

# FSC\_STOS CubeMX 工程(Hal 库)移植教程

——望穿秋水

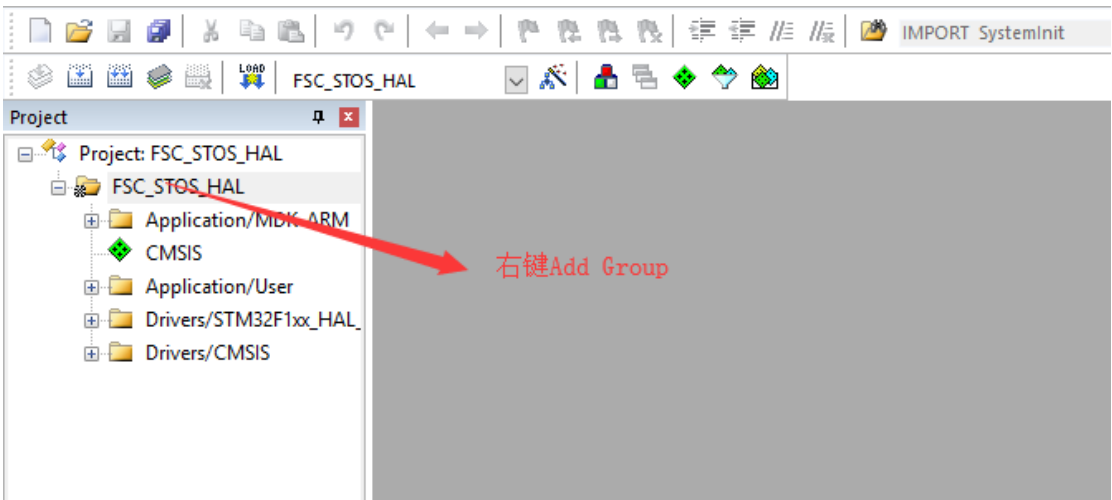
- 1.在裸机工程中加入 FSC\_STOS 内核: (基于 FSC\_STOS\_V3.4 版本)  
以 stm32f103c8t6 hal 库工程加入 FSC\_STOS 为例:  
<1>把 FSC\_STOS 专用移植文件夹复制到裸机工程根目录文件夹中

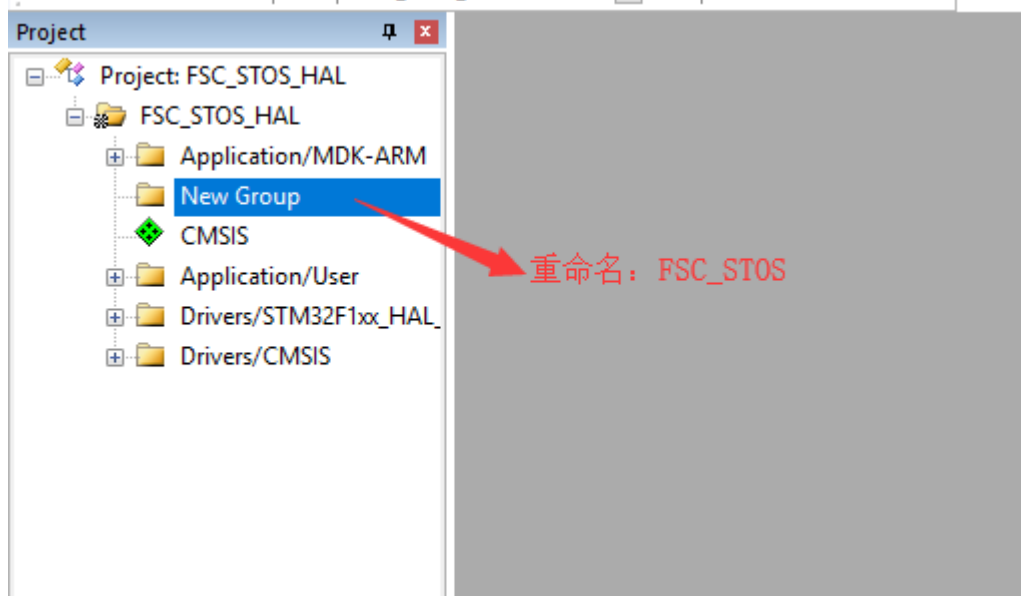
名称	修改日期	类型	大小
Drivers		文件夹	
FSC_STOS		文件夹	
Inc		文件夹	
MDK-ARM		文件夹	
Src		文件夹	

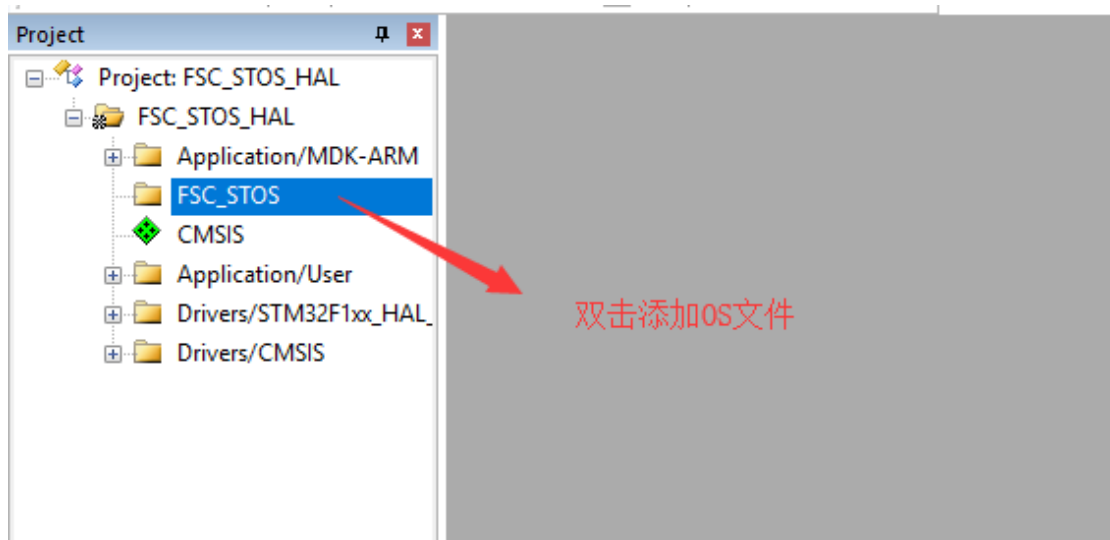
名称
APP.c
FSC_STOS.asm
FSC_STOS.c
FSC_STOS.h

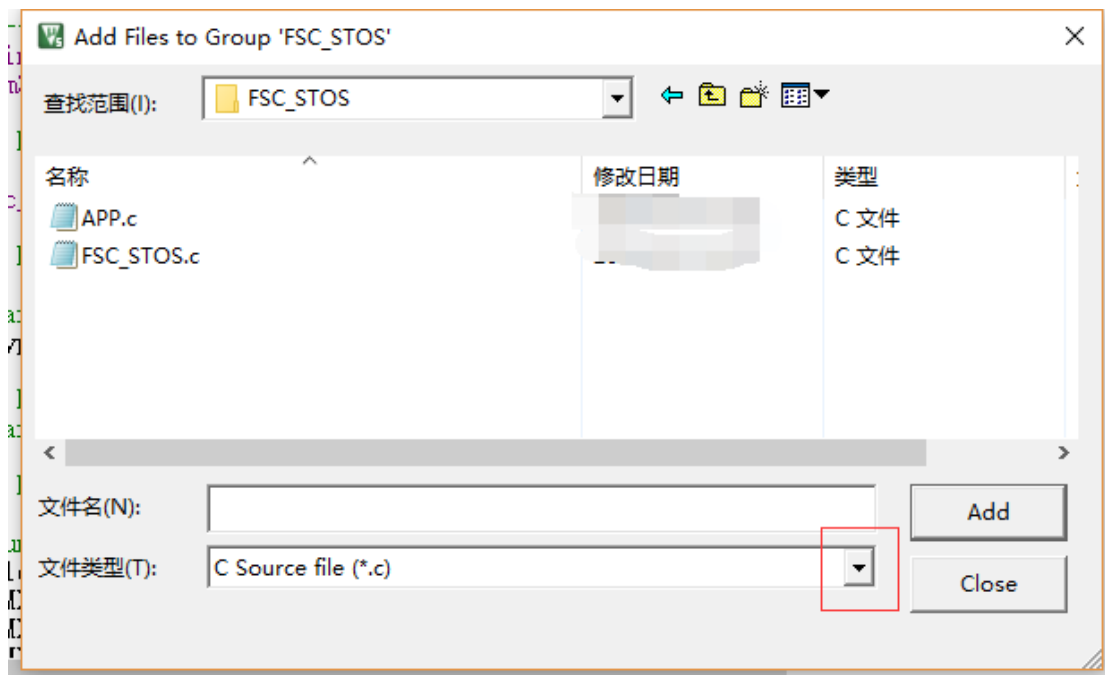
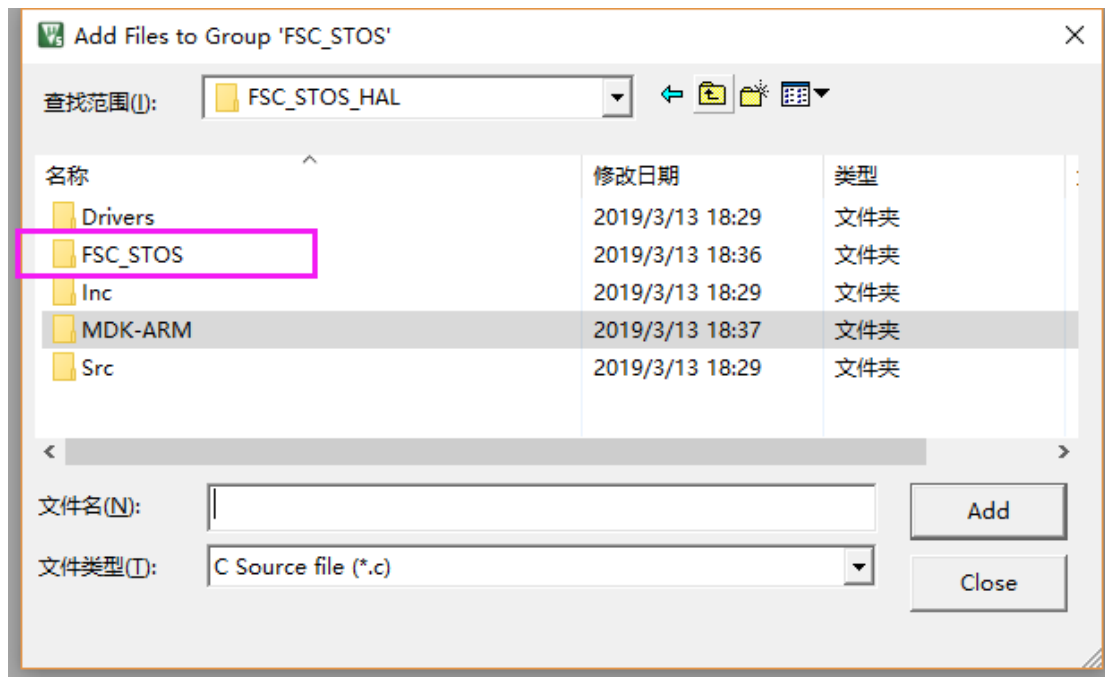
- <2>打开 hal 库 MDK 工程，在工程中新建 FSC\_STOS 文件夹，

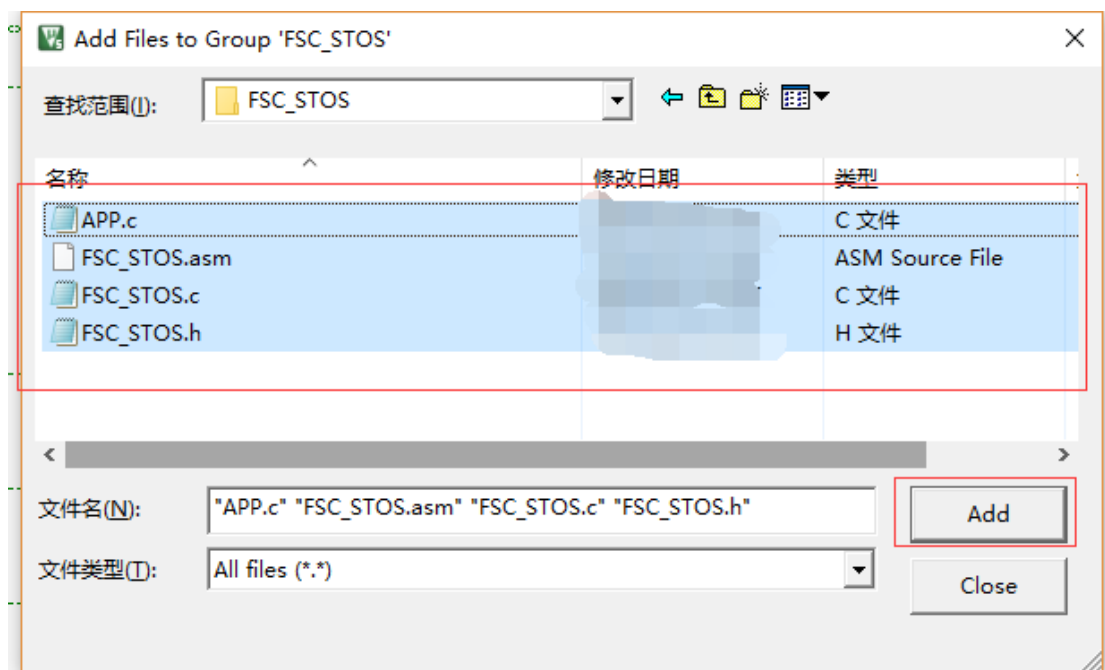
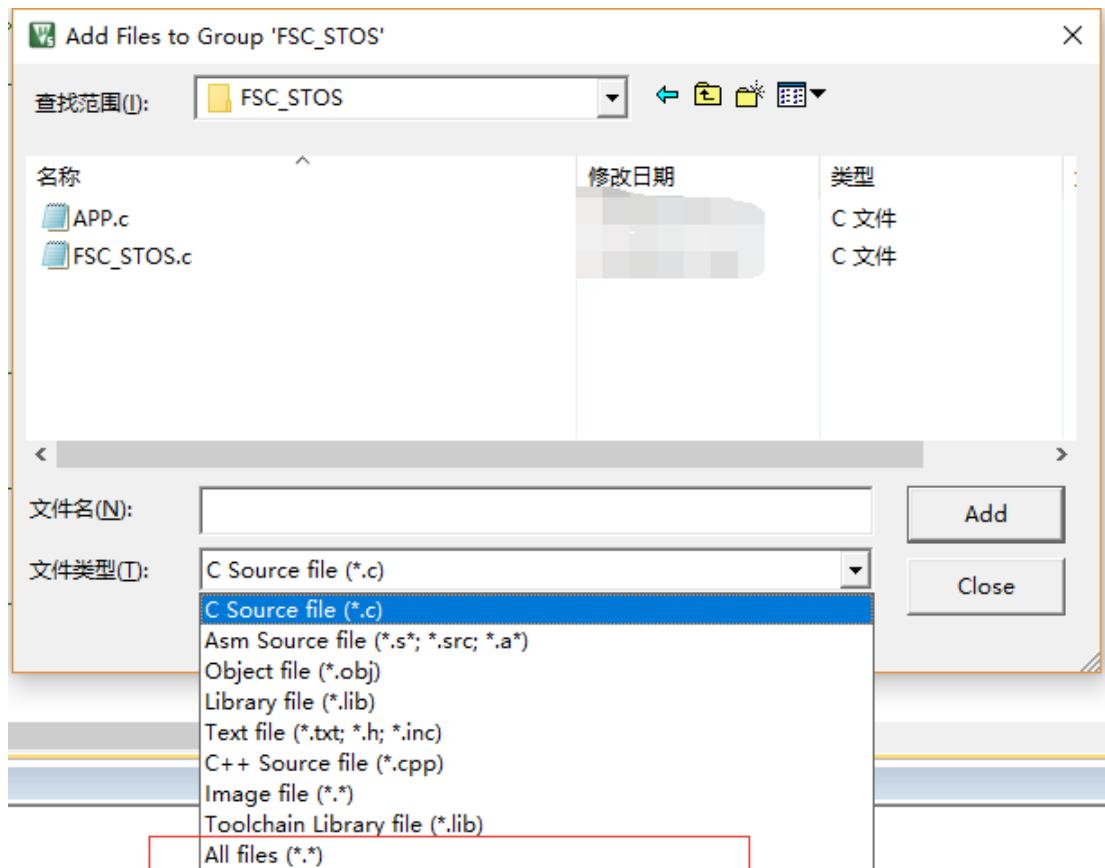




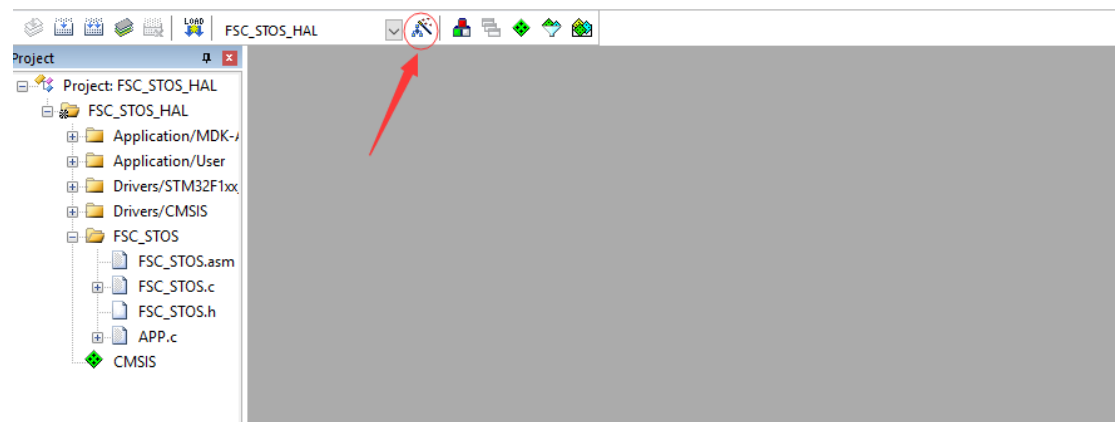
<3>文件夹加入 fsc\_stos.c、fsc\_stos.h、fsc\_stos.asm、main.c 共 4 个文件。



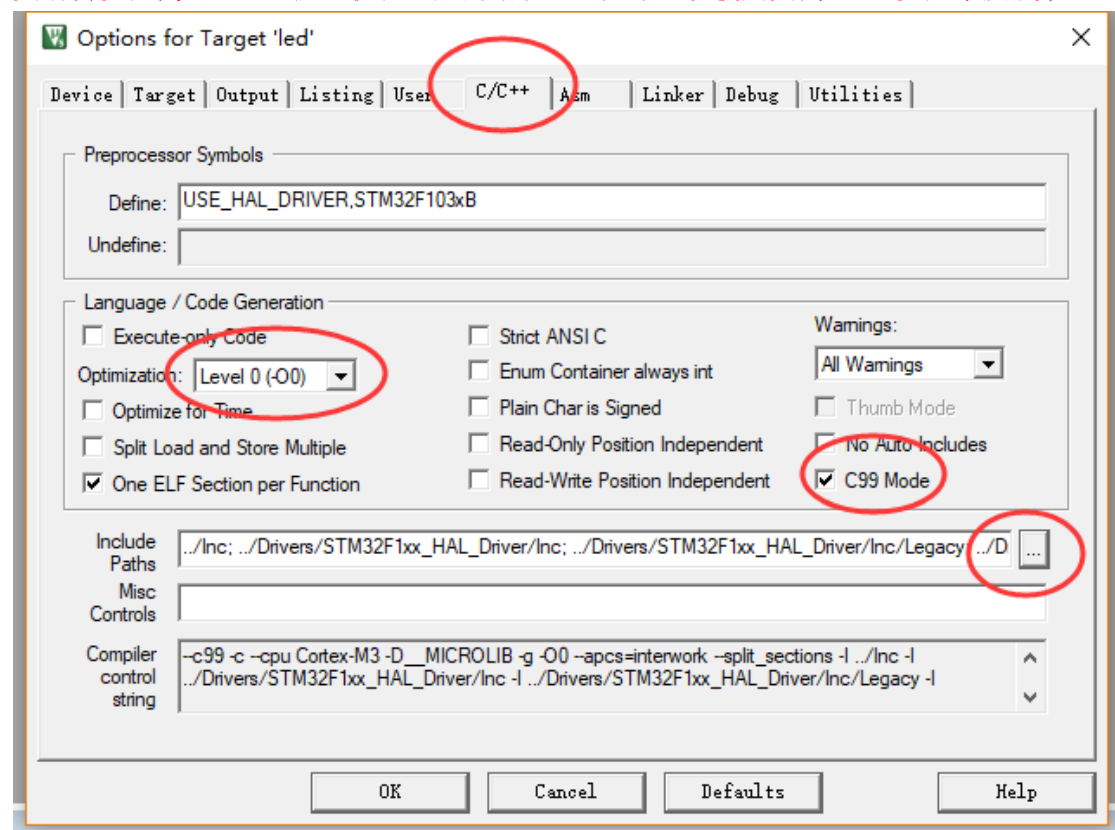


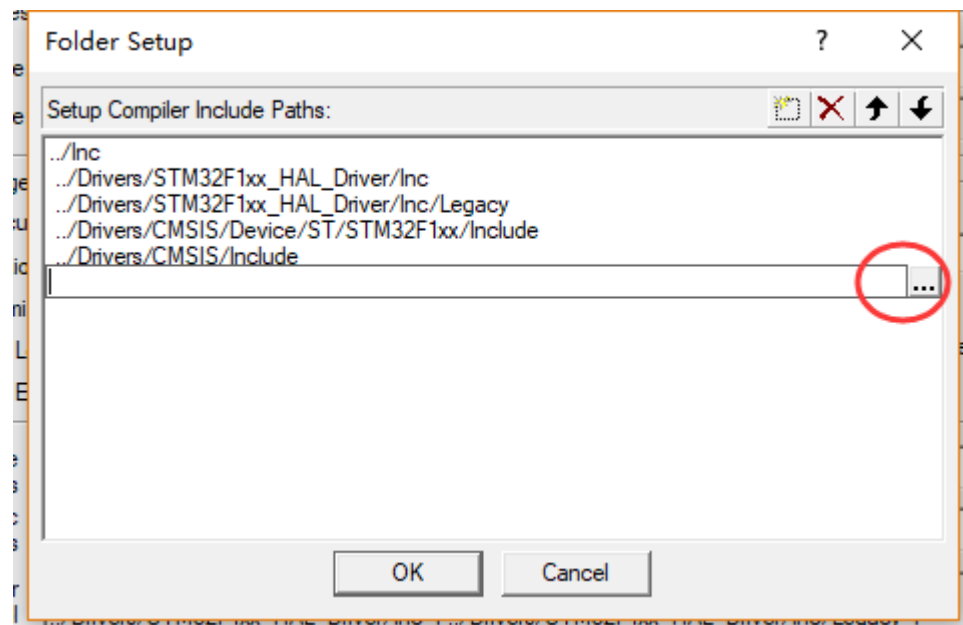
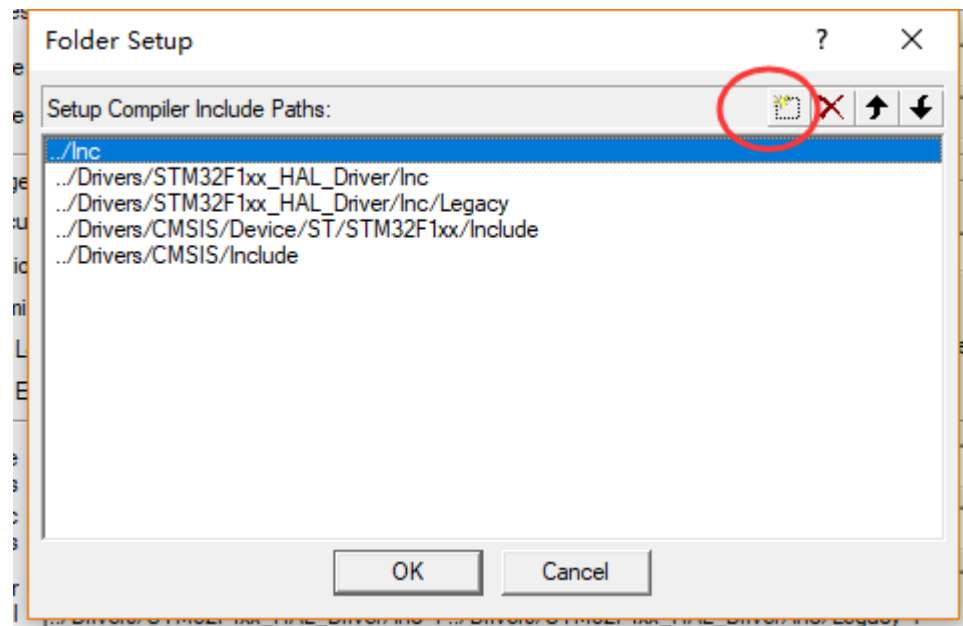


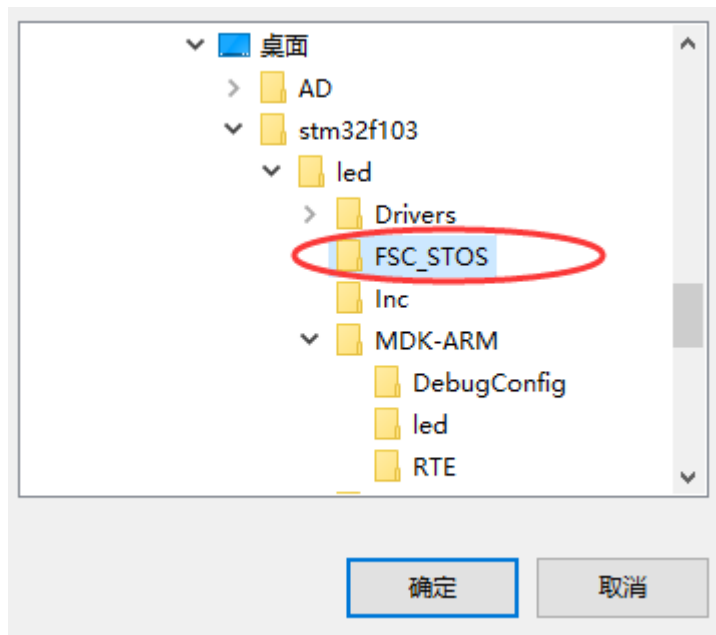
<4>Target 右键选择 Options for Target ->C/C++ 页面下 Include Paths 加入 FSC\_STOS 文件路径。  
或点击魔法棒图标打开。



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。





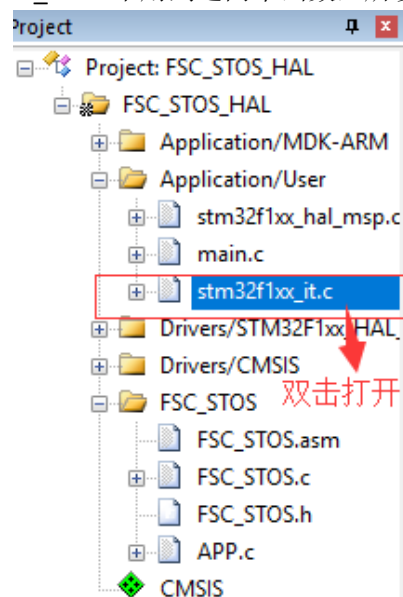


<5>在原 hal 工程 stm32f10x\_it.c / stm32f40x\_it.c 中注释掉

```
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)



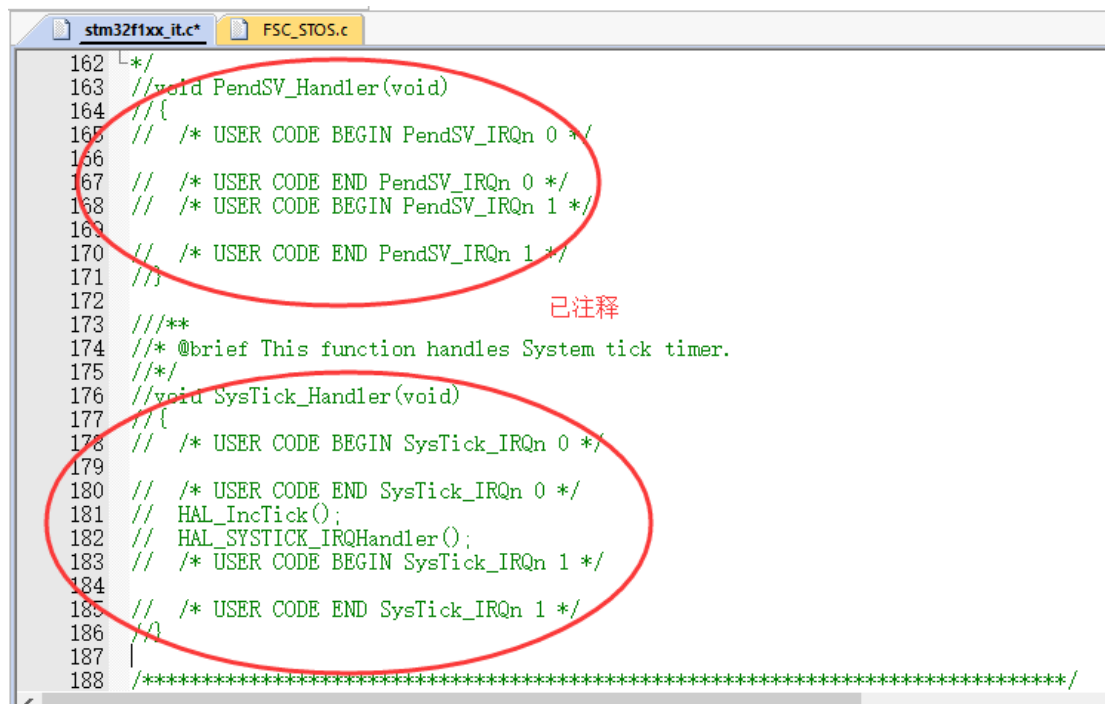
```
stm32f1xx_it.c
160 /**
161  * @brief This function handles Pendable request for system service.
162  */
163 void PendSV_Handler(void)
164 {
165     /* USER CODE BEGIN PendSV_IRQn 0 */
166
167     /* USER CODE END PendSV_IRQn 0 */
168     /* USER CODE BEGIN PendSV_IRQn 1 */
169
170     /* USER CODE 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     //Systick定时器初使化(使用其他定时器时,请修改为其他定时器)
36     char * Systick_priority = (char *)0xe00ed23; //Systick中断优先级寄存器
37     SysTick->LOAD = OSTIMER_CONT_lus* Nus; //Systick定时器重装载计数值
38     *Systick_priority = 0x00; //Systick定时器中断优先级
39     SysTick->VAL = 0; //Systick定时器计数器清0
40     SysTick->CTRL = 0x3; //Systick使用外部晶振时钟, 8分频 72MHz/8=9MHz 计数9000
41
42     OS_INT_ON();
43
44     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)
45     {
46         /*-----从stm32f1xx_it.c中的void SysTick_Handler(void)复制过来-----*/
47         HAL_IncTick();
48         HAL_SYSTICK_IRQHandler();
49         /*-----*/
50
51         OS_Timer_Handler();
52     }
53
54     /*-----*/
55     /*-----*/
56
57 }
```



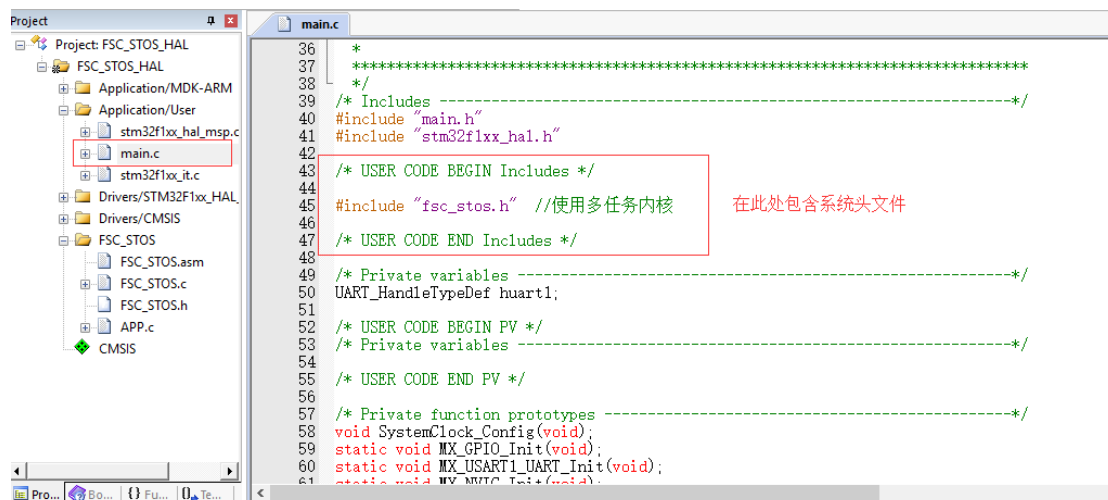


```
162 /*
163 //void PendSV_Handler(void)
164 //{
165 // /* USER CODE BEGIN PendSV_IRQn 0 */
166 // /* USER CODE END PendSV_IRQn 0 */
167 // /* USER CODE BEGIN PendSV_IRQn 1 */
168 // /* USER CODE END PendSV_IRQn 1 */
169 //}
170 //
171 //
172 //
173 /**
174  * @brief This function handles System tick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

已注释

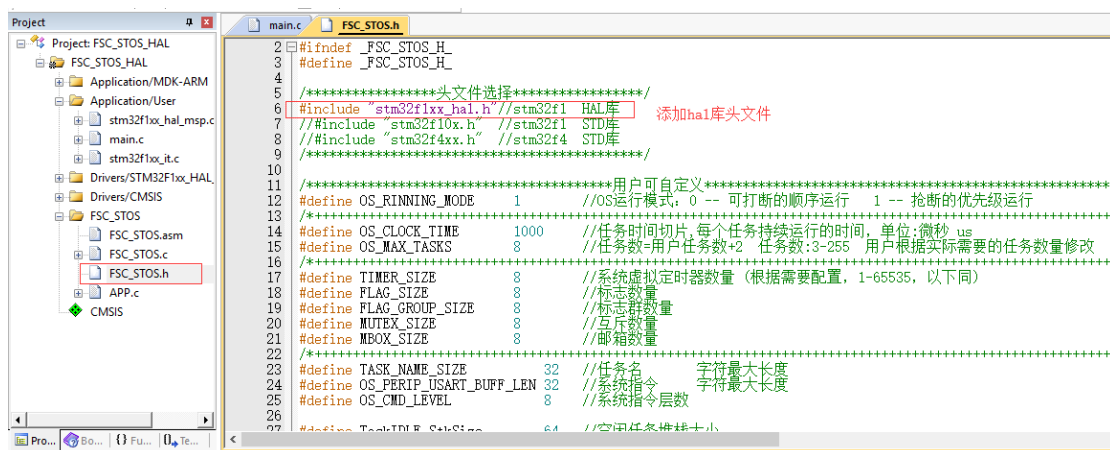
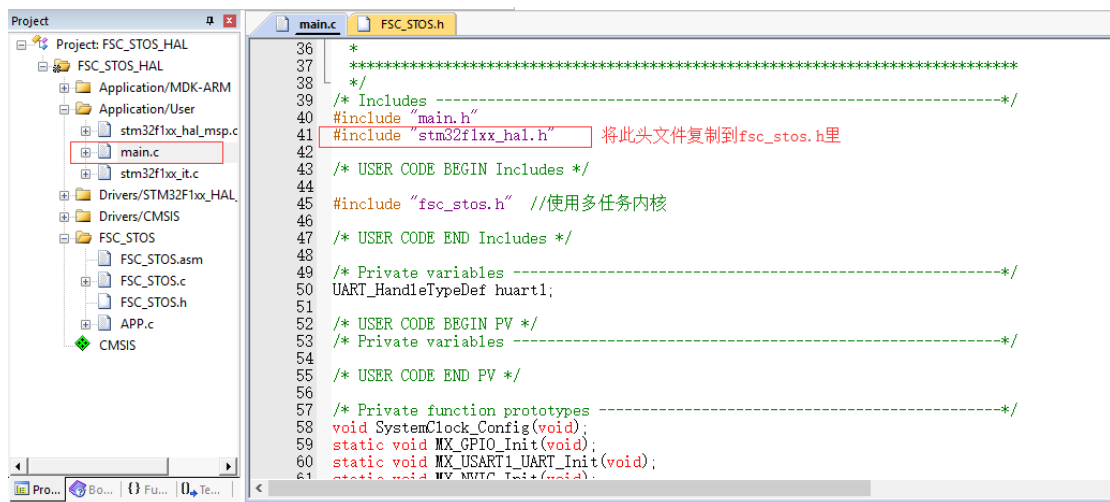
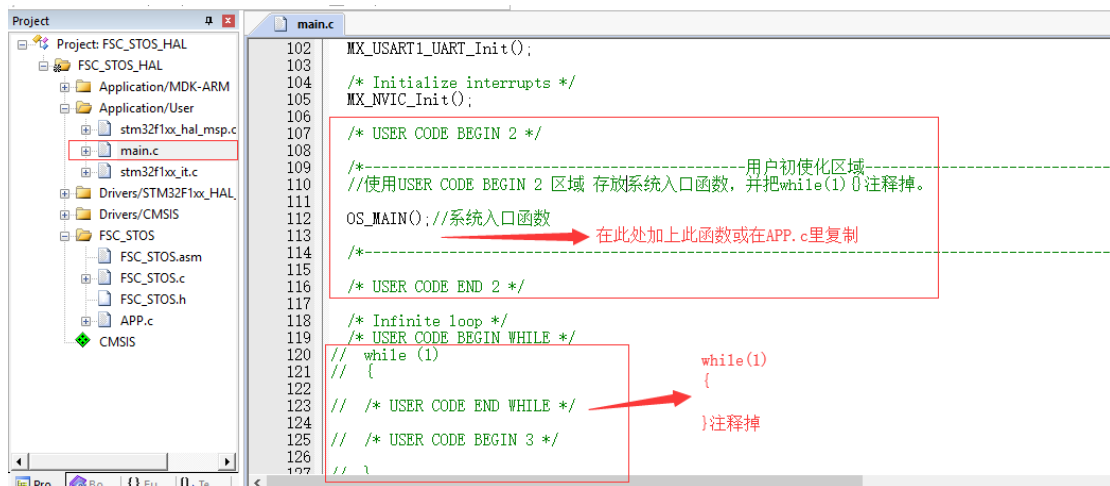
<6>把 OS\_MAIN()复制到 main 函数中:

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

在此处包含系统头文件



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）

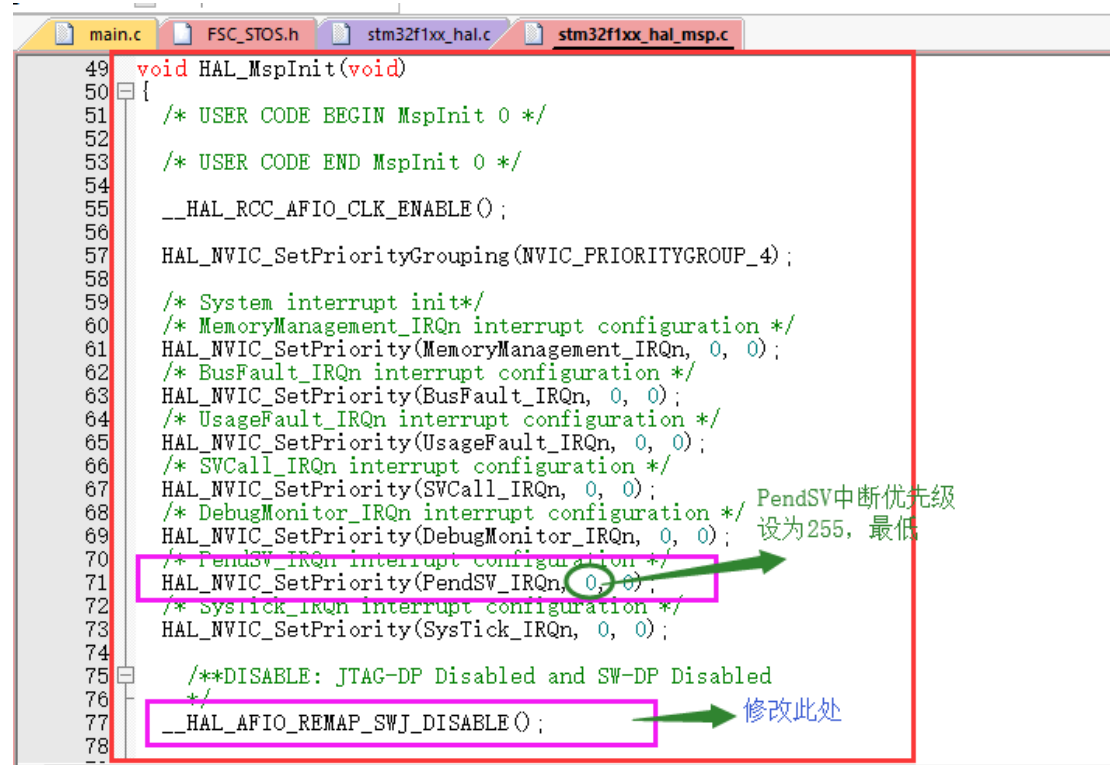
```
main.c FSC_STOS.h
161 {
162     /* USER CODE BEGIN 1 */
163
164     /* USER CODE END 1 */
165
166     /* MCU Configuration-----*/
167
168     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
169     HAL_Init();
170
171     /* USER CODE BEGIN Init */
172     /* USER CODE END Init */
173
174     /* Configure the system clock */
175     SystemClock_Config();
176
177     /* USER CODE BEGIN SysInit */
178
179     /* USER CODE END SysInit */
180
181     /* Initialize all configured peripherals */
182     MX_GPIO_Init();
183     MX_USART1_UART_Init();
184
185     /* Initialize interrupts */
186     MX_NVIC_Init();
187
188     /* USER CODE BEGIN 2 */
189 }
```

双击选中右键go to definition

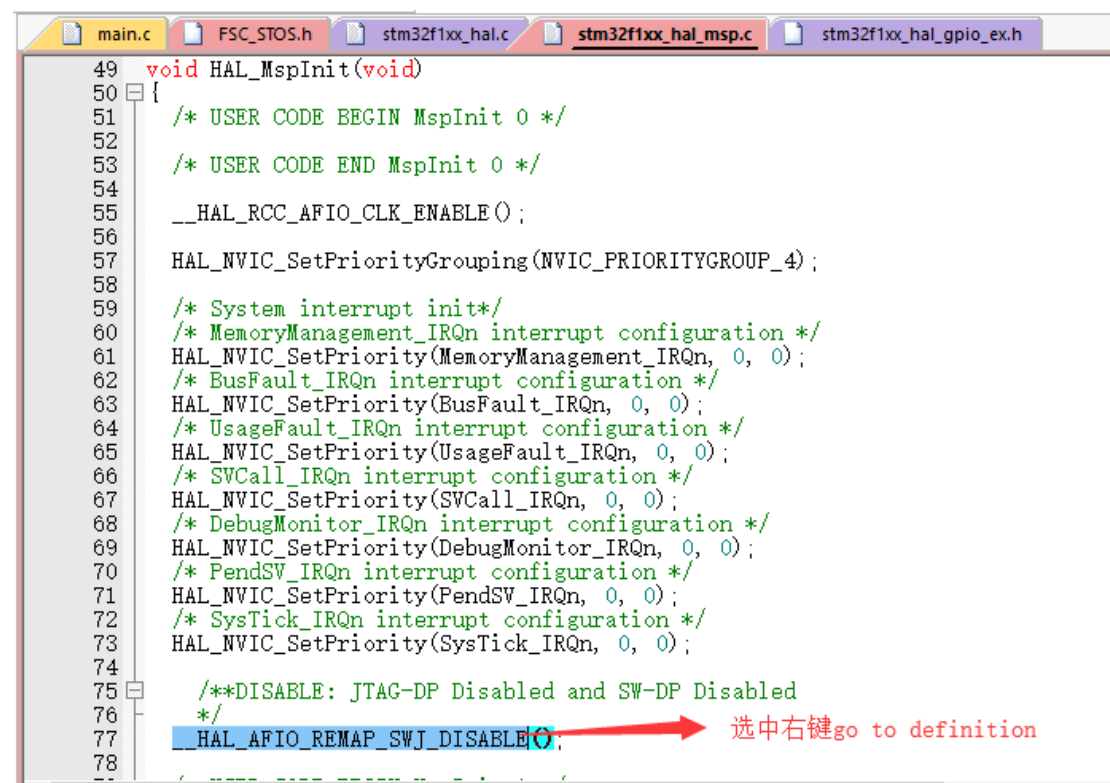
```
main.c FSC_STOS.h stm32f1xx_hal.c
155 * to have correct HAL operation.
156 * @retval HAL status
157 */
158 HAL_StatusTypeDef HAL_Init(void)
159 {
160     /* Configure Flash prefetch */
161     #if (PREFETCH_ENABLE != 0)
162     #if defined(STM32F101x6) || defined(STM32F101xB) || defined(STM32F101xE) || 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;
183 }
```

选中右键go to definition

PendSV 中断优先级设置这句注释掉或改成 255。



修改此处是为了打开 SWD 烧录，在没有设置 SYS 选项下，cubemx 默认生成的工程是关闭 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。



```
main.c FSC_STOS.h stm32f1xx_hal.c stm32f1xx_hal_msp.c stm32f1xx_hal_gpio_ex.h
484  * @retval None
485  */
486  #define __HAL_AFIO_REMAP_SWJ_ENABLE() AFIO_DBGAFR_CONFIG(AFIO_MAPR_SWJ_CFG_RESET)
487
488  /**
489   * @brief Enable the Serial wire JTAG configuration
490   * @note NONJTRST: Full SWJ (JTAG-DP + SW-DP) but without NJTRST
491   * @retval None
492   */
493  #define __HAL_AFIO_REMAP_SWJ_NONJTRST() AFIO_DBGAFR_CONFIG(AFIO_MAPR_SWJ_CFG_NONJTRST)
494
495  /**
496   * @brief Enable the Serial wire JTAG configuration
497   * @note NOJTAG: JTAG-DP Disabled and SW-DP Enabled
498   * @retval None
499   */
500  #define __HAL_AFIO_REMAP_SWJ_NOJTAG() AFIO_DBGAFR_CONFIG(AFIO_MAPR_SWJ_CFG_JTAGDISABLE)
501
502  /**
503   * @brief Disable the Serial wire JTAG configuration
504   * @note DISABLE: JTAG-DP Disabled and SW-DP Disabled
505   * @retval None
506   */
507  #define __HAL_AFIO_REMAP_SWJ_DISABLE() AFIO_DBGAFR_CONFIG(AFIO_MAPR_SWJ_CFG_DISABLE)
508
509  #if defined(AFIO_MAPR_SPI3_REMAP)
510  ...
511  ...
```

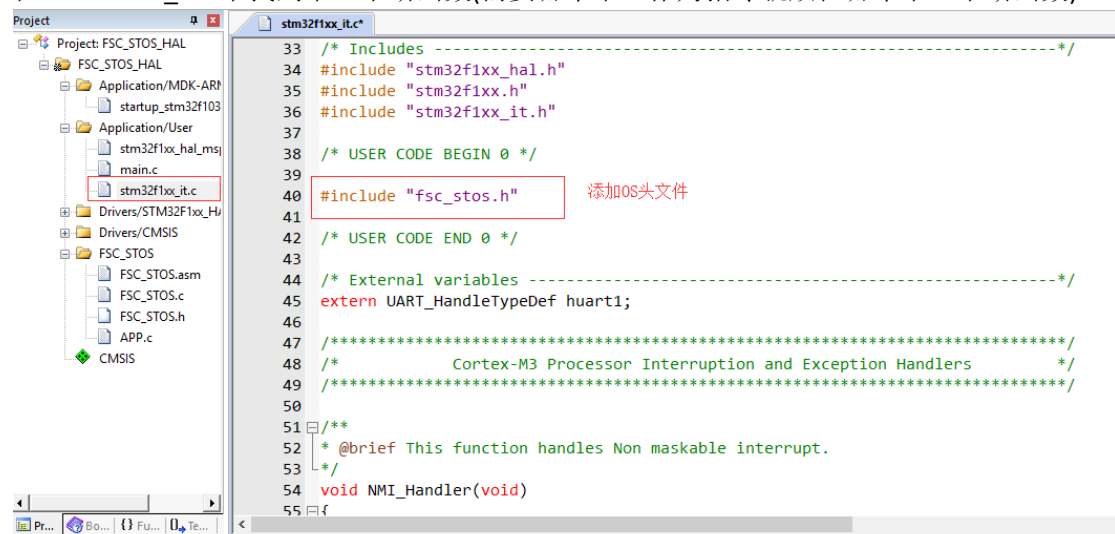
修改后如下:

```
main.c FSC_STOS.h stm32f1xx_hal.c stm32f1xx_hal_msp.c* stm32f1xx_hal_gpio_ex.h
55  __HAL_RCC_AFIO_CLK_ENABLE();
56
57  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
58
59  /* System interrupt init*/
60  /* MemoryManagement_IRQn interrupt configuration */
61  HAL_NVIC_SetPriority(MemoryManagement_IRQn, 0, 0);
62  /* BusFault_IRQn interrupt configuration */
63  HAL_NVIC_SetPriority(BusFault_IRQn, 0, 0);
64  /* UsageFault_IRQn interrupt configuration */
65  HAL_NVIC_SetPriority(UsageFault_IRQn, 0, 0);
66  /* SVCall_IRQn interrupt configuration */
67  HAL_NVIC_SetPriority(SVCall_IRQn, 0, 0);
68  /* DebugMonitor_IRQn interrupt configuration */
69  HAL_NVIC_SetPriority(DebugMonitor_IRQn, 0, 0);
70  /* PendSV_IRQn interrupt configuration */
71  HAL_NVIC_SetPriority(PendSV_IRQn, 0, 0);
72  /* SysTick_IRQn interrupt configuration */
73  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
74
75  /**DISABLE: JTAG-DP Disabled and SW-DP Disabled
76   */
77  __HAL_AFIO_REMAP_SWJ_NOJTAG();
78
79  /* USER CODE BEGIN MspInit 1 */
80
81  /* USER CODE END MspInit 1 */
82  }
83
84  void HAL_UART_MspInit(UART_HandleTypeDef* huart)
```

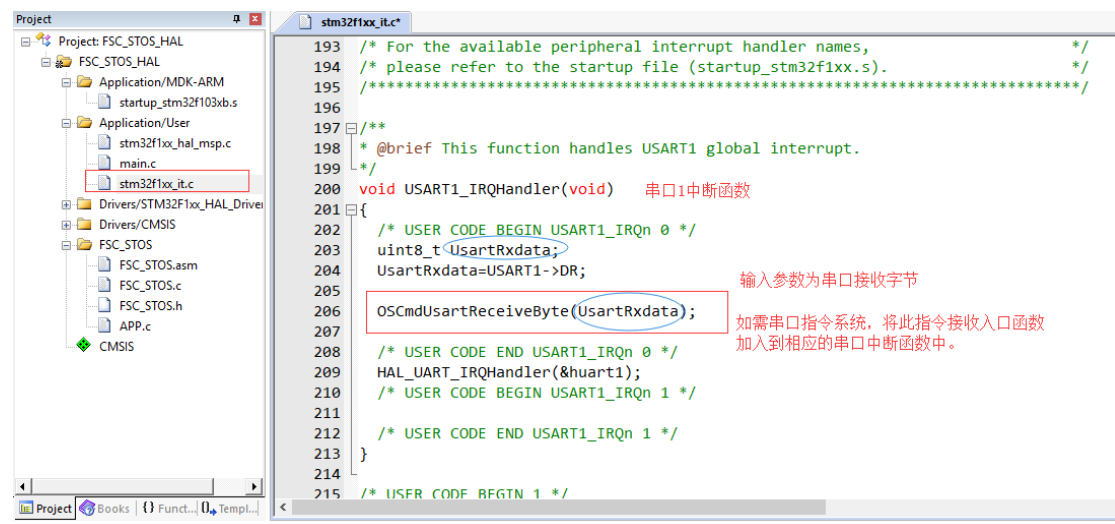
到此系统就可以正常运行了。由于新建 hal 库工程时，一般默认使用 systick 定时器用作 HAL\_Delay()毫秒级延时，其定时时间也刚好为 1ms，而系统默认系统定时器也使用 systick 定时器并且也是 1ms 定时中断，所以正好符合，如果需要修改系统时间切片为其他时间值，请在 hal 初使化中把 systick 初使化有关代码注释掉或在新建工程时把 systick 用作延时的选项去掉。

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

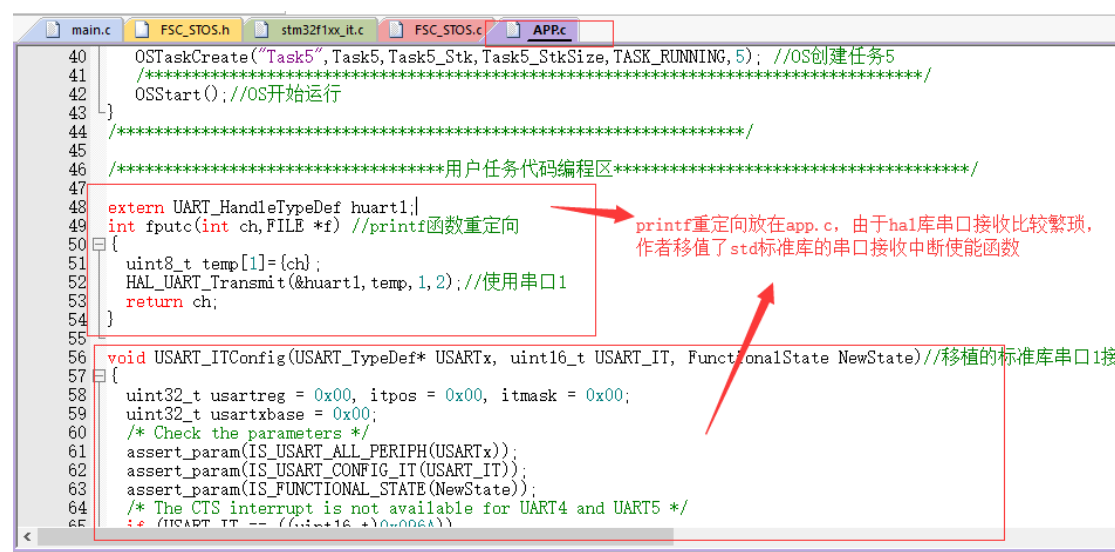
在 stm32fxxx\_it.c 中找到串口中断函数(需要哪个串口作为指令就放在哪个串口中断函数)



```
33 /* Includes -----*/
34 #include "stm32f1xx_hal.h"
35 #include "stm32f1xx.h"
36 #include "stm32f1xx_it.h"
37
38 /* USER CODE BEGIN 0 */
39
40 #include "fsc_stos.h" 添加OS头文件
41
42 /* USER CODE END 0 */
43
44 /* External variables -----*/
45 extern UART_HandleTypeDef huart1;
46
47 /*****
48  * Cortex-M3 Processor Interruption and Exception Handlers
49  *****/
50
51 /**
52  * @brief This function handles Non maskable interrupt.
53  */
54 void NMI_Handler(void)
55 {
```



```
193 /* For the available peripheral interrupt handler names,
194  * please refer to the startup file (startup_stm32f1xx.s).
195  */
196
197 /**
198  * @brief This function handles USART1 global interrupt.
199  */
200 void USART1_IRQHandler(void) 串口1中断函数
201 {
202     /* USER CODE BEGIN USART1_IRQn 0 */
203     uint8_t UsartRxdata;
204     UsartRxdata=USART1->DR;
205
206     OSCmdUsartReceiveByte(UsartRxdata); 输入参数为串口接收字节
207
208     /* USER CODE END USART1_IRQn 0 */
209     HAL_UART_IRQHandler(&huart1);
210     /* USER CODE BEGIN USART1_IRQn 1 */
211
212     /* USER CODE END USART1_IRQn 1 */
213 }
214
215 /* USER CODE BEGIN 1 */
```



```
40 OSTaskCreate("Task5", Task5, Task5_Stk, Task5_StkSize, TASK_RUNNING, 5); //OS创建任务5
41 /******
42  */
43 OSStart(); //OS开始运行
44
45 /*****
46  */
47
48 /******用户任务代码编程区*****
49
50 extern UART_HandleTypeDef huart1;
51 int fputc(int ch, FILE *f) //printf函数重定向
52 {
53     uint8_t temp[1]={ch};
54     HAL_UART_Transmit(&huart1, temp, 1, 2); //使用串口1
55     return ch;
56 }
57
58 void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState) //移植的标准库串口1接
59 {
60     uint32_t usartreg = 0x00, itpos = 0x00, itmask = 0x00;
61     uint32_t usartxbase = 0x00;
62     /* Check the parameters */
63     assert_param(IS_USART_ALL_PERIPH(USARTx));
64     assert_param(IS_USART_CONFIG_IT(USART_IT));
65     assert_param(IS_FUNCTIONAL_STATE(NewState));
66     /* The CTS interrupt is not available for UART4 and UART5 */
67     if ((USART_IT & ((uint16_t)0x0060)) != 0x0000)
68     {
```

```
main.c FSC_STOS.h stm32f1xx_it.c FSC_STOS.c APP.c
16 #define Task5_StkSize 128 //任务5堆栈大小
17
18 __align(8) OS_STK Task1_Stk[Task1_StkSize]; //任务1堆栈
19 __align(8) OS_STK Task2_Stk[Task2_StkSize]; //任务2堆栈
20 __align(8) OS_STK Task3_Stk[Task3_StkSize]; //任务3堆栈
21 __align(8) OS_STK Task4_Stk[Task4_StkSize]; //任务4堆栈
22 __align(8) OS_STK Task5_Stk[Task5_StkSize]; //任务5堆栈
23
24 void OS_MAIN(void)
25 {
26     /*-----用户全局初始化区-----*/
27     /*推荐把所有任务都使用到的初始化放在此处，Task独立用到的初始化放在Task内*/
28
29     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)
30
31
32     /*-----*/
33     /*-----*/
34     OSInit(); //系统初始化
35     /*-----在系统中创建任务-----*/
36     OSTaskCreate("Task1", Task1, Task1_Stk, Task1_StkSize, TASK_RUNNING, 1); //OS创建任务1
37     OSTaskCreate("Task2", Task2, Task2_Stk, Task2_StkSize, TASK_RUNNING, 2); //OS创建任务2
38     OSTaskCreate("Task3", Task3, Task3_Stk, Task3_StkSize, TASK_RUNNING, 3); //OS创建任务3
39     OSTaskCreate("Task4", Task4, Task4_Stk, Task4_StkSize, TASK_RUNNING, 4); //OS创建任务4
40     OSTaskCreate("Task5", Task5, Task5_Stk, Task5_StkSize, TASK_RUNNING, 5); //OS创建任务5
41     /*-----*/
42 }
```

到此移植完成。编译

```
Project
Project FSC_STOS_HAL
  Application/MDK-ARM
  Application/User
  stm32f1xx_hal_msp.c
  main.c
  stm32f1xx_it.c
  Drivers/STM32F1xx_HAL_Drv
  Drivers/CMSIS
    FSC_STOS.asm
    FSC_STOS.c
    FSC_STOS.h
    APP.c
  CMSIS

main.c FSC_STOS.h stm32f1xx_it.c FSC_STOS.c APP.c
16 #define Task5_StkSize 128 //任务5堆栈大小
17
18 __align(8) OS_STK Task1_Stk[Task1_StkSize]; //任务1堆栈
19 __align(8) OS_STK Task2_Stk[Task2_StkSize]; //任务2堆栈
20 __align(8) OS_STK Task3_Stk[Task3_StkSize]; //任务3堆栈
21 __align(8) OS_STK Task4_Stk[Task4_StkSize]; //任务4堆栈
22 __align(8) OS_STK Task5_Stk[Task5_StkSize]; //任务5堆栈
23
24 void OS_MAIN(void)
25 {
26     /*-----用户全局初始化区-----*/
27     /*推荐把所有任务都使用到的初始化放在此处，Task独立用到的初始化放在Task内*/
28
29     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)
30
31
32     /*-----*/
33     /*-----*/
34     OSInit(); //系统初始化
35     /*-----在系统中创建任务-----*/
36     OSTaskCreate("Task1", Task1, Task1_Stk, Task1_StkSize, TASK_RUNNING, 1); //OS创建任务1
37     OSTaskCreate("Task2", Task2, Task2_Stk, Task2_StkSize, TASK_RUNNING, 2); //OS创建任务2
38     OSTaskCreate("Task3", Task3, Task3_Stk, Task3_StkSize, TASK_RUNNING, 3); //OS创建任务3
39     OSTaskCreate("Task4", Task4, Task4_Stk, Task4_StkSize, TASK_RUNNING, 4); //OS创建任务4
40     OSTaskCreate("Task5", Task5, Task5_Stk, Task5_StkSize, TASK_RUNNING, 5); //OS创建任务5
41     /*-----*/
42 }
```

```
Build Output
compiling main.c...
linking...
Program Size: Code=14394 RO-data=498 RW-data=268 ZI-data=6812
FromELF: creating hex file...
*.Output\FSC_STOS_HAL.axf 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:04
```

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)

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)

\*\*\*\*\*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 CPU	使用栈 Used	空闲栈 Free	百分比 per	优先级 prio	任务名 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

单条发送 多条发送 协议传输 帮助

cmd/osmanage//

发送

清除发送