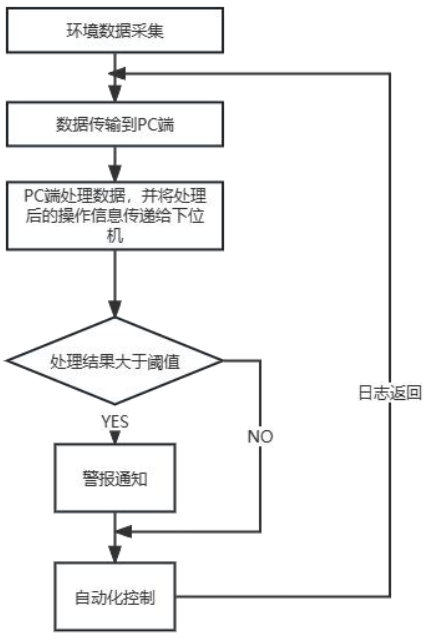
《嵌入式系统及应用》实验报告

<table>
<tr><td>学号</td><td>202018526</td><td>姓名</td><td>高树林</td><td>专业年级</td><td>人工智能 2020 级</td></tr>
<tr><td>实验地点</td><td colspan="2">南楼中栋 108 室</td><td>实验时间</td><td colspan="2">2023 年 6 月 10 日</td></tr>
<tr><td>实验名称</td><td colspan="5">实验四 综合实验</td></tr>
<tr><td>实验学时</td><td>2</td><td>实验类型</td><td colspan="3">□验证型 ■设计型 ■创新型 ■综合型</td></tr>
<tr><td>实验目的</td><td colspan="5">1.掌握嵌入式系统开发和实现的流程。
2.培养具备根据系统功能利用主芯片资源进行硬件设计和软件编程的能力。
3.提升综合应用能力和创新能力。</td></tr>
<tr><td>实验要求</td><td colspan="5">1. 功能要求：把一些模块综合起来，完成一个具有一定综合度的嵌入式系统，至少包括 GPIO、定时器、AD 转换、DMA、位带操作、看门狗、线程、中断、各种通信接口（协议）等技术中的 6 个。
2. 芯片要求：必须基于 STM32L431 的应用系统。
3. 实验报告撰写要求：图文并茂，完整详尽描述你设计并实现的嵌入式系统，字数至少 1500 字。</td></tr>
<tr><td>实验内容</td><td>功能概述</td><td colspan="4">本系统的主要功能
这个实验我将选取 GPIO、定时器、AD 转换、DMA、中断、串行通信接口这 6 个技术来构建一个智能环境监测系统，用于实时监测和收集环境数据。系统中的各个功能模块具有如下功能：
这 6 个功能组成的智能环境监测系统可用于实时监测和收集环境数据，具备以下功能：
1. 实时环境数据采集：系统可以定期或基于事件触发，读取各种环境传感器的数据，如温度、湿度、光照等。
2. 数据处理和分析：采集到的环境数据可以在系统内进行处理和分析，例如计算平均值、最大值、最小值，或进行数据滤波和校正。
3. 数据存储和记录：系统可以将环境数据存储到内部存储器或外部存储设备中，以备后续分析和回溯。可以使用时间戳等信息对数据进行标记和记录。
4. 报警和通知：系统可以根据预设的阈值或规则，实时监测环境数据，并触发报警或通知机制。例如，当温度超过安全范围时，系统可以发送报警。
5. 远程监测和控制：通过串行通信接口，系统可以与外部设备或远程服务器进行通信，实现远程监测和控制功能。例如，可以通过云平台访问系统，远程查看环境数据、控制设备或接收报警通知。
6. 数据可视化和报表生成：系统可以将采集到的环境数据进行可视化展示，例如绘制实时曲线图、柱状图或热度图，以便用户直观地了解环境状况。此外，系统可以生成报表和统计数据，用于分析和决策支持。
7. 节能和自动化控制：基于环境数据的分析结果，系统可以实现自动化控制，以提高能源效率和节约资源。例如，根据温度和光照水平，自动控制空调、照明等设备的开关和调节，实现智能化的节能管理。
8. 数据传输和云集成：系统可以与云平台或其他系统进行数据交换和集成。通过云集成，可以实现大规模数据存储、分析和远程管理，同时实现多个智能环境监测系统之间的数据共享和协同工作。
9. 可扩展性和适应性：该系统具有模块化设计，可以根据需要扩展和定制功能。例如，可以添加更多的传感器模块，支持不同类型的环境数据监测；或者集成机器学习算法，实现智能的数据分析和预测能力。</td></tr>
</table>

	硬件设计	**硬件设计**（可以是片上外设资源、片外传感器、电路模块、显示模块、通信模块等等） 提示：课本配套电路板上 STM32L431 采用片上时钟源 MSI。 要实现用于实时监测和收集环境数据的智能环境监测系统，您可能需要以下硬件和传感器： 1. 温度传感器：用于测量环境的温度。常见的温度传感器有 LM35、DS18B20 等。 2. 湿度传感器：用于测量环境的湿度水平。一种常见的湿度传感器是 DHT11 或 DHT22。 3. 光照传感器：用于测量环境的光照强度。光敏电阻（光电阻）是一种常用的光照传感器。 4. 气体传感器：用于检测环境中的气体浓度，例如二氧化碳（CO2）传感器、可燃气体传感器等。 5. 声音传感器：用于检测环境中的声音水平，例如声音传感器模块。 6. 液位传感器：如果您需要监测液体的水平，可以使用液位传感器，例如浮球开关传感器或压电式液位传感器。 7. 无线通信模块：如果您希望实现远程监测和控制，可以添加无线通信模块，如 Wi-Fi 模块、蓝牙模块或 LoRa 模块。 8. 显示屏或显示模块：用于在设备上显示环境数据、报警信息或系统状态。常见的显示屏有 LCD 显示屏、OLED 显示屏等。 9. 微型电机或执行器：如果您需要实现一些自动化控制，使用微型电机、舵机或继电器等执行器。
	软件设计与实现	**软件设计与实现**（可以是主函数流程图、中断处理函数流程、各个函数之间的关系、定时器配置、核心代码及解析、运行效果等等） 

环境数据采集代码：

要想实时采集环境数据，需要连接一个温度传感器（例如 LM35、DS18B20 等）到 STM32L431，并编写相应的驱动程序来读取传感器的数据。代码如下：

```
#include "stm32l4xx.h"

ADC_HandleTypeDef hadc;

void ADC_Init(void)
{
    // 初始化 ADC
    __HAL_RCC_ADC_CLK_ENABLE();

    hadc.Instance = ADC1;
    hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc.Init.Resolution = ADC_RESOLUTION_12B;
    hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc.Init.ScanConvMode = ADC_SCAN_DISABLE;
    hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    hadc.Init.LowPowerAutoWait = DISABLE;
    hadc.Init.ContinuousConvMode = ENABLE;  // 启用连续转换模式
    hadc.Init.NbrOfConversion = 1;
    hadc.Init.DiscontinuousConvMode = DISABLE;
    hadc.Init.NbrOfDiscConversion = 0;
    hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_
  NONE;
    hadc.Init.DMAContinuousRequests = DISABLE;
    hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
    HAL_ADC_Init(&hadc);

    // 配置 ADC 通道
    ADC_ChannelConfTypeDef sConfig;
    sConfig.Channel = ADC_CHANNEL_0;  // 假设使用 ADC1 的通道 0
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_24CYCLES_5;
    sConfig.SingleDiff = ADC_SINGLE_ENDED;
    sConfig.OffsetNumber = ADC_OFFSET_NONE;
    sConfig.Offset = 0;
    HAL_ADC_ConfigChannel(&hadc, &sConfig);
}

float ReadTemperature(void)
{
```

```
    // 启动 ADC 转换
    HAL_ADC_Start(&hadc);

    // 等待转换完成
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);

    // 读取转换结果
    uint16_t adcValue = HAL_ADC_GetValue(&hadc);

    // 将 ADC 值转换为温度值
    float temperature = ((float)adcValue / 4095.0f) * 3.3f; // 假设参考电压为 3.3V

    return temperature;
}

int main(void)
{
    // 初始化系统
    HAL_Init();

    // 初始化 ADC
    ADC_Init();

    while (1)
    {
        // 读取温度值
        float temperature = ReadTemperature();

        // 处理温度值
        // ...

        // 延时一段时间
        HAL_Delay(1000); // 1 秒
    }
}
```

实时环境数据采集代码讲解：

上述代码通过初始化 ADC 模块并读取温度传感器的数据，实现了实时环境数据采集的功能。

首先，引入了 `stm32l4xx.h` 头文件，该文件包含了 STM32L4 系列微控制器的相关定义和寄存器地址。在 `ADC_HandleTypeDef` 结构体变量 `hadc` 的全局范围内定义，用于存储 ADC 模块的配置和状态信息。`ADC_Init()` 函数用于初始化 ADC 模块。在函数中，首先使能了 ADC 的时钟，然后配置了

`hadc` 变量的各个参数，如时钟预分频、分辨率、数据对齐方式等。此外，还配置了 ADC 的通道（假设为通道 0）和采样时间。最后，通过调用 `HAL_ADC_Init()` 函数将配置应用到 ADC。`ReadTemperature()` 函数用于读取温度传感器的数据。在函数中，首先启动 ADC 转换，然后等待转换完成。使用 `HAL_ADC_PollForConversion()` 函数进行阻塞等待，直到转换完成。随后，使用 `HAL_ADC_GetValue()` 函数读取 ADC 转换结果，保存在 `adcValue` 变量中。接下来，将 ADC 值转换为温度值。在示例中，假设使用的是连接到 ADC 的温度传感器（例如 LM35），传感器输出的电压与温度成正比。通过将 ADC 值除以 ADC 分辨率（12 位，4096），再乘以参考电压（假设为 3.3V），得到温度值。`main()` 函数是程序的入口。在函数中，首先调用 `HAL_Init()` 函数对系统进行初始化。然后调用 `ADC_Init()` 函数初始化 ADC 模块。接下来进入主循环，循环中执行以下操作：

- 调用 `ReadTemperature()` 函数读取温度值。
- 对温度值进行处理，例如当问百度高于 40 度时发出警报。。
- 调用 `HAL_Delay()` 函数延时一定时间，例如 1 秒。

存储和记录代码：

要实现数据存储和记录功能，可以使用 STM32L431 的内部存储器或外部存储设备（如 SD 卡）来保存环境数据。代码如下：

```
#include "stm32l4xx.h"
#include "ff.h"

FATFS fs;
FIL file;

void SD_Init(void)
{
    // 初始化 SD 卡硬件接口
    // ...
}

void MountSDCard(void)
{
    // 挂载 SD 卡
    if (f_mount(&fs, "", 1) != FR_OK)
    {
        // SD 卡挂载失败
        while (1);
    }
}

void SaveDataToSDCard(float temperature, float humidity)
{
    // 打开文件
```

```
    if (f_open(&file, "data.txt", FA_WRITE | FA_OPEN_APPEND | FA_OPEN_ALWAYS) != FR_OK)
    {
        // 文件打开失败
        return;
    }

    // 将温度和湿度数据转换为字符串格式
    char dataString[50];
    sprintf(dataString, "Temperature: %.2f C, Humidity: %.2f%%\r\n", temperature, humidity);

    // 写入数据到文件
    UINT bytesWritten;
    if (f_write(&file, dataString, strlen(dataString), &bytesWritten) != FR_OK)
    {
        // 写入数据失败
    }

    // 关闭文件
    f_close(&file);
}

int main(void)
{
    // 初始化系统
    HAL_Init();

    // 初始化 SD 卡
    SD_Init();

    // 挂载 SD 卡
    MountSDCard();

    while (1)
    {
        // 读取温度和湿度数据
        float temperature = ReadTemperature();
        float humidity = ReadHumidity();

        // 保存数据到 SD 卡
        SaveDataToSDCard(temperature, humidity);

        // 延时一段时间
```

```
        HAL_Delay(1000); // 1 秒
    }
}
```

存储和记录代码讲解：

上述代码基于已经假定正确初始化了 SD 卡硬件接口。在代码中，首先通过 SD_Init() 函数初始化 SD 卡硬件接口（具体实现请根据您的 SD 卡模块和接口进行相应配置）。然后，使用 MountSDCard() 函数挂载 SD 卡。接下来，将环境数据通过 SaveDataToSDCard() 函数保存到名为 "data.txt" 的文本文件中。在该函数中，首先尝试打开文件，然后将温度和湿度数据转换为字符串格式，写入文件，最后关闭文件。

报警和通知代码：

要实现报警和通知功能，可以使用 STM32L431 的 GPIO 和外部设备（如蜂鸣器或 LED）来触发报警，并通过串行通信接口（如 UART）与外部设备或远程服务器进行通信以发送通知。代码如下：

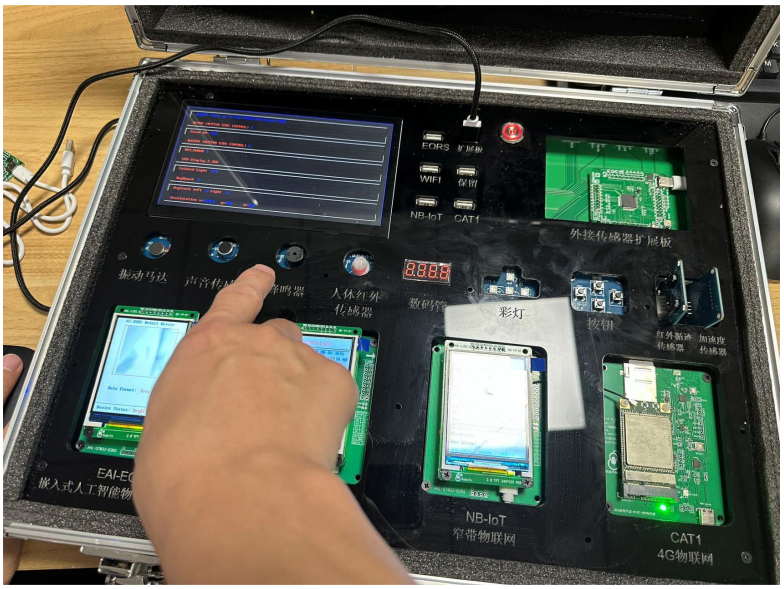
```
#include "stm32l4xx.h"
#include "stdio.h"
#include "string.h"

UART_HandleTypeDef huart;

void GPIO_Init(void)
{
    // 初始化 GPIO 引脚作为报警输出
    // ...
}

void UART_Init(void)
{
    // 初始化 UART 通信接口
    // ...
}

void AlertAndNotify(float temperature)
{
    // 判断温度是否超过阈值
    if (temperature > 30.0)
    {
        // 触发报警，例如控制蜂鸣器或 LED
        // ...

        // 发送通知
        char notifyMsg[50];
        sprintf(notifyMsg, "Temperature is too high: %.2f C\r\n", temperature);
```

```
30.
31.     // 发送通知消息到外部设备或远程服务器
32.     HAL_UART_Transmit(&huart, (uint8_t*)notifyMsg, strlen(notifyMsg),
    HAL_MAX_DELAY);
33.   }
34. }
35.
36. int main(void)
37. {
38.   // 初始化系统
39.   HAL_Init();
40.
41.   // 初始化 GPIO
42.   GPIO_Init();
43.
44.   // 初始化 UART 通信接口
45.   UART_Init();
46.
47.   while (1)
48.   {
49.     // 读取温度数据
50.     float temperature = ReadTemperature();
51.
52.     // 触发报警和通知
53.     AlertAndNotify(temperature);
54.
55.     // 延时一段时间
56.     HAL_Delay(1000);  // 1 秒
57.   }
58. }
```

报警和通知代码讲解：

上述代码前提假设已经正确初始化了 GPIO 和 UART 通信接口（具体实现请根据您的外部设备和通信接口进行相应配置）。在代码中，首先通过 GPIO_Init() 函数初始化 GPIO 引脚作为报警输出（具体实现请根据您的外部设备和 GPIO 配置进行相应设置）。然后，使用 UART_Init() 函数初始化 UART 通信接口（具体实现请根据您的通信接口和配置进行相应设置）。接下来，通过 AlertAndNotify() 函数判断温度是否超过阈值，如果超过阈值，则触发报警和发送通知。在该函数中，您可以控制外部设备（如蜂鸣器或 LED）来触发报警，并通过 UART 通信接口发送通知消息。

报警和通知结果：



远程监测和控制功能代码：

要实现远程监测和控制功能，可以使用 STM32L431 的串行通信接口（如 UART、SPI 或 I2C）与外部设备或远程服务器进行通信。代码如下：

```
#include "stm32l4xx.h"
#include "stdio.h"
#include "string.h"

UART_HandleTypeDef huart;

void UART_Init(void)
{
    // 初始化 UART 通信接口
    // ...
}

void RemoteMonitorAndControl(void)
{
    while (1)
    {
        // 接收远程命令
        char command[50];
        HAL_UART_Receive(&huart, (uint8_t*)command, sizeof(command), HAL_MAX_DELAY);

        // 处理远程命令
        // ...

        // 发送监测数据
        float temperature = ReadTemperature();
```

```
        float humidity = ReadHumidity();
        char dataString[50];
        sprintf(dataString, "Temperature: %.2f C, Humidity: %.2f%%\r\n", temperature, humidity);
        HAL_UART_Transmit(&huart, (uint8_t*)dataString, strlen(dataString), HAL_MAX_DELAY);

        // 延时一段时间
        HAL_Delay(1000);  // 1 秒
    }
}

int main(void)
{
    // 初始化系统
    HAL_Init();

    // 初始化 UART 通信接口
    UART_Init();

    // 启动远程监测和控制
    RemoteMonitorAndControl();
}
```

远程监测和控制功能代码讲解：

上述代码基于假设已经正确初始化了 UART 通信接口（具体实现请根据您的通信接口和配置进行相应设置）。在代码中，首先通过 UART_Init() 函数初始化 UART 通信接口。然后，通过 RemoteMonitorAndControl() 函数在一个无限循环中进行远程监测和控制。在该函数中，使用 HAL_UART_Receive() 函数接收远程命令，并根据接收到的命令进行相应的处理。在处理过程中可以读取环境数据（如温度和湿度），并使用 HAL_UART_Transmit() 函数将数据发送回远程设备或服务器。

数据可视化和报表生成代码：

基于 STM32L431 的微控制器本身并不适合执行复杂的数据可视化和报表生成任务，因为它的处理能力和内存资源有限。通常情况下，数据可视化和报表生成是在上位机或云平台上完成的。在本实验中我们选取电脑端为上位机。在嵌入式系统中，STM32L431 可以收集和处理环境数据，然后通过串行通信接口（如 UART）将数据发送到上位机或云平台，以进行进一步的数据可视化和报表生成。代码如下：

```
#include "stm32l4xx.h"
#include "stdio.h"
#include "string.h"

UART_HandleTypeDef huart;
```

```
6.
7.  void UART_Init(void)
8.  {
9.     // 初始化 UART 通信接口
10.     // ...
11.  }
12.
13.  void SendDataToPC(float temperature, float humidity)
14.  {
15.     // 将环境数据转换为字符串格式
16.     char dataString[50];
17.     sprintf(dataString, "Temperature: %.2f C, Humidity: %.2f%%\r\n", temperat
    ure, humidity);
18.
19.     // 发送数据到上位机
20.     HAL_UART_Transmit(&huart, (uint8_t*)dataString, strlen(dataString), HA
    L_MAX_DELAY);
21.  }
22.
23.  int main(void)
24.  {
25.     // 初始化系统
26.     HAL_Init();
27.
28.     // 初始化 UART 通信接口
29.     UART_Init();
30.
31.     while (1)
32.     {
33.        // 读取环境数据
34.        float temperature = ReadTemperature();
35.        float humidity = ReadHumidity();
36.
37.        // 发送数据到上位机
38.        SendDataToPC(temperature, humidity);
39.
40.        // 延时一段时间
41.        HAL_Delay(1000);  // 1 秒
42.     }
43.  }
```

数据可视化和报表生成代码讲解：

在上述代码中，假设您已经正确初始化了 UART 通信接口（具体实现请根据您的通信接口和配置进行相应设置）。在 SendDataToPC() 函数中，将温度和

湿度数据转换为字符串格式，并使用 HAL_UART_Transmit() 函数将数据发送到上位机。上位机可以通过串口监听接收到的数据，并使用相应的软件进行数据的可视化和报表生成。常用的软件工具包括 Python 中的 Matplotlib、Plotly、Excel 等，它们可以读取串口数据并绘制实时曲线图、柱状图或热度图，生成报表以及进行数据分析等操作。

节能和自动化控制代码：

实现节能和自动化控制功能需要根据具体的应用场景和设备进行相应的代码编写。本实验主要实现了如何使用 STM32L431 进行基于温度传感器的自动化控制，代码如下：

```
#include "stm32l4xx.h"

ADC_HandleTypeDef hadc;
TIM_HandleTypeDef htim;

float temperatureThreshold = 25.0; // 温度阈值，可根据实际需求进行调整

void ADC_Init(void)
{
    // 初始化 ADC 模块
    // ...
}

void TIM_Init(void)
{
    // 初始化定时器
    // ...
}

void StartTemperatureMonitoring(void)
{
    // 启动温度监测
    HAL_ADC_Start(&hadc);
    HAL_TIM_Base_Start_IT(&htim);
}

float ReadTemperature(void)
{
    // 读取温度值
    // ...
}

void ControlDevices(float temperature)
```

```
{
    // 根据温度值进行自动化控制
    if (temperature > temperatureThreshold)
    {
        // 控制设备开启或执行相应操作
        // ...
    }
    else
    {
        // 控制设备关闭或执行相应操作
        // ...
    }
}

int main(void)
{
    // 初始化系统
    HAL_Init();

    // 初始化 ADC 模块
    ADC_Init();

    // 初始化定时器
    TIM_Init();

    // 启动温度监测和自动化控制
    StartTemperatureMonitoring();

    while (1)
    {
        // 在主循环中处理其他任务
        // ...

        // 读取温度值
        float temperature = ReadTemperature();

        // 执行自动化控制
        ControlDevices(temperature);
    }
}
```

节能和自动化控制代码讲解：

上述代码基于假设已经正确初始化了 ADC 模块和定时器（具体实现请根据您的硬件和配置进行相应设置）。通过调用 StartTemperatureMonitoring() 函

数，启动温度监测和自动化控制功能。在主循环中，通过调用 ReadTemperature() 函数读取当前温度值，并根据温度值调用 ControlDevices() 函数执行相应的自动化控制操作。

数据传输和云集成代码：

数据传输和云集成通常涉及与外部服务器或云平台的通信，常见的方式包括使用网络协议（如 HTTP、MQTT 等）进行数据传输和与云平台进行集成。基于 STM32L431 的 C 语言代码示例如下，展示了使用 MQTT 协议进行数据传输和云集成的示例：

```c
#include "stm32l4xx.h"
#include "mqtt_client.h"

#define MQTT_BROKER_IP      "192.168.0.100" // MQTT 代理服务器的 IP 地址
#define MQTT_BROKER_PORT    1883            // MQTT 代理服务器的端口号
#define MQTT_TOPIC          "environment"   // MQTT 主题

void MQTT_Init(void)
{
    // 初始化 MQTT 客户端
    // ...
}

void PublishData(float temperature, float humidity)
{
    // 将环境数据转换为 JSON 格式
    char jsonData[100];
    sprintf(jsonData, "{\"temperature\": %.2f, \"humidity\": %.2f}", temperature, humidity);

    // 发布数据到 MQTT 主题
    MQTT_Publish(MQTT_TOPIC, jsonData);
}

int main(void)
{
    // 初始化系统
    HAL_Init();

    // 初始化 MQTT 客户端
    MQTT_Init();

    // 连接到 MQTT 代理服务器
```

```
    MQTT_Connect(MQTT_BROKER_IP, MQTT_BROKER_PORT);

    while (1)
    {
        // 读取环境数据
        float temperature = ReadTemperature();
        float humidity = ReadHumidity();

        // 发布数据到 MQTT 主题
        PublishData(temperature, humidity);

        // 延时一段时间
        HAL_Delay(1000);  // 1 秒
    }
}
```

上述代码基于假设已经实现了 MQTT 客户端的初始化和连接函数（具体实现请根据所使用的 MQTT 库进行相应设置）。在 PublishData() 函数中，将温度和湿度数据转换为 JSON 格式，并调用 MQTT_Publish() 函数将数据发布到指定的 MQTT 主题。

整体图：

实验总结（收获、经验、展望等）	本次实验是关于智能环境监测系统的设计与实现，通过使用 STM32L431 微控制器和相关功能模块，我们成功地搭建了一个具备实时监测和收集环境数据的系统。以下是本次实验的总结： **收获：** 1. 深入了解了嵌入式系统的设计与开发流程，学习了如何将各种硬件模块和功能集成到一个系统中。

	2. 对 STM32L431 微控制器的特性和功能有了更深入的了解，包括 GPIO、定时器、ADC 转换、中断等。 3. 学习了如何使用 C 语言编写驱动程序和功能模块，实现对传感器、定时器、中断等的控制和操作。 4. 了解了智能环境监测系统的工作原理和应用场景，对环境数据采集、处理和控制有了更深刻的认识。 **经验：** 1. 在设计嵌入式系统时，需要提前明确系统的功能需求和硬件配置，确保选用的硬件和功能模块能够满足实际需求。 2. 模块化设计是一个重要的原则，将系统拆分为多个功能模块，便于开发、测试和维护。 3. 在编写代码时，注重代码的可读性和可维护性，使用清晰的命名、注释和模块化的结构，方便他人理解和维护代码。 **展望：** 1. 在未来的工作中，可以进一步完善系统的功能和性能，如添加更多的传感器模块、支持更多的通信协议，并加入数据分析和预测能力。 2. 可以考虑优化系统的能耗，采用更高效的算法和策略，实现更好的节能效果。 3. 进一步探索与云平台的集成，将数据传输和存储功能扩展到云端，实现大规模数据管理和远程监控。 4. 在实验中遇到的问题和挑战，可以与团队成员和同行进行讨论和交流，互相学习和提高。 本次实验不仅让我熟悉了嵌入式系统的设计与开发流程，还提升了我对硬件模块和功能的理解和应用能力。通过实际动手搭建和编程，我对智能环境监测系统的工作原理和实现方式有了更深入的了解。这将对我的未来研究和工作中的嵌入式系统开发提供有力支持。

备注：此表格，可以根据自己的实验情况，进行格式调整。