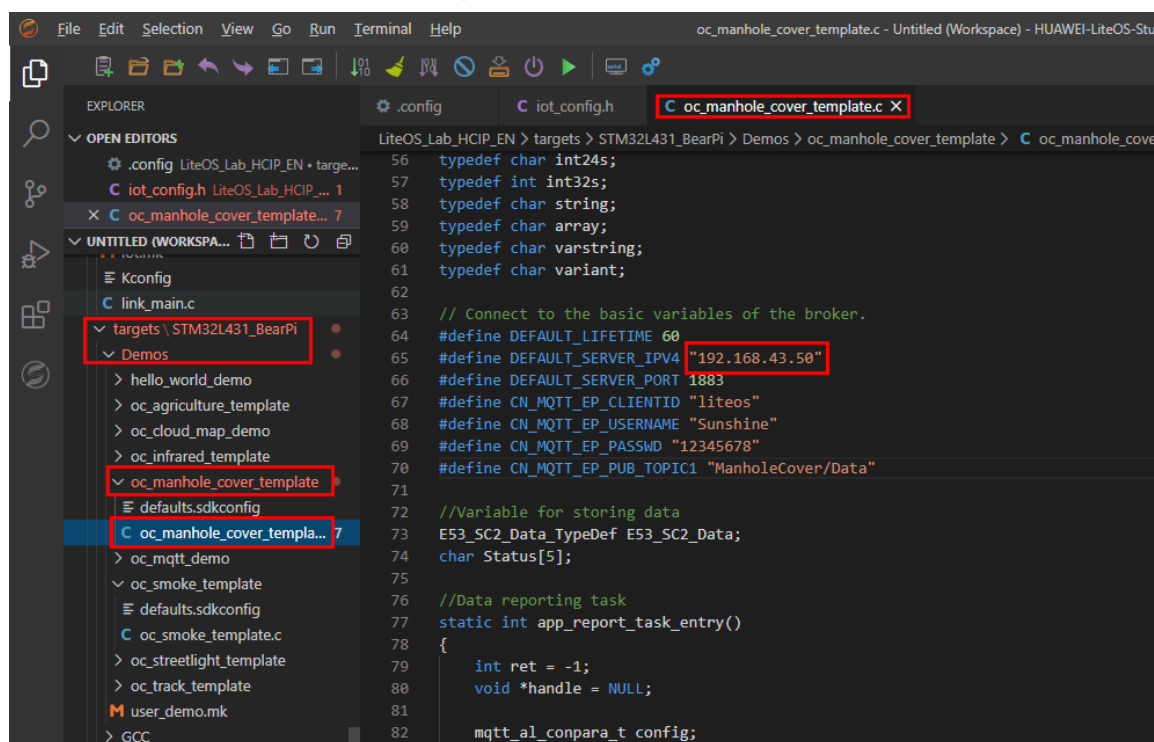
Wireless LAN adapter WLAN:

    Connection-specific DNS Suffix  . :
    IPv4 Address. . . . . : 192.168.43.50
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.43.1
  
```

Step 2 Change the IP address in the code.

Choose STM32L431\_BearPi > Demos > oc\_manhole\_cover\_template > oc\_manhole\_cover\_template.c.

Set DEFAULT\_SERVER\_IPV4 to the queried IP address.



```


File Edit Selection View Go Run Terminal Help
oc_manhole_cover_template.c - Untitled (Workspace) - HUAWEI-LiteOS-Stu

EXPLORER
OPEN EDITORS
  .config LiteOS_Lab_HCIP_EN + targe...
  C:\iot_config.h LiteOS_Lab_HCIP_... 1
  X C:\oc_manhole_cover_template... 7
  C:\link_main.c
  C:\Kconfig
  C:\link_main.c
  C:\targets\STM32L431_BearPi
  C:\Demos
    > hello_world_demo
    > oc_agriculture_template
    > oc_cloud_map_demo
    > oc_infrared_template
    > oc_manhole_cover_template
    > defaults.sdkconfig
    C:\oc_manhole_cover_templa... 7
    > oc_mqtt_demo
    > oc_smoke_template
    > defaults.sdkconfig
    C:\oc_smoke_template.c
    > oc_streetlight_template
    > oc_track_template
    > user_demo.mk
    > GCC

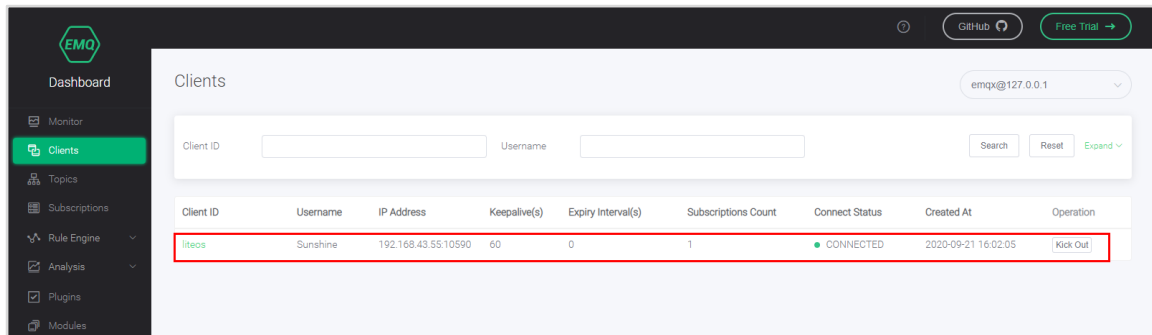
C:\oc_manhole_cover_template.c X
  56 typedef char int24s;
  57 typedef int int32s;
  58 typedef char string;
  59 typedef char array;
  60 typedef char varstring;
  61 typedef char variant;
  62
  63 // Connect to the basic variables of the broker.
  64 #define DEFAULT_LIFETIME 60
  65 #define DEFAULT_SERVER_IPV4 "192.168.43.50"
  66 #define DEFAULT_SERVER_PORT 1883
  67 #define CN_MQTT_EP_CLIENTID "liteos"
  68 #define CN_MQTT_EP_USERNAME "Sunshine"
  69 #define CN_MQTT_EP_PASSWD "12345678"
  70 #define CN_MQTT_EP_PUB_TOPIC1 "ManholeCover/Data"
  71
  72 //Variable for storing data
  73 E53_SC2_Data_TypeDef E53_SC2_Data;
  74 char Status[5];
  75
  76 //Data reporting task
  77 static int app_report_task_entry()
  78 {
  79     int ret = -1;
  80     void *handle = NULL;
  81
  82     mqtt_al_conpara_t config;
  
```

## 5.2.3 Compiling and Burning the Program

Step 1 Compile and burn the program and view the result.

Enable the Wi-Fi hotspot on the mobile phone and click  to recompile the program.

Burn the program and check whether the device is online in EMQ Broker.

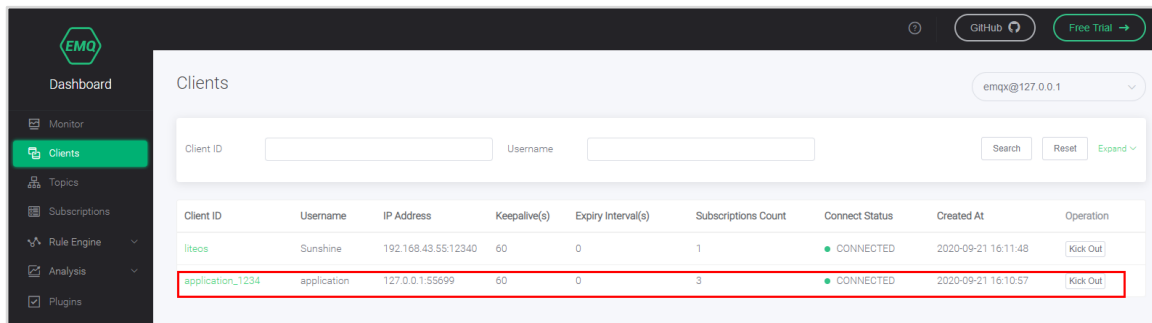


## 5.2.4 Running Applications

Step 1 Run the Java program to view the smart manhole cover data.

Right-click **HCIP-IoTEN** and choose **Run As > Java Application** from the shortcut menu.

On the Broker page, the device is online.



View the real-time data on the console and view the status change by flipping the development board.

```
Content of the received message:Temperature:21,Status:Level
Receive Message Subject:ManholeCover/Data
Content of the received message:Temperature:21,Status:Level
```

# 6 Wi-Fi-based Human Body Sensor Exercise

## 6.1 Introduction

### 6.1.1 About This Exercise

In this exercise, you will use Wi-Fi to implement a human body sensor case, which involves collecting real-time data, responding to command delivery, and implementing device-cloud synergy.

### 6.1.2 Objectives

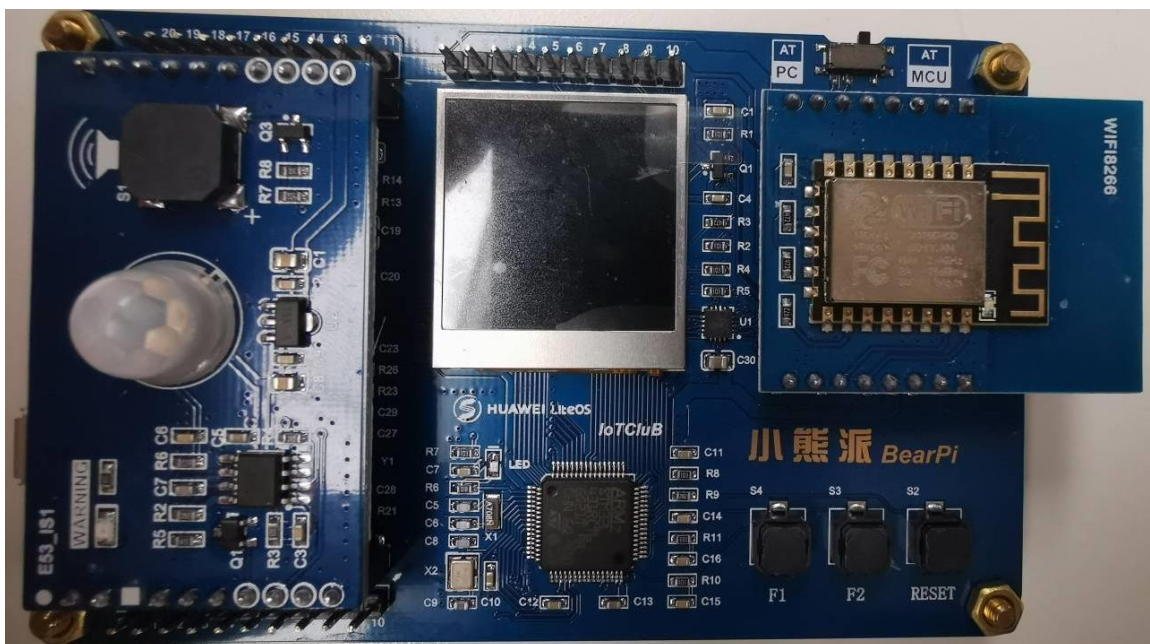
- Master how to configure the Wi-Fi communication mode.
- Master how to develop human body sensor cases.

## 6.2 Tasks

### 6.2.1 Configuring a Human Body Sensor Case

Step 1 Install the human body sensor expansion board E53\_IS1.

Insert the expansion board E53\_IS1 into the BearPi development board.



## Step 2 Modify the .config file.

Choose **targets > STM32L431\_BearPi > .config**.

Set **CONFIG\_USER\_DEMO** to **oc\_infrared\_template**.

Set **CONFIG\_ESP8266\_SSID** to the Wi-Fi username.

Set **CONFIG\_ESP8266\_PWD** to the Wi-Fi password.

Press **Ctrl+S** to save the **.config** file.

## Step 3 Modify the **iot\_config.h** file.

Choose **targets > STM32L431\_BearPi > iot\_config.h**.

Set **CONFIG\_USER\_DEMO** to **oc\_infrared\_template**.

Set **CONFIG\_ESP8266\_SSID** to the Wi-Fi username.

Set **CONFIG\_ESP8266\_PWD** to the Wi-Fi password.

Press **Ctrl+S** to save the **iot\_config.h** file.

## 6.2.2 Configuring the IP Address of the Broker

### Step 1 Query the IP address of the Broker.

The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\>ipconfig

Windows IP Configuration

Ethernet adapter :

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Wireless LAN adapter * 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Wireless LAN adapter * 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

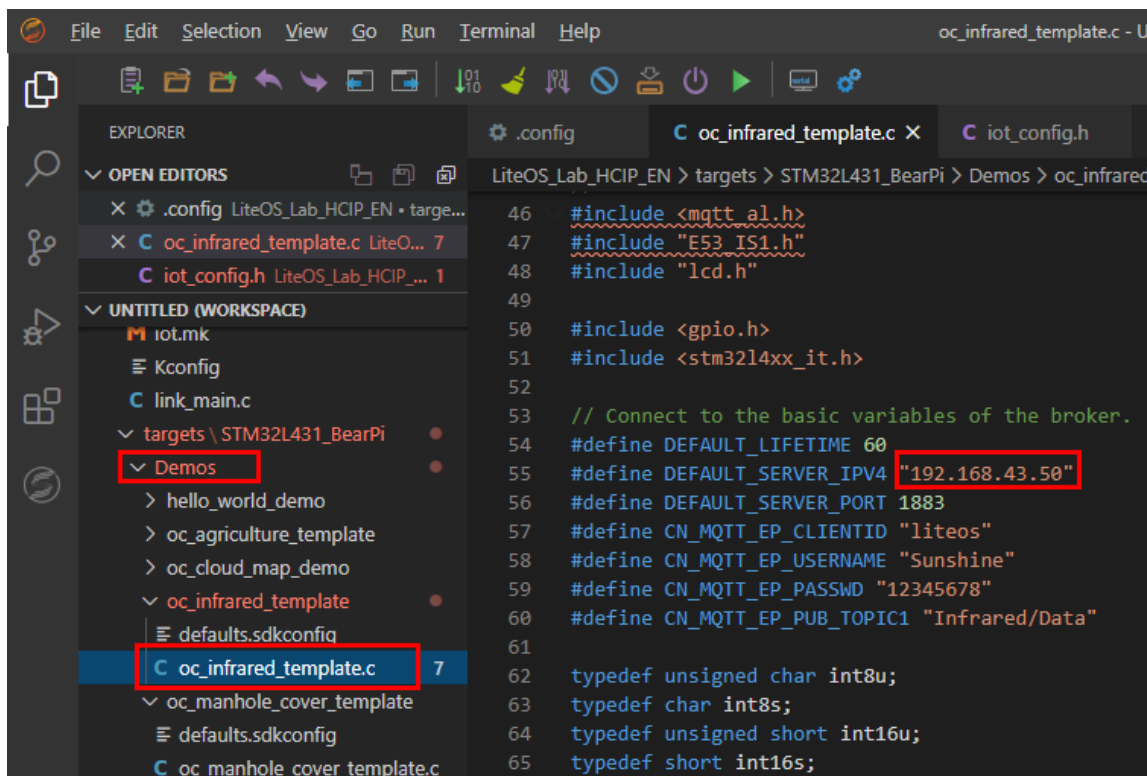
Wireless LAN adapter WLAN:

    Connection-specific DNS Suffix . :
    IPv4 Address. . . . . : 192.168.43.50
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.43.1
```

### Step 2 Change the IP address in the code.


Choose **STM32L431\_BearPi > Demos > oc\_infrared\_template > oc\_infrared\_template.c**.

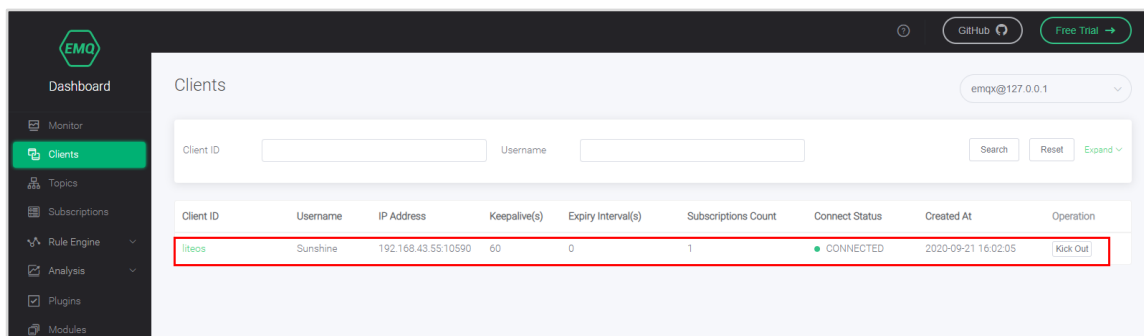
Set **DEFAULT\_SERVER\_IPV4** to the queried IP address.



## 6.2.3 Compiling and Burning the Program

Step 1 Compile and burn the program and view the result.

Enable the Wi-Fi hotspot on the mobile phone and click  to recompile the program. Burn the program and check whether the device is online in EMQ Broker.

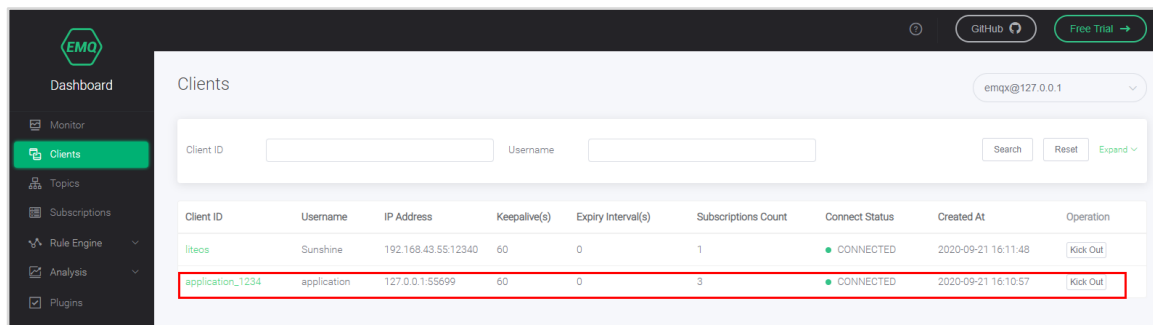


## 6.2.4 Running Applications

Step 1 Run the Java program to view human body sensor data.

Right-click **HCIP-IoTEN** and choose **Run As > Java Application** from the shortcut menu. On the Broker page, the device is online.





EMQX Dashboard

Monitor

Clients

Topics

Subscriptions

Rule Engine

Analysis

Plugins

Clients

emqx@127.0.0.1

Client ID Username IP Address Keepalive(s) Expiry Interval(s) Subscriptions Count Connect Status Created At Operation

Client ID	Username	IP Address	Keepalive(s)	Expiry Interval(s)	Subscriptions Count	Connect Status	Created At	Operation
liteos	Sunshine	192.168.43.55:12340	60	0	1	CONNECTED	2020-09-21 16:11:48	Kick Out
application_1234	application	127.0.0.1:55699	60	0	3	CONNECTED	2020-09-21 16:10:57	Kick Out

View the real-time status on the console. When a human body enters the sensing area, the status changes.

```

receive message subject:infrared/data
Content of the received message:Status: NO
Receive Message Subject:Infrared/Data
Content of the received message:Status:Have
  
```

# 7

## Wi-Fi-based Vending Machine Exercise

---

### 7.1 Introduction

#### 7.1.1 About This Exercise

In this exercise, you will use Wi-Fi to implement a vending machine case, which involves collecting real-time data, responding to command delivery, and implementing device-cloud synergy.

#### 7.1.2 Objectives

- Master how to configure the Wi-Fi communication mode.
- Master how to develop vending machine cases.

### 7.2 Tasks

#### 7.2.1 Configuring a Vending Machine Case

Step 1 Remove the sensor expansion board.

The vending machine case does not require a sensor expansion board.

Step 2 Modify the `.config` file.

Choose **targets > STM32L431\_BearPi > .config**.

Set `CONFIG_USER_DEMO` to `oc_vending_machine_template`.

Set `CONFIG_ESP8266_SSID` to the Wi-Fi username.

Set `CONFIG_ESP8266_PWD` to the Wi-Fi password.

Press **Ctrl+S** to save the `.config` file.

Step 3 Modify the `iot_config.h` file.

Choose **targets > STM32L431\_BearPi > iot\_config.h**.

Set `CONFIG_USER_DEMO` to `oc_vending_machine_template`.

Set `CONFIG_ESP8266_SSID` to the Wi-Fi username.

Set `CONFIG_ESP8266_PWD` to the Wi-Fi password.

Press **Ctrl+S** to save the `iot_config.h` file.



## 7.2.2 Configuring the IP Address of the Broker

Step 1 Query the IP address of the Broker.

The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\>ipconfig

Windows IP Configuration

Ethernet adapter :

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Wireless LAN adapter * 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Wireless LAN adapter * 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

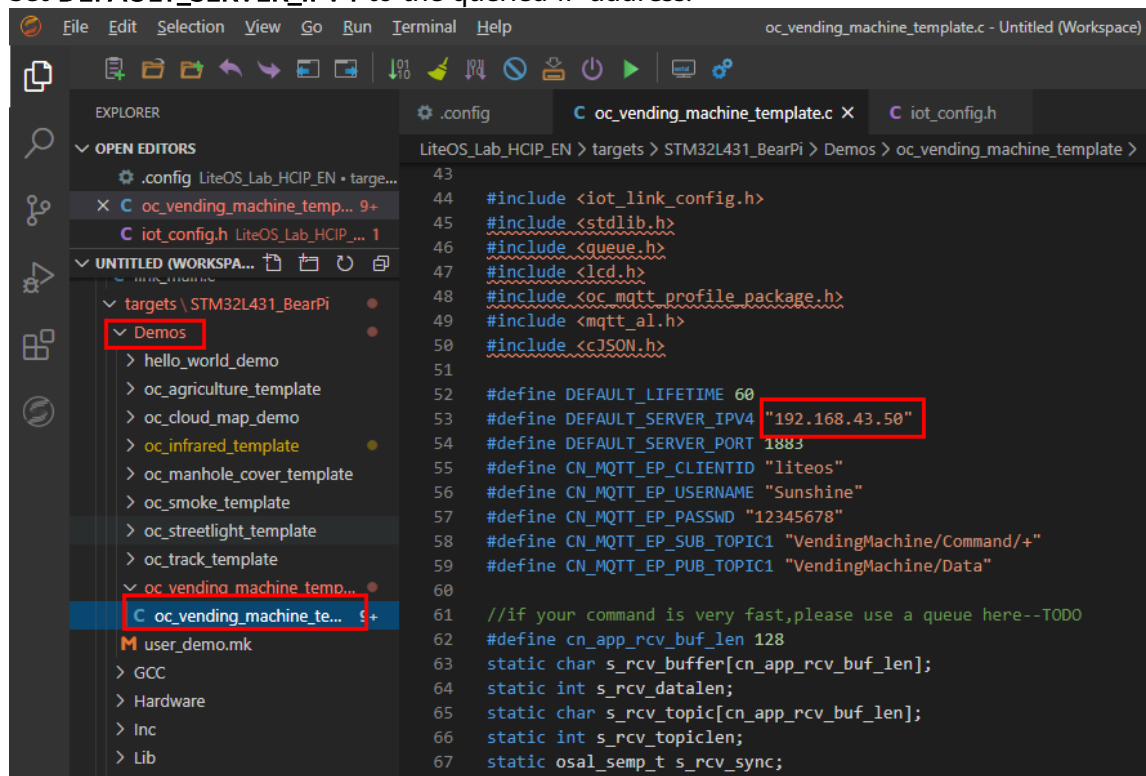
Wireless LAN adapter WLAN:

    Connection-specific DNS Suffix . :
    IPv4 Address. . . . . : 192.168.43.50
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.43.1
```

Step 2 Change the IP address in the code.

Choose **STM32L431\_BearPi** > **Demos** > **oc\_vending\_machine\_template** > **oc\_vending\_machine\_template.c**.

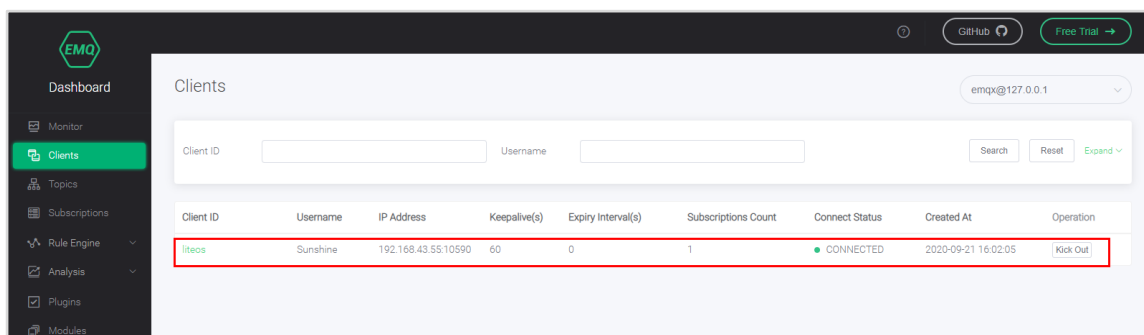
Set **DEFAULT\_SERVER\_IPV4** to the queried IP address.



## 7.2.3 Compiling and Burning the Program

Step 1 Compile and burn the program and view the result.

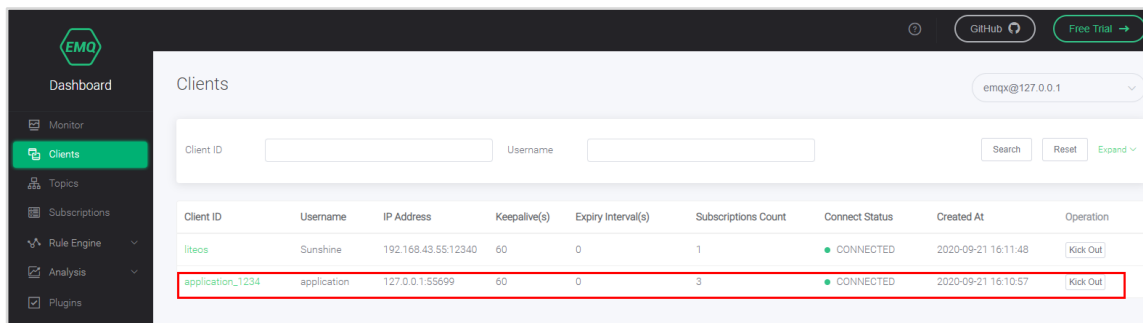
Enable the Wi-Fi hotspot on the mobile phone and click to recompile the program. Burn the program and check whether the device is online in EMQ Broker.



## 7.2.4 Running Applications

Step 1 Run the Java program to view the vending machine data.

Right-click **HCIP-IoTEN** and choose **Run As > Java Application** from the shortcut menu. On the Broker page, the device is online.



Client ID	Username	IP Address	Keepalive(s)	Expiry Interval(s)	Subscriptions Count	Connect Status	Created At	Operation
liteos	Sunshine	192.168.43.55:12340	60	0	1	CONNECTED	2020-09-21 16:11:48	Kick Out
application_1234	application	127.0.0.1:55699	60	0	3	CONNECTED	2020-09-21 16:10:57	Kick Out

View the real-time data on the console.

```
Receive Message Subject:VendingMachine/Data
Content of the received message:{"services":[{"service_id":"order","properties":{"orderId":"10000001","userID":"377743","userAge":23,"deviceId":"WZ_1-001
```

Step 2 Run the Java program to deliver the command for changing the offering sequence.

Choose HCIP-IoTEN > src > app > Application.java.

Uncomment the command for changing the offering sequence and run the program again.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    MqttClient client = connect();
    subscribe(client);
    // Command(client, pubTopicAgricultureLight, "ON");
    // Command(client, pubTopicAgricultureLight, "OFF");
    // Command(client, pubTopicAgricultureMotor, "ON");
    // Command(client, pubTopicAgricultureMotor, "OFF");
    // Command(client, pubTopicSmokeBeep, "ON");
    // Command(client, pubTopicSmokeBeep, "OFF");
    // Command(client, pubTopicTrackBeep, "ON");
    // Command(client, pubTopicTrackBeep, "OFF");
    Command(client, pubTopicVendingMachine, "9123456780");
}
```

The offering sequence on the LCD of the development board is changed.

# 8 Comprehensive Exercise

---

## 8.1 Introduction

### 8.1.1 About This Exercise

In this exercise, you will implement data reporting and command delivery based on device-cloud synergy in the previous exercises.

### 8.1.2 Objectives

- Master how to use the IoT platform and Huawei LiteOS to implement device-cloud synergy in different cases.

## 8.2 Tasks

### 8.2.1 Wi-Fi-based Smart Logistics Exercise

Step 1 Change the cases in `.config` and `iot_config.h` to `oc_track_template`.

Step 2 Modify the code in `oc_track_template.c` to use the MQTT protocol to report data and process command responses.

### 8.2.2 Wi-Fi-based Smart Street Lamp Exercise

Step 1 Change the cases in `.config` and `iot_config.h` to `oc_streetlight_template`.

Step 2 Modify the code in `oc_streetlight_template.c` to use the MQTT protocol to report data and process command responses.