FSC_STOS CubeMX 工程(Hal 库)移植教程

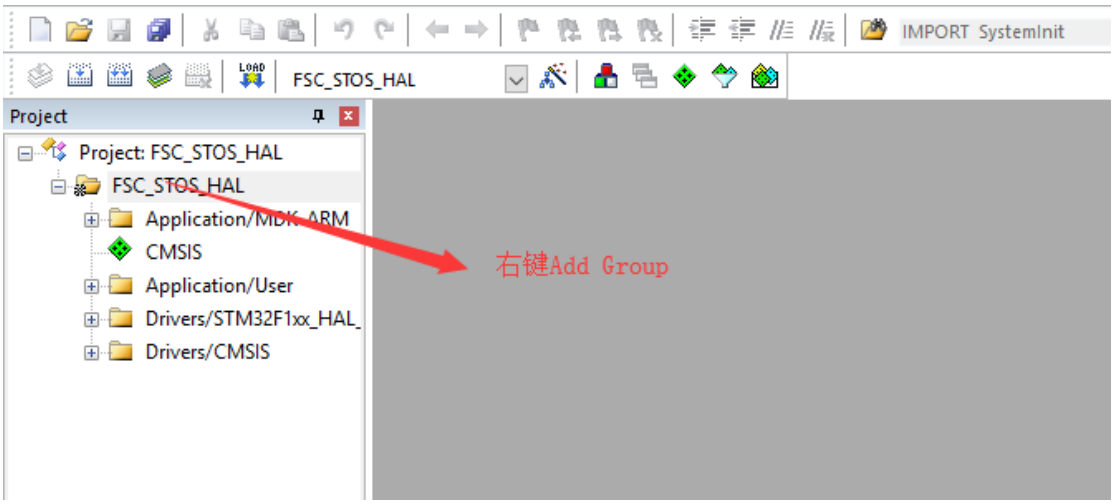
——望穿秋水

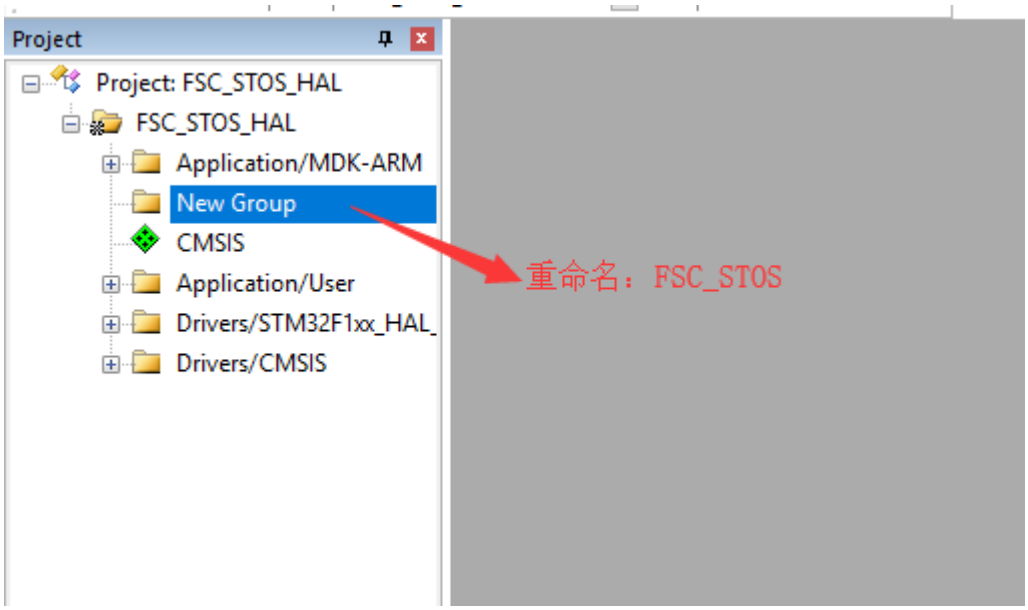
1.在裸机工程中加入 FSC_STOS 内核: (基于 FSC_STOS_V3.4 版本)

以 stm32f103c8t6　hal 库工程加入 FSC_STOS 为例:

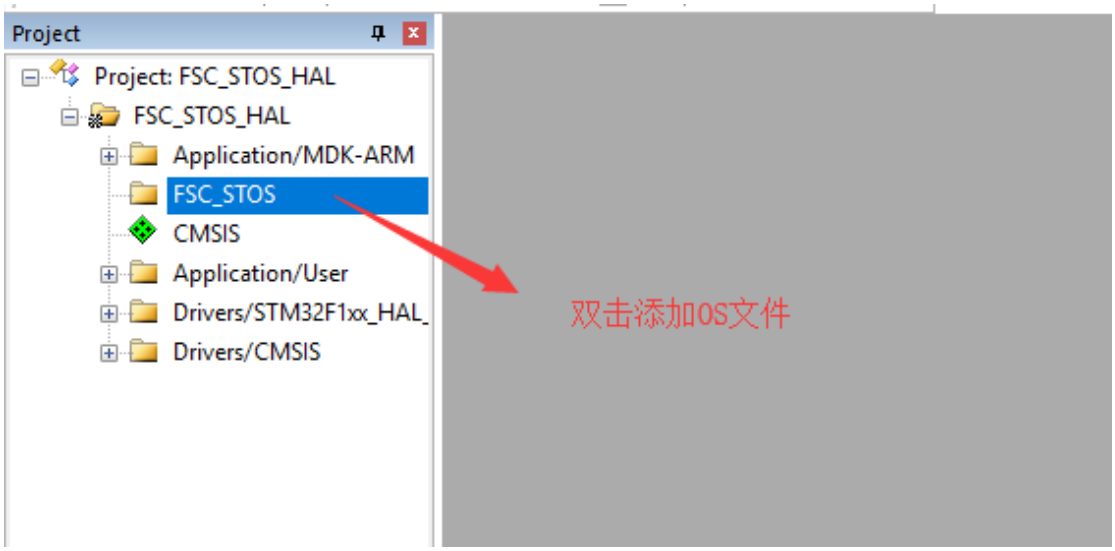
<1>把 FSC_STOS 专用移植文件夹复制到裸机工程根目录文件夹中

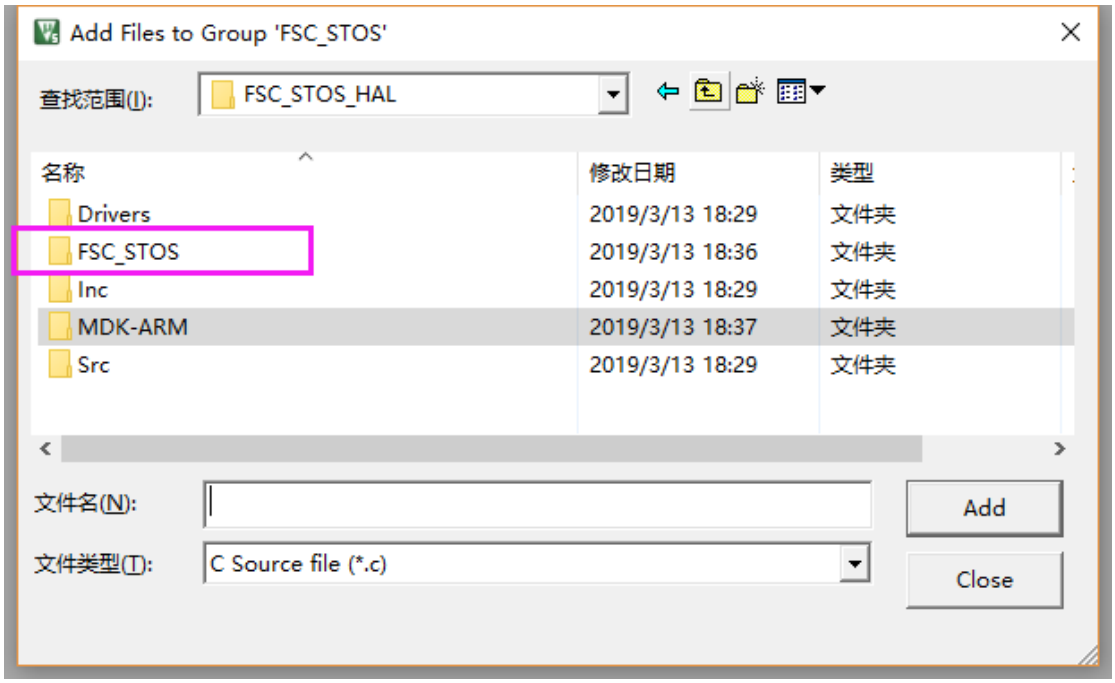
<2>打开 hal 库 MDK 工程，在工程中新建 FSC_STOS 文件夹，

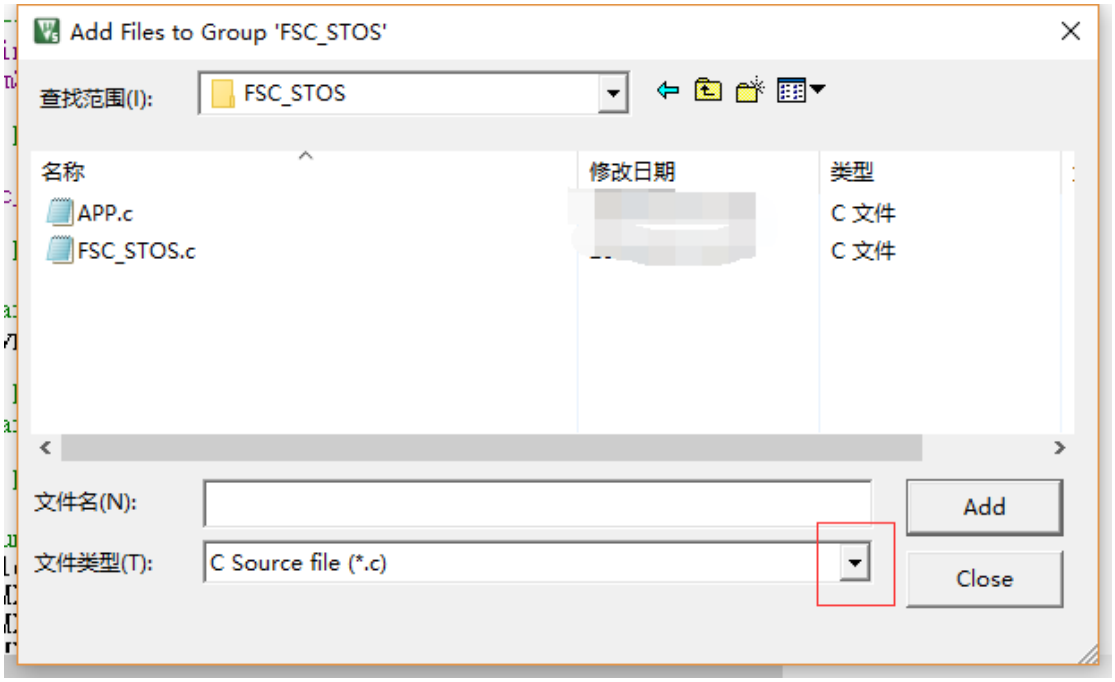


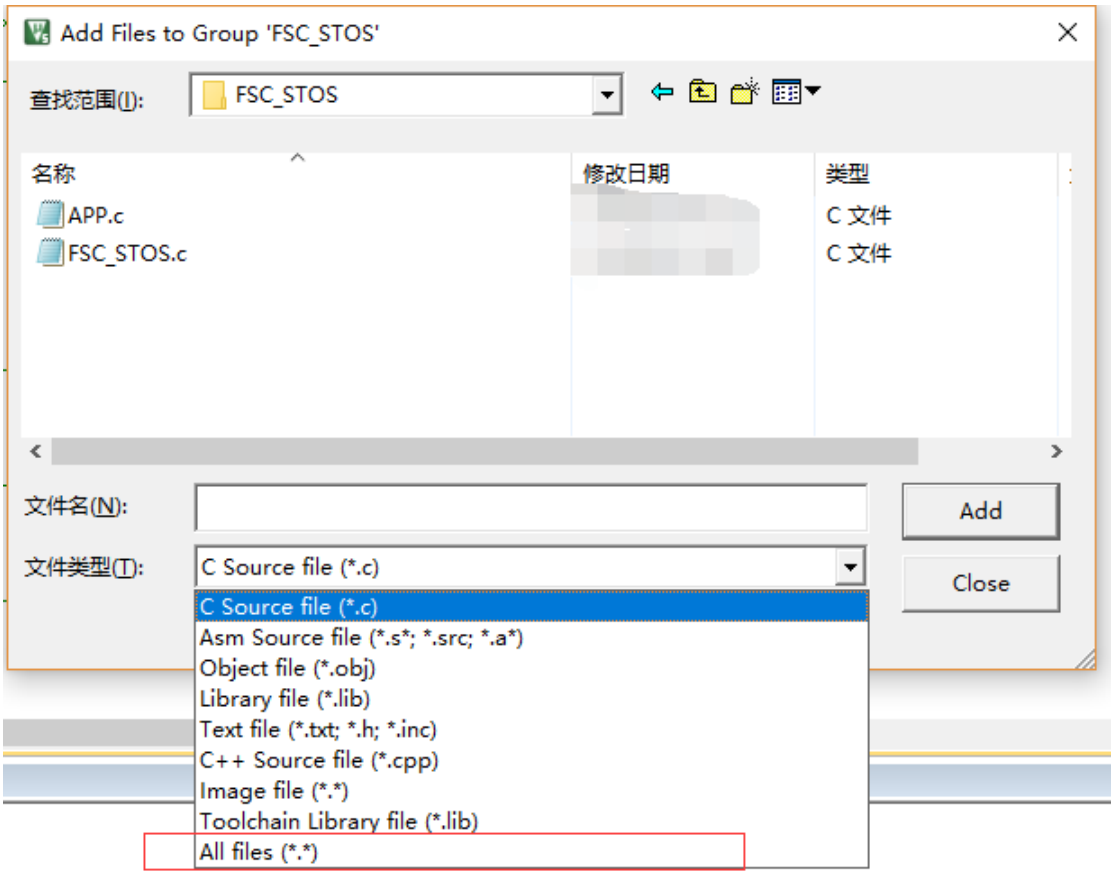


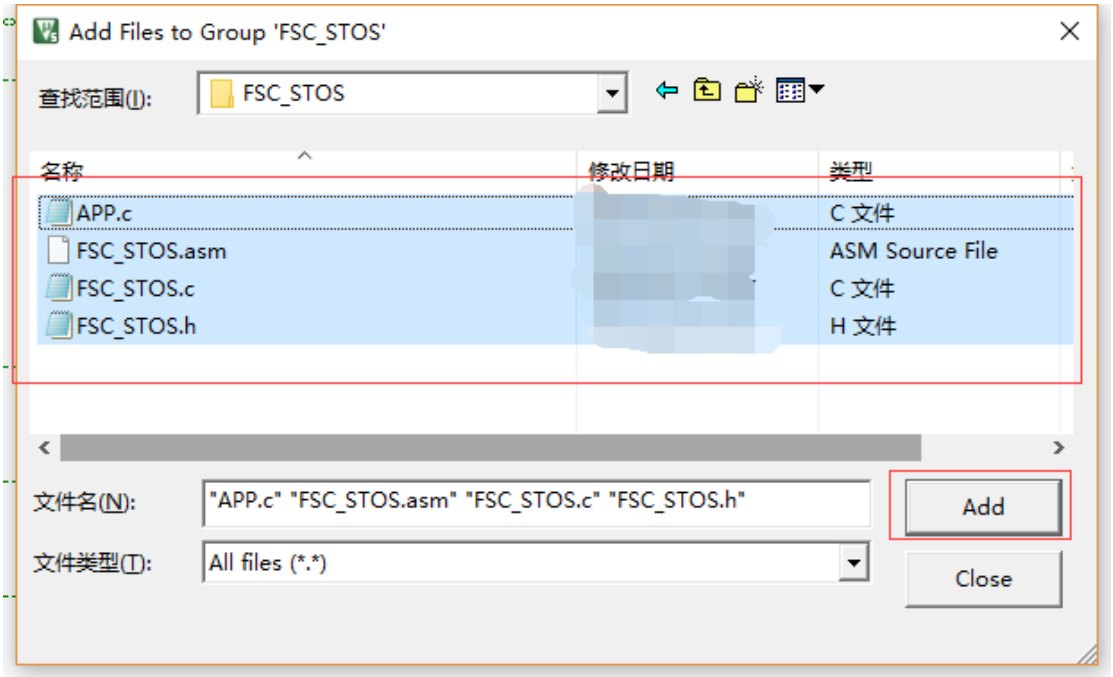
<3>文件夹加入 fsc_stos.c、fsc_stos.h、fsc_stos.asm、main.c 共 4 个文件。



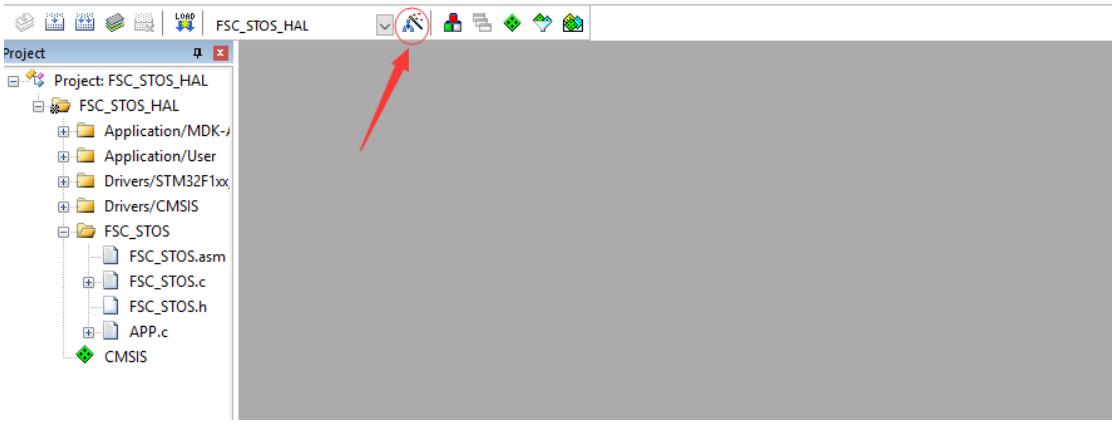

Add Files to Group 'FSC_STOS'
查找范围(I):
FSC_STOS_HAL
名称
修改日期
类型
Drivers
2019/3/13 18:29
文件夹
FSC_STOS
2019/3/13 18:36
文件夹
Inc
2019/3/13 18:29
文件夹
MDK-ARM
2019/3/13 18:37
文件夹
Src
2019/3/13 18:29
文件夹
文件名(N):
文件类型(T):
C Source file (*.c)
Add
Close


Add Files to Group 'FSC_STOS'
查找范围(I):
FSC_STOS
名称
修改日期
类型
APP.c
C 文件
FSC_STOS.c
C 文件
文件名(N):
文件类型(T):
C Source file (*.c)
Add
Close

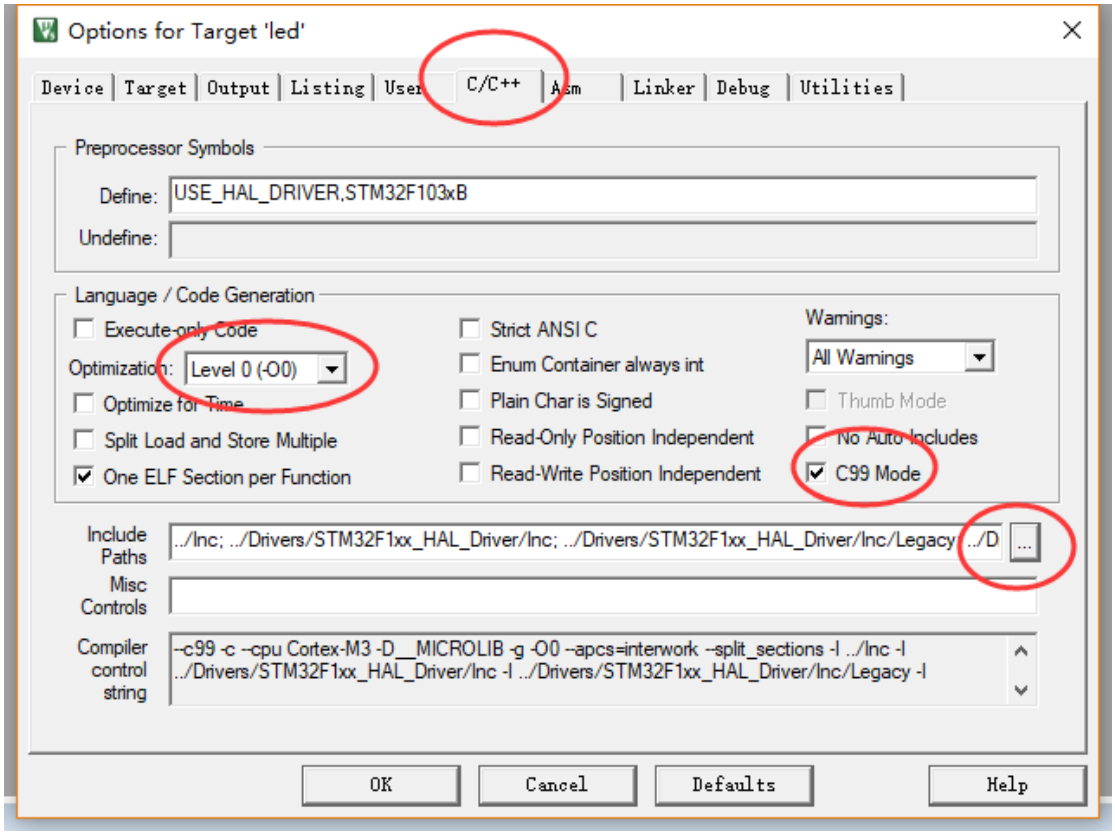

Add Files to Group 'FSC_STOS'
查找范围(I):
FSC_STOS
名称
修改日期
类型
APP.c
C 文件
FSC_STOS.c
C 文件
文件名(N):
Add
文件类型(T):
C Source file (*.c)
Close
C Source file (*.c)
Asm Source file (*.s*; *.src; *.a*)
Object file (*.obj)
Library file (*.lib)
Text file (*.txt; *.h; *.inc)
C++ Source file (*.cpp)
Image file (*.*)
Toolchain Library file (*.lib)
All files (*.*)

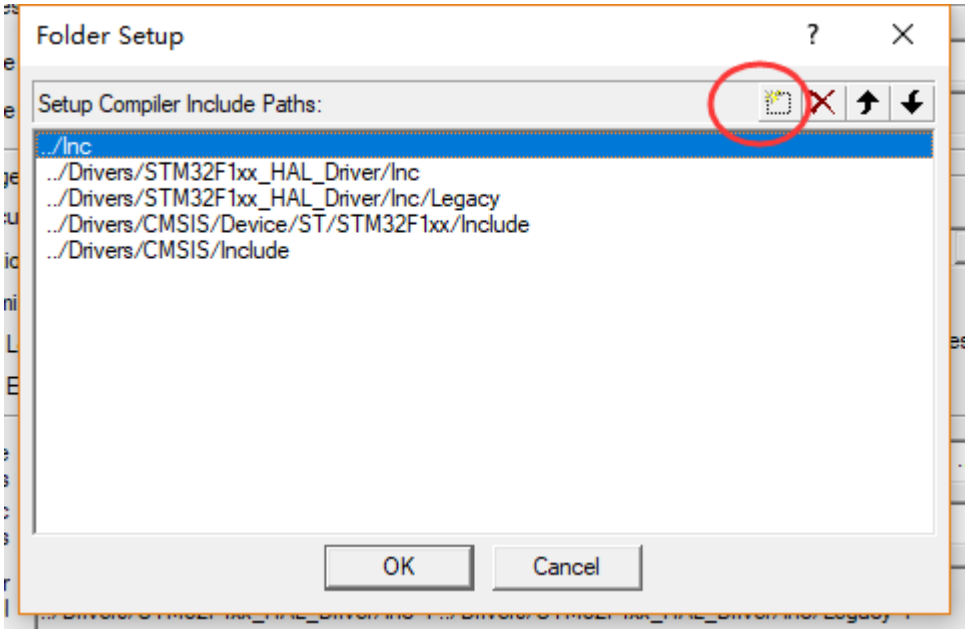

Add Files to Group 'FSC_STOS'
查找范围(I):
FSC_STOS
名称
修改日期
类型
APP.c
C 文件
FSC_STOS.asm
ASM Source File
FSC_STOS.c
C 文件
FSC_STOS.h
H 文件
文件名(N):
"APP.c" "FSC_STOS.asm" "FSC_STOS.c" "FSC_STOS.h"
Add
文件类型(T):
All files (*.*)
Close

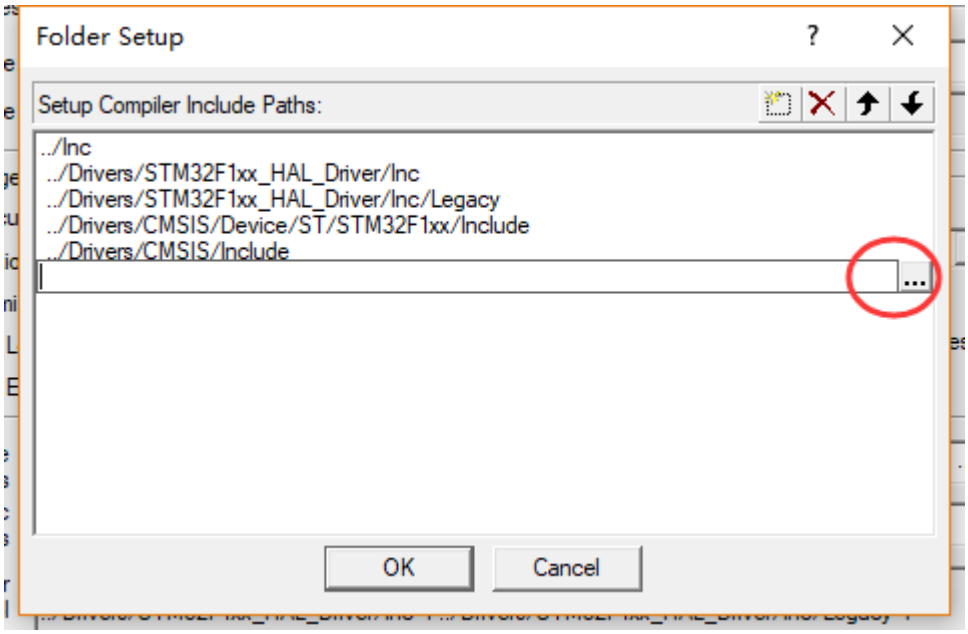
<4>Target 右键选择 Options for Target ->C/C++ 页面下 Include Paths 加入 FSC_STOS 文件路径。
或点击魔法棒图标打开。

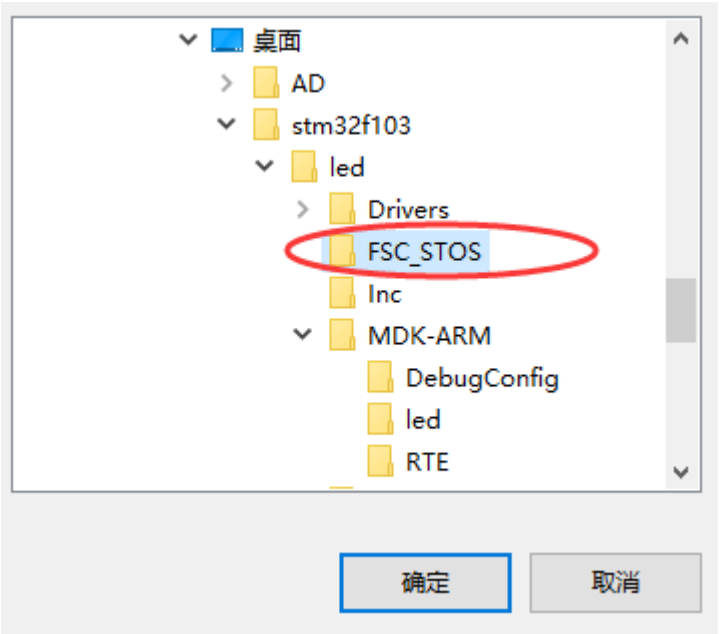


此步骤中的 Level 设置非常关键，这里要设为 0，是为了防止它把系统指针优化掉，下版本更新会优化代码，加入防止优化造成的错误，让大家可以使用更高 level 优化等级服务。




Folder Setup
Setup Compiler Include Paths:
../Inc
../Drivers/STM32F1xx_HAL_Driver/Inc
../Drivers/STM32F1xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32F1xx/Include
../Drivers/CMSIS/Include
OK
Cancel


Folder Setup
Setup Compiler Include Paths:
../Inc
../Drivers/STM32F1xx_HAL_Driver/Inc
../Drivers/STM32F1xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32F1xx/Include
../Drivers/CMSIS/Include
OK
Cancel

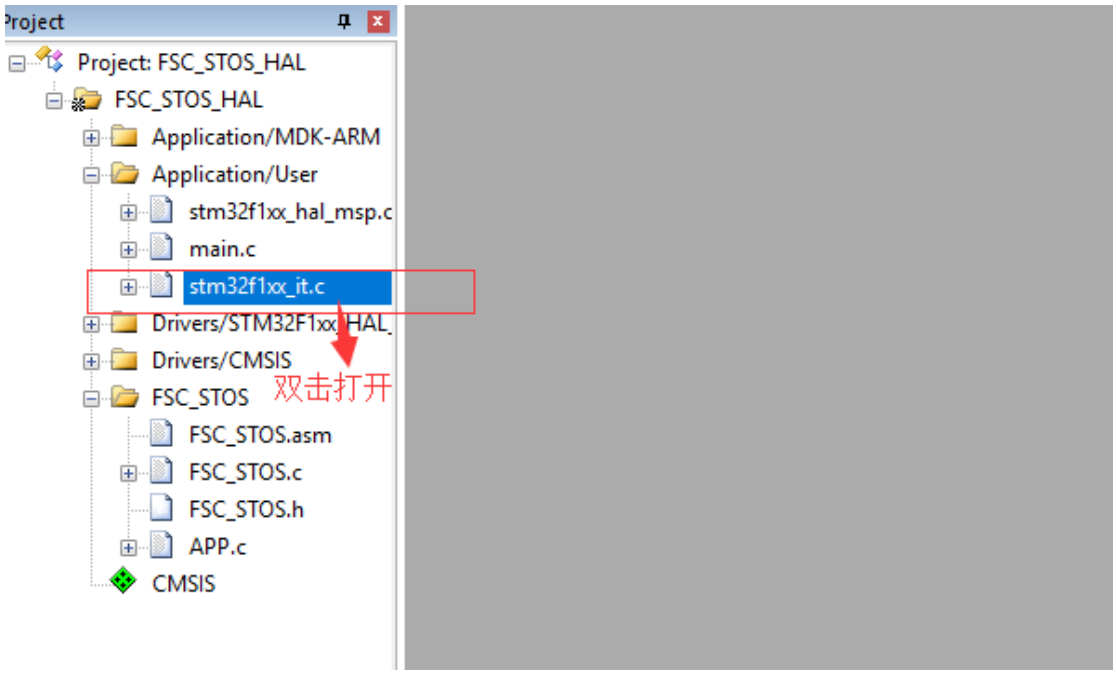


<5>在原 hal 工程 stm32f10x_it.c / stm32f40x_it.c 中注释掉

void PendSV_Handler(void){ }

void SysTick_Handler(void){ }

两个函数，并复制 SysTick_Handler 中的内容到 fsc_stos.c 中的 SysTick_Handler 函数里。(FSC_STOS 中用到这两个函数，所以用户工程中要注释掉，不然会产生冲突)



stm32f1xx_it.c

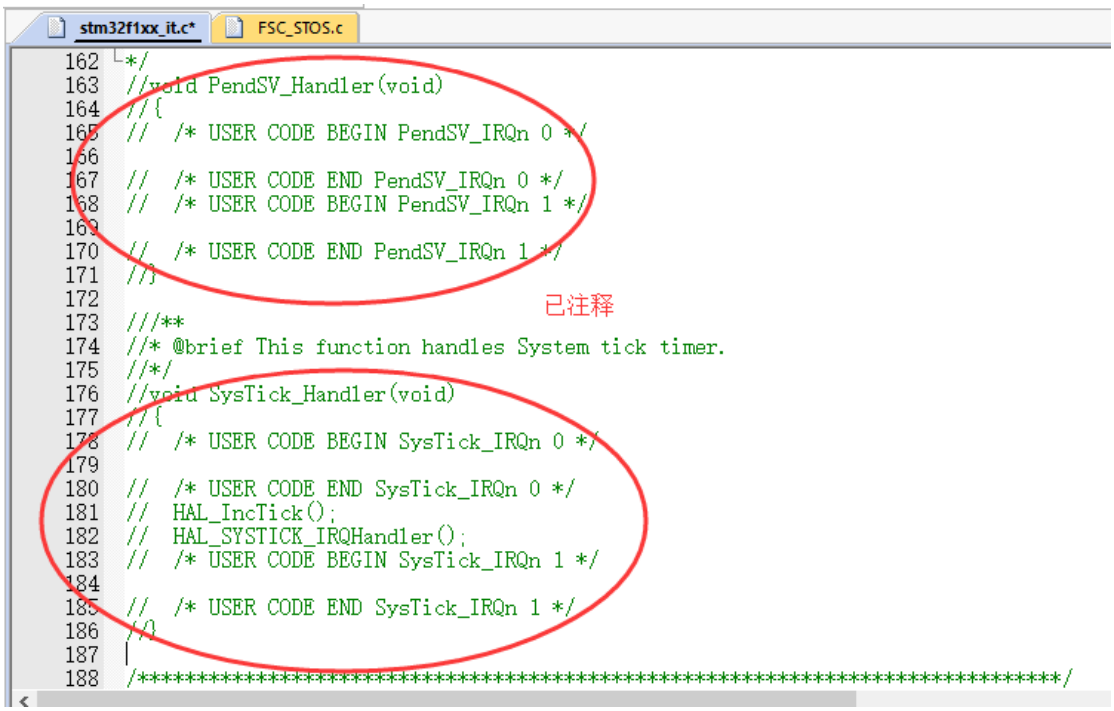
```
160 /**
161   * @brief This function handles Pendable request for system service.
162   */
163 void PendSV_Handler(void)
164 {
165   /* USER CODE BEGIN PendSV_IRQn 0 */
166
167   /* USER CODE END PendSV_IRQn 0 */
168   /* USER CODE BEGIN PendSV_IRQn 1 */
169
170   /* USER CODE END PendSV_IRQn 1 */
171 }
172
173 /**
174   * @brief This function handles System tick timer.
175   */
176 void SysTick_Handler(void)
177 {
178   /* USER CODE BEGIN SysTick_IRQn 0 */
179
180   /* USER CODE END SysTick_IRQn 0 */
181   HAL_IncTick();
182   HAL_SYSTICK_IRQHandler();
183   /* USER CODE BEGIN SysTick_IRQn 1 */
184
185   /* USER CODE END SysTick_IRQn 1 */
186 }
```

注释掉

复制到fsc_stos.c中的systick中断函数里

main.c* | FSC_STOS.h | stm32f1xx_it.c | FSC_STOS.c*

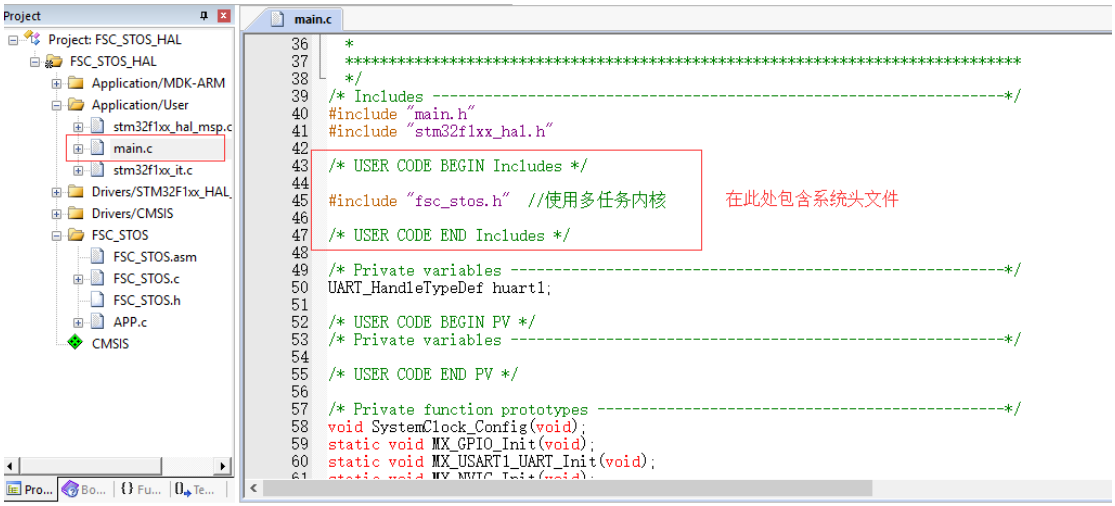
```
32 {
33   OS_INT_OFF();
34
35
36   //Systick定时器初使化(使用其他定时器时，请修改为其他定时器)
37   char * Systick_priority = (char *)0xe000ed23;       //Systick中断优先级寄存器
38   SysTick->LOAD  = OSTIMER_CONT_1us* Nus;             //Systick定时器重装载计数值
39   *Systick_priority = 0x00;                           //Systick定时器中断优先级
40   SysTick->VAL   = 0;                                 //Systick定时器计数器清0
41   SysTick->CTRL = 0x3;                                //Systick使用外部晶振时钟，8分频  72MHz/8=9MHz  计数9000次
42
43   OS_INT_ON();
44 }
45 void SysTick_Handler(void) //Systick定时器中断函数(使用其他定时器时,请修改为其他定时器的中断函数名)
46 {
47   /*----------从stm32f1xx_it.c中的void SysTick_Handler(void)复制过来----------*/
48   HAL_IncTick();
49   HAL_SYSTICK_IRQHandler();
50   /*----------------------------------------------------------------------*/
51
52   OS_Timer_Handler();
53 }
54 /*-------------------------------------------------------------------------------*/
55 /*********************************************************************************************************/
56
57
```



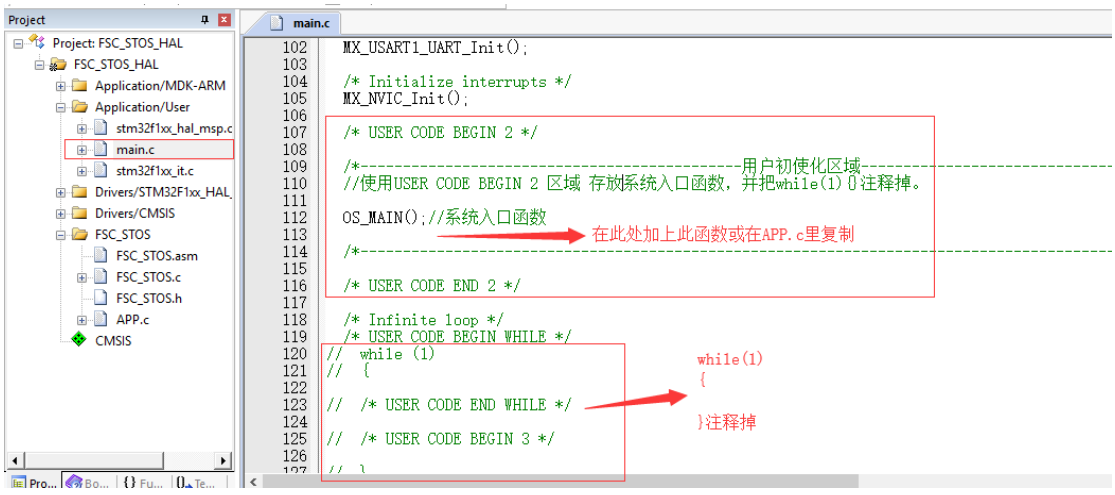

```
162 */
163 //void PendSV_Handler(void)
164 //{
165 //  /* USER CODE BEGIN PendSV_IRQn 0 */
166
167 //  /* USER CODE END PendSV_IRQn 0 */
168 //  /* USER CODE BEGIN PendSV_IRQn 1 */
169
170 //  /* USER CODE END PendSV_IRQn 1 */
171 //}
172
173 ///**
174 //* @brief This function handles System tick timer.
175 //*/
176 //void SysTick_Handler(void)
177 //{
178 //  /* USER CODE BEGIN SysTick_IRQn 0 */
179
180 //  /* USER CODE END SysTick_IRQn 0 */
181 //  HAL_IncTick();
182 //  HAL_SYSTICK_IRQHandler();
183 //  /* USER CODE BEGIN SysTick_IRQn 1 */
184
185 //  /* USER CODE END SysTick_IRQn 1 */
186 //}
187
188 /*************************************************************************************/
```

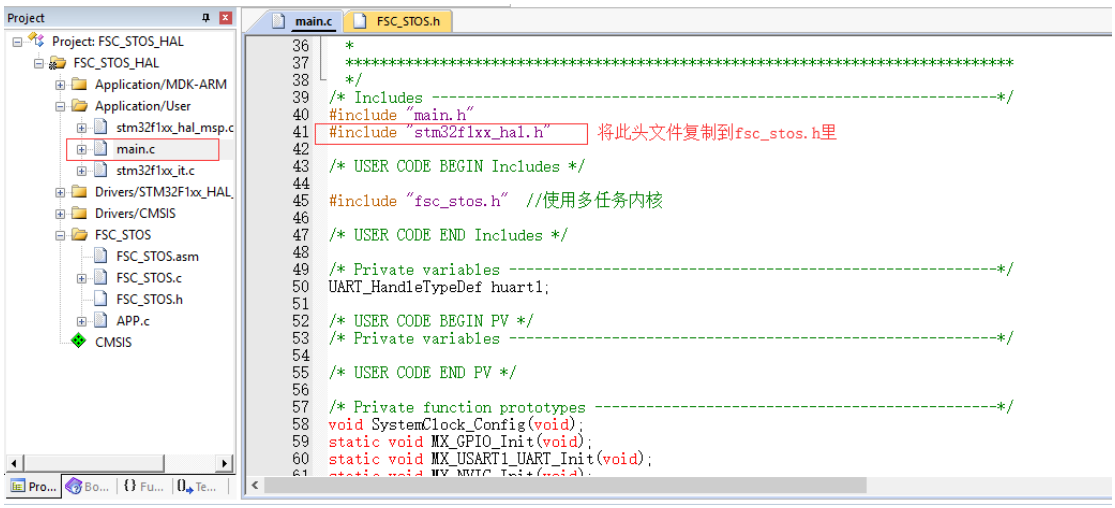
<6>把 OS_MAIN()复制到 main 函数中：

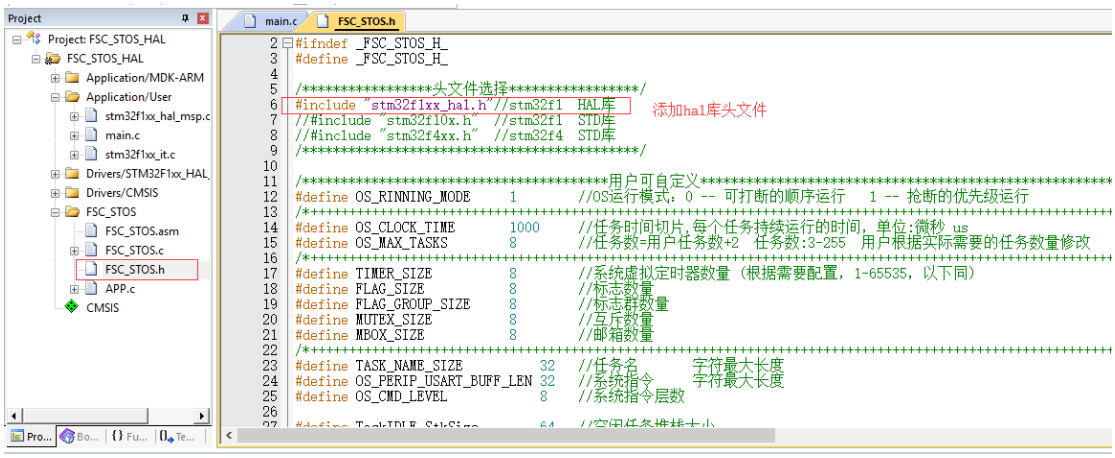
1>注释或删除 hal 工程中 main()中的 while(1){　}，把 APP.c 中的 OS_MAIN()函数复制过去。




```
36   *
37   ******************************************************************************
38   */
39 /* Includes ------------------------------------------------------------------*/
40 #include "main.h"
41 #include "stm32f1xx_hal.h"
42
43 /* USER CODE BEGIN Includes */
44
45 #include "fsc_stos.h"  //使用多任务内核
46
47 /* USER CODE END Includes */
48
49 /* Private variables ---------------------------------------------------------*/
50 UART_HandleTypeDef huart1;
51
52 /* USER CODE BEGIN PV */
53 /* Private variables ---------------------------------------------------------*/
54
55 /* USER CODE END PV */
56
57 /* Private function prototypes -----------------------------------------------*/
58 void SystemClock_Config(void);
59 static void MX_GPIO_Init(void);
60 static void MX_USART1_UART_Init(void);
```







此时基本移植好，但是还要做一些重要修改。（对 systick 和 PendSV 的初使化修改）

main.c FSC_STOS.h

```
161 {
162   /* USER CODE BEGIN 1 */
163
164   /* USER CODE END 1 */
165
166   /* MCU Configuration--------------------------------------------------------*/
167
168   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
169   HAL_Init();
170
171   /* USER CODE BEGIN Init */
172
173   /* USER CODE END Init */
174
175   /* Configure the system clock */
176   SystemClock_Config();
177
178   /* USER CODE BEGIN SysInit */
179
180   /* USER CODE END SysInit */
181
182   /* Initialize all configured peripherals */
183   MX_GPIO_Init();
184   MX_USART1_UART_Init();
185
186   /* Initialize interrupts */
187   MX_NVIC_Init();
188
```

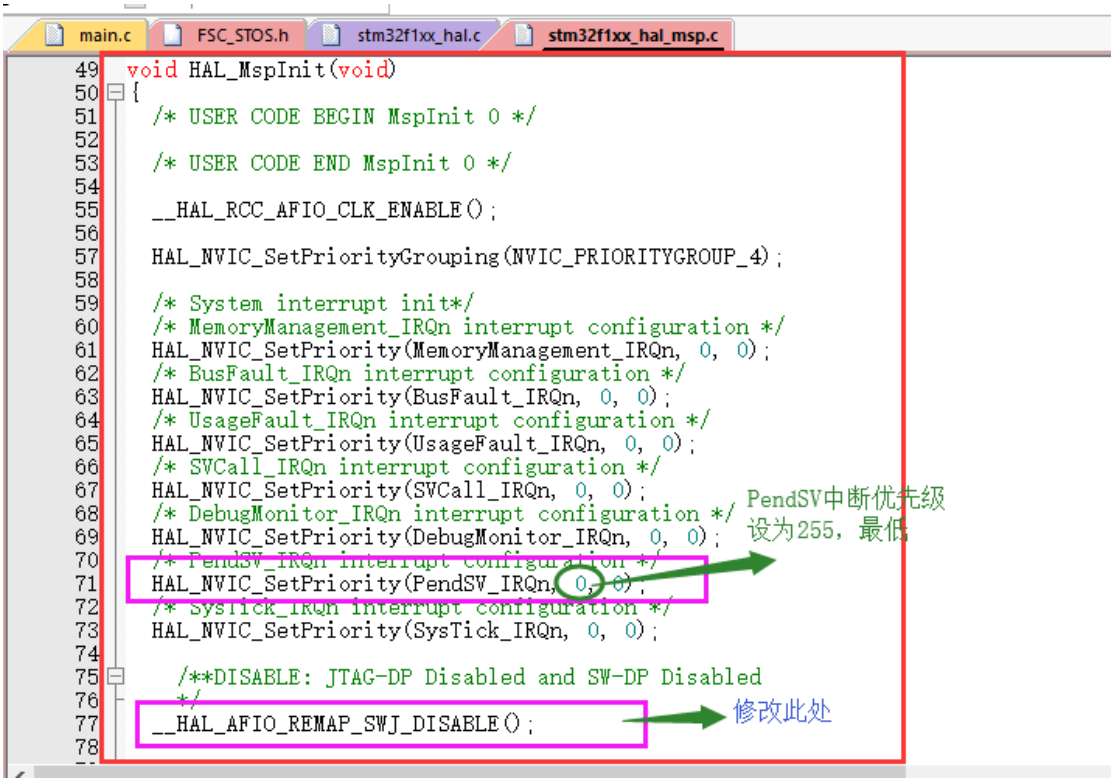
双击选中右键go to definition

main.c FSC_STOS.h stm32f1xx_hal.c

```
155   *         to have correct HAL operation.
156   * @retval HAL status
157   */
158 HAL_StatusTypeDef HAL_Init(void)
159 {
160   /* Configure Flash prefetch */
161 #if (PREFETCH_ENABLE != 0)
162 #if defined(STM32F101x6) || defined(STM32F101xB) || defined(STM32F101xE) || defined(STM32F101xG) || \
163     defined(STM32F102x6) || defined(STM32F102xB) || \
164     defined(STM32F103x6) || defined(STM32F103xB) || defined(STM32F103xE) || defined(STM32F103xG) || \
165     defined(STM32F105xC) || defined(STM32F107xC)
166
167   /* Prefetch buffer is not available on value line devices */
168   __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
169 #endif
170 #endif /* PREFETCH_ENABLE */
171
172   /* Set Interrupt Group Priority */
173   HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
174
175   /* Use systick as time base source and configure 1ms tick (default clock after Reset is HSI) */
176   HAL_InitTick(TICK_INT_PRIORITY);
177
178   /* Init the low level hardware */
179   HAL_MspInit();
180
181   /* Return function status */
182   return HAL_OK;
```

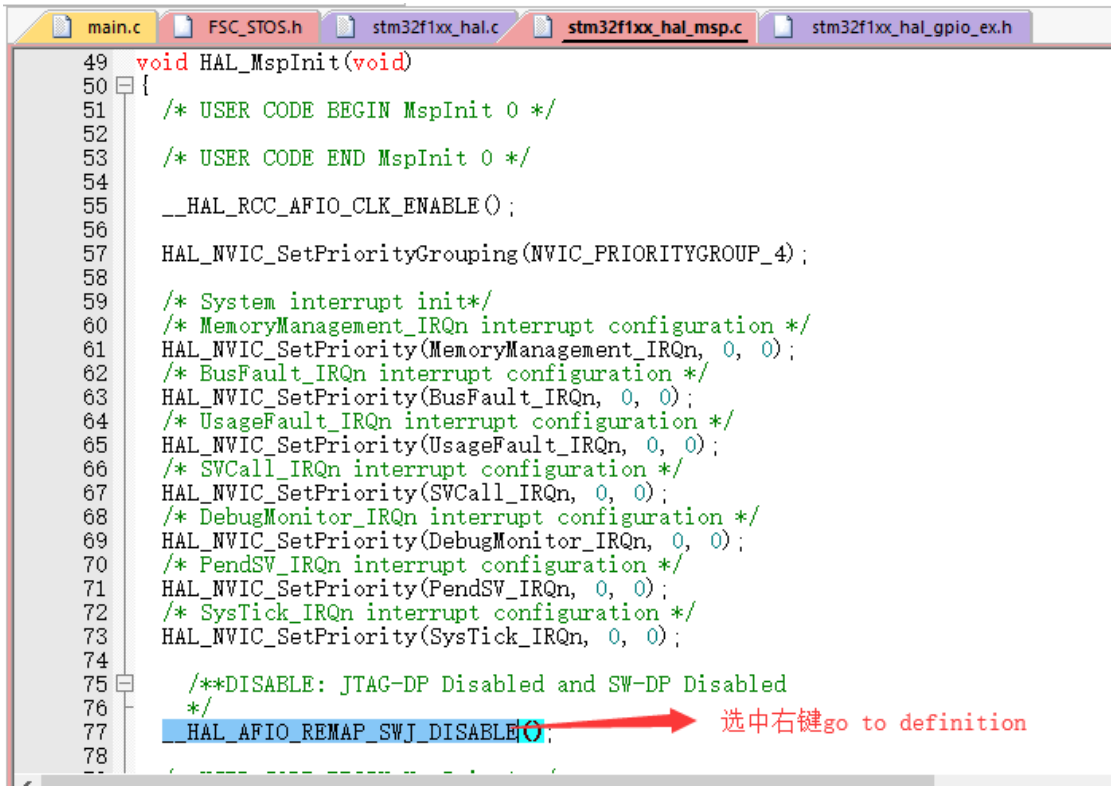
选中右键go to definiton

PendSV 中断优先级设置这句注释掉或改成 255。

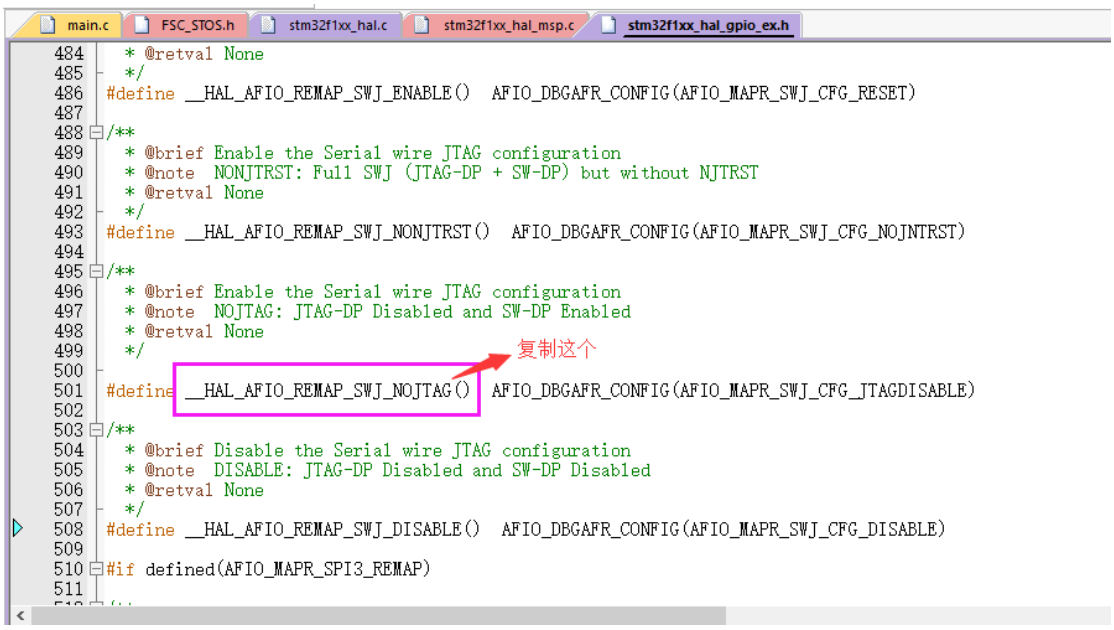



```
void HAL_MspInit(void)
{
  /* USER CODE BEGIN MspInit 0 */

  /* USER CODE END MspInit 0 */

  __HAL_RCC_AFIO_CLK_ENABLE();

  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* System interrupt init*/
  /* MemoryManagement_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(MemoryManagement_IRQn, 0, 0);
  /* BusFault_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(BusFault_IRQn, 0, 0);
  /* UsageFault_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(UsageFault_IRQn, 0, 0);
  /* SVCall_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SVCall_IRQn, 0, 0);
  /* DebugMonitor_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DebugMonitor_IRQn, 0, 0);
  /* PendSV_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(PendSV_IRQn, 0, 0);
  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);

    /**DISABLE: JTAG-DP Disabled and SW-DP Disabled
    */
  __HAL_AFIO_REMAP_SWJ_DISABLE();
```

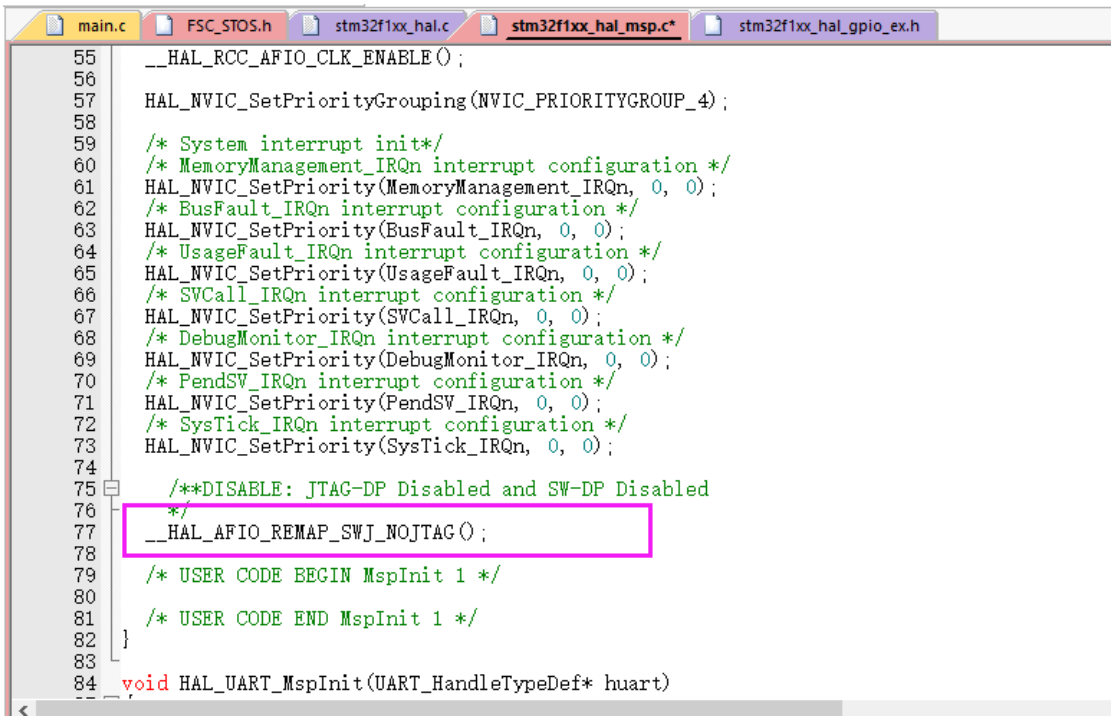
修改此处是为了打开 SWD 烧录，在没有设置 SYS 选项下，cubemx 默认成生的工程是关闭 SWD 下载功能的（是个小坑）。如果没有修改此处，第一次能烧录进去，下次就不能烧录，如果不小心中招了，恢复方法如下：按下面图示方法修改好，然后按住开发板上的复位按键，点 load 下载，点击下载看到烧录进度条后要立即松开按键，等程序强制下载进去就恢复了。




```
void HAL_MspInit(void)
{
  /* USER CODE BEGIN MspInit 0 */

  /* USER CODE END MspInit 0 */

  __HAL_RCC_AFIO_CLK_ENABLE();

  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* System interrupt init*/
  /* MemoryManagement_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(MemoryManagement_IRQn, 0, 0);
  /* BusFault_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(BusFault_IRQn, 0, 0);
  /* UsageFault_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(UsageFault_IRQn, 0, 0);
  /* SVCall_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SVCall_IRQn, 0, 0);
  /* DebugMonitor_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DebugMonitor_IRQn, 0, 0);
  /* PendSV_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(PendSV_IRQn, 0, 0);
  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);

    /**DISABLE: JTAG-DP Disabled and SW-DP Disabled
    */
  __HAL_AFIO_REMAP_SWJ_DISABLE();
```



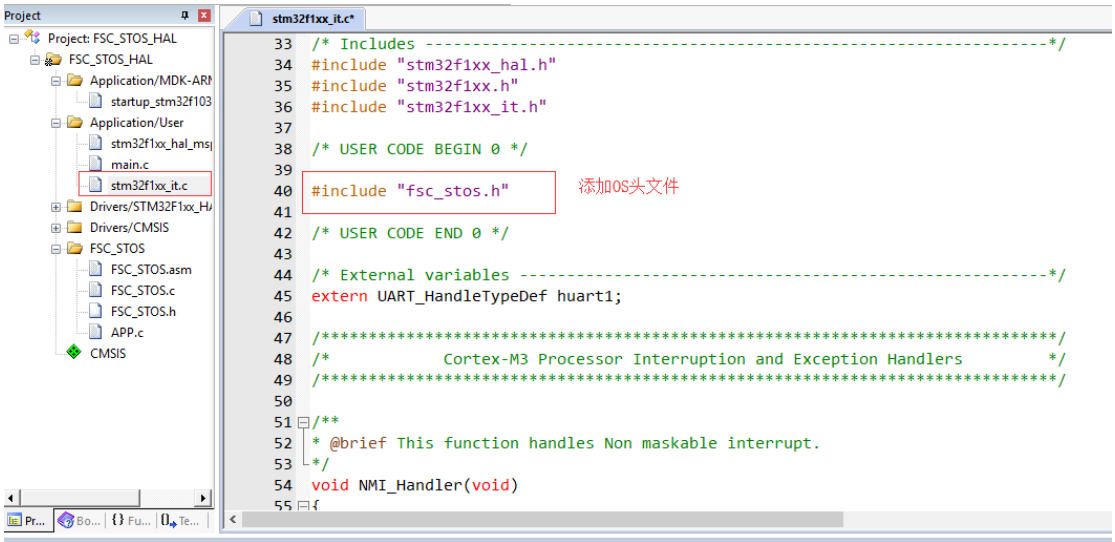
修改后如下：

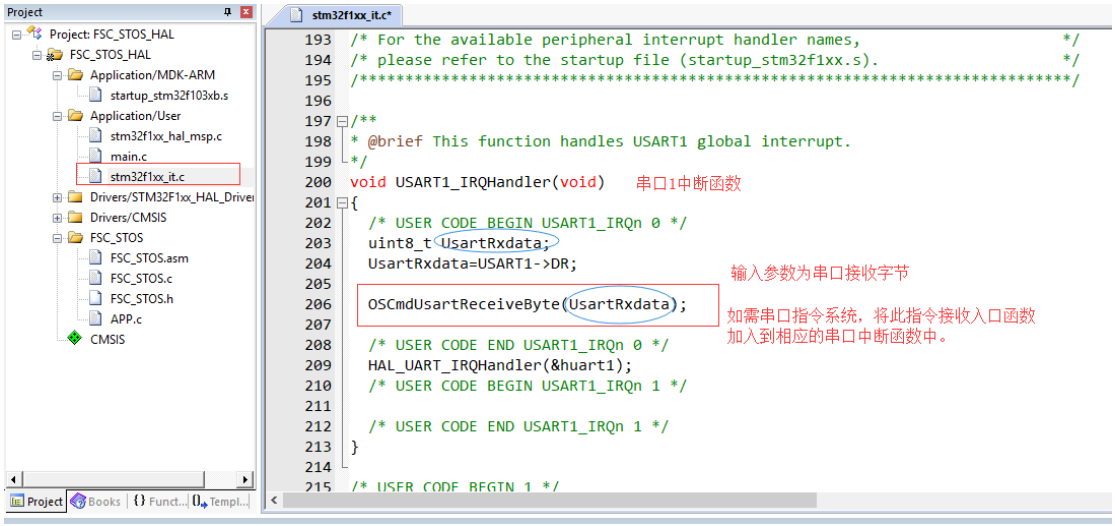


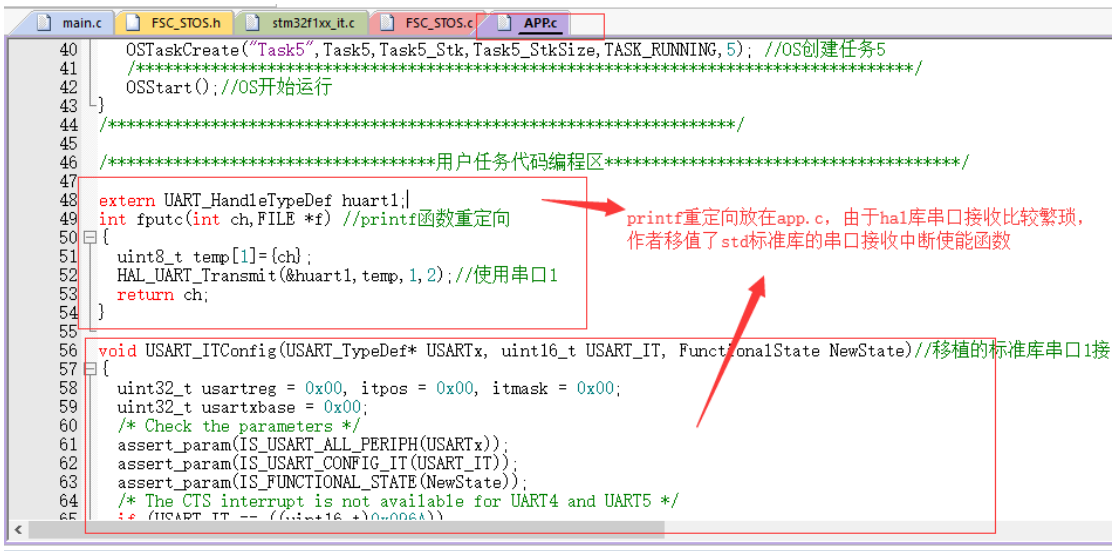
到此系统就可以正常运行了。由于新建 hal 库工程时，一般默认使用 systick 定时器用作 HAL_Delay()毫秒级延时，其定时时间也刚好为 1ms，而系统默认系统定时器也使用 systick 定时器并且也是 1ms 定时中断，所以正好符合，如果需要修改系统时间切片为其他时间值，请在 hal 初使化中把 systick 初使化有关代码注释掉或在新建工程时把 systick 用作延时的选项去掉。

接下来是 OSprintf 函数和系统串口指令接入到 hal 库串口的设置。

在 stm32fxxx_it.c 中找到串口中断函数(需要哪个串口作为指令就放在哪个串口中断函数)

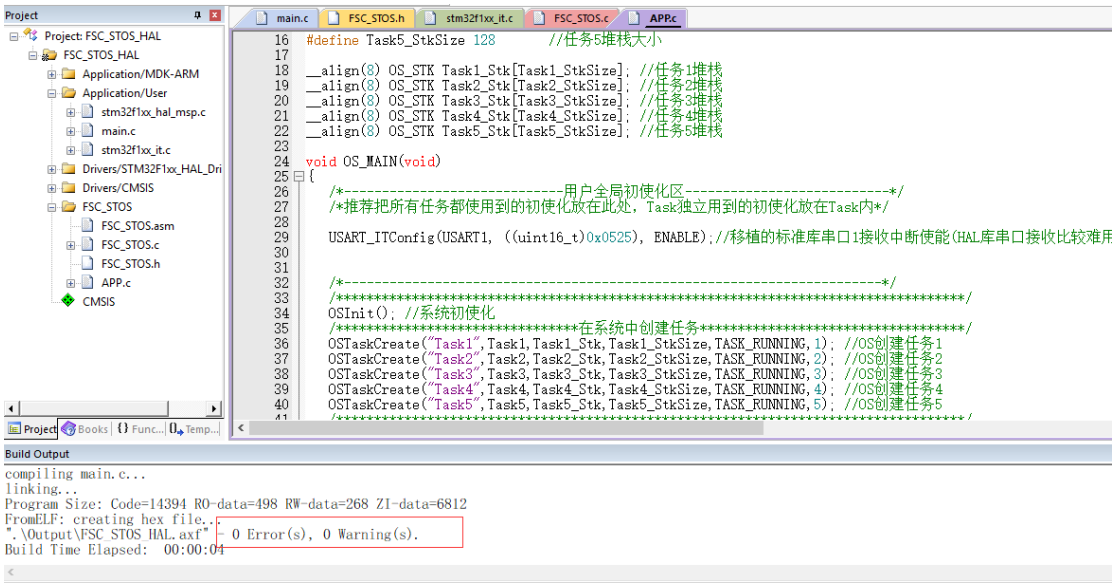






```c
#define Task5_StkSize 128       //任务5堆栈大小

__align(8) OS_STK Task1_Stk[Task1_StkSize]; //任务1堆栈
__align(8) OS_STK Task2_Stk[Task2_StkSize]; //任务2堆栈
__align(8) OS_STK Task3_Stk[Task3_StkSize]; //任务3堆栈
__align(8) OS_STK Task4_Stk[Task4_StkSize]; //任务4堆栈
__align(8) OS_STK Task5_Stk[Task5_StkSize]; //任务5堆栈

void OS_MAIN(void)
{
    /*-----------------------------用户全局初使化区-----------------------------*/
    /*推荐把所有任务都使用到的初使化放在此处，Task独立用到的初使化放在Task内*/

    USART_ITConfig(USART1, ((uint16_t)0x0525), ENABLE);//移植的标准库串口1接收中断使能(HAL库串口接收比较难用)


    /*--------------------------------------------------------------------------*/
    /*****************************************************************************/
    OSInit(); //系统初使化
    /*********************************在系统中创建任务*********************************/
    OSTaskCreate("Task1",Task1,Task1_Stk,Task1_StkSize,TASK_RUNNING,1); //OS创建任务1
    OSTaskCreate("Task2",Task2,Task2_Stk,Task2_StkSize,TASK_RUNNING,2); //OS创建任务2
    OSTaskCreate("Task3",Task3,Task3_Stk,Task3_StkSize,TASK_RUNNING,3); //OS创建任务3
    OSTaskCreate("Task4",Task4,Task4_Stk,Task4_StkSize,TASK_RUNNING,4); //OS创建任务4
    OSTaskCreate("Task5",Task5,Task5_Stk,Task5_StkSize,TASK_RUNNING,5); //OS创建任务5
```

到此移植完成。编译



串口发送 cmd/osmanage//查看系统状态信息(注意波特率要与实际匹配)


XCOM V2.0
++++++++++++++++++++++系统状态信息++++++++++++++++++++++
OS Running Mode: Order
TaskTimeSliceCnt: 4183
CPU占用率: 14.9% CPU最大占用率: 17.9%
利用率 使用栈 空闲栈 百分比 优先级 任务名
CPU Used Free per prio Taskname
80.0% 16 48 25.0% 0 Task_Idle
12.2% 20 140 12.5% 1 Task_Manage
0.2% 18 110 14.1% 1 Task1
3.9% 59 69 46.1% 2 Task2
1.5% 53 75 41.4% 3 Task3
3.1% 55 73 43.0% 4 Task4
0.0% 18 110 14.1% 5 Task5
串口选择
COM5:USB-SERIAL
波特率
9600
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
16进制显示
白底黑字
RTS
DTR
时间戳(以换行回车断帧)
单条发送
多条发送
协议传输
帮助
cmd/osmanage//
发送
清除发送