LiteOS Porting

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example is based on the GPIO_OutputInput code in the BSP, porting LiteOS and creating 2 threads.
BSP Version	M031_Series_BSP_CMSIS_V3.07.000
Hardware	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. Overview

This example describes how to port LiteOS and create 2 threads under the Keil MDK compilation environment based on the GPIO_OutputInput reference code in the BSP. Through UART, Thread 1 outputs "This is task1, count is ***" every 1000 ms; Thread 2 outputs "This is task2, count is *** " every 2000 ms.

1.1 Principle

LiteOS is a lightweight operating system developed by Huawei, primarily targeting IoT devices. Its characteristics include:

- Compact size, low memory usage (minimum kernel can be less than 10 KB).
- Strong real-time capability, suitable for scenarios requiring fast response.
- Supports multiple mainstream chip architectures (e.g., ARM Cortex-M, RISC-V, etc.).
- Supports multiple network protocols (e.g., Wi-Fi, NB-IoT, LoRa, etc.).
- Open-source code (some versions), easy for secondary development and customization.

LiteOS is similar to other IoT RTOS like FreeRTOS, RT-Thread, and Zephyr but has unique advantages in localization, ecosystem compatibility, and integration with Huawei's IoT ecosystem. It is widely used in smart homes, wearables, vehicular networks, urban public services, manufacturing, etc., significantly reducing device deployment and maintenance costs, effectively lowering the development threshold, and shortening the development cycle.

This example uses the M031SE3AE microcontroller, which has 128 KB Flash and 16 KB RAM. It describes how to port LiteOS in Keil MDK and create 2 threads.

1.2 Demo Result

After compiling and programming the example code, the UART prints information as shown in Figure 1-1.

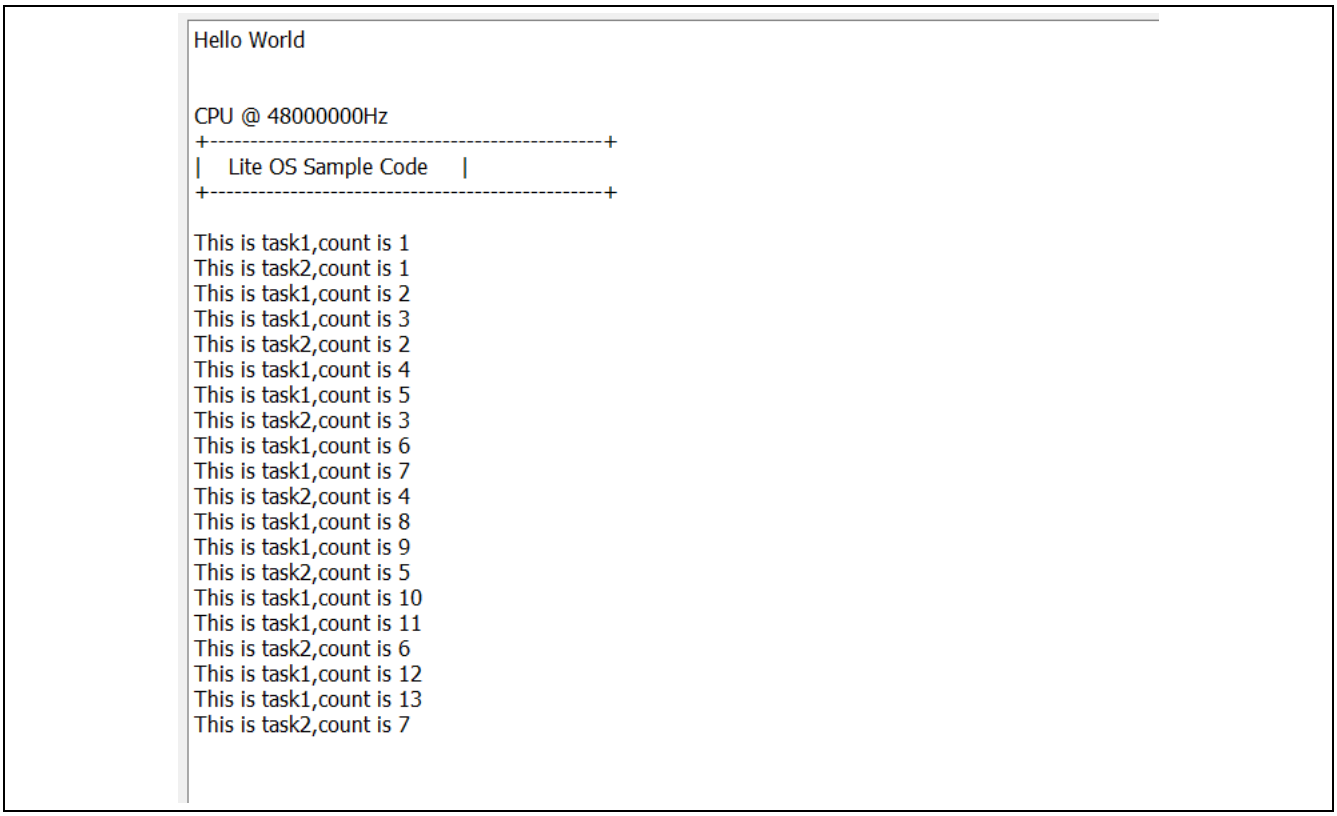

Figure 1-1 UART Printed Information

2. Code Description

2.1 Download LiteOS Source Code

Download the LiteOS source code:

- Use the develop branch, link: https://github.com/LiteOS/LiteOS/tree/develop;
- Use the LiteOSV200R001C50B038 tag, link: https://github.com/LiteOS/LiteOS/tree/LiteOSV200R001C50B038 .

Both links can be used. This example uses the develop branch. The source code structure after extraction is shown in Figure 2-1.

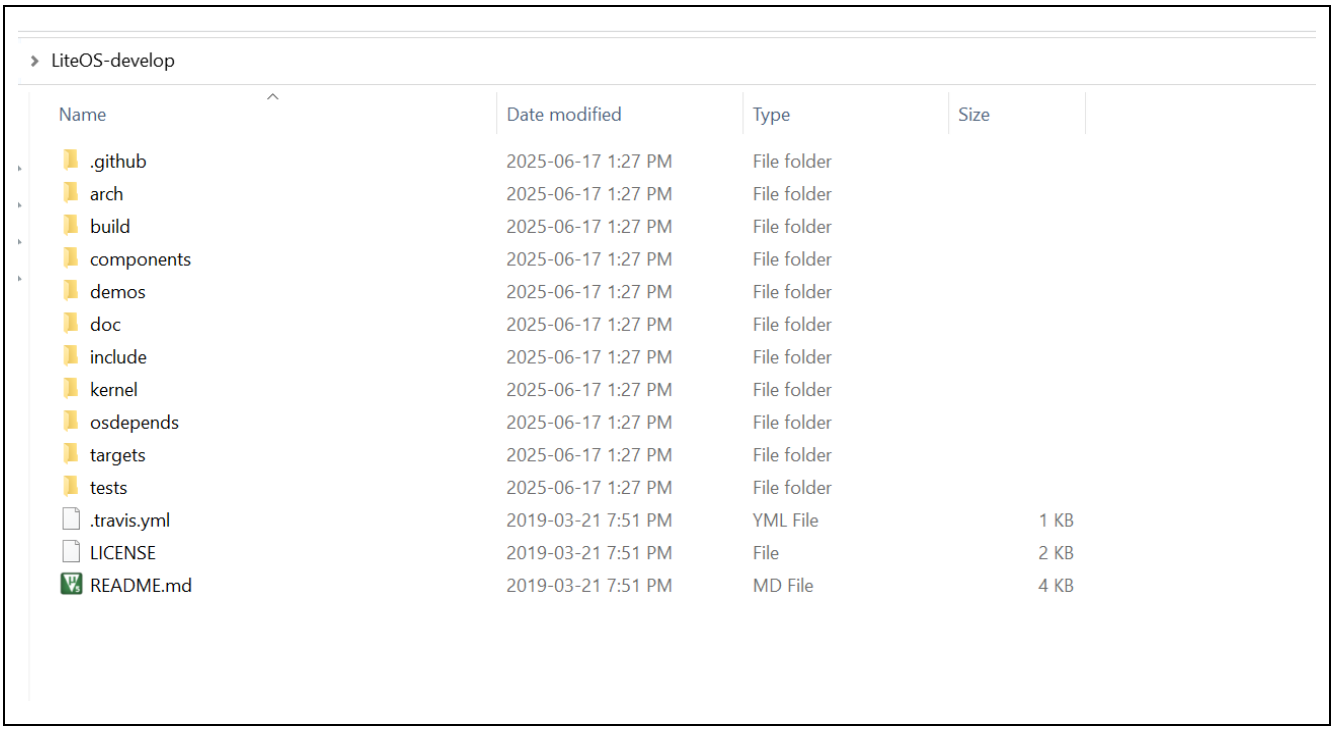

Figure 2-1 LiteOS Source Code

2.2 Copy LiteOS Source Files to BSP Path

First, create a subdirectory named *LiteOS-develop* under *BSP\ThirdParty* to store the LiteOS source code. Within the newly created *LiteOS-develop* directory, create four subdirectories named *kernel*, *arch*, *cmsis*, and *OS_CONFIG*, as shown in Figure 2-2.

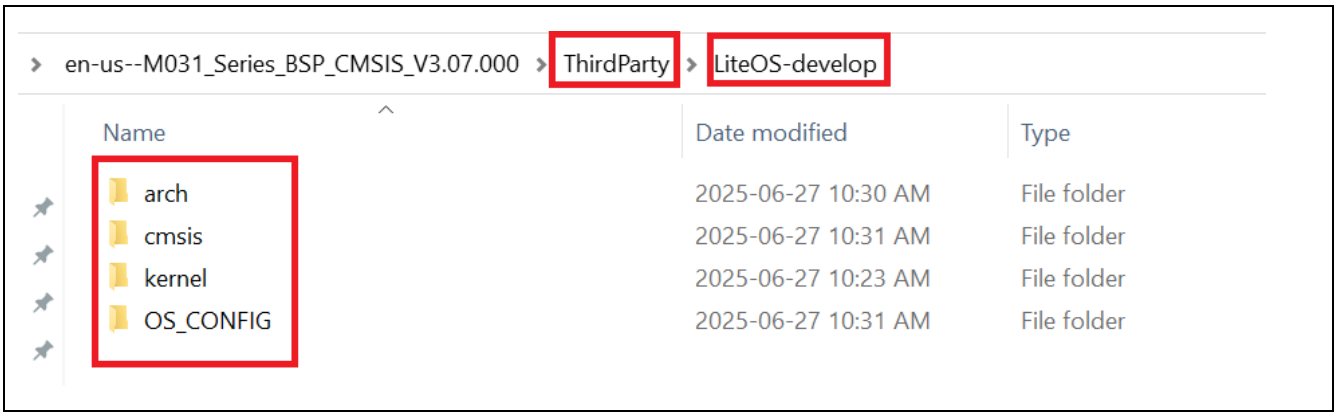

Figure 2-2 Location of the LiteOS Source Code

Copy all contents from *.\kernel* in the LiteOS source tree to the project directory *.\LiteOS-develop\kernel*. Then delete the two subdirectories *bestfit* and *tlsf* from *.\LiteOS-develop\Kernel\base\mem,* as shown in Figure 2-3.

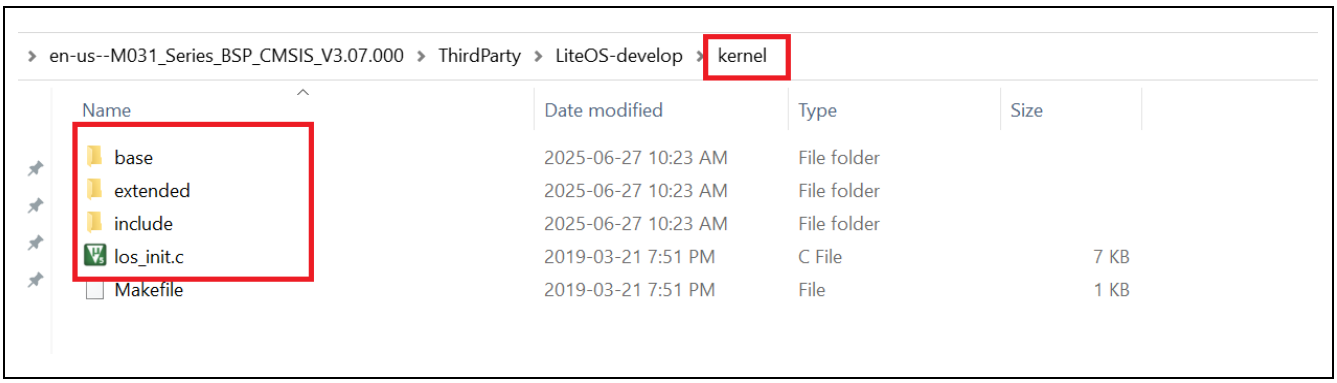

Figure 2-3 Kernel Directory Structure after Changes

Copy the two directories *src* and *include* from LiteOS source *.\arch\arm\arm-m* to the project directory *.\LiteOS-develop\arch*. Copy *.\arch\arm\arm-m\contex-m0\keil\los_dispatch_keil.S* to *.\LiteOS-develop\arch* in the project, as shown in Figure 2-4.

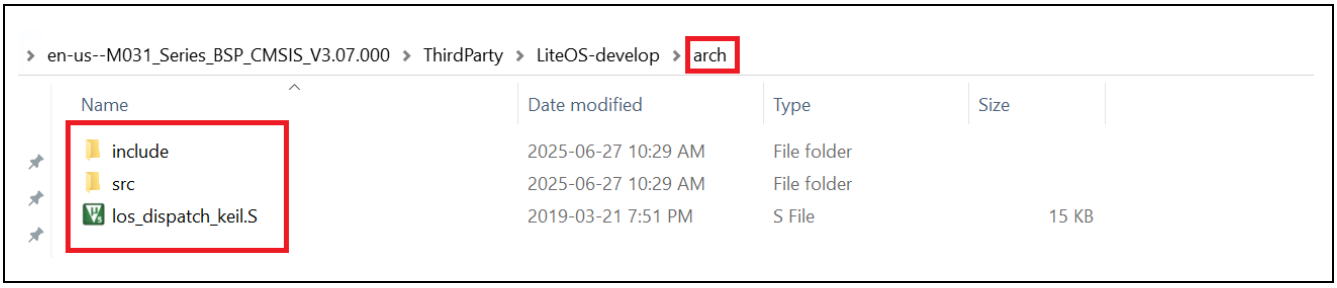

Figure 2-4 Arch Directory Structure after Changes

Copy the files from the LiteOS source directory *osdepends\liteos\cmsis* to the project directory *.\LiteOS-develop\cmsis*, as shown in Figure 2-5.

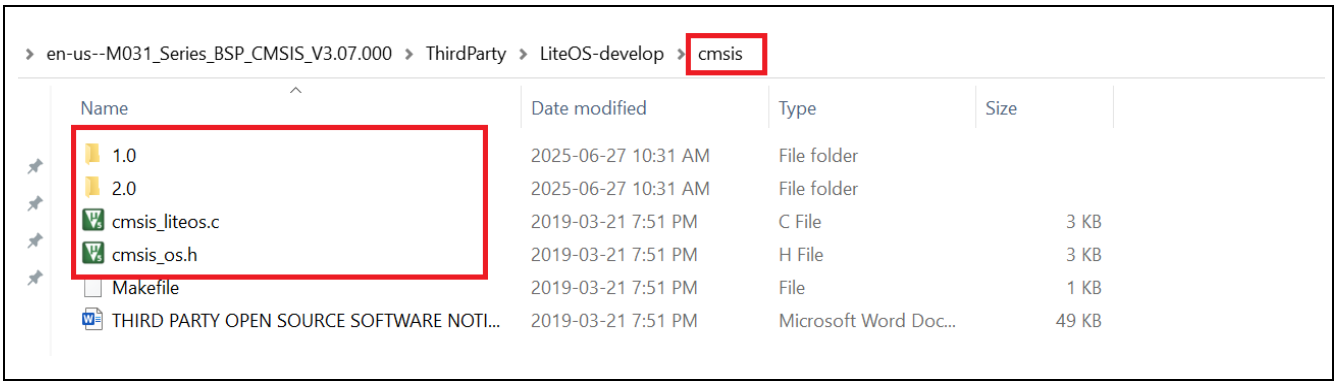

Figure 2-5 cmsis Directory Structure after Changes

Copy the header files from LiteOS source *targets\STM32F103VET6_NB_GCC\OS_CONFIG* to the project directory *.\LiteOS\OS_CONFIG*, as shown in Figure 2-6.

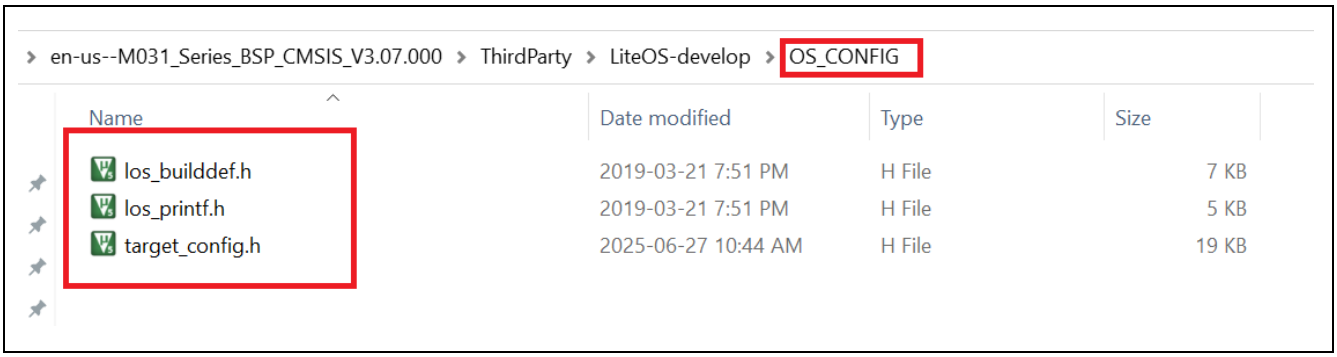

Figure 2-6 OS_CONFIG Directory Structure after Changes

The files required for porting LiteOS are now prepared.

2.3 Create LiteOS Keil Project

Open the GPIO_OutputInput project in the BSP package and rename it to LiteOS, as shown in Figure 2-7 and Figure 2-8.

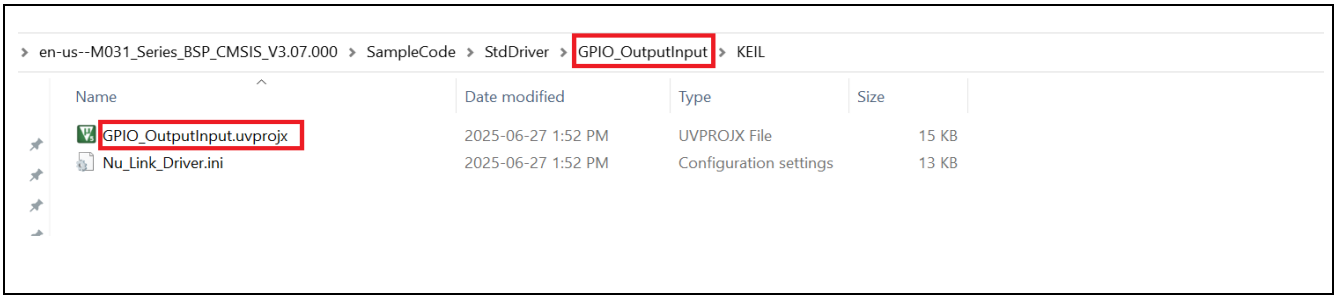

Figure 2-7 Project Directory prior to Modification as GPIO_OutputInput

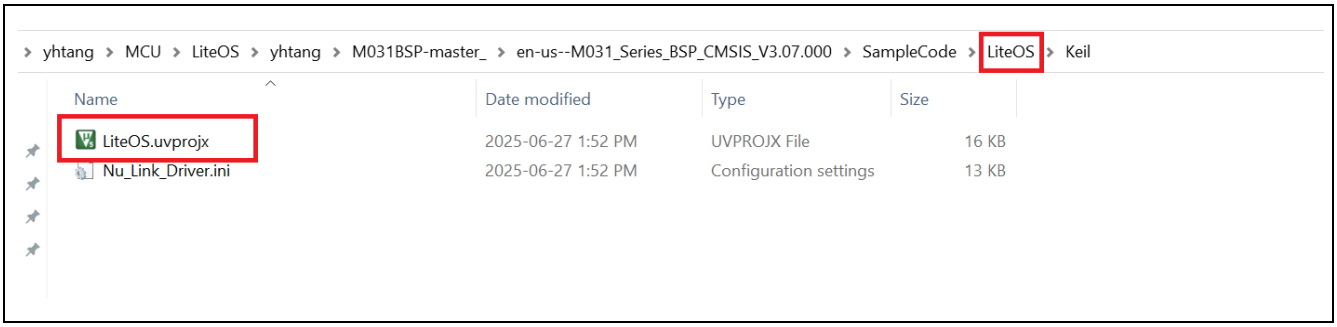

Figure 2-8 Project Directory after Modification as LiteOS

Open the LiteOS project file, change the Project Target to LiteOS, and create four groups — *LiteOS/Arch*, *LiteOS/Kernel*, *LiteOS/CMSIS*, and *LiteOS/Config*, as shown in Figure 2-9.

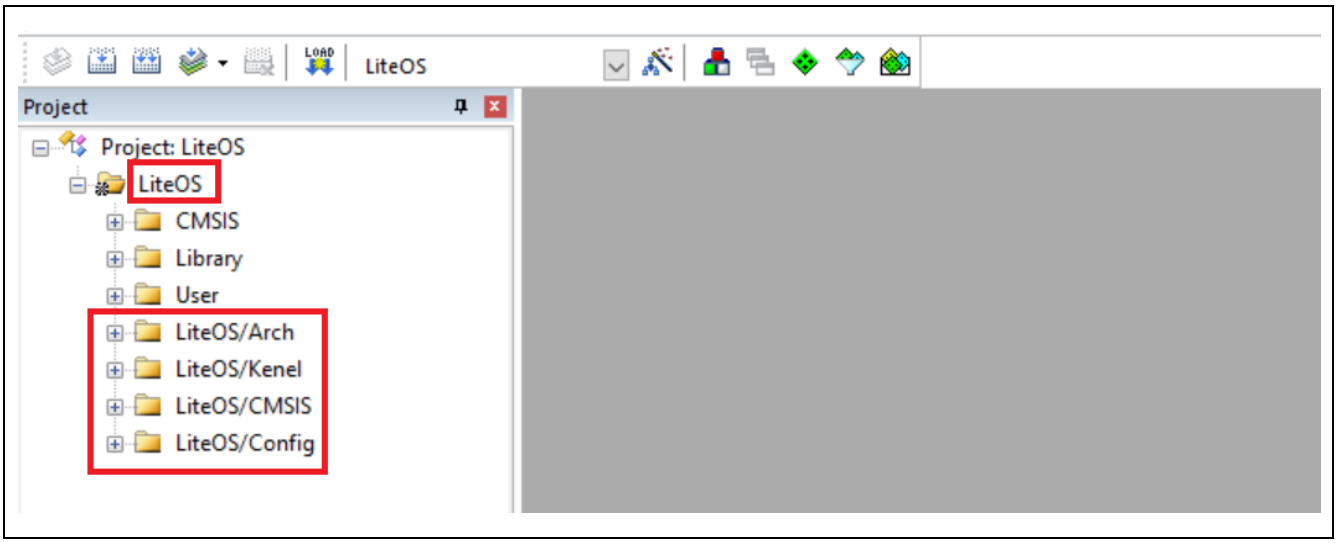

Figure 2-9 LiteOS Project Items

Under the *LiteOS/Kernel* group, add all *.c* files under the *LiteOS\Kernel* directory (including subdirectories).

Under the *LiteOS/Arch* group, add all *.c* files under the *LiteOS\Arch* directory (including subdirectories), and the *los_dispatch_keil.S* file.

Under the *LiteOS/CMSIS* group, *add cmsis_liteos.c* from the *LiteOS\CMSIS* directory.

Under the *LiteOS/Config* group, add all *.h* header files from the *LiteOS\Config* directory.

The files under each Group after being added are shown in Figure 2-10.

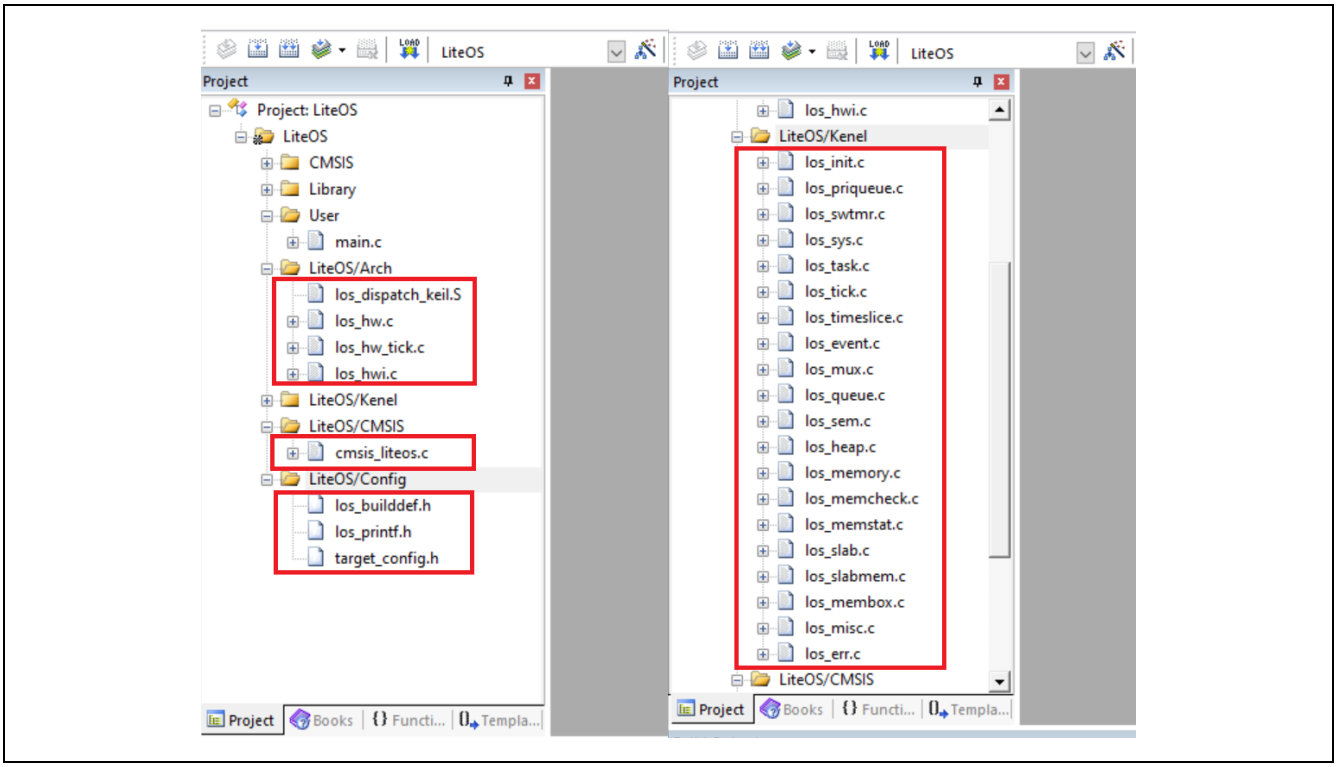

Figure 2-10 Added LiteOS Files

2.4 Modify Keil Option

Modify the Target options, as shown in Figure 2-11.

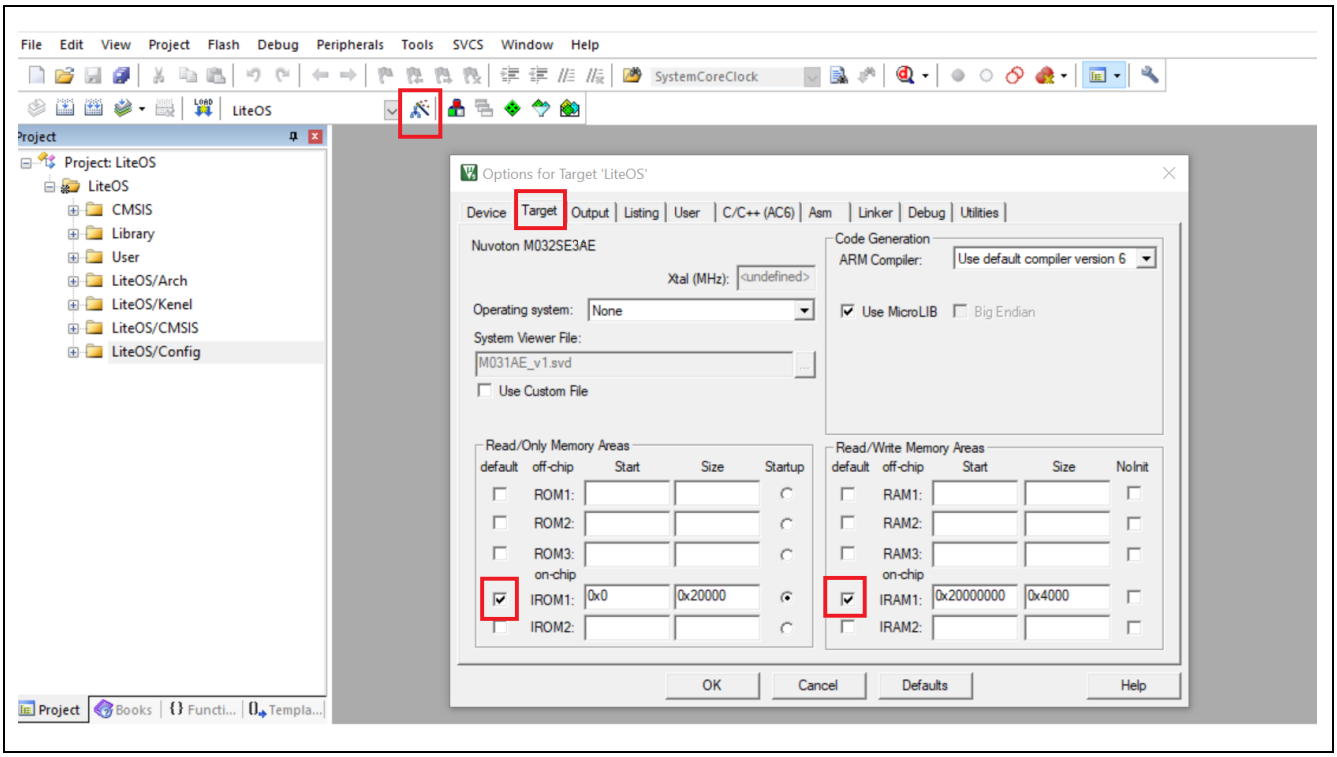

Figure 2-11 Modify Keil Option->Target

Modify the Linker options, as shown in Figure 2-12.

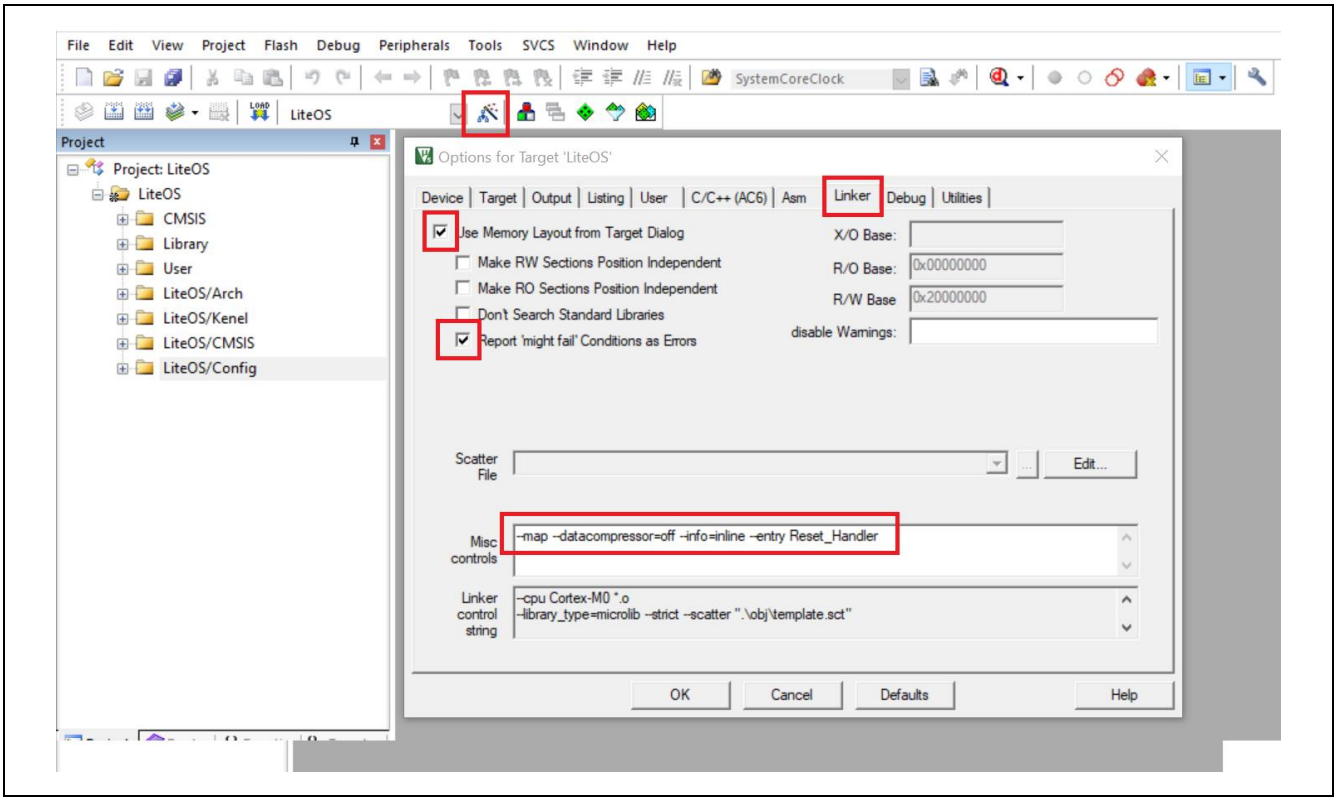

Figure 2-12 Modify Keil Option->Linker

Modify the Output options, as shown in Figure 2-13

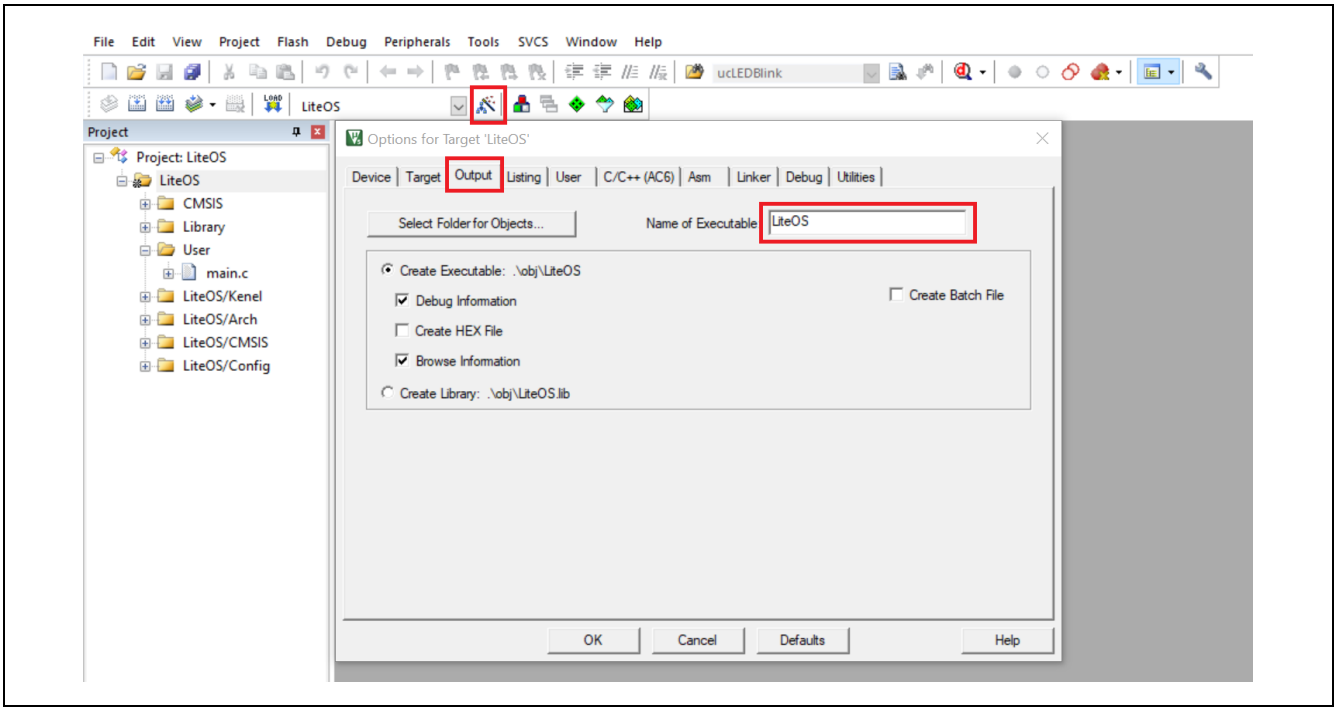

Figure 2-13 Modify Keil Option->Output

Add the include file directories, as shown in Figure 2-14.

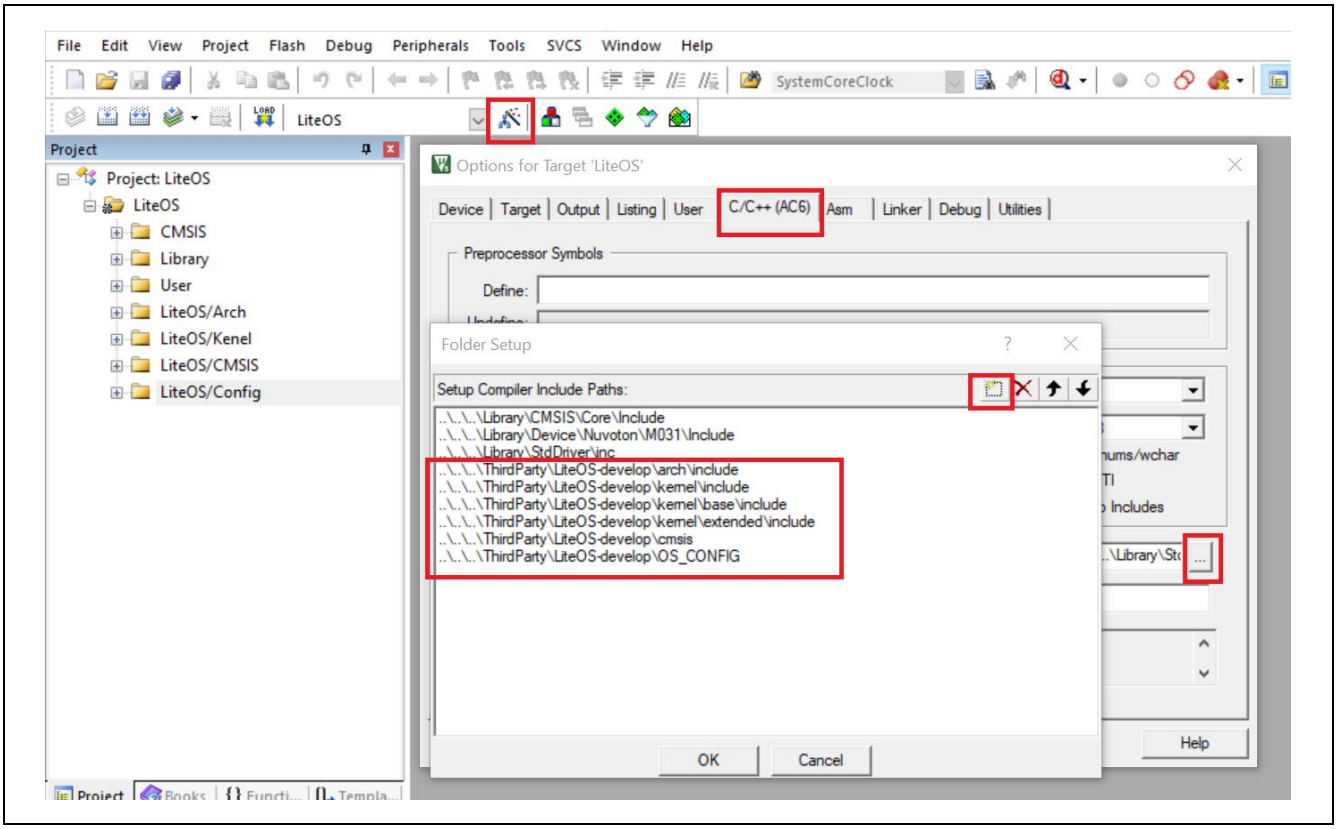

Figure 2-14 Update the Header Search Paths under Keil Option->C/C++(AC6)

2.5 Modify target_config.h

This file is located in *\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\OS_CONFIG*. Modify it to include *NuMicro.h.*

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

Since the M031SE3AE SRAM is 16 KB, also modify the BOARD_SRAM_SIZE_KB definition to 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)
```

To match the Keil V6 compiler, modify the heap memory compilation condition.

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

2.6 Modify los_typedef.h

This file is located in *\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\kernel\include*. Modify it to include *NuMicro.h*.

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

2.7 Modify los_hw.h

This file is located in *\EC_M031_LiteOS_V1.00\ThirdParty\LiteOS-develop\arch\include*. The M031 series does not have an FPU, so the code defining FPU_EXIST needs to be changed as follows:

```
/**
 * @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 Modify main.c

This file is located in *\EC_M031_LiteOS_V1.00\SampleCode\ExampleCode\LiteOS*. Modify it to include *los_base.h, los_sys.h, los_typedef.h, los_task.h* files.

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

Define the thread-related global variable.

```
   uint32_t g_TskHandle;
```

Create Thread 1, which executes every 1000 ms and prints "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;
}
```

Create Thread 2, which executes every 2000 ms and prints "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;
}
```

Redesign the main function.

```
/*
 * 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 Modify Directory Name

Modify the corresponding directory names and delete unnecessary files, as shown in Figure 2-15.

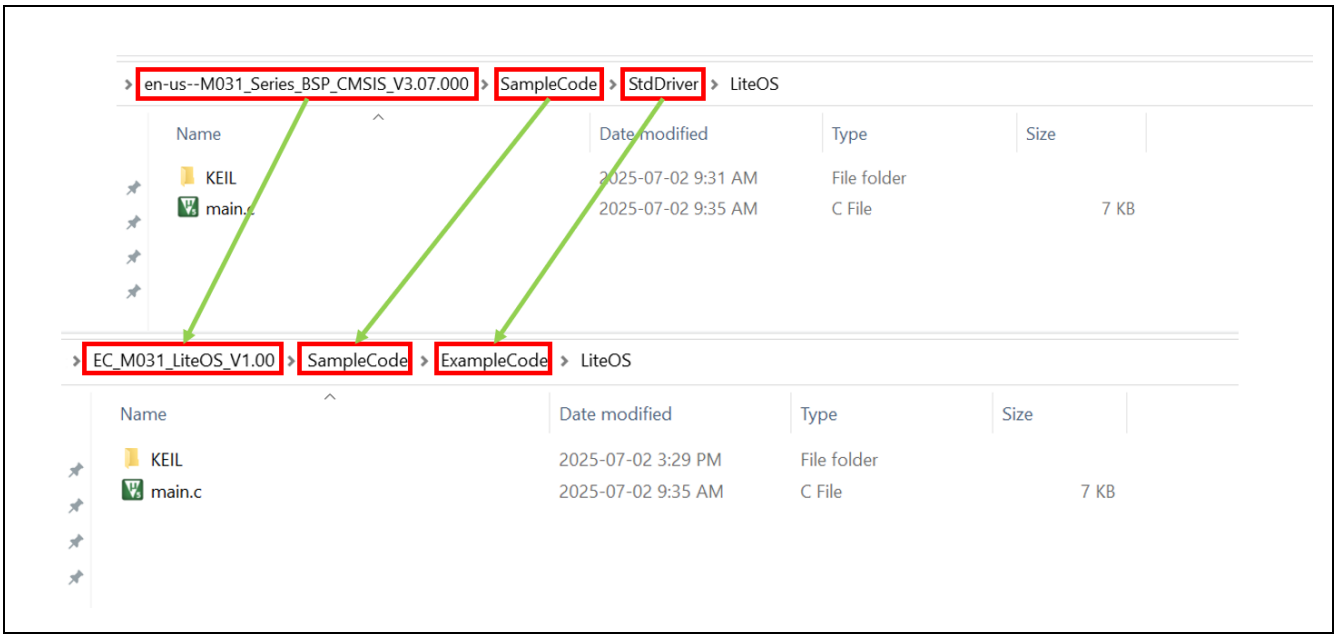

Figure 2-15 Rename Each Directory

3. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M031SE3AE V1.1
- Pin Connect
  - You need to enable the USB-to-serial port function by setting the DIP switch on the Nu-Link2-Me.

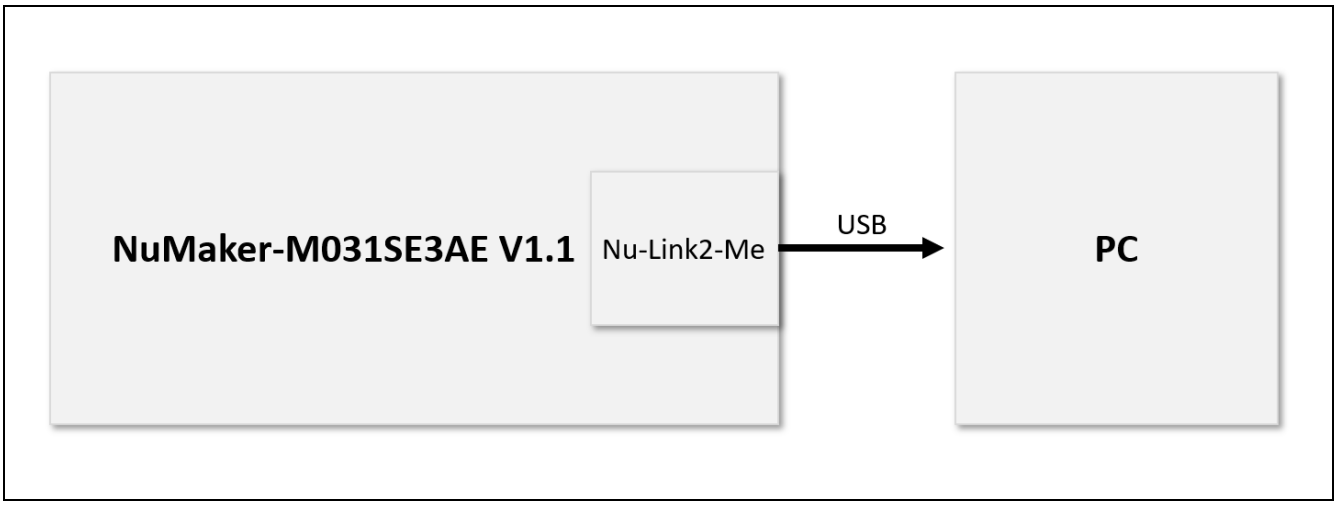

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

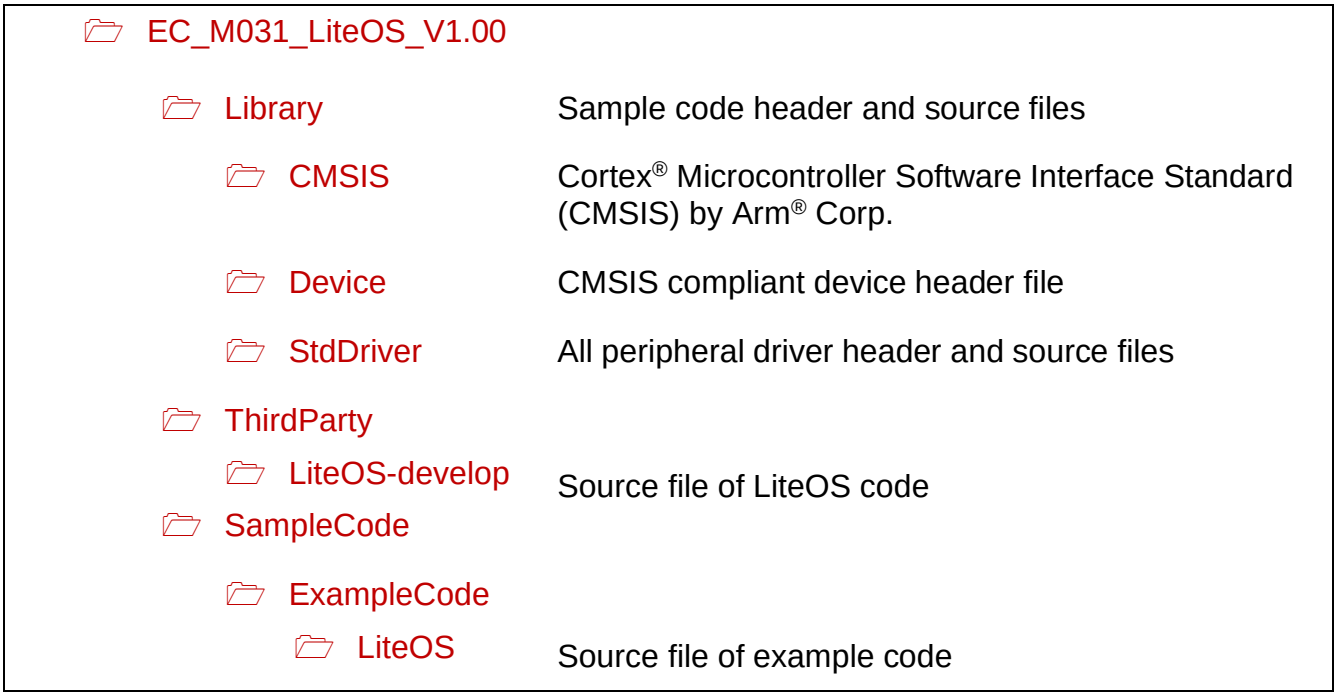

Figure 4-1 Directory Structure

5. Example Code Execution

Browse the sample code folder as described in the Directory Information section and double-click *LiteOS.uvprojx*.

1. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
2. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.07.24	1.00	Initial version.

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.