Huawei IoT Certification Training

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

ISSUE: 2.5

Huawei Technologies Co., Ltd.

Trademarks and Permissions

HUAWEI 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
Huawei Certification
ICT Vertical Certification
Finance
Public Safety
Platform and Service Certification
Big Data
AI
IoT
Intelligent Vision
Enterprise Communication
Kunpeng Application Developer
GaussDB
Cloud Computing
Cloud Service
ICT Infrastructure Certification
Data Center
Storage
Intelligent Computing
Security
Datacom
WLAN
SDN
Transmission
Access
LTE
5G
HCIE
Huawei Certified ICT Expert
HCIP
Huawei Certified ICT Professional
HCIA
Huawei Certified ICT Associate

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

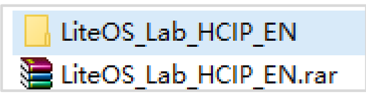
- Master how to use LiteOS Studio.
- Master how to execute LiteOS tasks.
- Become familiar with the LCD.
- Become familiar with the LED and buttons of the development board.

1.2 Tasks

1.2.1 Opening a LiteOS Project

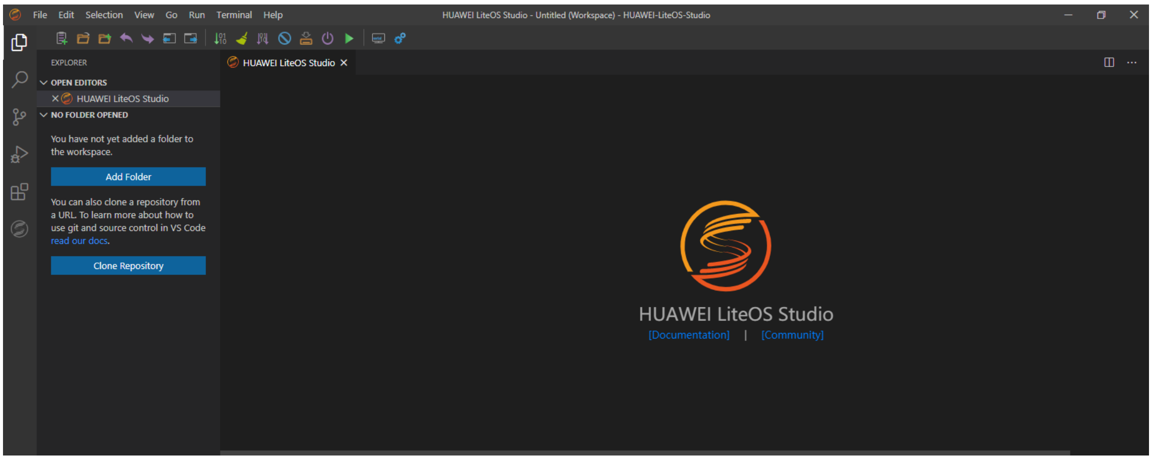
Step 1 Decompress the downloaded project.

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.

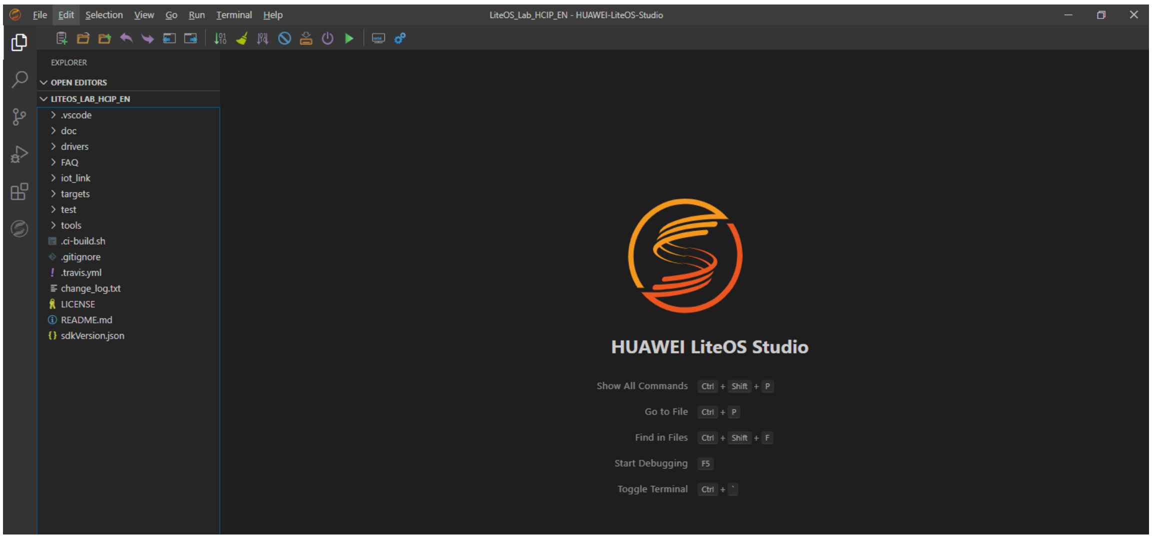


Step 2 Open the project.

Open the installed LiteOS Studio.



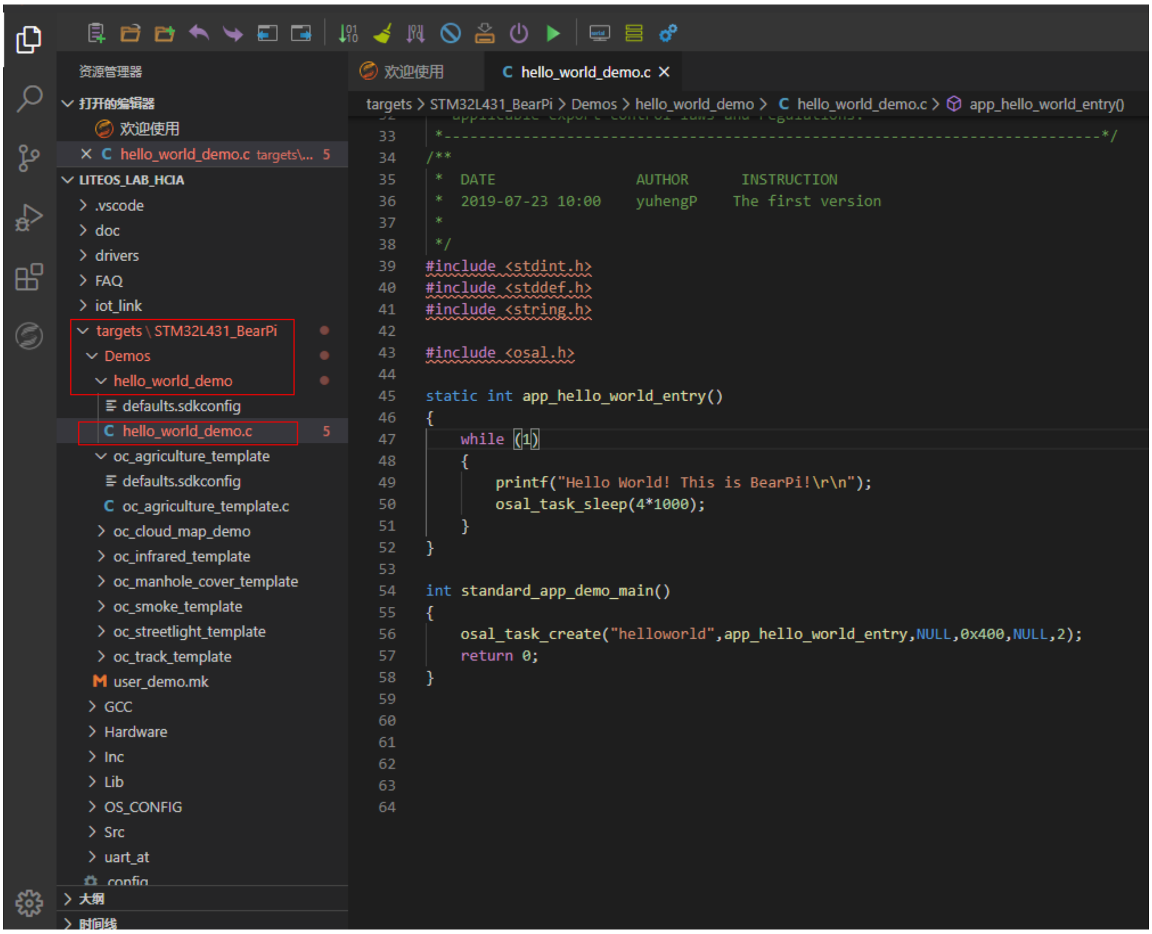
Choose **File** > **Open Folder** in the upper left corner and select the **LiteOS_Lab_HCIP_EN** folder.



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**.



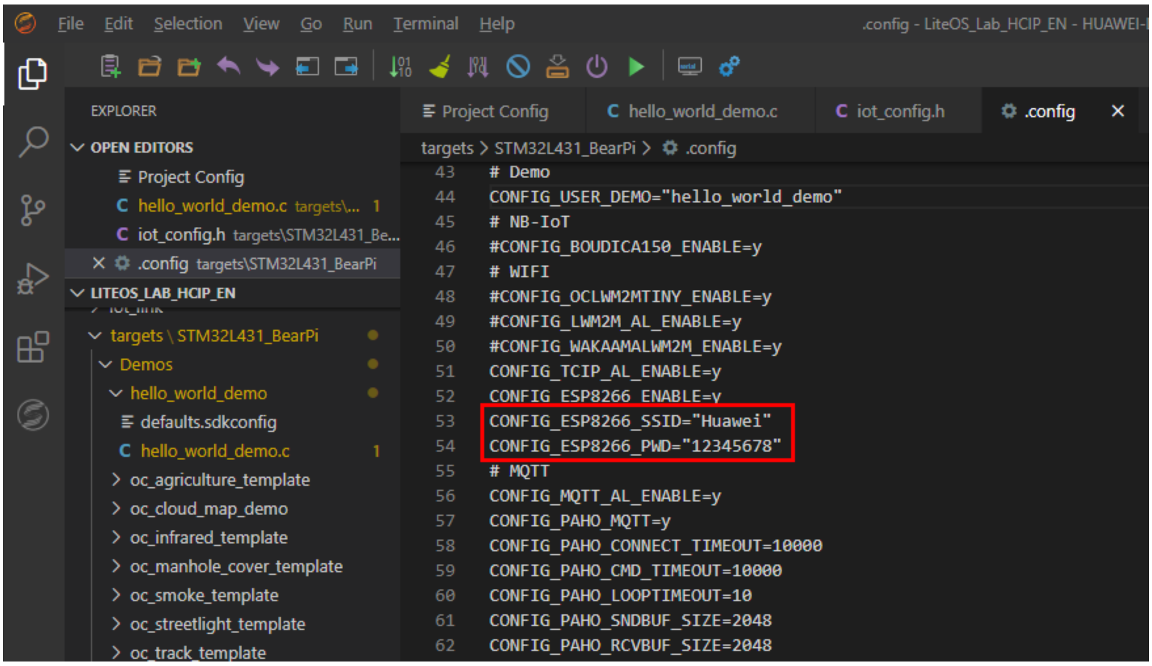
Step 2 Modify the **.config** file.

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.



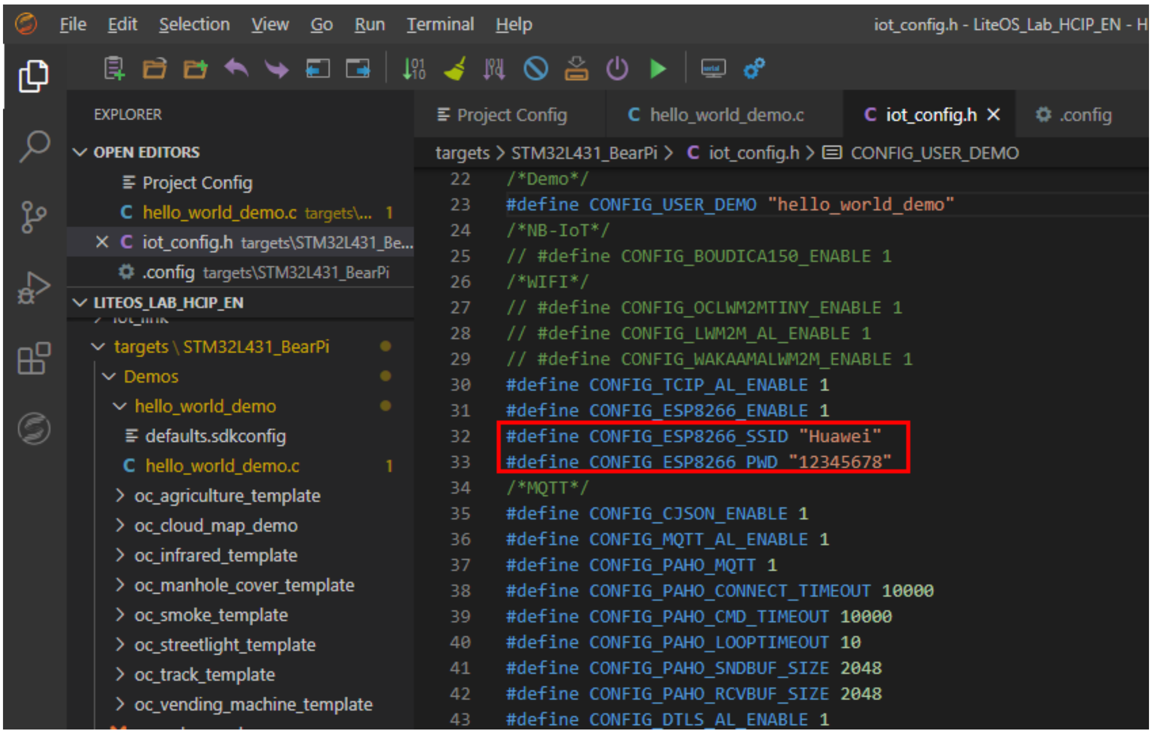
Step 3 Modify the **iot_config.h** file.

Choose **targets** > **STM32L431_BearPi** > **iot_config.h**.

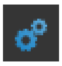
Set **CONFIG_ESP8266_SSID** to the Wi-Fi username.

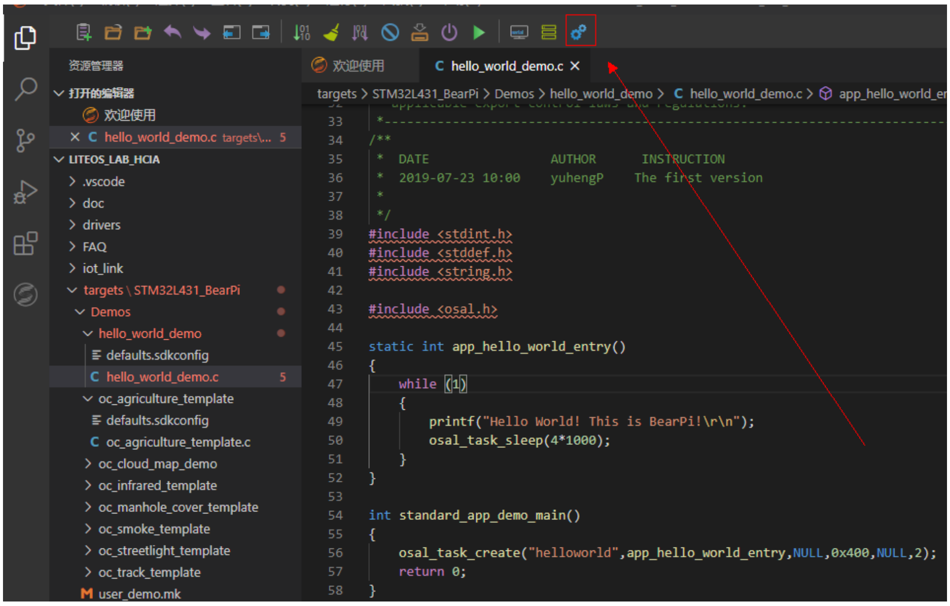
Set **CONFIG_ESP8266_PWD** to the Wi-Fi password.

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

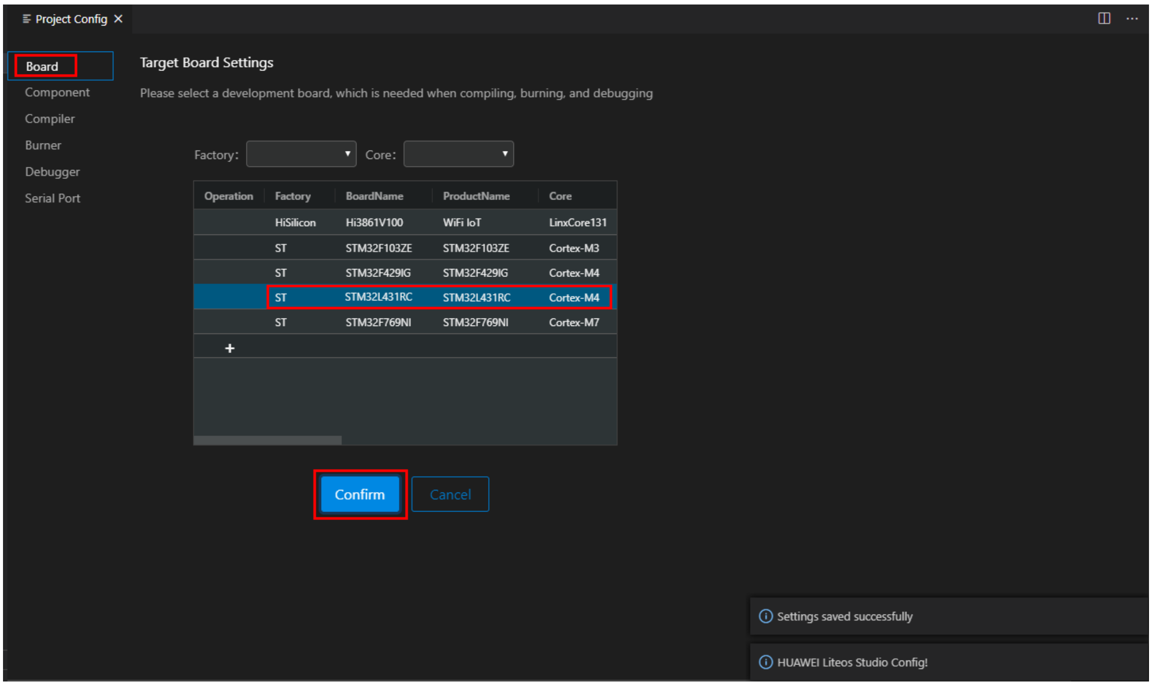


Step 4 Configure the project.

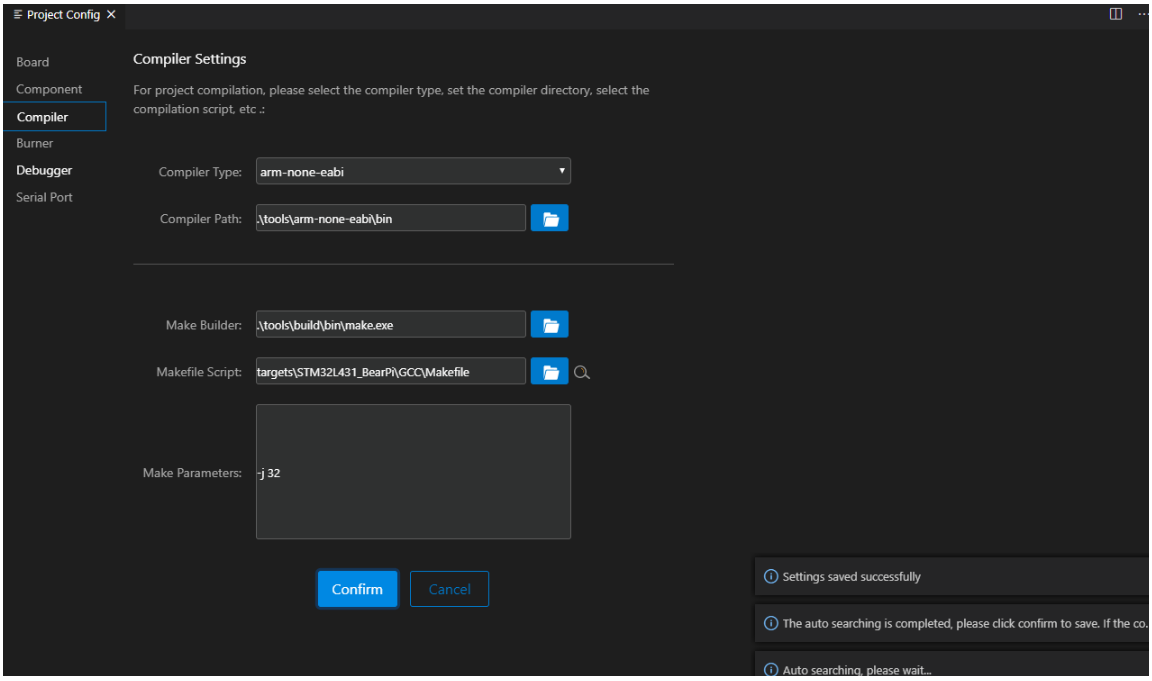
Click  on the toolbar.



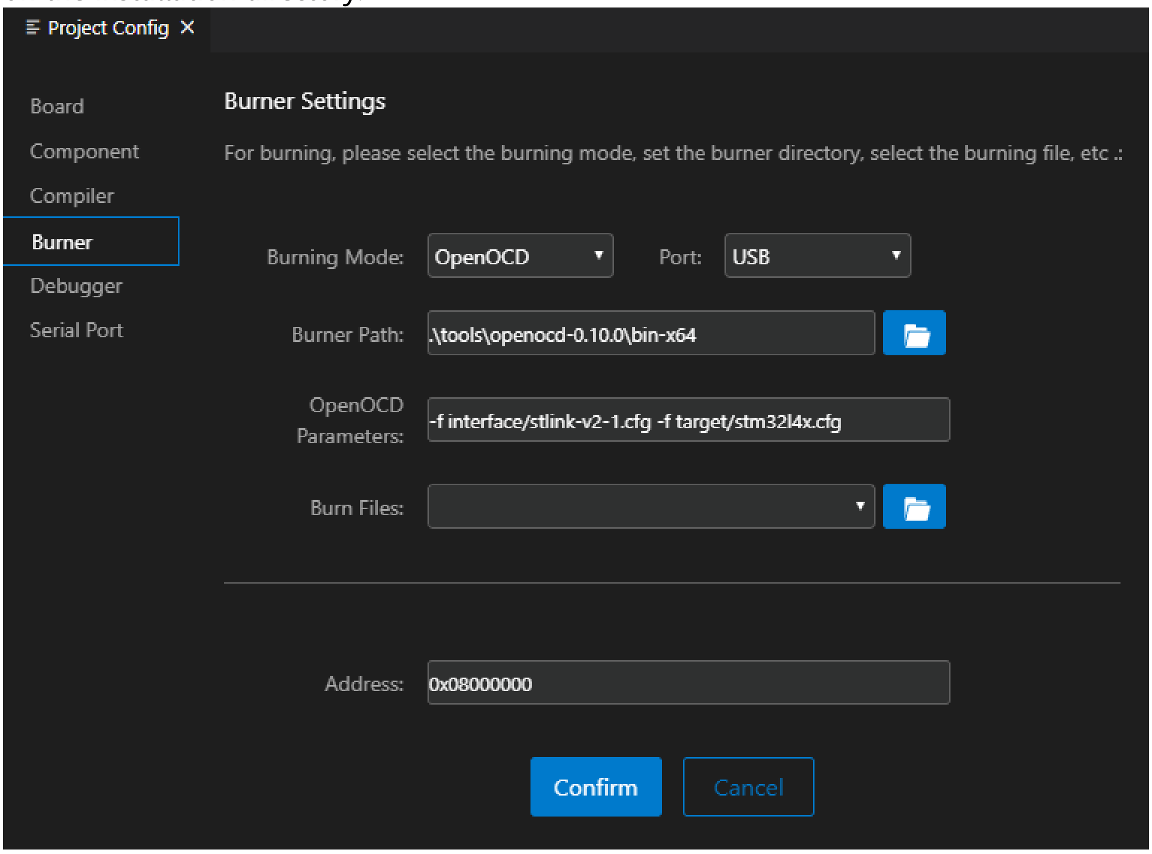
Select **STM32L431RC** as the target board and click **Confirm**.



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**.

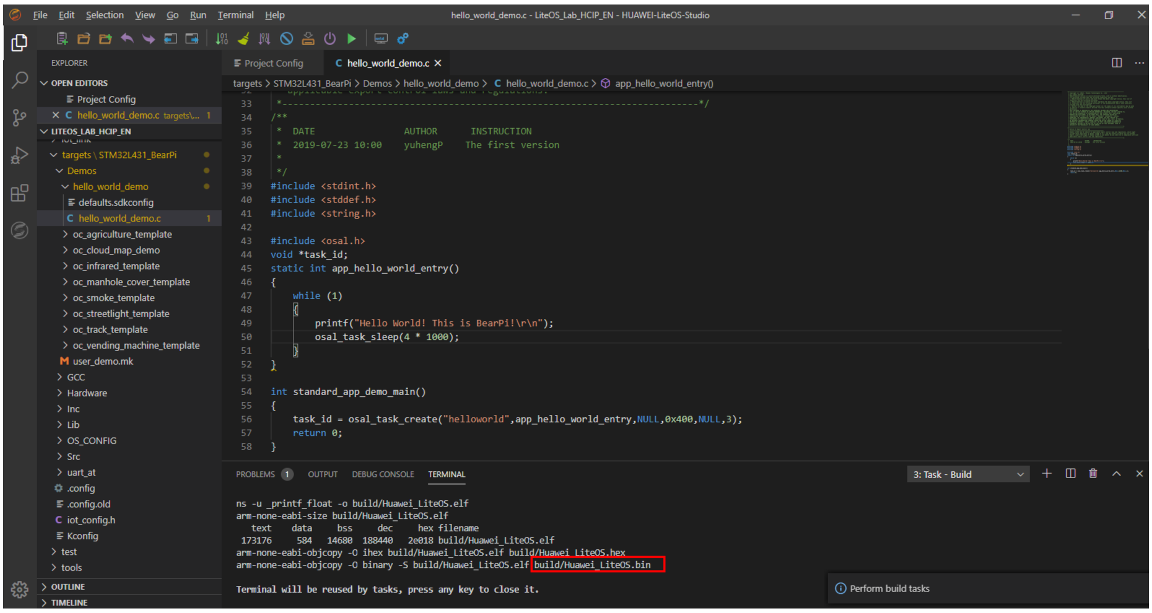


Choose **Burner**, set **Burning Mode** to **OpenOCD**, and click **Confirm**. Set **Burner Path** based on the installation directory.



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.

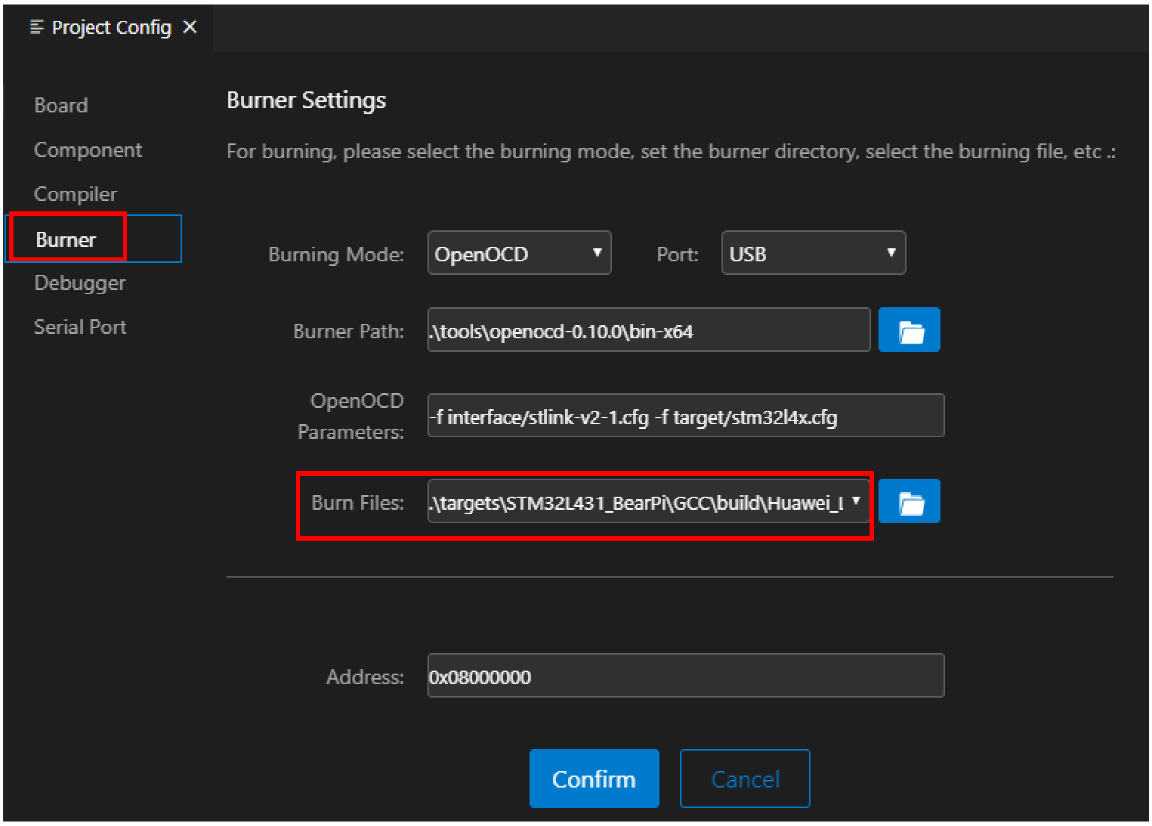


Step 6 Configure the development board.

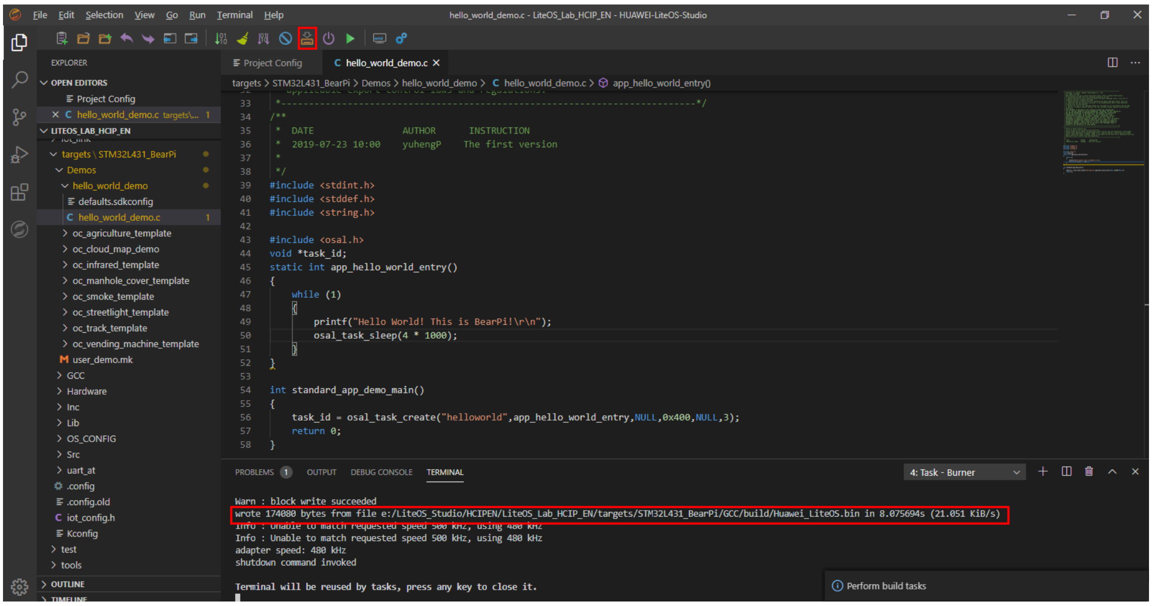
Set the switch of the serial port mode to AT<->MCU, and connect the development board to the PC using a USB cable.

Step 7 Burn the program.

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



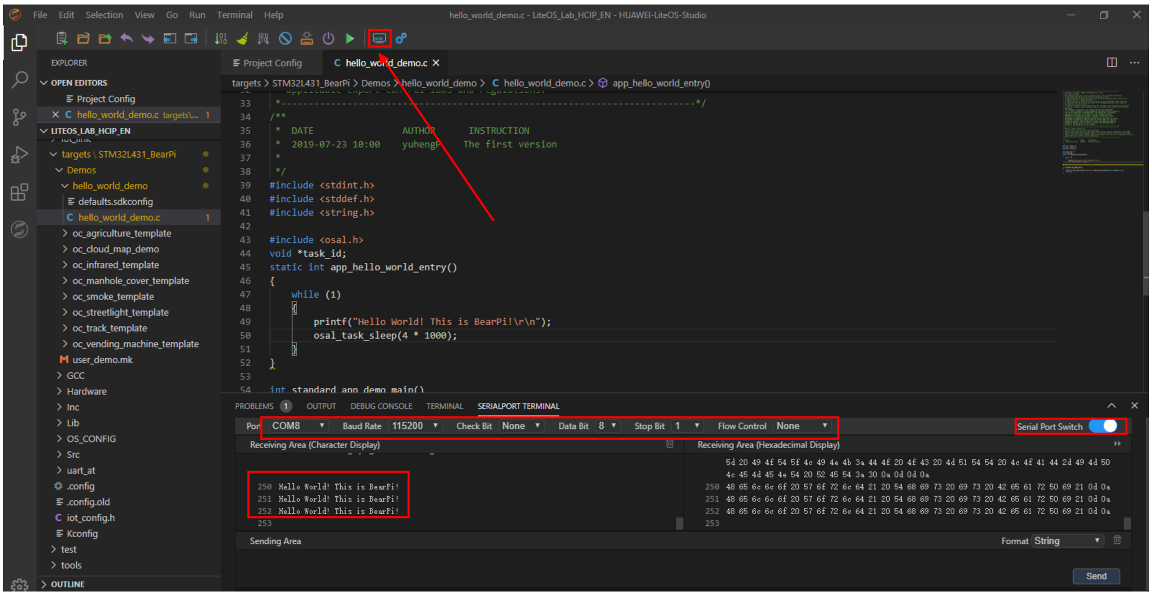
Open the hotspot on the phone, click  on the toolbar, and wait for the burning to complete.



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**.

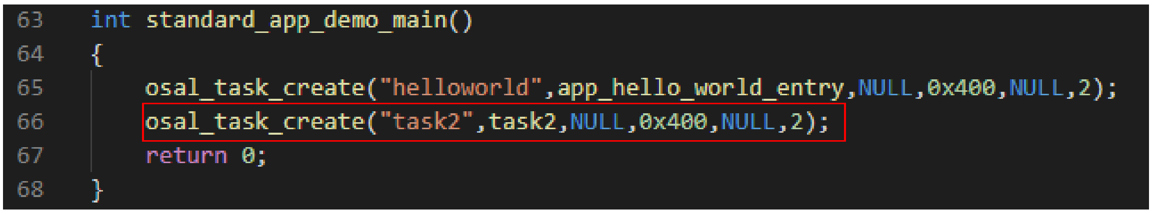
In **hello_world_demo.c**, add the code for executing **task2**.

```
static int task2()
```

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

Step 2 Create **task2**.

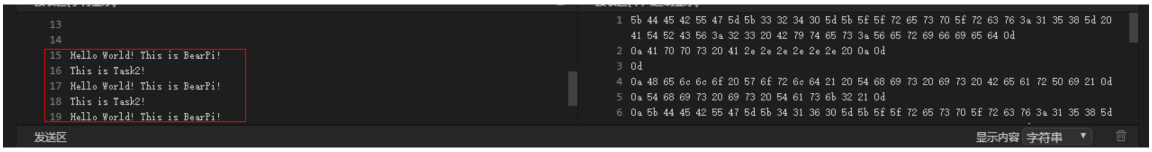
```
osal_task_create("task2",task2,NULL,0x400,NULL,2);
```



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.



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

Add a task ID.

```
void* task_id;
```

```
#include <stdint.h>
#include <stddef.h>
#include <string.h>

#include <osal.h>
void* task_id;
static int app_hello_world_entry()
{
    while (1)
    {
        printf("Hello World! This is BearPi!\r\n");
        osal_task_sleep(4*1000);
    }
}
```

Get the task ID.

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

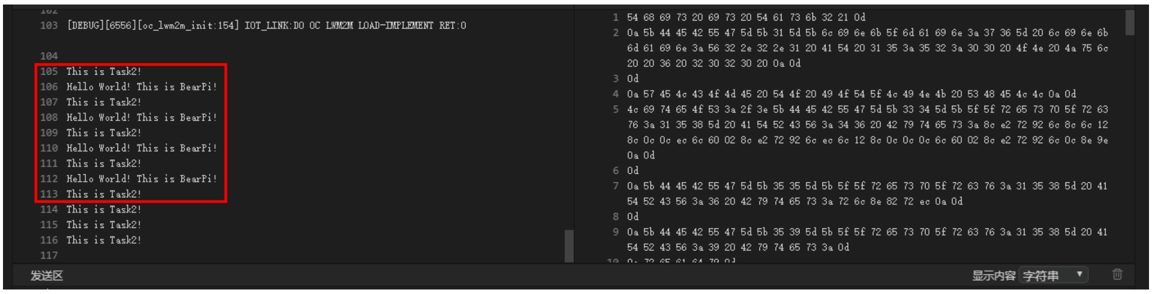
Add the code for deleting the task.

```
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);
}
```

```
static int task2()
{
    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);
    }
}
```

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.



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;
}
```

```
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;
}
```

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

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

```
    printf("\r\ntask2: 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;
}
```

```

static int mutex_task2_entry()
{
 while (1)
 {
  if(true == osal_mutex_lock(public_value_mutex))
  {
   printf("\r\ntask2: 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;
}

int standard_app_demo_main()
{
```

Step 3 Comment out the code for task management

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);
    return 0;
}
```

Step 4 Create a mutex.

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

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    osal_mutex_create(&public_value_mutex);
    return 0;
}
```

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);
```

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    osal_mutex_create(&public_value_mutex);
    osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);

    return 0;
}
```

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.

```
task2: lock a mutex.
task2: public_value = 5.
task2: unlock a mutex.


task1: lock a mutex.
task1: public_value = 15.
task1: sleep...
task1: continue...
task1: unlock a mutex.


task2: lock a mutex.
task2: public_value = 20.
task2: unlock a mutex.


task1: lock a mutex.
task1: public_value = 30.
task1: sleep...
task1: continue...
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;
        }
    }
}
```

```
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;
        }
    }
}

int standard_app_demo_main()
{
```

Step 2 Comment out the code for creating a mutex.

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);

    return 0;
}
```

Step 3 Create a memory task.

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

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);

    return 0;
}
```

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.

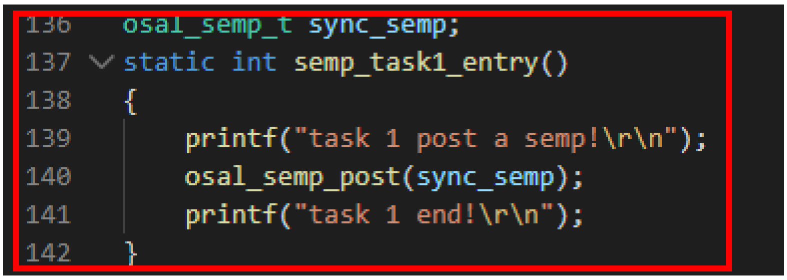
```
access 1 bytes memory success!
free memory success!
access 2 bytes memory success!
free memory success!
access 4 bytes memory success!
free memory success!
access 8 bytes memory success!
free memory success!
access 16 bytes memory success!
free memory success!
access 32 bytes memory success!
free memory success!
access 64 bytes memory success!
free memory success!
```

1.2.6 Creating a Semaphore

Step 1 Add the code of **task1**.

```
osal_semp_t sync_semp;
```

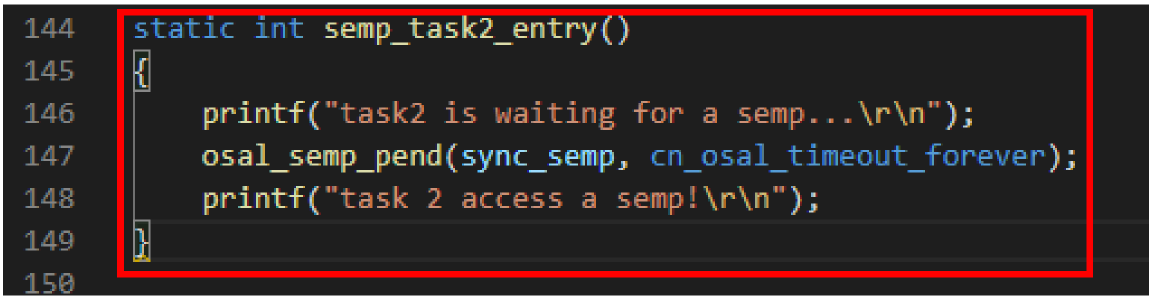
```
static int semp_task1_entry()
{
    printf("task 1 post a semp!\r\n");
    osal_semp_post(sync_semp);
    printf("task 1 end!\r\n");
}
```



```
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");
}
```

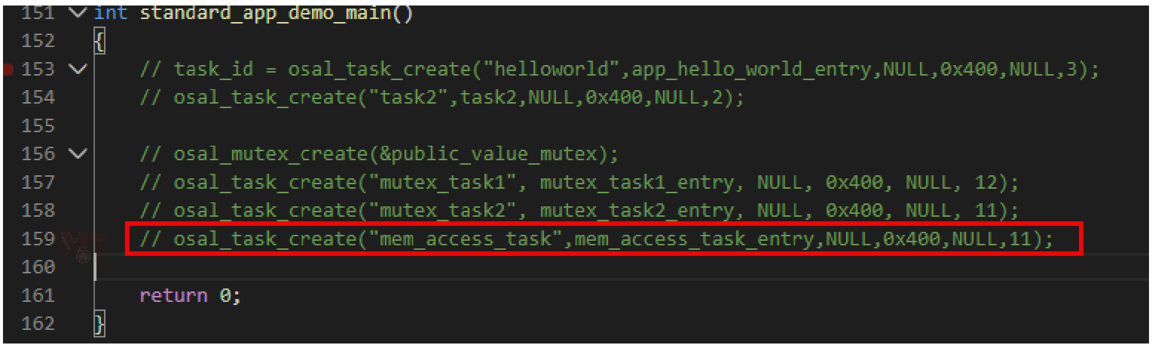
Step 2 Add the code of **task2**.

```
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");
}
```



```
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");
}
```

Step 3 Comment out the code for memory management.



```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);

    return 0;
}
```

Step 4 Create a semaphore.

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

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
    osal_semp_create(&sync_semp, 1, 0);
    printf("sync_semp semp create success.\r\n");

    return 0;
}
```

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);
```

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
    osal_semp_create(&sync_semp, 1, 0);
    printf("sync_semp semp create success.\r\n");
    osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
    osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);

    return 0;
}
```

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.

```
task2 is waiting for a semp...
task 1 post a semp!
task 2 access a semp!
task 1 end!
```

Step 7 Comment out the code for creating a semaphore.

```
int standard_app_demo_main()
{
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
    // osal_semp_create(&sync_semp, 1, 0);
    // printf("sync_semp semp create success.\r\n");
    // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
    // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);

    return 0;
}
```

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"
```

```
#include <stdint.h>
#include <stddef.h>
#include <string.h>

#include <osal.h>
#include "lcd.h"
void *task_id;
static int app_hello_world_entry()
{
    while (1)
    {
        printf("Hello World! This is BearPi!\r\n");
        osal_task_sleep(4 * 1000);
    }
}
```

Step 2 Add the code for clearing the LCD.

```
LCD_Clear(BLACK);
```

```
int standard_app_demo_main()
{
    LCD_Clear(BLACK);
    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
    // osal_semp_create(&sync_semp, 1, 0);
    // printf("sync_semp semp create success.\r\n");
    // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
    // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);

    return 0;
}
```

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

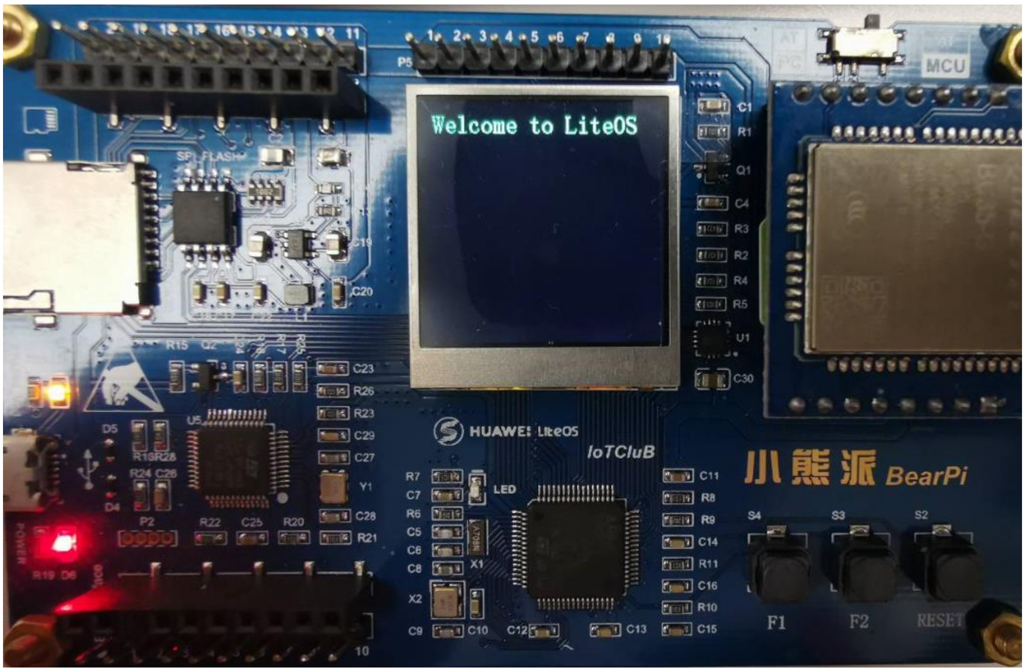
```
POINT_COLOR = GREEN;
LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");
```

```
int standard_app_demo_main()
{
    LCD_Clear(BLACK);
    POINT_COLOR = GREEN;
    LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");

    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);
```

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_InitStruct;
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    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);
    }
}
```

```c

static int led_task()
{
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    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);
    }
}

int standard_app_demo_main()
```

Step 2 Create an LED blinking task.

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

```c
int standard_app_demo_main()
{
    LCD_Clear(BLACK);
    POINT_COLOR = GREEN;
    LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");

    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
    // osal_semp_create(&sync_semp, 1, 0);
    // printf("sync_semp semp create success.\r\n");
    // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
    // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
    osal_task_create("led_task",led_task,NULL,0x400,NULL,2);

    return 0;
}
```

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.

```
static int led_task()
{
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    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);
    }
}
```

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);
}
```

```
static int led_task()
{
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    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);
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2)==GPIO_PIN_RESET)//查询按键KEY1低电平
        {
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
        }else if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_3)==GPIO_PIN_RESET)//查询按键KEY2低电平
        {
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
        }
    }
}
```

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);
}
```

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

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);
}
```

```
static Key2_interrupt_entry()
{
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    __HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_3);
}

```

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

```
int standard_app_demo_main()
{
    LCD_Clear(BLACK);
    POINT_COLOR = GREEN;
    LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");

    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
    // osal_semp_create(&sync_semp, 1, 0);
    // printf("sync_semp semp create success.\r\n");
    // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
    // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
    // osal_task_create("led_task",led_task,NULL,0x400,NULL,2);

    return 0;
}
```

Step 4 Create the interrupt functions.

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

```
int standard_app_demo_main()
{
    LCD_Clear(BLACK);
    POINT_COLOR = GREEN;
    LCD_ShowString(10, 10, 200, 16, 24, "Welcome to LiteOS");

    // task_id = osal_task_create("helloworld",app_hello_world_entry,NULL,0x400,NULL,3);
    // osal_task_create("task2",task2,NULL,0x400,NULL,2);

    // osal_mutex_create(&public_value_mutex);
    // osal_task_create("mutex_task1", mutex_task1_entry, NULL, 0x400, NULL, 12);
    // osal_task_create("mutex_task2", mutex_task2_entry, NULL, 0x400, NULL, 11);
    // osal_task_create("mem_access_task",mem_access_task_entry,NULL,0x400,NULL,11);
    // osal_semp_create(&sync_semp, 1, 0);
    // printf("sync_semp semp create success.\r\n");
    // osal_task_create("semp_task1",semp_task1_entry,NULL,0x400,NULL,12);
    // osal_task_create("semp_task2",semp_task2_entry,NULL,0x400,NULL,11);
    // osal_task_create("led_task",led_task,NULL,0x400,NULL,2);
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    osal_int_connect( EXTI2_IRQn, 3, NULL, Key1_interrupt_entry, NULL);
    osal_int_connect( EXTI3_IRQn, 4, NULL, Key2_interrupt_entry, NULL);

    return 0;
}
```

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

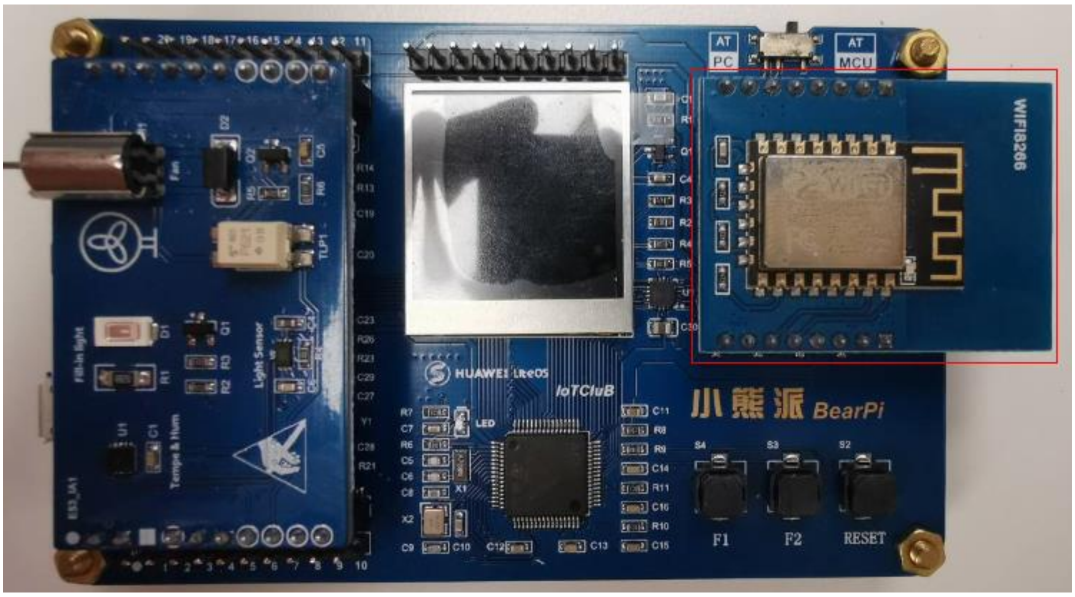
- Master how to configure the Wi-Fi communication mode.
- Master how to develop smart agriculture cases.

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.

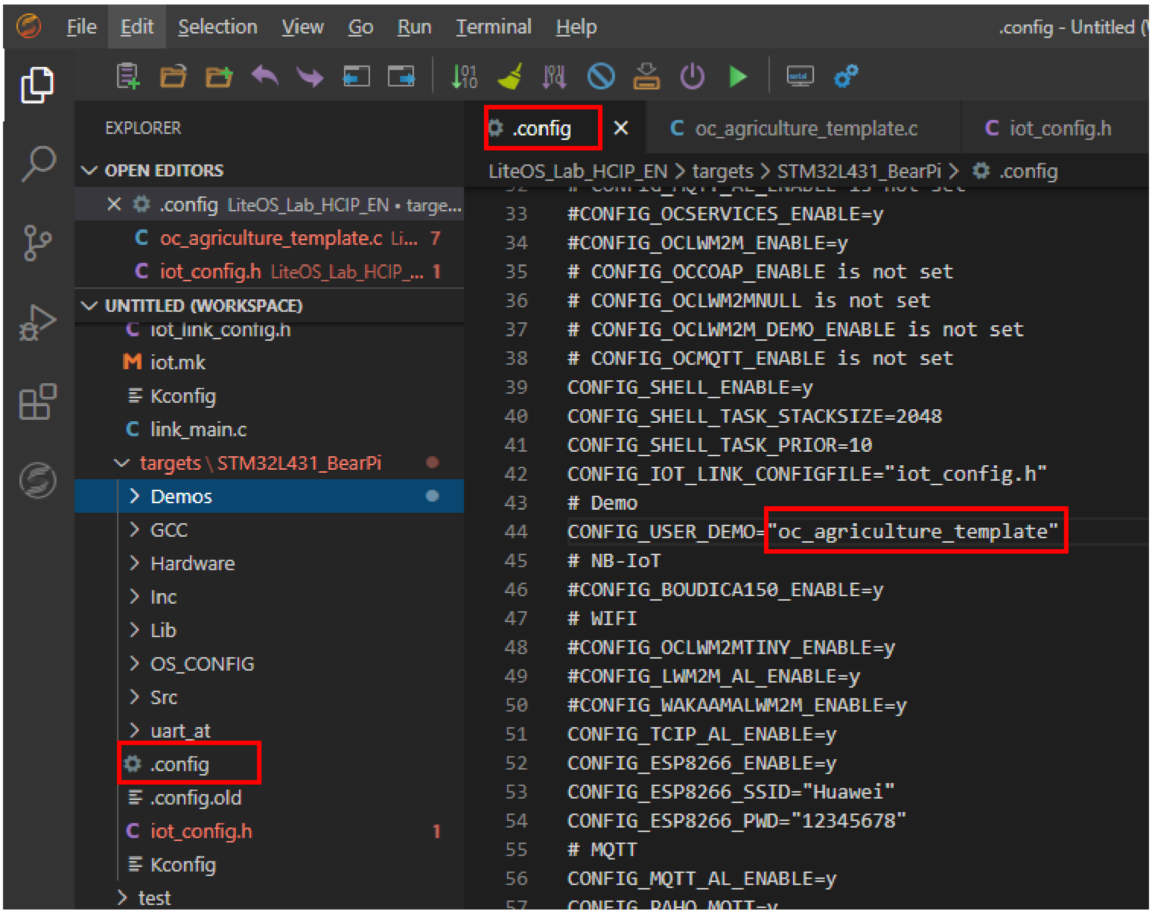
Choose **targets** > **STM32L431_BearPi** > **.config**.

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.

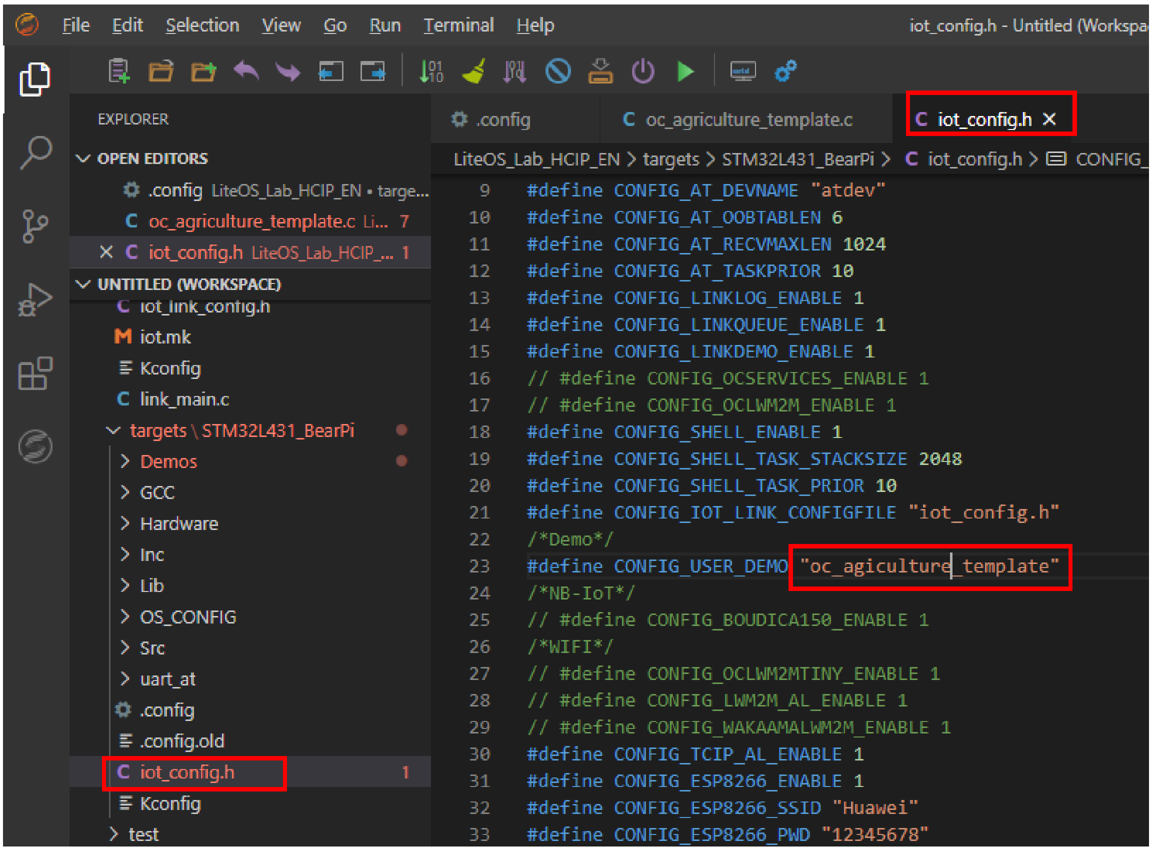
Choose **targets** > **STM32L431_BearPi** > **iot_config.h**.

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.

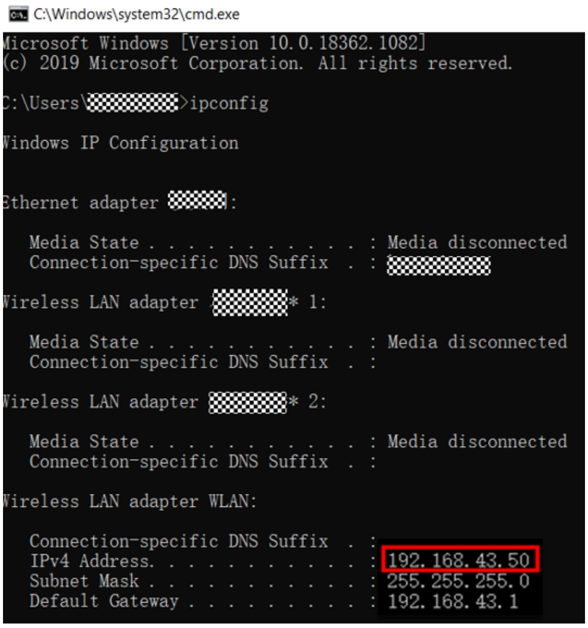
Press **Ctrl+S** to save the **iot_config.h** file.



3.2.2 Configuring the IP Address of the Broker

Step 1 Query the IP address of the Broker.

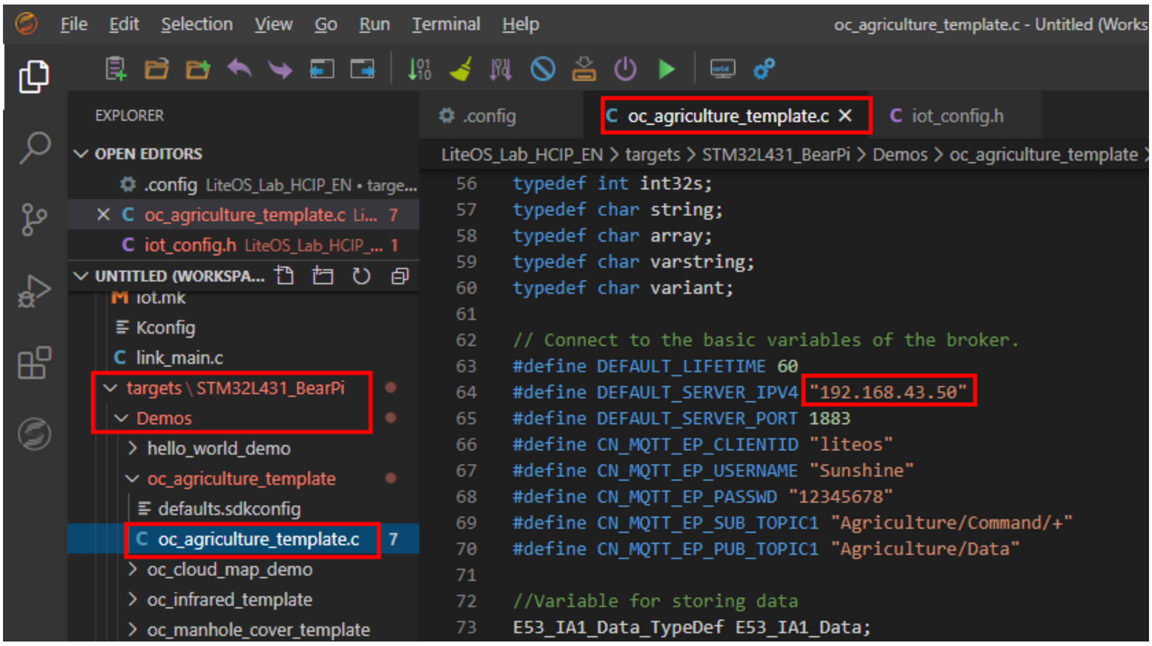
The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.



Step 2 Change the IP address in the code.

Choose **STM32L431_BearPi** > **Demos** > **oc_agriculture_template** > **oc_agriculture_template.c**.

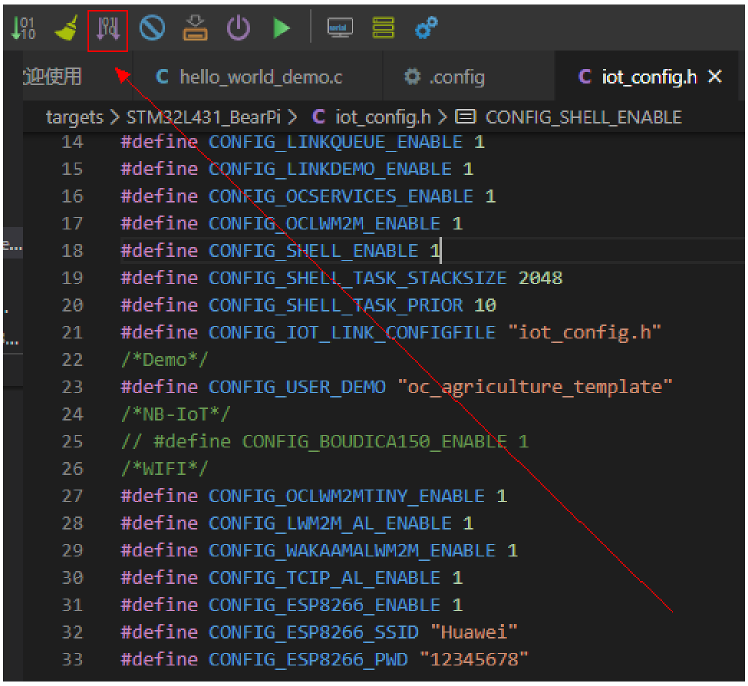
Set **DEFAULT_SERVER_IPV4** to the queried IP address.



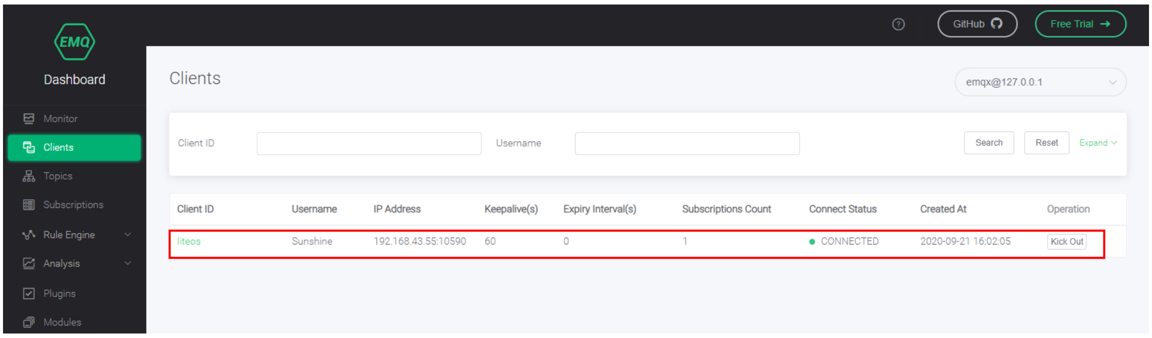
3.2.3 Compiling and Burning the Program

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.

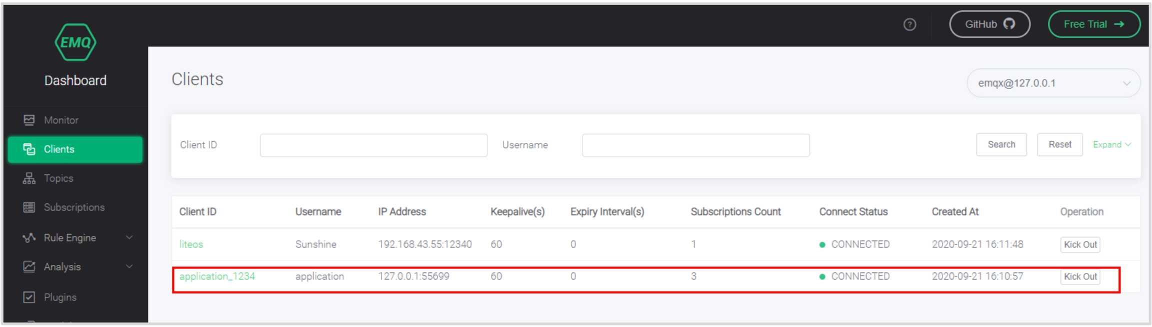


3.2.4 Running Applications

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

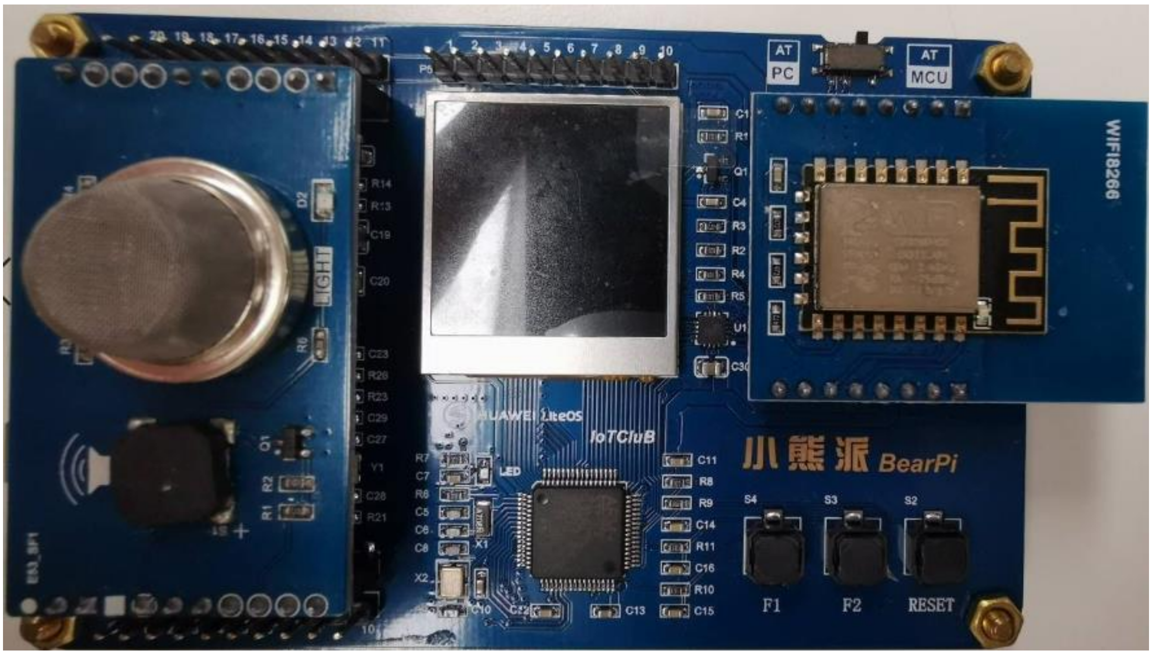
- Master how to configure the Wi-Fi communication mode.
- Master how to develop smart smoke detector cases.

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.

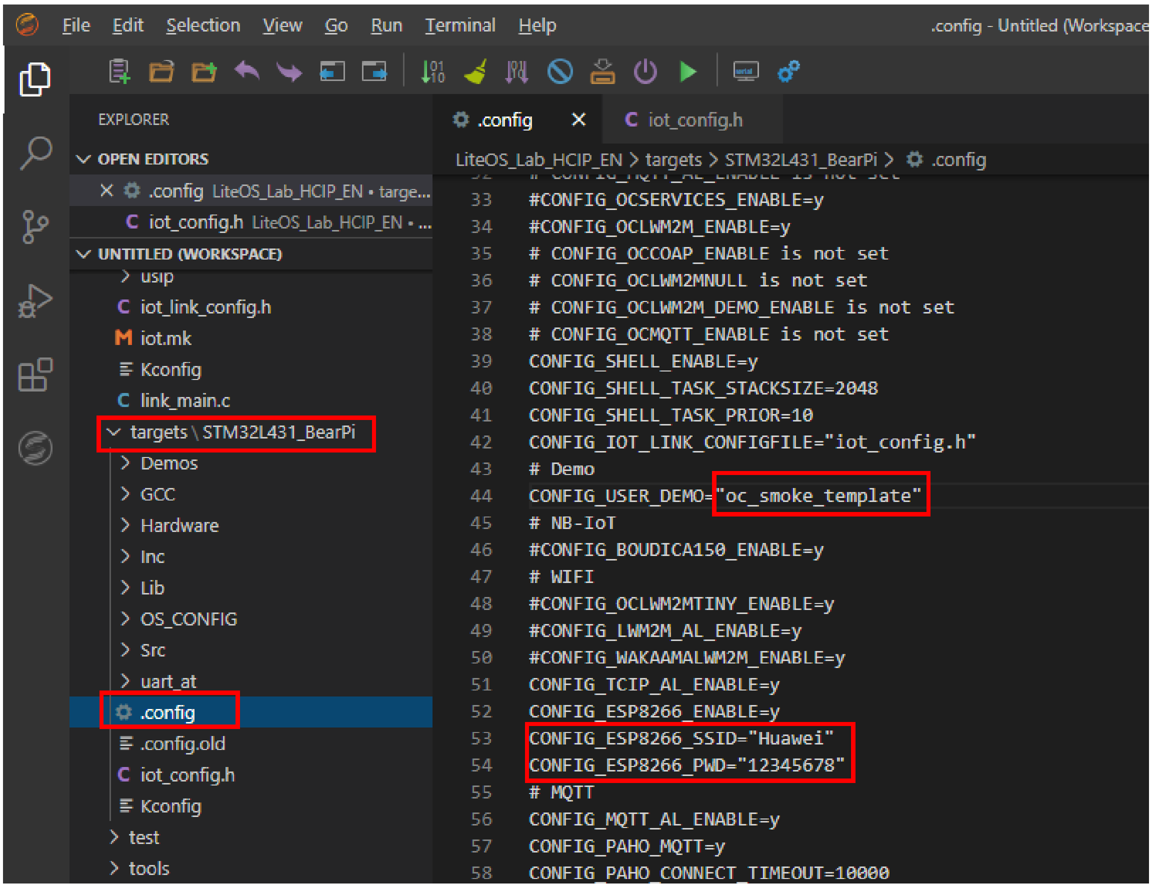
Choose **targets** > **STM32L431_BearPi** > **.config**.

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.

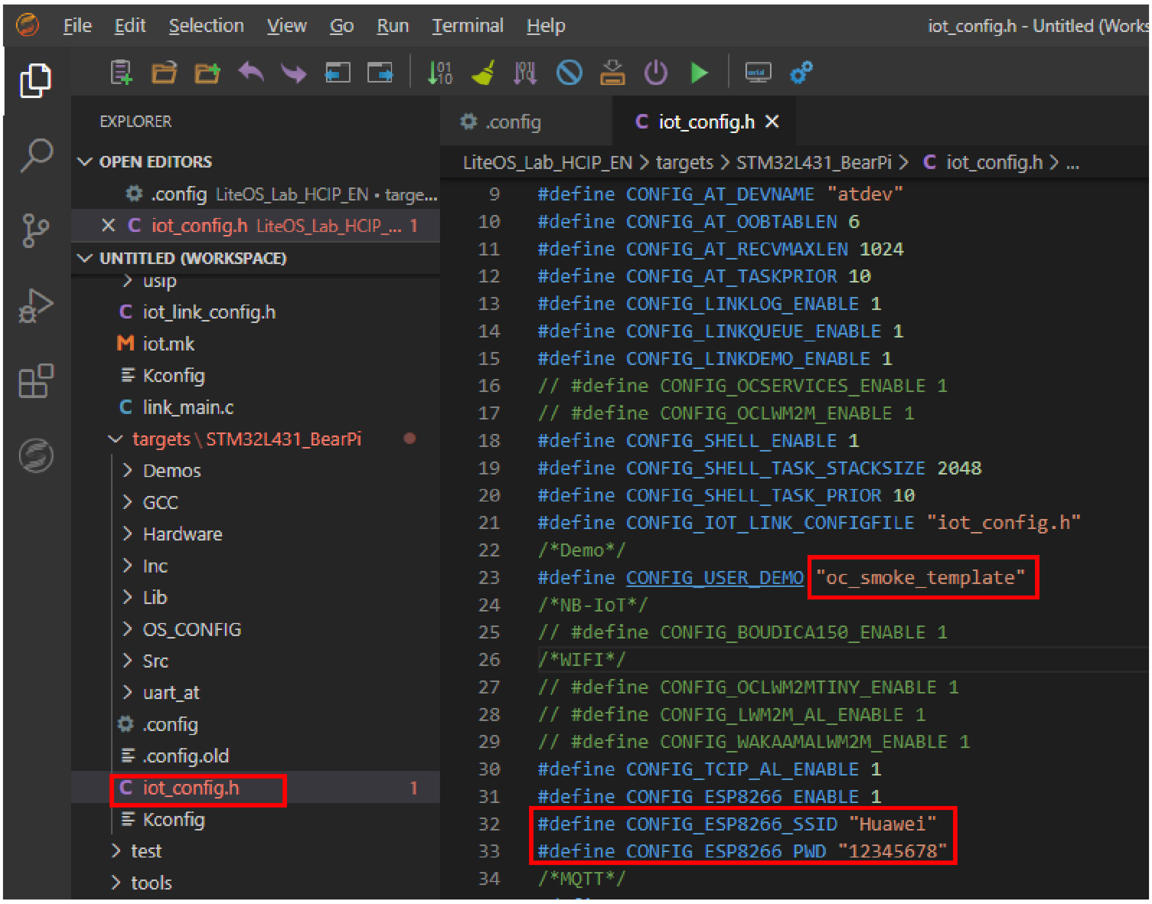
Choose **targets** > **STM32L431_BearPi** > **iot_config.h**.

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.

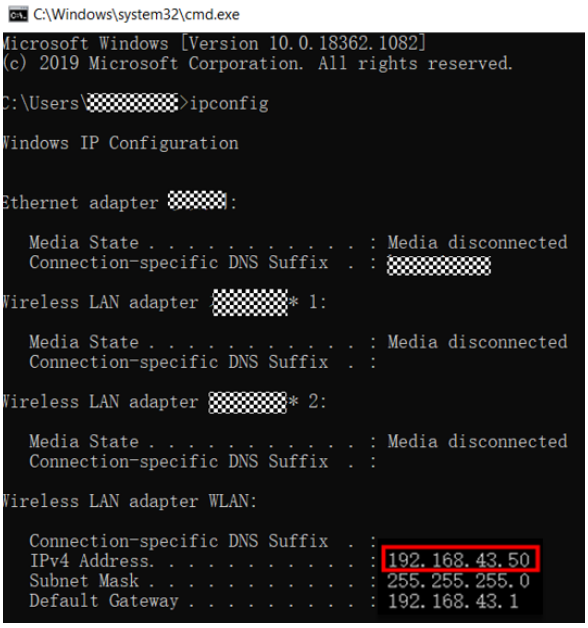
Press **Ctrl+S** to save the **iot_config.h** file.



4.2.2 Configuring the IP Address of the Broker

Step 1 Query the IP address of the Broker.

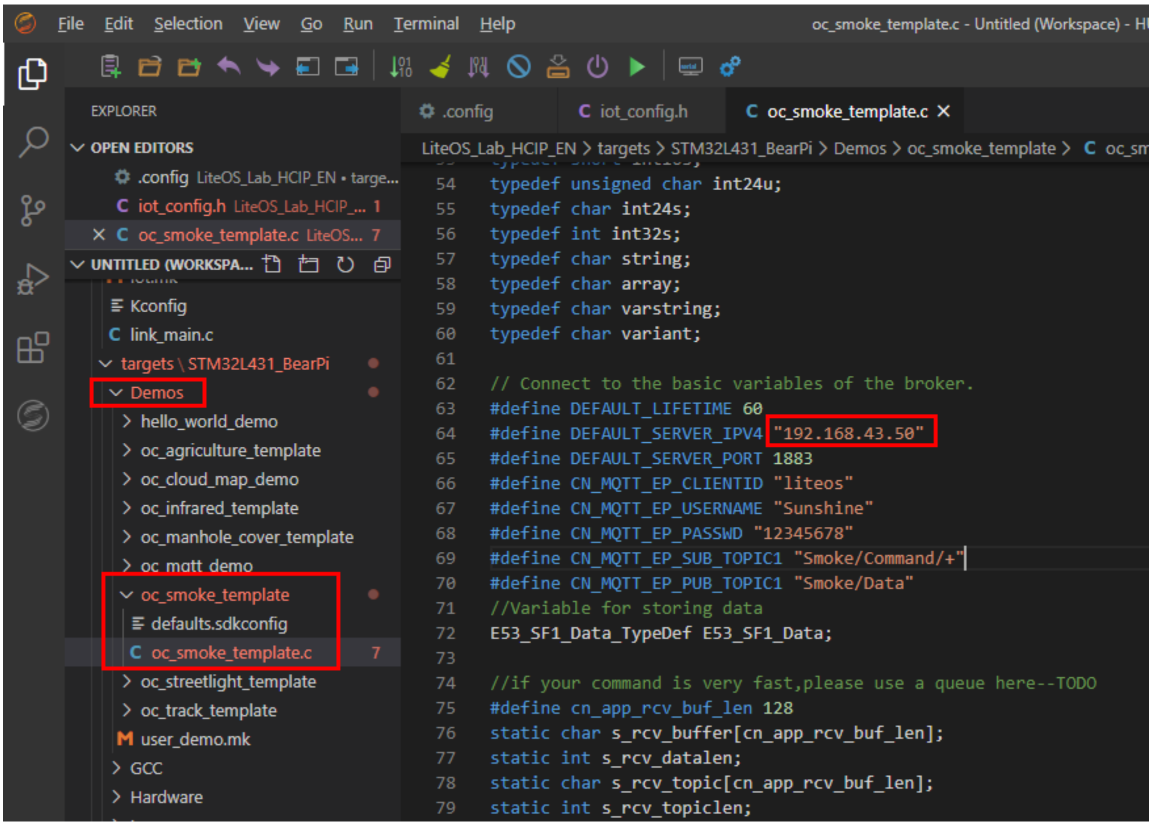
The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.



Step 2 Change the IP address in the code.

Choose **STM32L431_BearPi** > **Demos** > **oc_smoke_template** > **oc_smoke_template.c**.

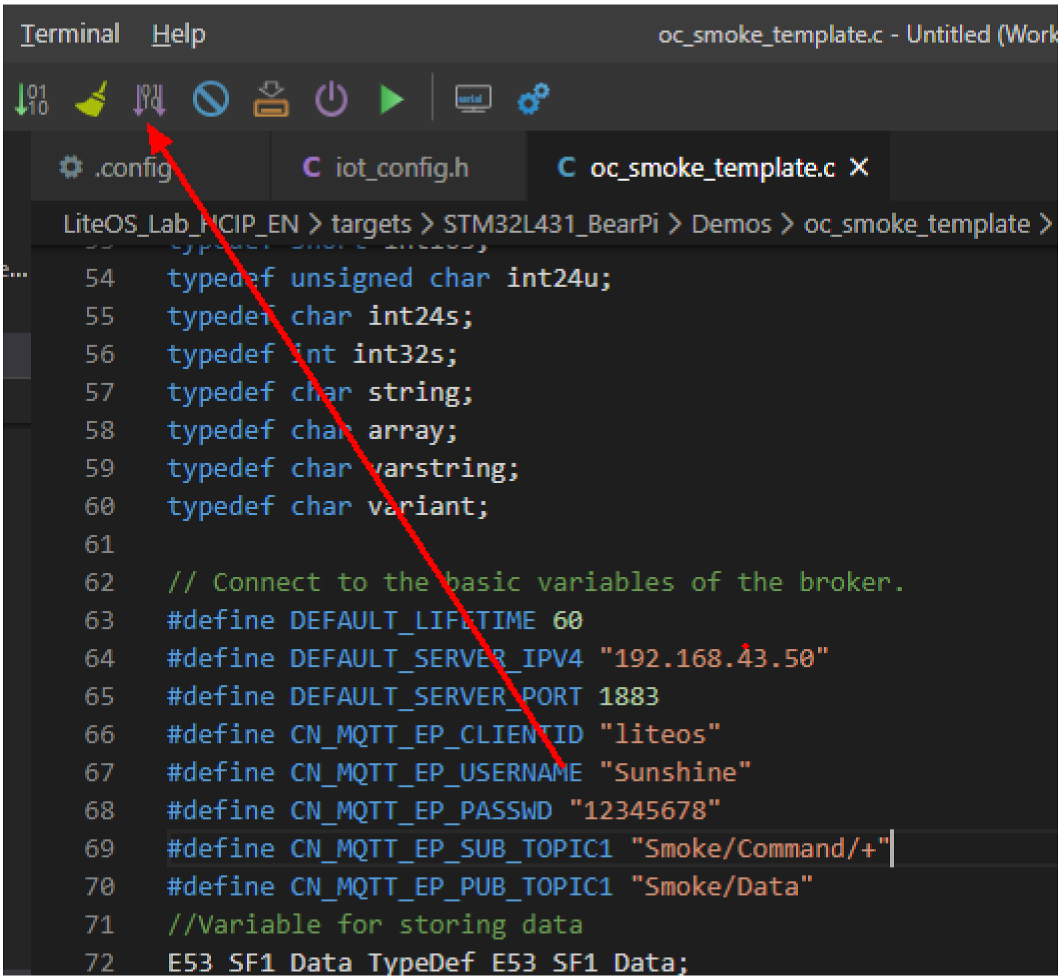
Set **DEFAULT_SERVER_IPV4** to the queried IP address.



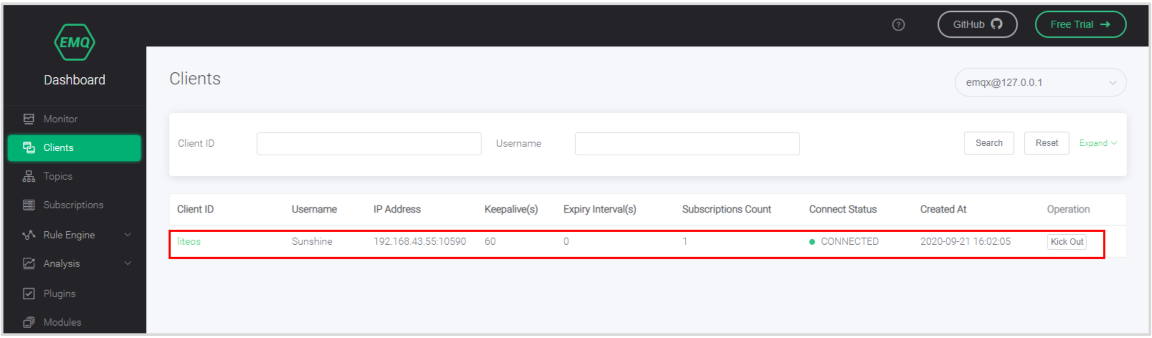
4.2.3 Compiling and Burning the Program

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.



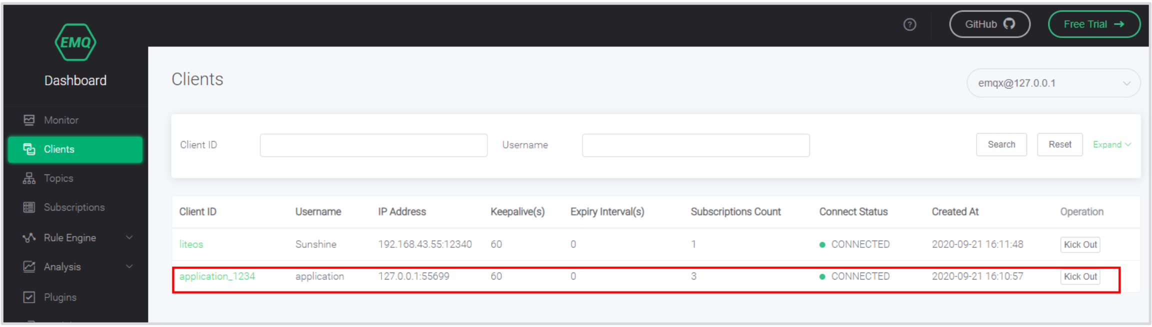
Burn the program and check whether the device is online in EMQ Broker.



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.



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

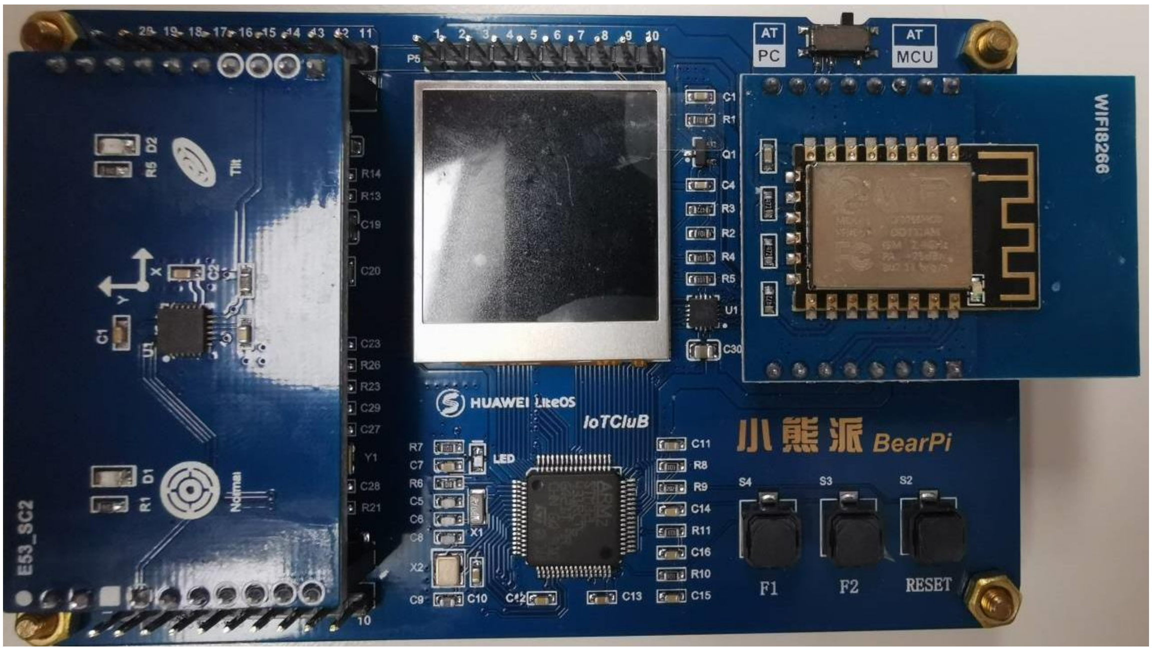
- Master how to configure the Wi-Fi communication mode.
- Master how to develop smart manhole cover cases.

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.

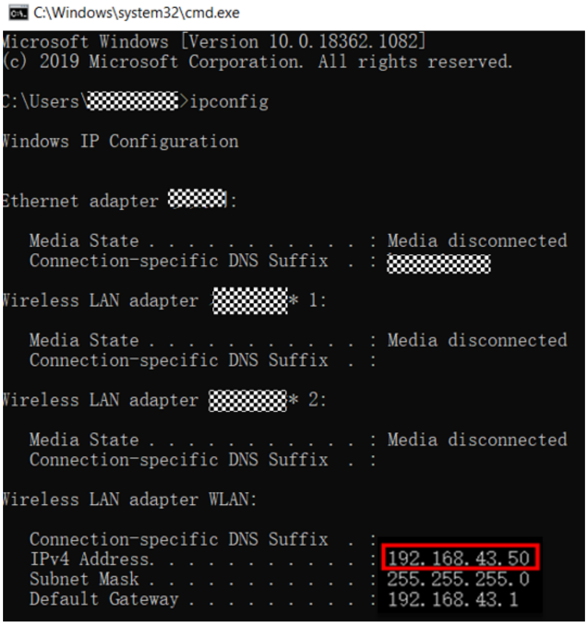
Set **CONFIG_ESP8266_PWD** to the Wi-Fi password.

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.

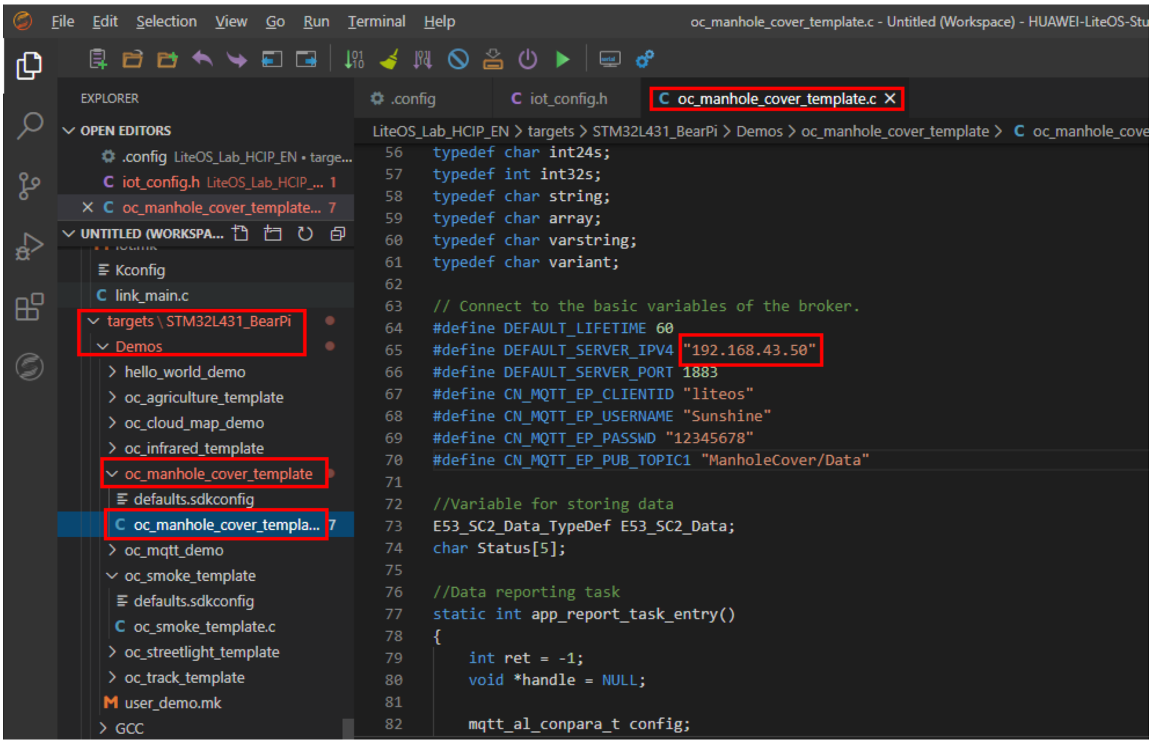
The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.



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.

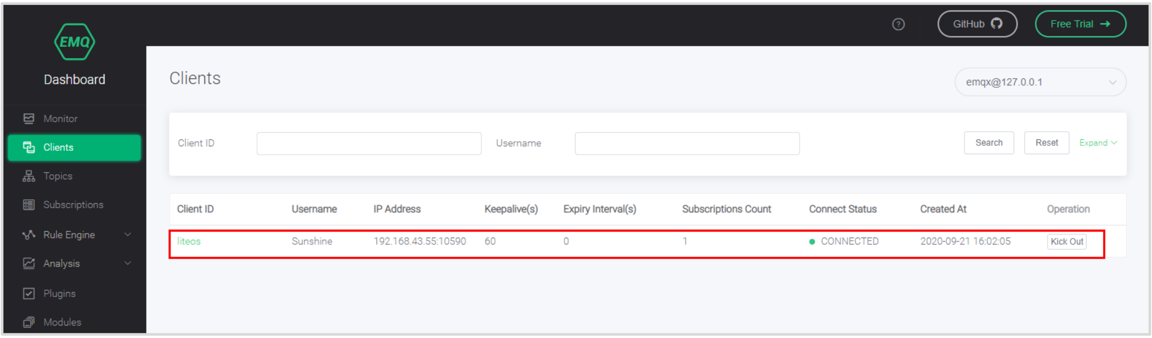


5.2.3 Compiling and Burning the Program

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.

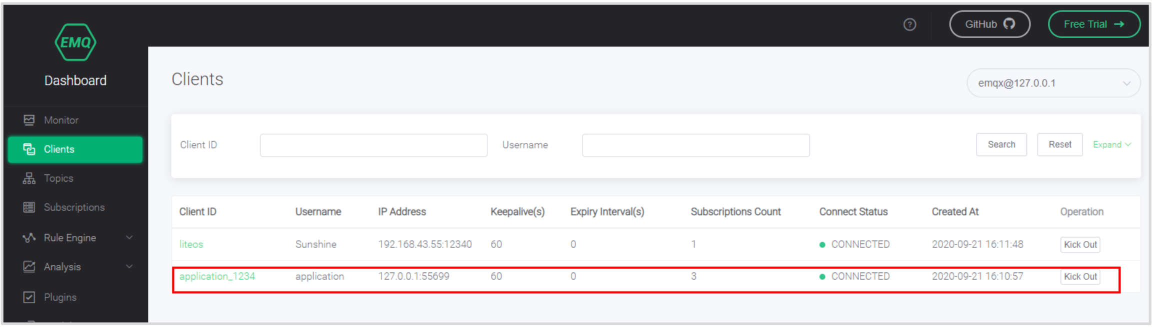


5.2.4 Running Applications

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

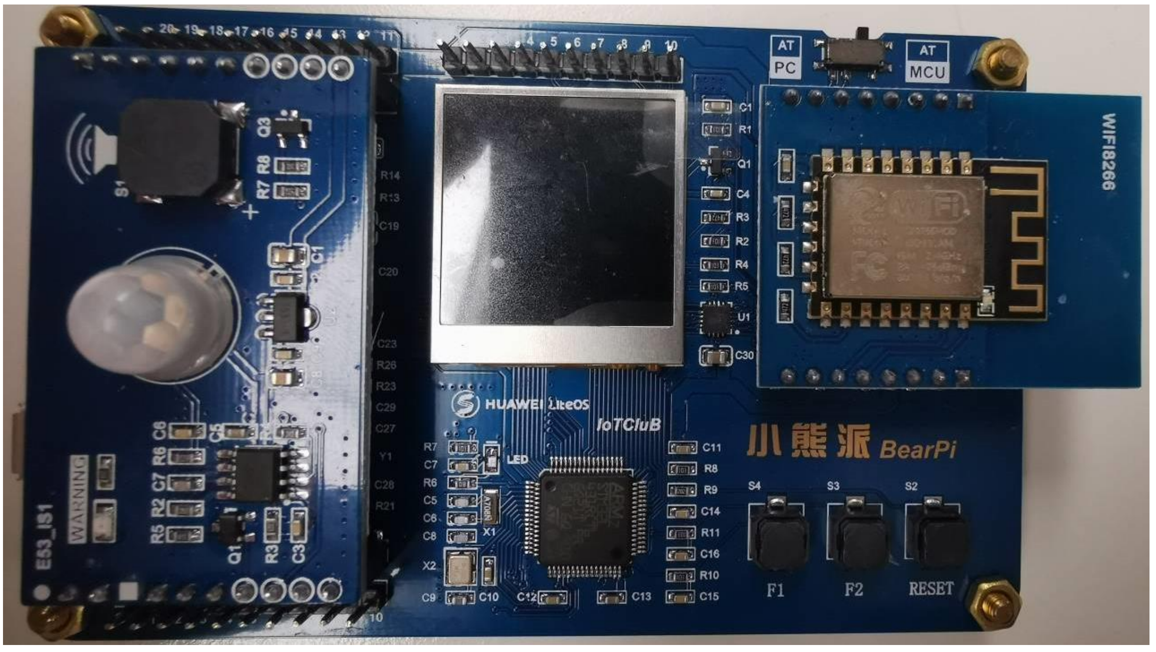
- Master how to configure the Wi-Fi communication mode.
- Master how to develop human body sensor cases.

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.

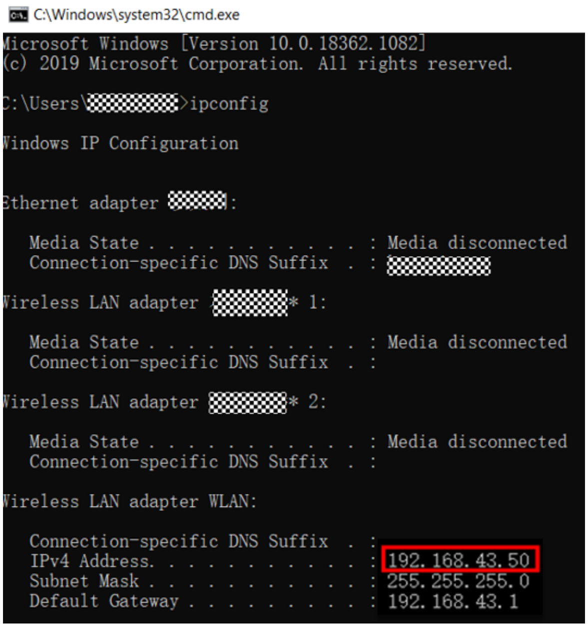
Set **CONFIG_ESP8266_PWD** to the Wi-Fi password.

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.

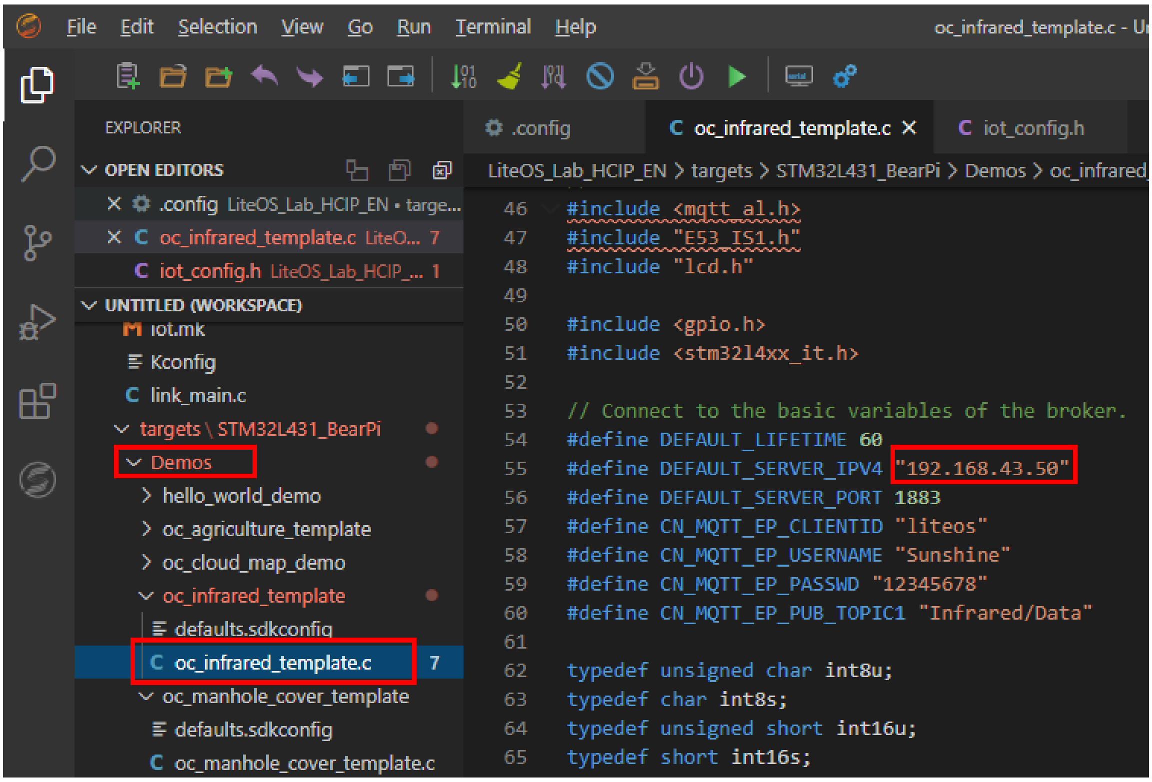
The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.



Step 2 Change the IP address in the code.

Choose **STM32L431_BearPi** > **Demos** > **oc_infrared_template** > **oc_infrared_template.c**.

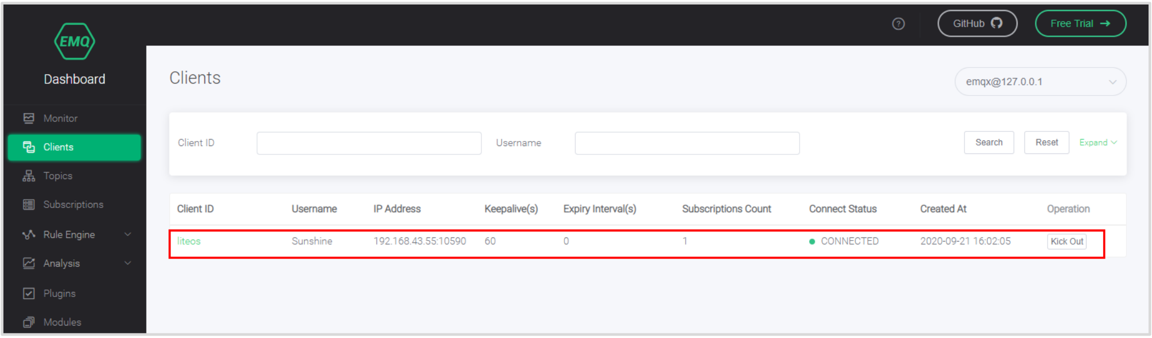
Set **DEFAULT_SERVER_IPV4** to the queried IP address.



6.2.3 Compiling and Burning the Program

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

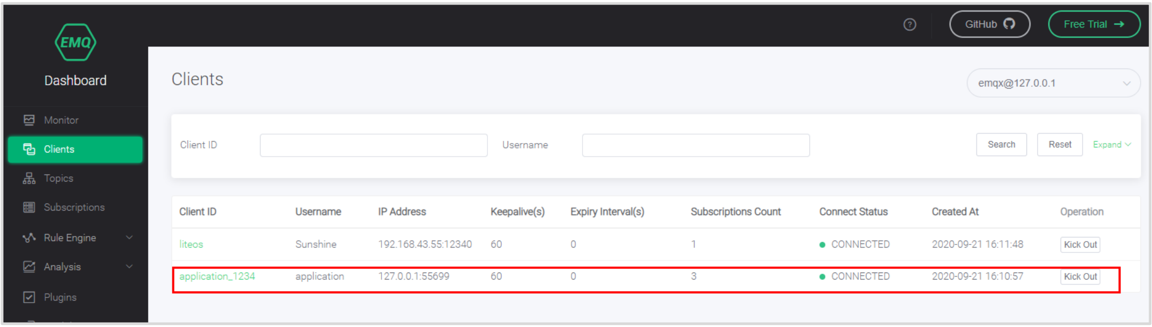
Enable the Wi-Fi hotspot on the mobile phone and click [icon] to recompile the program. Burn the program and check whether the device is online in EMQ Broker.



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.



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

```
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.

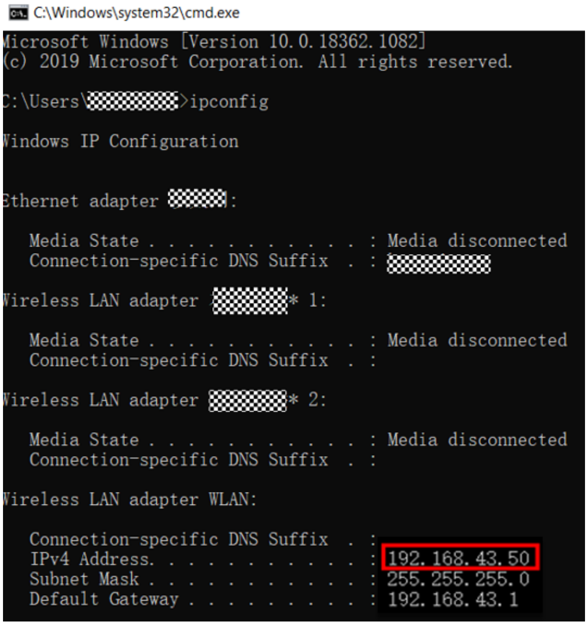
Set **CONFIG_ESP8266_PWD** to the Wi-Fi password.

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.

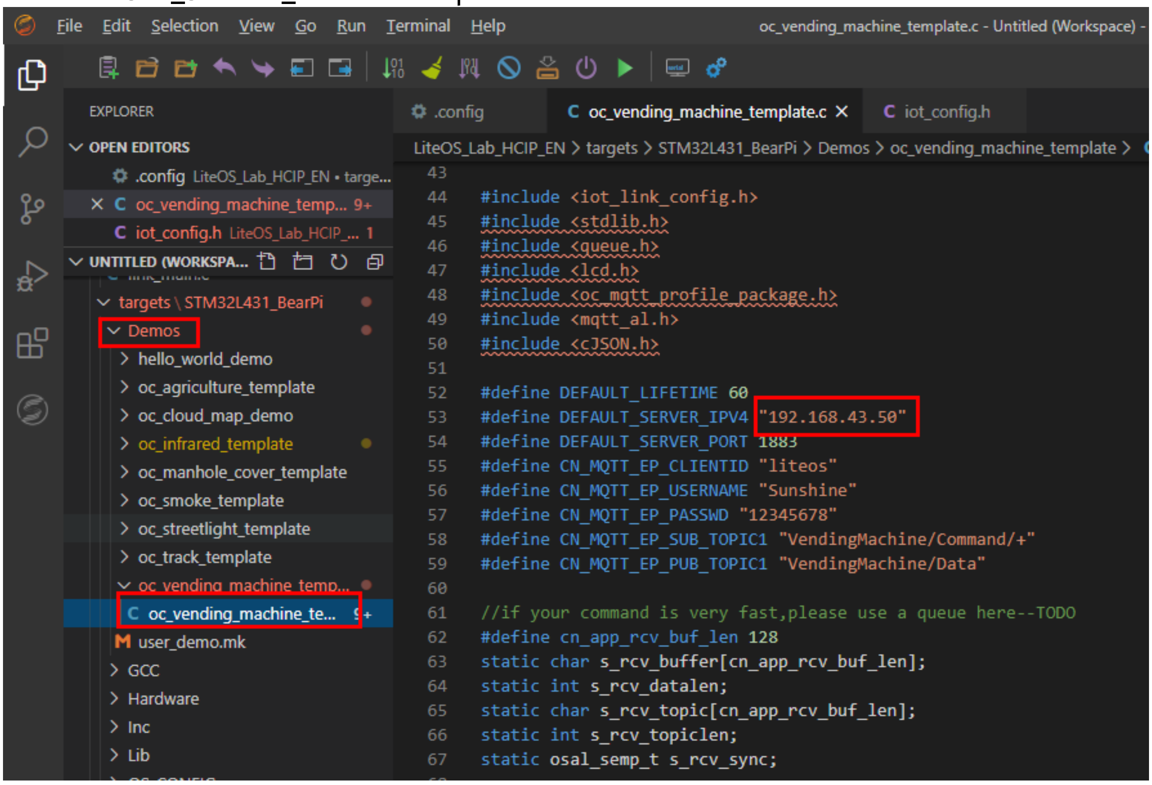
The IP address of the Broker is the IP address of the PC. Ensure that the PC and the development board are in the same LAN.



Step 2 Change the IP address in the code.

Choose **STM32L431_BearPi** > **Demos** > **oc_vending_machine_template** > **oc_vending_machine_template.c**.

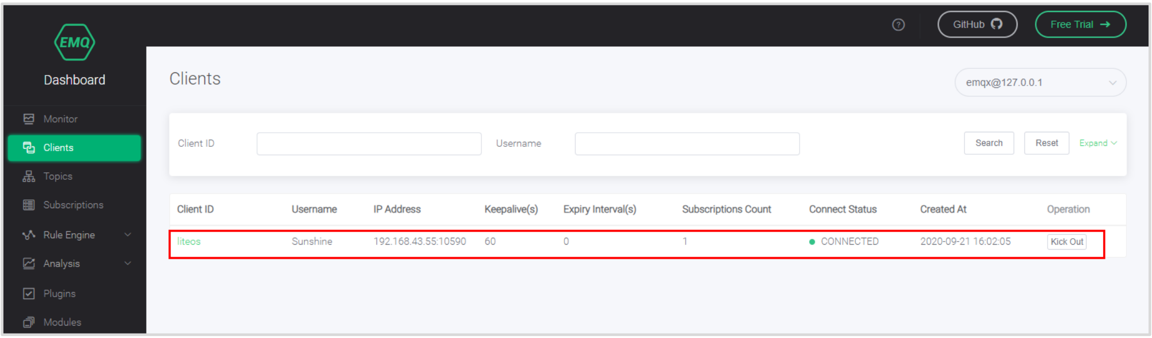
Set **DEFAULT_SERVER_IPV4** to the queried IP address.



7.2.3 Compiling and Burning the Program

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

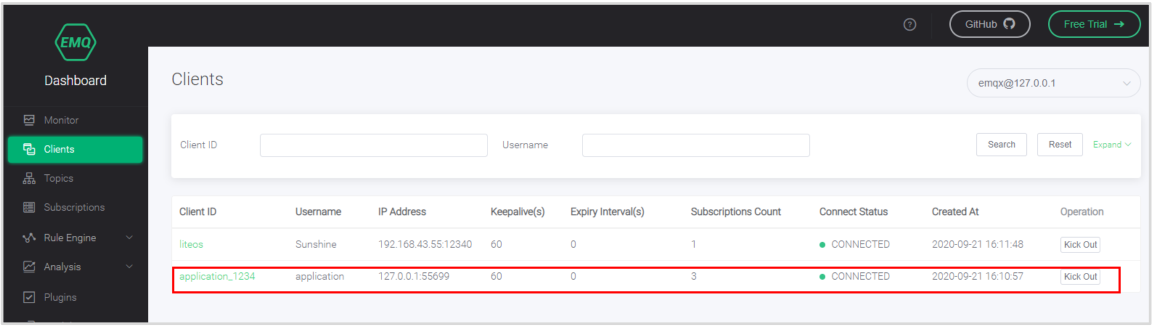
Enable the Wi-Fi hotspot on the mobile phone and click to recompile the program. Burn the program and check whether the device is online in EMQ Broker.



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.



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.