LiteOS 移植

NuMicro® 32 位元系列微控制器範例程式碼介紹

文件資訊

應用簡述	本範例基於 BSP 中 GPIO_OutputInput 參考程式碼，介紹如何移植 LiteOS 並創建 2 個執行緒。
BSP 版本	M031_Series_BSP_CMSIS_V3.07.000
開發平台	NuMaker-M031SE3AE V1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本範例基於 BSP 中 GPIO_OutputInput 參考程式碼，介紹如何在 Keil MDK 編譯環境下移植 LiteOS 並創建 2 個執行緒。執行緒 1 通過串口，每 1000 ms 輸出“This is task1,count is ***”；執行緒 2 通過串口，每 2000 ms 輸出“This is task2,count is ***”。

1.1 原理

LiteOS 是華為開發的一款輕量級作業系統，主要面向物聯網（IoT）設備。特點：

- 體積小巧，記憶體佔用低（最小內核可小於 10 KB）；
- 即時性強，適合對回應速度有要求的場景；
- 支援多種主流晶片架構（如 ARM Cortex-M、RISC-V 等）；
- 支持多種網路通訊協定（如 Wi-Fi、NB-IoT、LoRa 等）；
- 代碼開源（部分版本），易於二次開發和定制。

LiteOS 與 FreeRTOS、RT-Thread、Zephyr 等其他物聯網 RTOS 類似，但在國產化、生態相容性和與華為 IoT 生態的集成方面有獨特優勢，廣泛應用于智慧家居、個人穿戴、車聯網、城市公共服務、製造業等領域，大幅降低設備佈置及維護成本，有效降低開發門檻、縮短開發週期。

本範例使用 M031SE3AE 晶片，內置 128 KB Flash，16 KB RAM，介紹如何在 Keil MDK 中移植 LiteOS 並創建管理 2 個執行緒。

1.2 執行結果

本範例執行編譯和燒錄後，串口列印資訊如圖 1-1 所示。

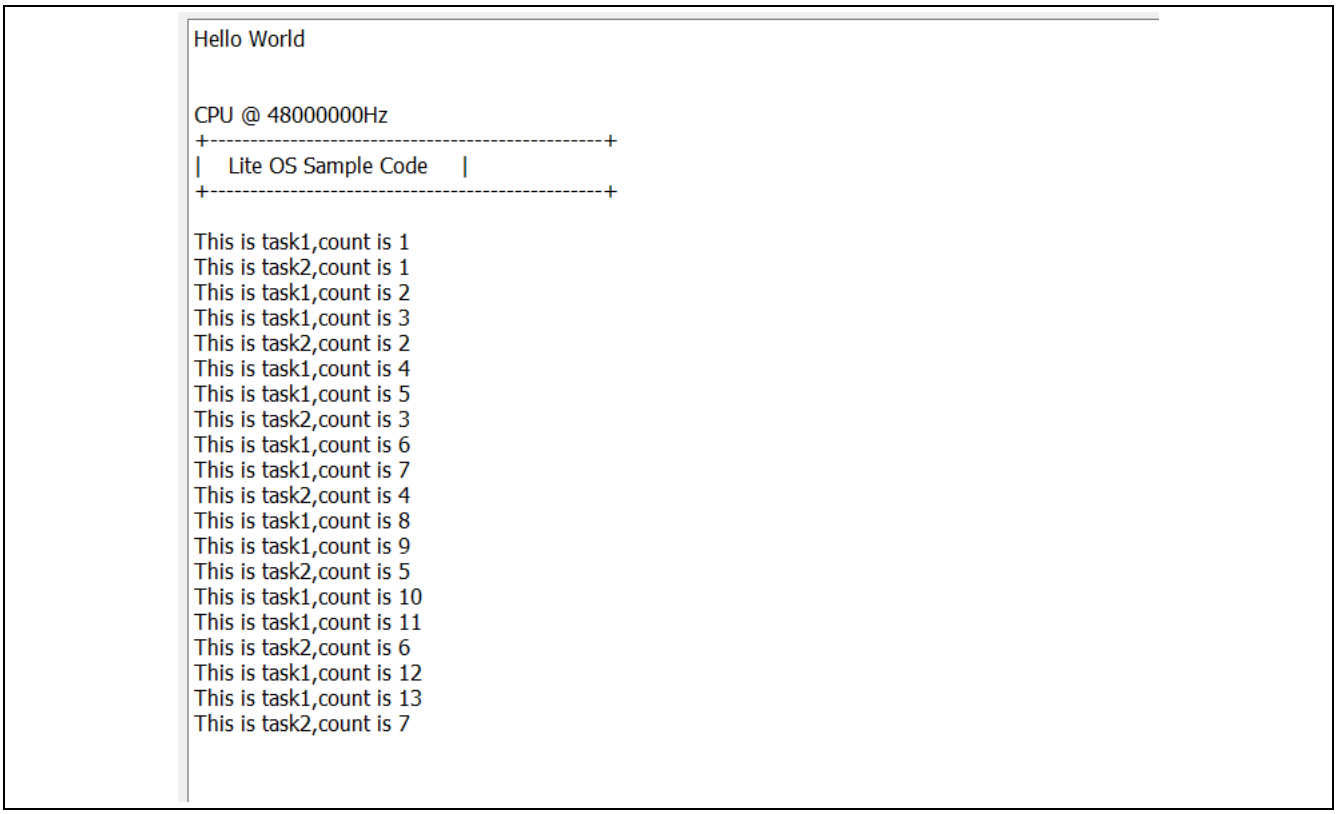

圖 1-1 串口列印資訊

2. 程式碼介紹

2.1 下載 LiteOS 原始碼

下載 LiteOS 原始碼：

- 使用 develop 分支，連結為：https://github.com/LiteOS/LiteOS/tree/develop；
- 使用 LiteOSV200R001C50B038 標籤，連結為：https://github.com/LiteOS/LiteOS/tree/LiteOSV200R001C50B038 。

以上兩個連結均可使用，本範例中使用了 develop 分支，原始碼解壓後如圖 2-1 所示。

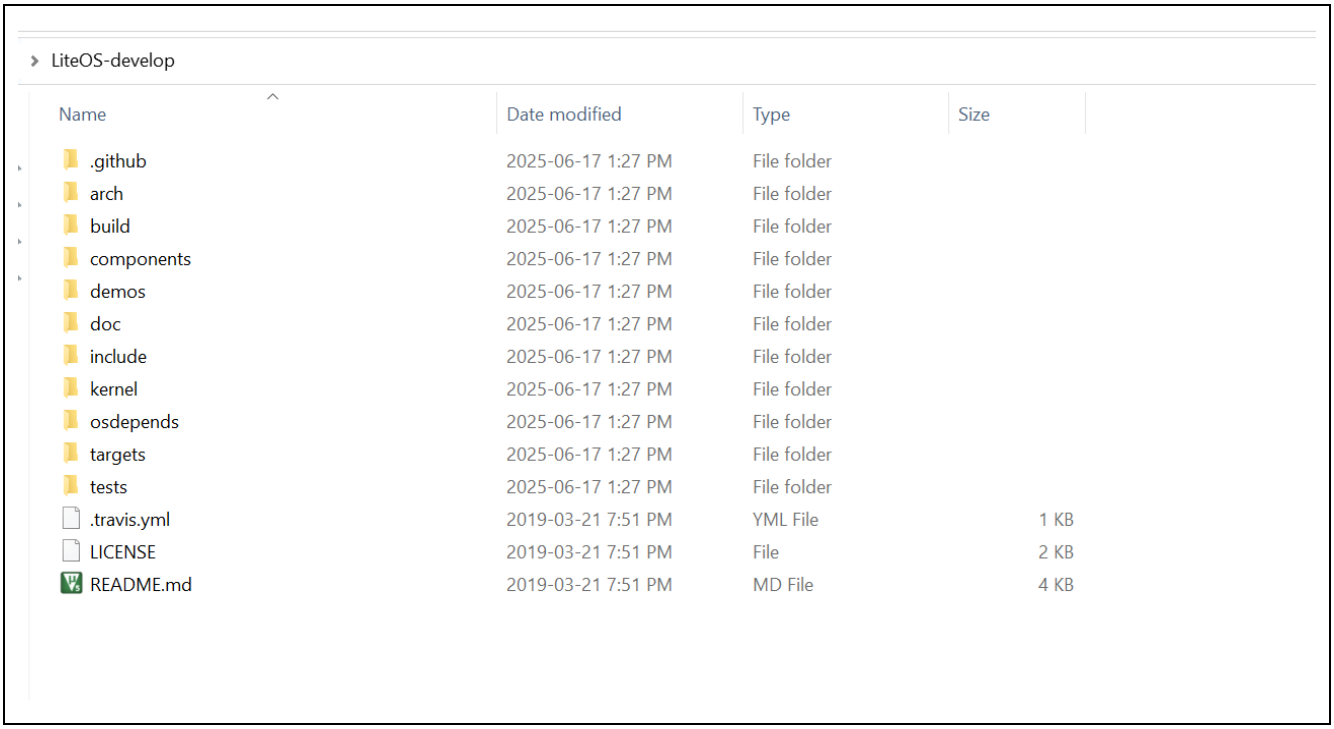

圖 2-1 LiteOS 原始碼檔案結構

2.2 拷貝 LiteOS 原始檔案到 BSP 路徑下

先在 BSP\ThirdParty 目錄下，建一個子目錄 *LiteOS-develop*，用來放置 LiteOS 的原始碼。在新建 *LiteOS-develop* 目錄下，再建四個子目錄，分別叫 *kernel*、*arch*、*cmsis*、*OS_CONFIG*，如圖 2-2 所示。

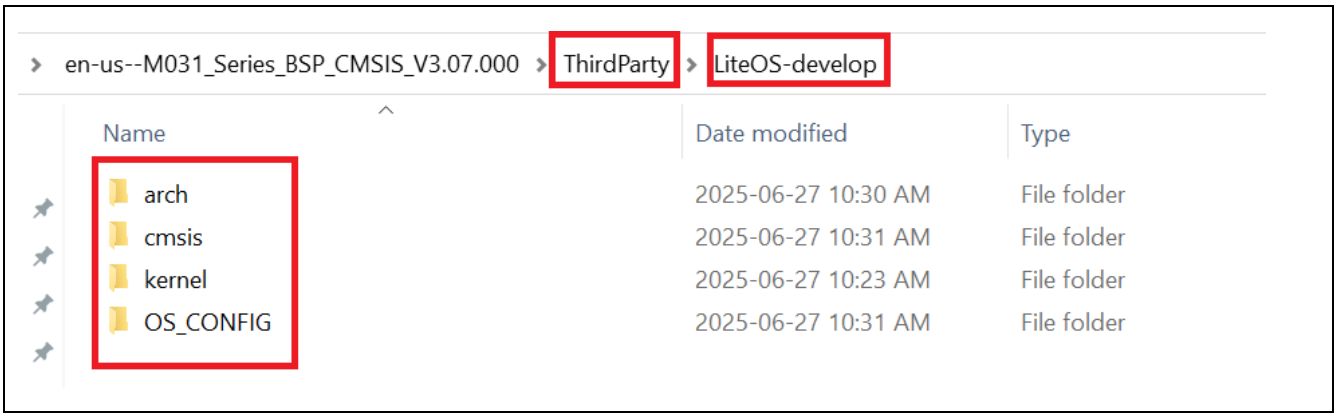

圖 2-2 LiteOS 原始碼存放路徑

把 LiteOS 原始碼.*kernel* 目錄下的內容全部複製到專案裡的 *LiteOS-develop\kernel* 目錄下，然後再把.*LiteOS-develop\Kernel\base\mem* 目錄下的兩個子目錄 *bestfit*、*tlsf* 刪除，如圖 2-3 所示。

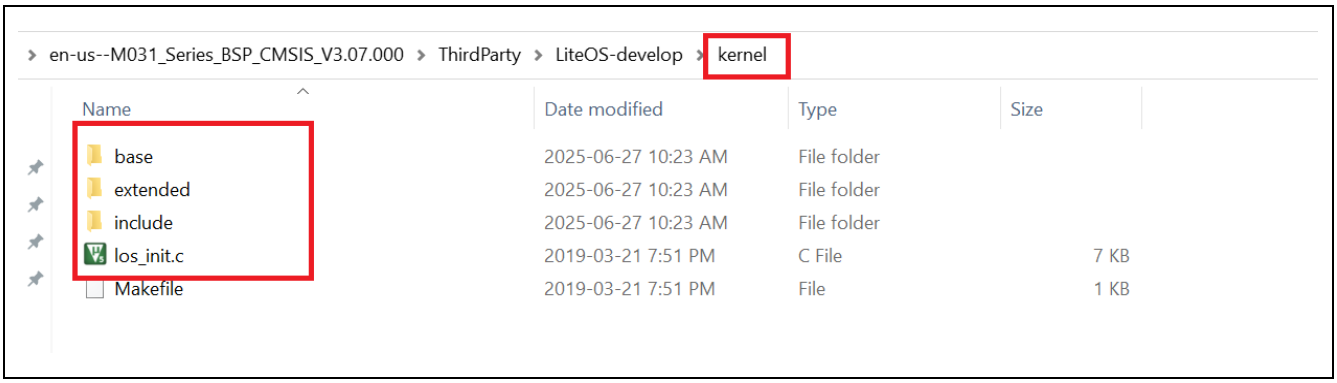

圖 2-3 更改後 kernel 目錄下檔案結構

從 LiteOS 原始碼.*arch\arm\arm-m* 中複製 *src*、*include* 兩個目錄到專案 *LiteOS-develop\arch* 目錄下。再複製 *arch\arm\arm-m\contex-m0\keil\los_dispatch_keil.S* 到專案 *LiteOS-develop\arch* 目錄下，如圖 2-4 所示。

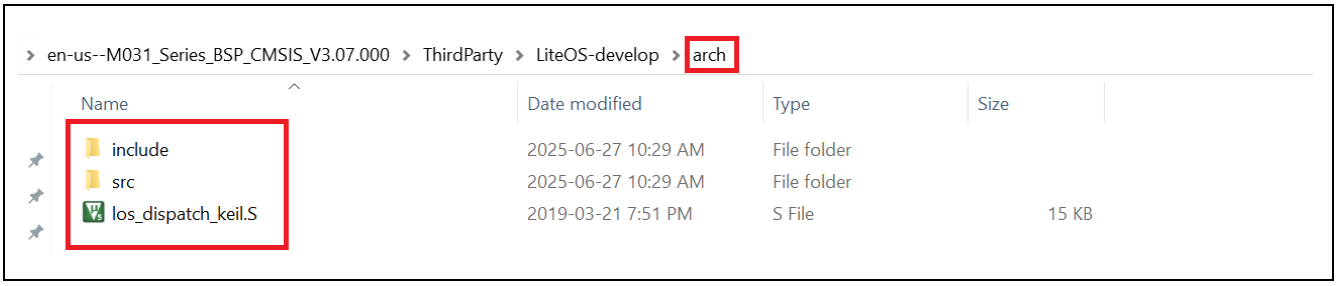

圖 2-4 更改後 arch 目錄下檔案結構

複製 LiteOS 原始碼 *osdepends\liteos\cmsis* 目錄中的檔案到專案 *LiteOS-develop\cmsis* 目錄下，如圖 2-5 所示。

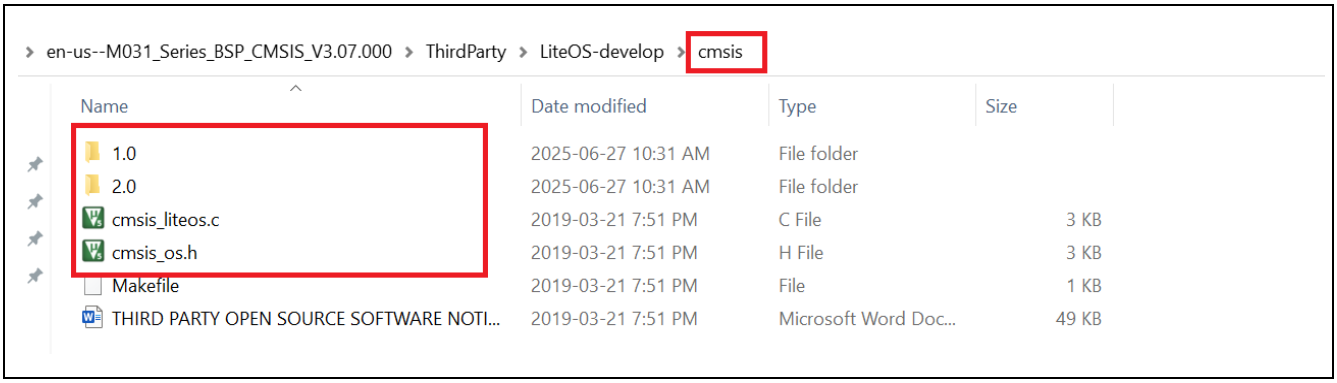

圖 2-5 更改後 cmsis 目錄下檔案結構

複製 LiteOS 源碼 *targets\STM32F103VET6_NB_GCC\OS_CONFIG* 目錄下標頭檔（.h 檔）到專案 *LiteOS\OS_CONFIG* 目錄下，如圖 2-6 所示。

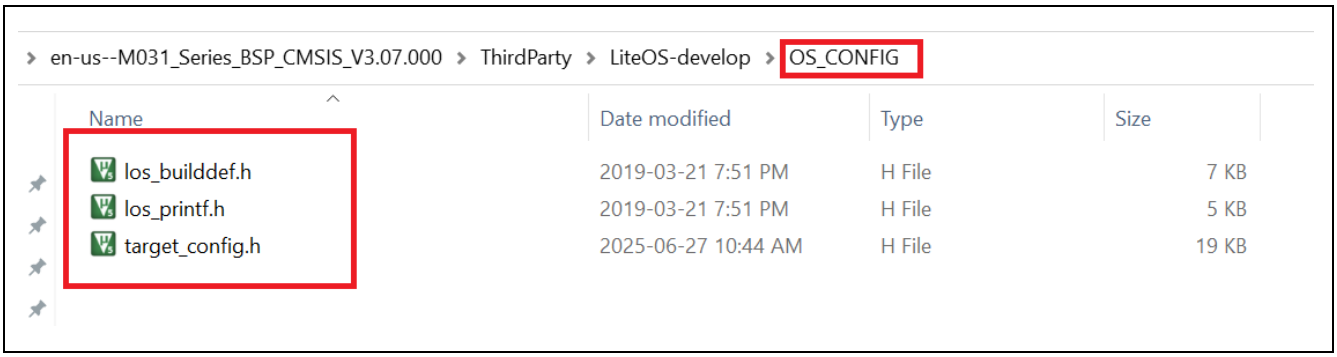

圖 2-6 更改後 OS_CONFIG 目錄下檔案結構

到此，移植 LiteOS 所需的檔案就準備好了。

2.3 建立 LiteOS Keil project

打開在 BSP 包中 GPIO_OutputInput 專案，將其名稱修改為 LiteOS，如圖 2-7、圖 2-8 所示。

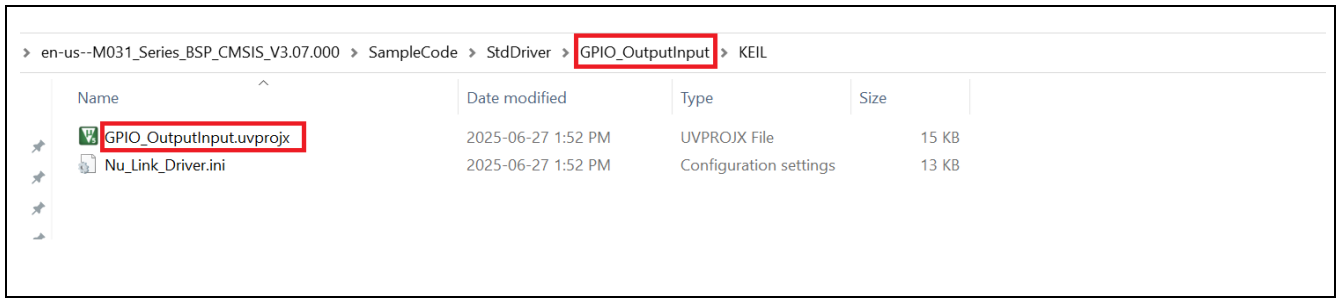

圖 2-7 修改前為 GPIO_OutputInput 專案目錄

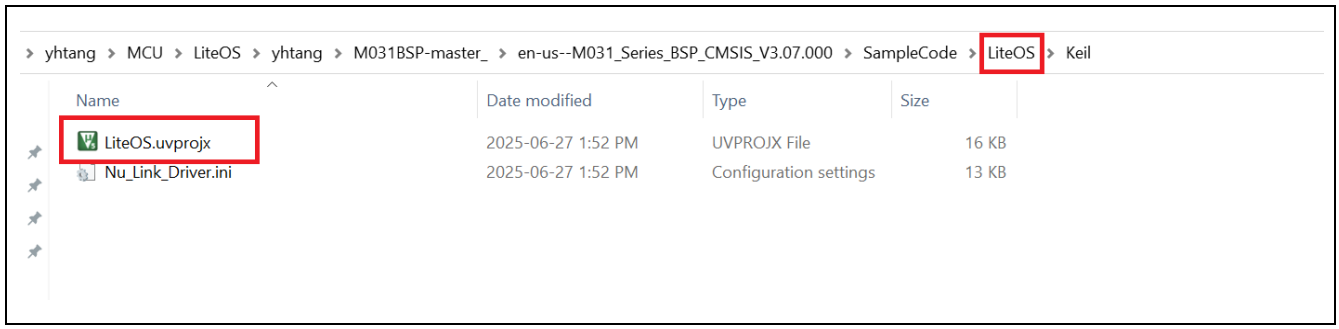

圖 2-8 修改後為 LiteOS 專案目錄

打開LiteOS專案檔，修改Project Target為LiteOS，並創建*LiteOS/Arch* 、*LiteOS/Kernel* 、*LiteOS/CMSIS LiteOS/Config*四個Group，如圖 2-9所示。

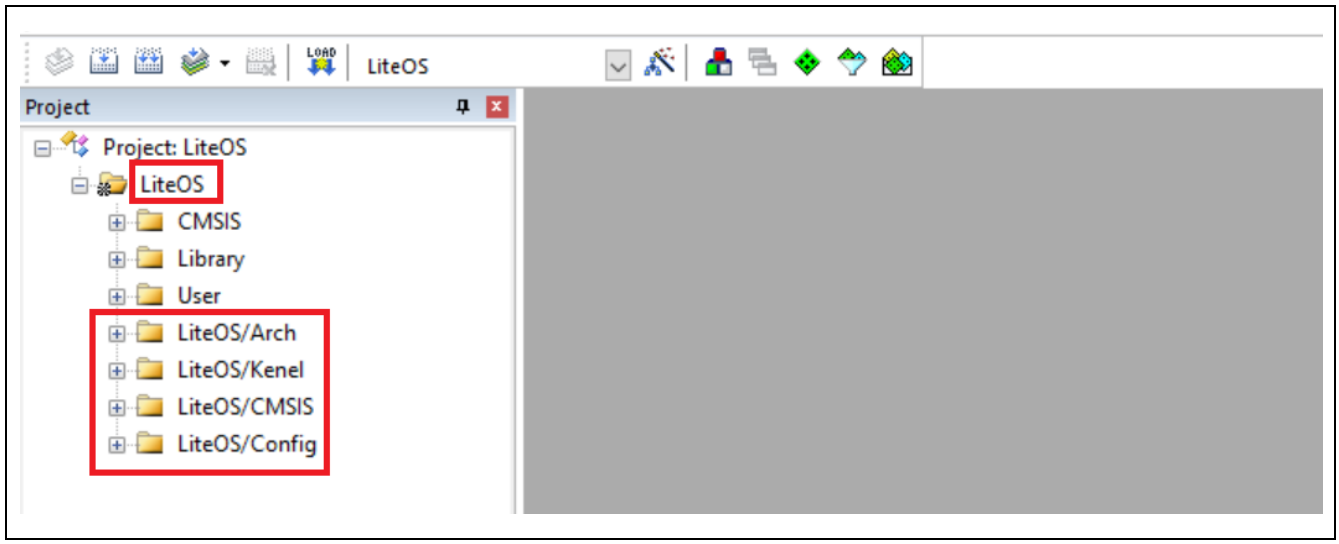

圖 2-9 LiteOS Project Items

在 LiteOS/Kernel 分組下，添加 *LiteOS\Kernel* 目錄下（包含子目錄下）的所有.c 檔案。

在 *LiteOS/Arch* 分組下，添加 *LiteOS\Arch* 目錄下（包含子目錄下）所有.c 檔案，以及 *los_dispatch_keil.S* 檔案。

在 *LiteOS/CMSIS* 分組下，添加 *LiteOS\CMSIS* 目錄下的 *cmsis_liteos.c*。

在 *LiteOS/Config* 分組下，添加 *LiteOS\Config* 目錄下的所有.h 的標頭檔。

添加完成後各 Group 下的檔案如圖 2-10 所示。

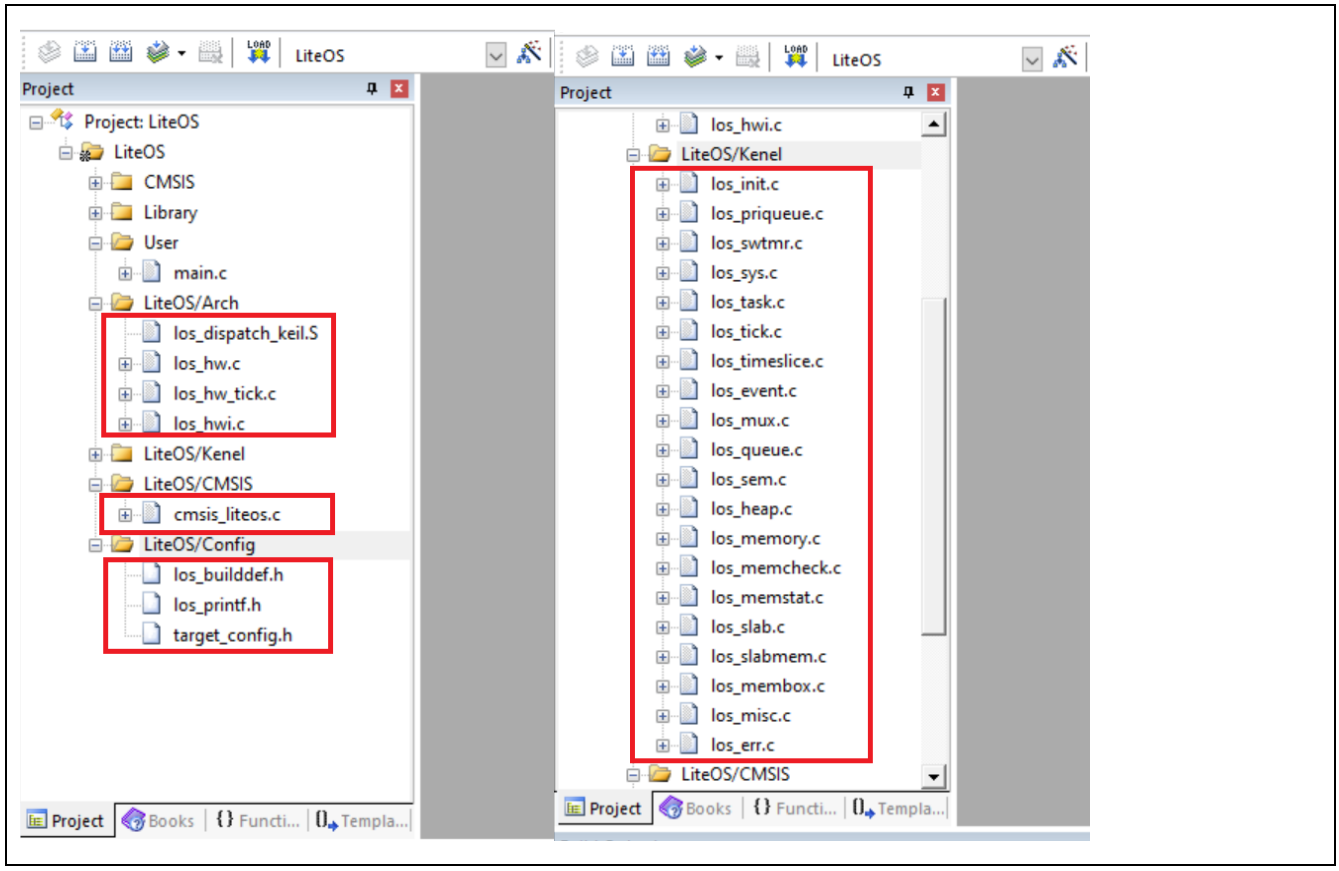

圖 2-10 添加的 LiteOS 檔案

2.4 修改 Keil Option

修改 Target 選項，如圖 2-11 所示。

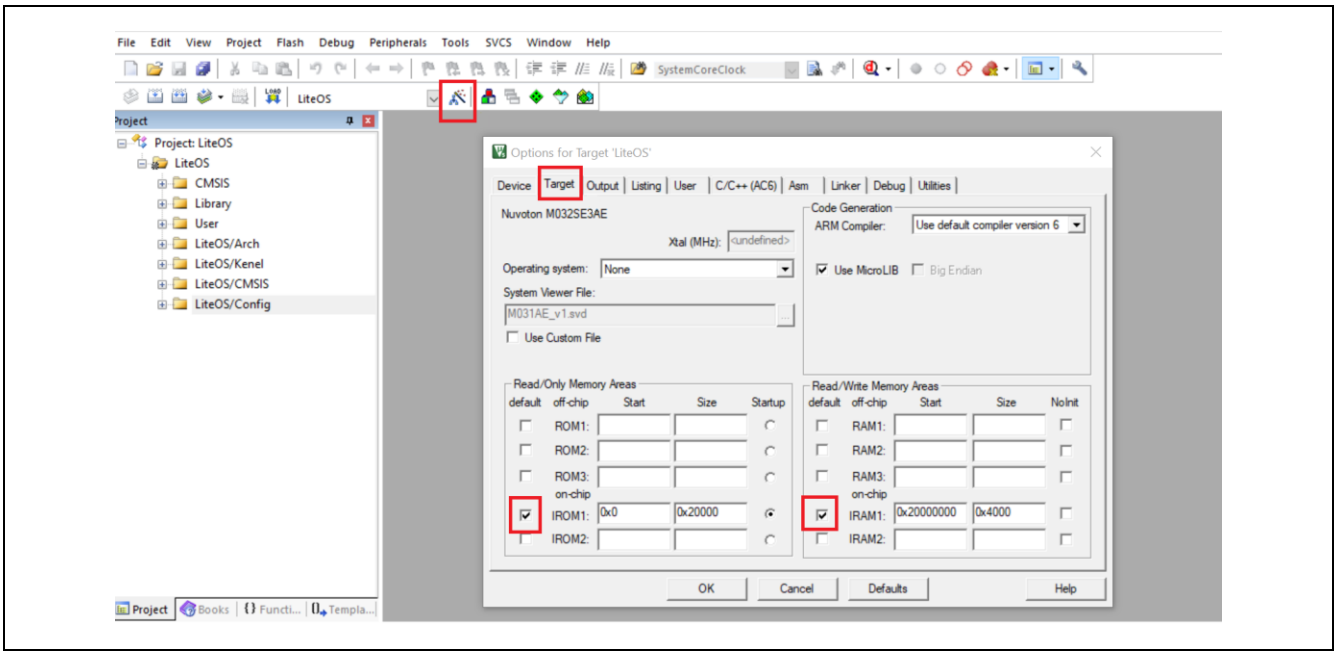

圖 2-11 修改 Keil Option->Target

修改 Linker 選項，如圖 2-12 所示。

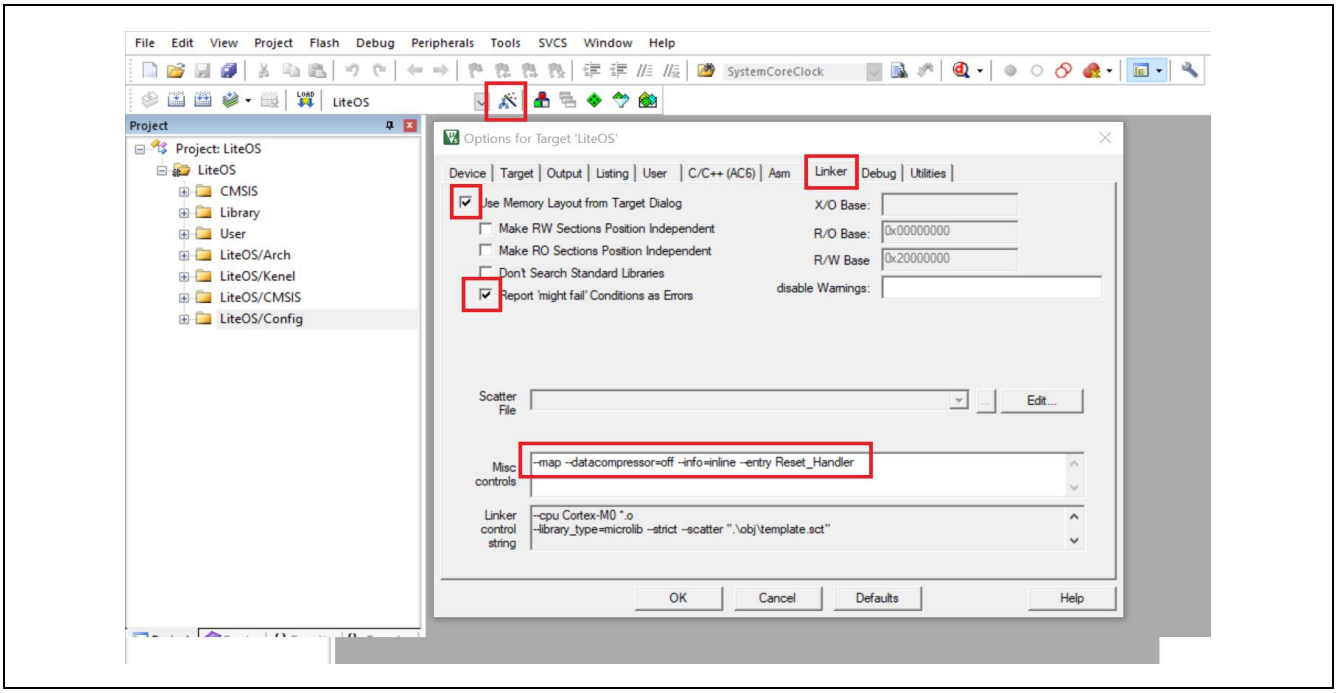

圖 2-12 修改 Keil Option->Linker

修改 Output 選項，如圖 2-13 所示。

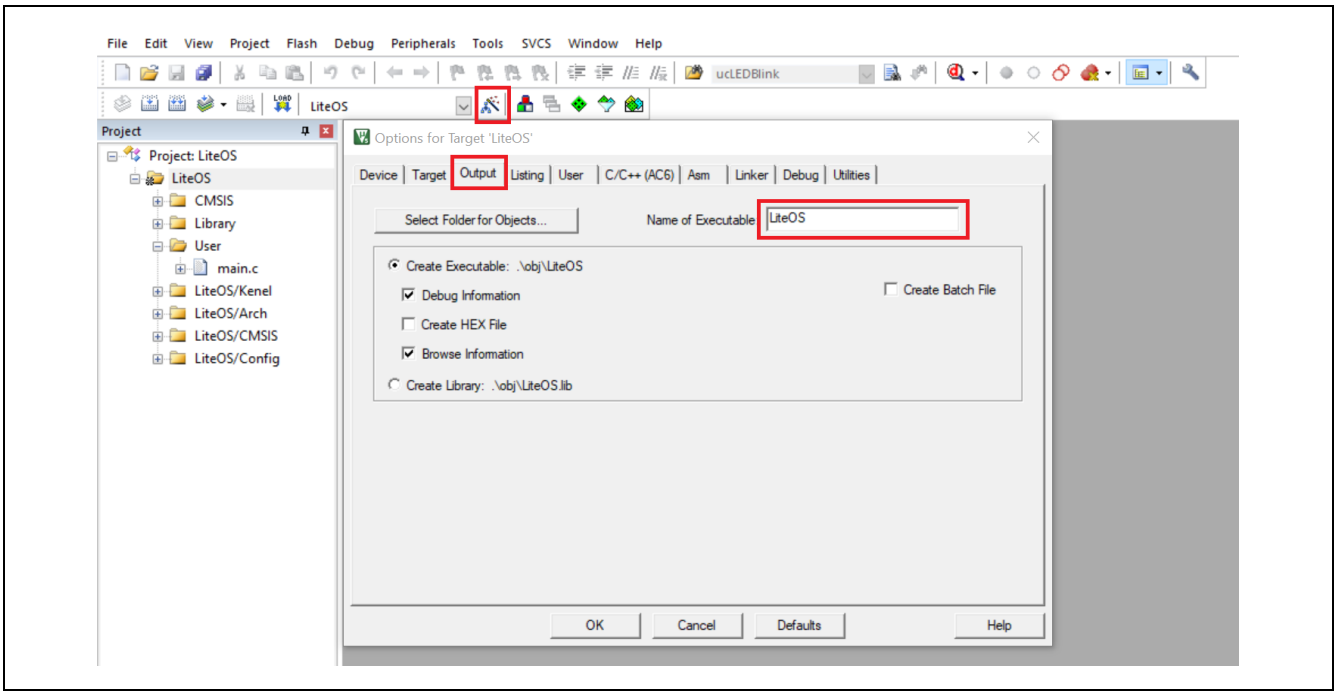

圖 2-13 修改 Keil Option->Output

添加標頭檔目錄，如圖 2-14所示。

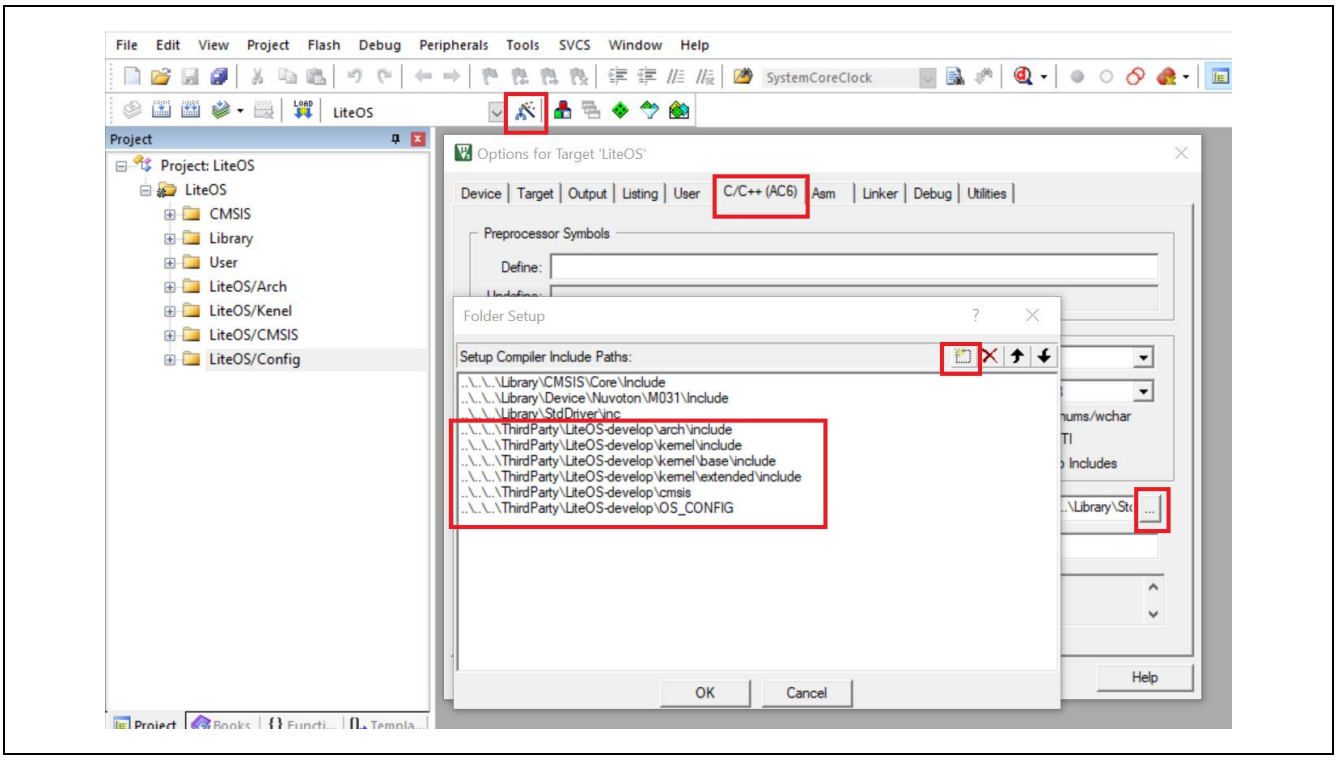

圖 2-14 修改標頭檔路徑 Keil Option->C/C++(AC6)

2.5 修改 target_config.h 檔案

檔案位於*\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\OS_CONFIG* 目錄下，修改該檔案，新增包含 *NuMicro.h*。

```
#include "NuMicro.h"
#include "los_typedef.h"
#include <stdio.h>
#include <string.h>
```

因為 M031SE3AE SRAM 為 16 KB，還需修改程式中 BOARD_SRAM_SIZE_KB 為 16 KB。

```
/**
 * Without modify default link script and startup file, In order to use all the remaining
 * SRAM space as LiteOS`s heap, User must config the start address and size of the board`s
 * memory, Because in KEIL compiler, the program can`t get the end address of SRAM.
 */
#define BOARD_SRAM_START_ADDR     0x20000000
#define BOARD_SRAM_SIZE_KB        16
#define BOARD_SRAM_END_ADDR       (BOARD_SRAM_START_ADDR + 1024 * BOARD_SRAM_SIZE_KB)
```

匹配 Keil V6 compiler，需修改 heap memory 編譯條件。

```
/**
 * Config the start address and size of the LiteOS`s heap memory
 */
#if defined ( __CC_ARM )||((__ARMCC_VERSION) && (__ARMCC_VERSION >= 6000000))
```

2.6 修改 los_typedef.h 檔案

檔案位於*\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\kernel\include* 目錄下，修改該檔案，新增包含 *NuMicro.h* 。

```
#ifndef _LOS_TYPEDEF_H
#define _LOS_TYPEDEF_H
#include "NuMicro.h"
#include "los_builddef.h"
```

2.7 修改 los_hw.h 檔案

檔案位於*\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\arch\include* 目錄下， M031 系列晶片沒有浮點運算單元，因此檔中定義 FPU_EXIST 的代碼需更改，更改後如下。

```
/**
 * @ingroup los_hw
 * FPU exist macro: If it exist, save EXC_RETURN value in the context of the task to check
 * the size of the stack frame(BIT_4: 1 ---> 8 words, 0 ---> 26 words) used by the task.
 */
#define FPU_EXIST  0   // FPU does not exist
```

2.8 修改 main.c 檔案

檔案位於*\EC_M031_LiteOS_V1.00\SampleCode\ExampleCode\LiteOS* 目錄下，修改該檔案，新增包含 *los_base.h* 、 *los_sys.h* 、*los_typedef.h* 、*los_task.h*。

```
/* ---------------------- LiteOS includes ----------------------------------*/
#include "los_base.h"
#include "los_sys.h"
#include "los_typedef.h"
#include "los_task.h"
```

定義執行緒相關全部變數。

```
uint32_t g_TskHandle;
```

創建執行緒 1，執行緒 1 函數中每隔 1000 ms 執行一次並列印“This is task1,count is ***”。

```
/*---------------------------------------------------------------------------*/
/* Print message per 1000ms                                                  */
/*---------------------------------------------------------------------------*/
void task1(void)
{
    uint32_t count = 1;

    while (1)
    {
        printf("This is task1,count is %d\r\n", count++);
        LOS_TaskDelay(1000);
    }
}

/*---------------------------------------------------------------------------*/
/* Creat LiteOS task1                                                        */
/*---------------------------------------------------------------------------*/
uint32_t creat_task1(void)
{
    uint32_t uwRet = LOS_OK;
    TSK_INIT_PARAM_S task_init_param;

    task_init_param.usTaskPrio = 0;                         /**< Task priority         */
    task_init_param.pcName = "task1";                       /**< Task name             */
    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task1; /**< Task entrance function*/
    task_init_param.uwStackSize = 0x200;                    /**< Task stack size       */

    uwRet = LOS_TaskCreate(&g_TskHandle, &task_init_param);

    if (LOS_OK != uwRet)
    {
        return uwRet;
    }

    return uwRet;
}
```

創建執行緒 2，執行緒 2 函數中每隔 2000 ms 執行一次並列印“This is task2,count is ***”。

```
/*---------------------------------------------------------------------------------------*/
/* Print message per 2000ms                                                              */
/*---------------------------------------------------------------------------------------*/
void task2(void)
{
    uint32_t count = 1;

    while (1)
    {
        printf("This is task2,count is %d\r\n", count++);
        LOS_TaskDelay(2000);
    }
}

/*---------------------------------------------------------------------------------------*/
/* Creat LiteOS task2                                                                    */
/*---------------------------------------------------------------------------------------*/
uint32_t creat_task2(void)
{
    uint32_t uwRet = LOS_OK;
    TSK_INIT_PARAM_S task_init_param;

    task_init_param.usTaskPrio = 1;                       /**< Task priority         */
    task_init_param.pcName = "task2";                     /**< Task name             */
    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)task2; /**< Task entrance function*/
    task_init_param.uwStackSize = 0x200;                  /**< Task stack size       */

    uwRet = LOS_TaskCreate(&g_TskHandle, &task_init_param);

    if (LOS_OK != uwRet)
    {
        return uwRet;
    }

    return uwRet;
}
```

主函數 main 重新設計。

```
/*
 * This is a LiteOS project for M031 series MCU. Users could based on this project to
 * create their own application without worry about the Keil project settings.
 * This project shows how to runs LiteOS on M031.
 */
int main(void)
{
    uint32_t uwRet = 0;
    SYS_Init();

    /* Init UART0 to 115200-8n1 for print message */
    UART_Open(UART0, 115200);

    /* Connect UART to PC, and open a terminal tool to receive following message */
    printf("Hello World\n");
    printf("\n\nCPU @ %dHz\n", SystemCoreClock);
    printf("+-----------------------------------------------+\n");
    printf("|    Lite OS Sample Code     |\n");
    printf("+-----------------------------------------------+\n\n");
    /* Init LiteOS Kernel */
    uwRet = LOS_KernelInit();
```

```
    if (uwRet != LOS_OK)
    {
        return LOS_NOK;
    }

    /* Creat task1 */
    uwRet = creat_task1();

    if (uwRet != LOS_OK)
    {
        return LOS_NOK;
    }

    /* Creat task2 */
    uwRet = creat_task2();

    if (uwRet != LOS_OK)
    {
        return LOS_NOK;
    }

    /* Start LiteOS */
    LOS_Start();

    /* Got no where to go, just loop forever */
    while (1);
}
```

2.9 修改各目錄名稱

對各目錄名稱進行相應修改，並刪除非必要檔案，如圖 2-15 所示。

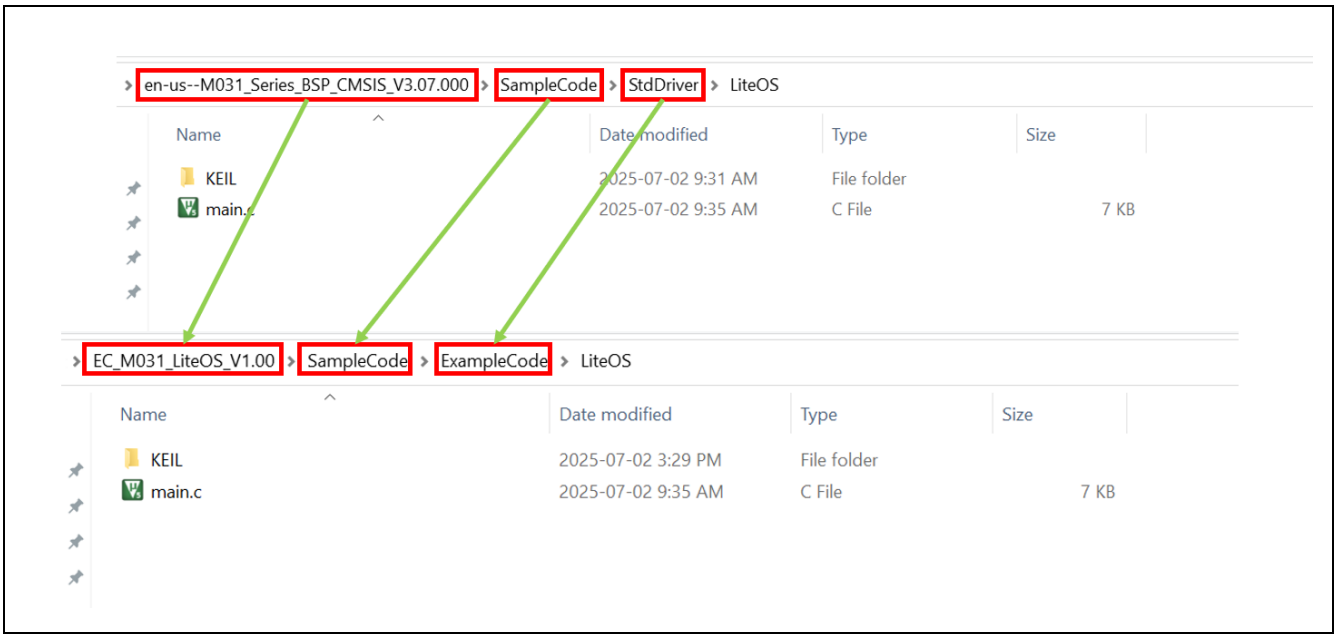

圖 2-15 修改各目錄名稱

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M031_Series_BSP_CMSIS_V3.07.000
- IDE 版本
  - Keil uVersion 5.41

3.2 硬體需求

- 電路元件
  - NuMaker-M031SE3AE V1.1
- 線路示意圖
  - 通過設置 Nu-Link2-Me 上的撥碼開關，使能其 USB 轉串口功能。

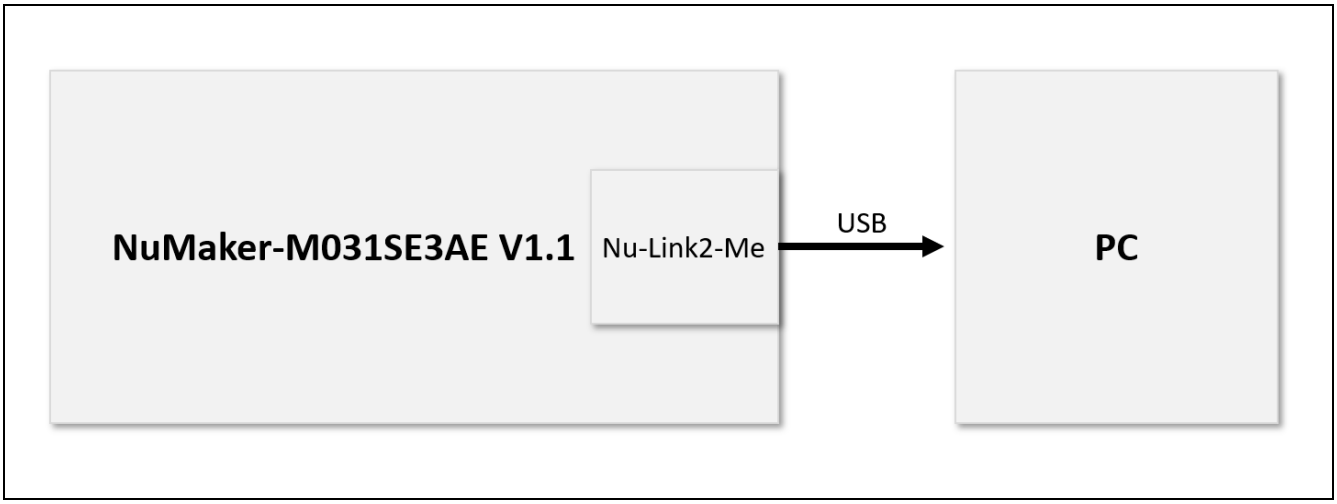

圖 3-1 線路示意圖

4. 目錄資訊

目錄結構如圖 4-1 所示。

EC_M031_LiteOS_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
ThirdParty	
LiteOS-develop	Source file of LiteOS code
SampleCode	
ExampleCode	
LiteOS	Source file of example code

圖 4-1 目錄資訊

5. 範例程式執行

根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *LiteOS.uvprojx*。

1. 進入編譯模式介面
   - 編譯
   - 下載程式碼至記憶體
   - 進入 / 離開模擬模式
2. 進入模擬模式介面
   - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2025.07.24	1.00	初始發佈。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.