uC/OS_II 在 M480系列单片机上的移植

NuMicro® 32位系列微控制器范例代码介绍

文件信息

应用简述	在 M480 系列单片机上移植 uC/OS_V2.93 的示例代码
BSP 版本	M480_Series_BSP_CMSIS_V3.05.003
开发平台	NuMaker-M483KG V1.1

1. 概述

本代码介绍在 M480 系列单片机上移植 UC/OS_II 需要添加哪些文件。

本工程建立了 3 个任务，前俩任务相互通信，第 3 个任务负责打印前俩任务发来的打印信息。

1.1 原理

为何要用 uC/OS? 因为用了 uC/OS,每个任务都像是独占了 CPU，因此，一个应用可以分成由不同人编写的多个任务。

当一个任务被另一任务打断时，就像是 CPU 去执行中断代码了，过一会儿 CPU 就会回来。每个任务都有独立的栈区，在任务开始调度前，每个栈区被初始化成像是刚刚发生中断一样，CPU 寄存器被压入了栈区。

```
OS_STK  *OSTaskStkInit(void (*task)(void *p_arg),  void*p_arg,  OS_STK*ptos,  INT16U opt)
{
    OS_STK  *p_stk;

    (void)opt;                                  /* 'opt' is not used, prevent warning */
    p_stk = ptos + 1u;                          /* Load stack pointer    */
    p_stk = (OS_STK *)((OS_STK)(p_stk) & 0xFFFFFFF8u);   /* Align the stack to 8-bytes. */
                                               /* Registers stacked as if auto-saved on exception */
#if (OS_CPU_ARM_FP_EN > 0u)                    /* FPU auto-saved registers. */
    --p_stk;
    *(--p_stk) = (OS_STK)0x02000000u;                /* FPSCR  */
                                             /* Initialize S0-S15 floating point registers */
    *(--p_stk) = (OS_STK)0x41700000u;                                /* S15   */
    *(--p_stk) = (OS_STK)0x41600000u;                                /* S14   */
    *(--p_stk) = (OS_STK)0x41500000u;                                /* S13   */
    *(--p_stk) = (OS_STK)0x41400000u;                                /* S12   */
    *(--p_stk) = (OS_STK)0x41300000u;                                /* S11   */
    *(--p_stk) = (OS_STK)0x41200000u;                                /* S10   */
    *(--p_stk) = (OS_STK)0x41100000u;                                /* S9    */
    *(--p_stk) = (OS_STK)0x41000000u;                                /* S8    */
    *(--p_stk) = (OS_STK)0x40E00000u;                                /* S7    */
    *(--p_stk) = (OS_STK)0x40C00000u;                                /* S6    */
    *(--p_stk) = (OS_STK)0x40A00000u;                                /* S5    */
    *(--p_stk) = (OS_STK)0x40800000u;                                /* S4    */
    *(--p_stk) = (OS_STK)0x40400000u;                                /* S3    */
    *(--p_stk) = (OS_STK)0x40000000u;                                /* S2    */
    *(--p_stk) = (OS_STK)0x3F800000u;                                /* S1    */
    *(--p_stk) = (OS_STK)0x00000000u;                                /* S0    */
#endif
    *(--p_stk) = (OS_STK)0x01000000uL;                               /* xPSR,  T=1   */
    *(--p_stk) = (OS_STK)task;                                       /* Entry Point  */
    *(--p_stk) = (OS_STK)OS_TaskReturn;                              /* R14 (LR)     */
    *(--p_stk) = (OS_STK)0x12121212uL;                               /* R12          */
```

```
    *(--p_stk) = (OS_STK)0x03030303uL;                          /* R3 */
    *(--p_stk) = (OS_STK)0x02020202uL;                          /* R2  */
    *(--p_stk) = (OS_STK)0x01010101uL;                          /* R1  */
    *(--p_stk) = (OS_STK)p_arg;                                 /* R0 : argument  */

#if (OS_CPU_ARM_FP_EN > 0u)           /* Initialize S16-S31 floating point registers */
    *(--p_stk) = (OS_STK)0x41F80000u;                           /* S31 */
    *(--p_stk) = (OS_STK)0x41F00000u;                           /* S30 */
    *(--p_stk) = (OS_STK)0x41E80000u;                           /* S29 */
    *(--p_stk) = (OS_STK)0x41E00000u;                           /* S28 */
    *(--p_stk) = (OS_STK)0x41D80000u;                           /* S27 */
    *(--p_stk) = (OS_STK)0x41D00000u;                           /* S26 */
    *(--p_stk) = (OS_STK)0x41C80000u;                           /* S25 */
    *(--p_stk) = (OS_STK)0x41C00000u;                           /* S24 */
    *(--p_stk) = (OS_STK)0x41B80000u;                           /* S23 */
    *(--p_stk) = (OS_STK)0x41B00000u;                           /* S22 */
    *(--p_stk) = (OS_STK)0x41A80000u;                           /* S21 */
    *(--p_stk) = (OS_STK)0x41A00000u;                           /* S20 */
    *(--p_stk) = (OS_STK)0x41980000u;                           /* S19 */
    *(--p_stk) = (OS_STK)0x41900000u;                           /* S18 */
    *(--p_stk) = (OS_STK)0x41880000u;                           /* S17 */
    *(--p_stk) = (OS_STK)0x41800000u;                           /* S16 */

    *(--p_stk) = (OS_STK)0xFFFFFFEDuL;   /*R14,See Note5 */
#else
    *(--p_stk) = (OS_STK)0xFFFFFFFDuL;   /*R14,See Note5 */
#endif
                                         /* Remaining registers saved on process stack */
    *(--p_stk) = (OS_STK)0x11111111uL;                          /* R11 */
    *(--p_stk) = (OS_STK)0x10101010uL;                          /* R10 */
    *(--p_stk) = (OS_STK)0x09090909uL;                          /* R9 */
    *(--p_stk) = (OS_STK)0x08080808uL;                          /* R8 */
    *(--p_stk) = (OS_STK)0x07070707uL;                          /* R7 */
    *(--p_stk) = (OS_STK)0x06060606uL;                          /* R6 */
    *(--p_stk) = (OS_STK)0x05050505uL;                          /* R5  */
    *(--p_stk) = (OS_STK)0x04040404uL;                          /* R4 */

    return (p_stk);
}
```

建好所有的任务并且栈区初始化好以后，CPU 开始执行最高优先级任务代码，就像是刚从中断返回来一样，这个工作是在函数 OSStart(), OSStartHighRdy() 和 PendSV_Handler 中做的。

任务的切换在函数 PendSV_Handler()中。最高优先级任务栈区的首地址被读出来写入栈指针 PSP，然后执行中断返回，CPU 就返回到最高优先级任务去执行代码了。

```
PendSV_Handler
    CPSID   I                       ;Cortex-M4 errata notice. See Note #5
    MOV32 R2,OS_KA_BASEPRI_Boundary ;SetBASEPRI priority required for exception preemption
    LDR     R1, [R2]
    MSR     BASEPRI, R1
    DSB
```

```
    ISB
    CPSIE   I

    MRS     R0, PSP                         ; PSP is process stack pointer
    CBZ     R0, OS_CPU_PendSVHandler_nosave

IF {FPU} != "SoftVFP"
    TST     LR, #0x10
    IT      EQ
    VSTMDBEQ R0!,{s16-s31}
ENDIF
    STMFD   R0!, {R4-R11,LR}     ; Save remaining regs r4-11, R14 on process stack

    LDR     R5, =OSTCBCur      ; OSTCBCur->OSTCBStkPtr = SP;
    LDR     R1, [R5]
    STR     R0, [R1]           ; R0 is SP of process being switched out

OS_CPU_PendSVHandler_nosave     ; At this point, entire context of process has been saved
    PUSH    {LR}               ; Save LR exc_return value
    BL      OSTaskSwHook       ; Call OSTaskSwHook()
    POP     {LR}

    LDR     R0, =OSPrioCur     ; OSPrioCur = OSPrioHighRdy;
    LDR     R1, =OSPrioHighRdy
    LDRB    R2, [R1]
    STRB    R2, [R0]

    LDR     R0, =OSTCBCur
    LDR     R1, =OSTCBHighRdy  ; OSTCBCur  = OSTCBHighRdy;
    LDR     R2, [R1]
    STR     R2, [R0]

    LDR     R0,  [R2]          ; R0 is new process SP; SP = OSTCBHighRdy->OSTCBStkPtr;

    LDMFD   R0!, {R4-R11,LR}    ; Restore r4-11, R14 from new process stack
IF {FPU} != "SoftVFP"
    TST     LR, #0x10
    IT      EQ
    VLDMIAEQ R0!,{s16-s31}
ENDIF

    MSR     PSP, R0             ; Load PSP with new process SP

    MOV32   R2, #0              ; Restore BASEPRI priority level to 0
    CPSID   I
    MSR     BASEPRI, R2
    DSB
    ISB
    CPSIE   I

    BX      LR                  ; Exception return will restore remaining context
    ALIGN           ; Removes warning[A1581W]: added <no_padbytes> of padding at <address>
    END
```

PendSV 中断代码的流程图见 图 1-1. 进入 PendSV 中断后, 如果 PSP 不等 0，表明 CPU 刚从一个任务跳转过来，要先保存当前任务的上下文，包括栈指针 SP，然后再获取新任务的上下文。

PSP=0 表明刚从 OSStart()开始调度任务，不需要保存当前任务的上下文。读出当前最高优先级任务的上下文，栈指针 PSP 指向最高优先任务的栈区，然后执行中断返回，CPU 就去执行最高优先级任务去了。

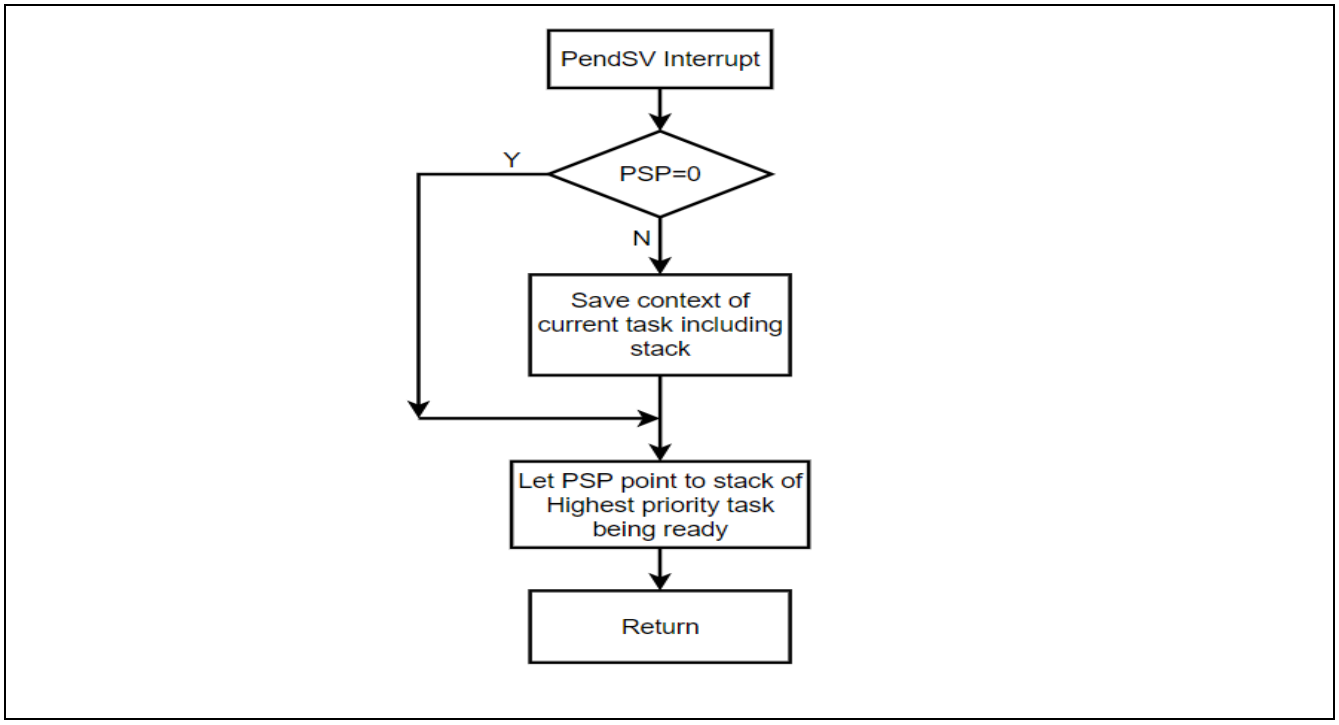

图 1-1 PendSV 中断代码流程图

1.2 执行结果

任务 2 让黄色 LED 闪烁，闪烁周期由根据任务 1 发过来，100 或 500 毫秒长短交替。任务 3 负责打印其它任务发来的打印信息如图 1-2。

图 1-2 UART 打印信息

2. 代码介绍

工程结构见图 2-1，工程中只包含必要文件。
由于版权问题，软件包中不含uC/OS内核代码文件，请到相关网站下载uC/OS内核代码，按如下目录添加到工程中即可。

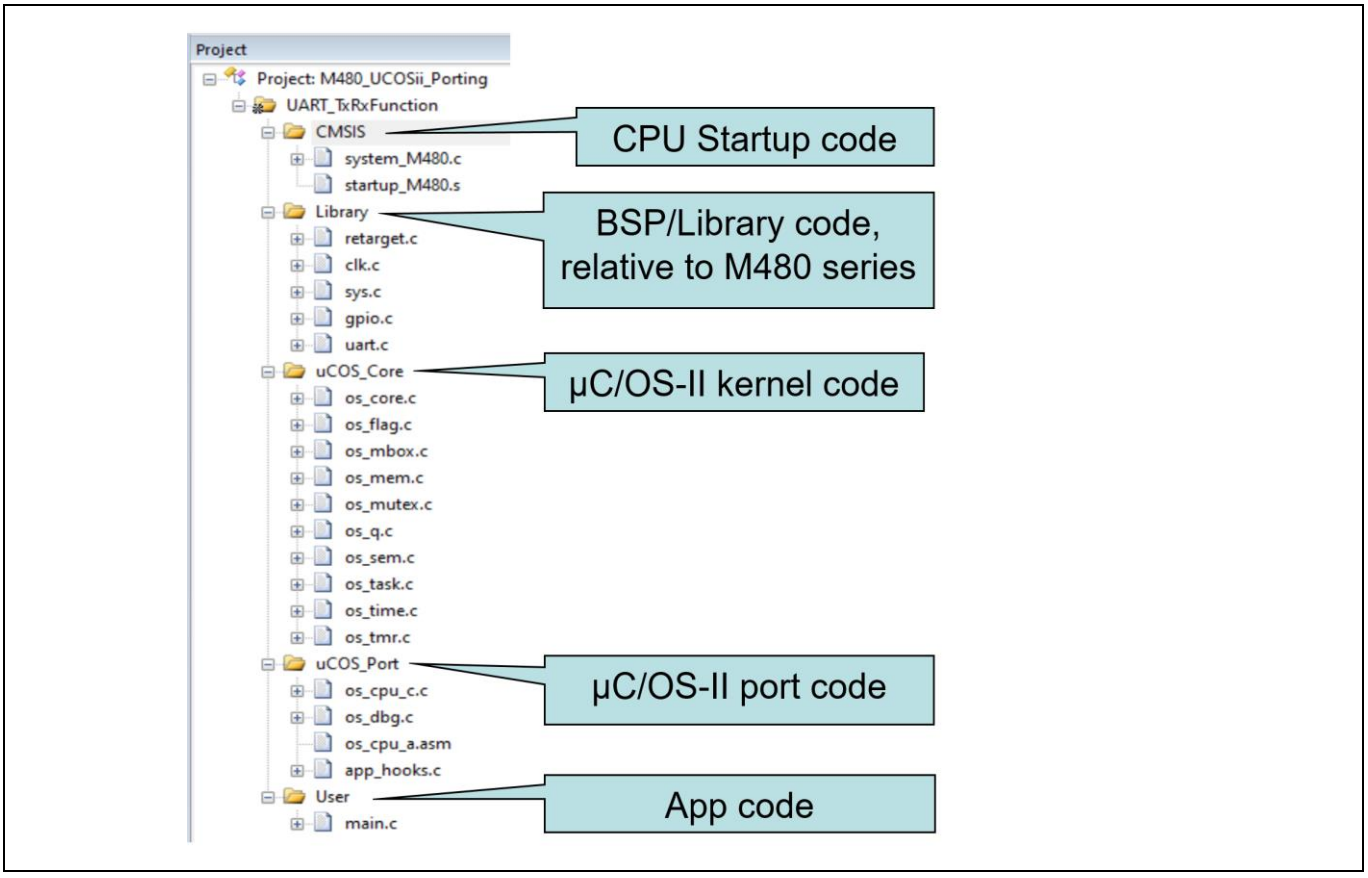

图 2-1 工程结构

M480 的各外设初始化后，先建立信号量、邮箱和队列，然后建立 任务。执行 OSStart()之后，代码开始任务调度。

```
int main(void)
{
    SYS_UnlockReg();                    // Unlock protected registers
    SYS_Init();                         // Init System, peripheral clock and multi-function I/O
    SYS_LockReg();                      // Lock protected registers

    UART_Open(UART0, 115200);                           //  Init UART0 for printf
    printf("\n\nCPU @ %d Hz\n", SystemCoreClock);
    PH->DOUT |= 0x30 ;
    GPIO_SetMode(PH, BIT4 | BIT5, GPIO_MODE_OUTPUT);

    OSInit();                                            // uC/OS initialization.
    p_MailBox_1 = OSMboxCreate((void *)0);               // create a mail box
```

```
    pQ_print   = OSQCreate(Print_Message, 10);        // create a Q which have 10 messages

    OSTaskCreateExt((void(*)(void *))task1_Highest,   // create the highest priority task
                    (void *)0,
                    (OS_STK *)&task1_stk_H[TASK_STK_SIZE - 1],
                    (INT8U)TASK1_LEDR_PRIO_Highest,
                    (INT16U)TASK1_LEDR_PRIO_Highest,
                    (OS_STK *)&task1_stk_H[0],
                    (INT32U)TASK_STK_SIZE,
                    (void *)0,
                    (INT16U)OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR |
OS_TASK_OPT_SAVE_FP);

    OSTaskCreate((void(*)(void *))task2_LED,              // create another task
                 (void *)0,
                 (OS_STK *)&task2_stk_led[TASK_STK_SIZE -1],
                 (INT8U)TASK2_LEDY_PRIO);

    OSTaskCreate((void(*)(void *))task3_Print,            // create another task
                 (void *)0,
                 (OS_STK *)&task3_stk_led[TASK_STK_SIZE -1],
                 (INT8U)TASK3_PRINT_PRIO);
    OSStart();                                            // Start schedule

    while (1) {} ;                          // This sentence would be never executed.
}
```

每个任务都是一个 while(1){ } 循环。在循环中必须调用系统函数，如 OSTimeDaley(), OSMboxPost()等，如此，其它任务才有时间执行。

系统节拍定时器必须在最高优先级的任务里启动，否则，如果在任务调度之前发生了节拍定时器中断，系统可能崩溃。

任务 1 优先级最高，开头启动 SysTick 定时器。while(1)循环里，每 3 秒钟给任务 2 发一次数据。

```
void task1_Highest(void *pvParameters)
{                                                 // MUST active tick-timer in highest task
    SysTick_Config(SystemCoreClock / OS_TICKS_PER_SEC);    // active tick-timer

    while (1)
    {
        OSTimeDlyHMSM(0, 0, 3, 0);                          // Delay 3 seconds

        if (Mail_Message[0] != 100) Mail_Message[0] = 100;
        else Mail_Message[0] = 500 ;                    // Configure message in mail box

        OSMboxPost(p_MailBox_1, Mail_Message);          // Send message via mail box
    }
}
```

任务 2 是让 LED 闪烁，闪烁周期，按任务 1 发送过来的时间。

```
void task2_LED(void *pvParameters)
{
    int32_t *ptr ;
    int32_t  static Time = 300 ;

    while (1)
    {
        ptr = OSMboxAccept(p_MailBox_1);   // Check whether there are message in mail box

        if (ptr != (void *)0)             // Received a mail message
        {
            Time = *ptr ;                // Read out the message in mailbox. To update Time
            OSQPost(pQ_print, (int8_t *)Message_Task2); // Send print-message to task3
        }

        OSTimeDlyHMSM(0, 0, 0, Time);             // Delay time according to message.
        LED_Y_TOGGLE() ;                          // LED_Y toggle(flash)
    }
}
```

任务 3, 打印从其它任务发过来的信息。

```
void task3_Print(void *pvParameters)
{
    uint8_t err ;
    uint8_t *ptr ;

    while (1)
    {
        ptr = OSQAccept(pQ_print, &err);          // Check whether there are messages in Q

        if (ptr == 0)OSTimeDlyHMSM(0, 0, 0, 100); // Have no message, wait for 100ms
        else                                      // Received a string to print.
        {
            while (*ptr != 0)                     // Not the last character of a string.
            {
                while (UART_IS_TX_FULL(UART0)) ;  // If UART FiFo has been full, waitting
                UART0->DAT = *ptr++ ;             // write a character into Tx_FiFO
            }
        }
    }
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M480_Series_BSP_CMSIS_V3.05.003
- IDE 版本
  - Keil uVersion 5.26

3.2 硬件需求

- 电路组件
  - NuMaker- M483KG V1.1

4. 目录信息

EC_ M480_uCOS_II_Porting_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
SampleCode	
ExampleCode	Source file of example code
uCOS_II	Files relative to uC/OS

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *M480_uCOS_II_Porting.uvproj*。
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2022.09.13	1.00	初始发布。

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.