

uC/OS_II 在 M480系列單片機上的移植

NuMicro® 32位系列微控制器範例代碼介紹

文件資訊

應用簡述	在 M480 系列單片機上移植 uC/OS_V2.93 的示例代碼
BSP 版本	M480_Series_BSP_CMSIS_V3.05.003
開發平台	NuMaker-M483KG 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

本代碼介紹在 M480 系列單片機上移植 UC/OS_II 需要添加哪些文件。

本工程建立了 3 個任務，前倆任務相互通信，第 3 個任務負責打印前倆任務發來的打印信息。

1.1 原理

為何要用 uC/OS? 因為用了 uC/OS,每個任務都像是獨占了 CPU，因此，一個應用可以分成幾個任務，而這些任務，可以由不同的人編寫。

當一個任務被另一任務搶先時，就像是 CPU 去執行中斷代碼了，過一會兒 CPU 就會回來。每一個任務都有獨立的堆疊，在任務開始調度前，每個堆疊被初始化成像是剛剛發生中斷一樣，CPU 寄存器被壓入了堆疊。

```

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 */
    /* FPU auto-saved registers. */
#ifdef OS_CPU_ARM_FP_EN > 0u
    --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
    #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)0xFFFFFEDuL;           /*R14,See Note5 */
#else
*(--p_stk) = (OS_STK)0xFFFFFEDuL;           /*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);
}

```

建好所有的任務並且堆疊初始化好以後，CPU 開始執行最高優先級任務代碼，就像是剛從中斷返回來一樣，這個工作是在函數 OSStart(), OSStartHighRdy() 和 PendSV_Handler 中做的。

任務的切換在函數 PendSV_Handler()中。最高優先級任務堆疊的首地址被讀出來寫入棧指針 PSP，然後執行中斷返回，CPU 就返回到最高優先級任務去執行代碼了。

```

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 中斷代碼的流程圖見 圖 1-1. 進入 PendSV 中斷後, 如果 PSP 不等 0, 表明 CPU 剛從一個任務跳轉過來, 要先保存當前任務的上下文, 包括棧指針 SP, 然後再獲取新任務的上下文。

PSP=0 表明剛從 OSStart()開始調度任務, 不需要保存當前任務的上下文。讀出當前最高優先級任務的上下文, 棧指針 PSP 指向最高優先任務的堆疊, 然後執行中斷返回, CPU 就去執行最高優先級任務去了。

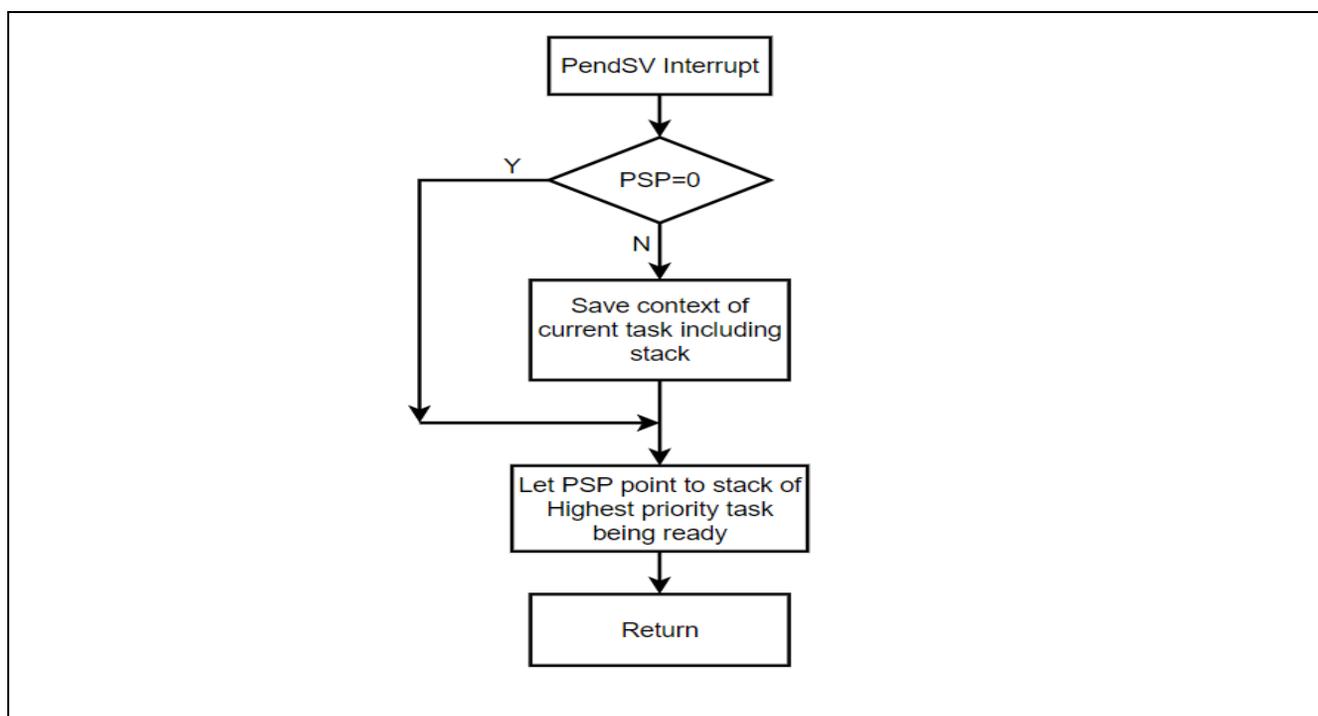


圖 1-1 PendSV 中斷代碼流程圖

1.2 執行結果

任務 2 讓黃色 LED 閃爍，閃爍周期由根據任務 1 發過來，100 或 500 毫秒長短交替。任務 3 負責打印其它任務發來的打印信息如圖 1-2。



圖 1-2 UART 打印信息

2. 代碼介紹

工程結構見圖 2-1, 工程中只包含必要文件。

由於版權問題，軟件包中不含uC/OS內核代碼文件，請到相關網站下載uC/OS內核代碼，按如下目錄添加到工程中即可。

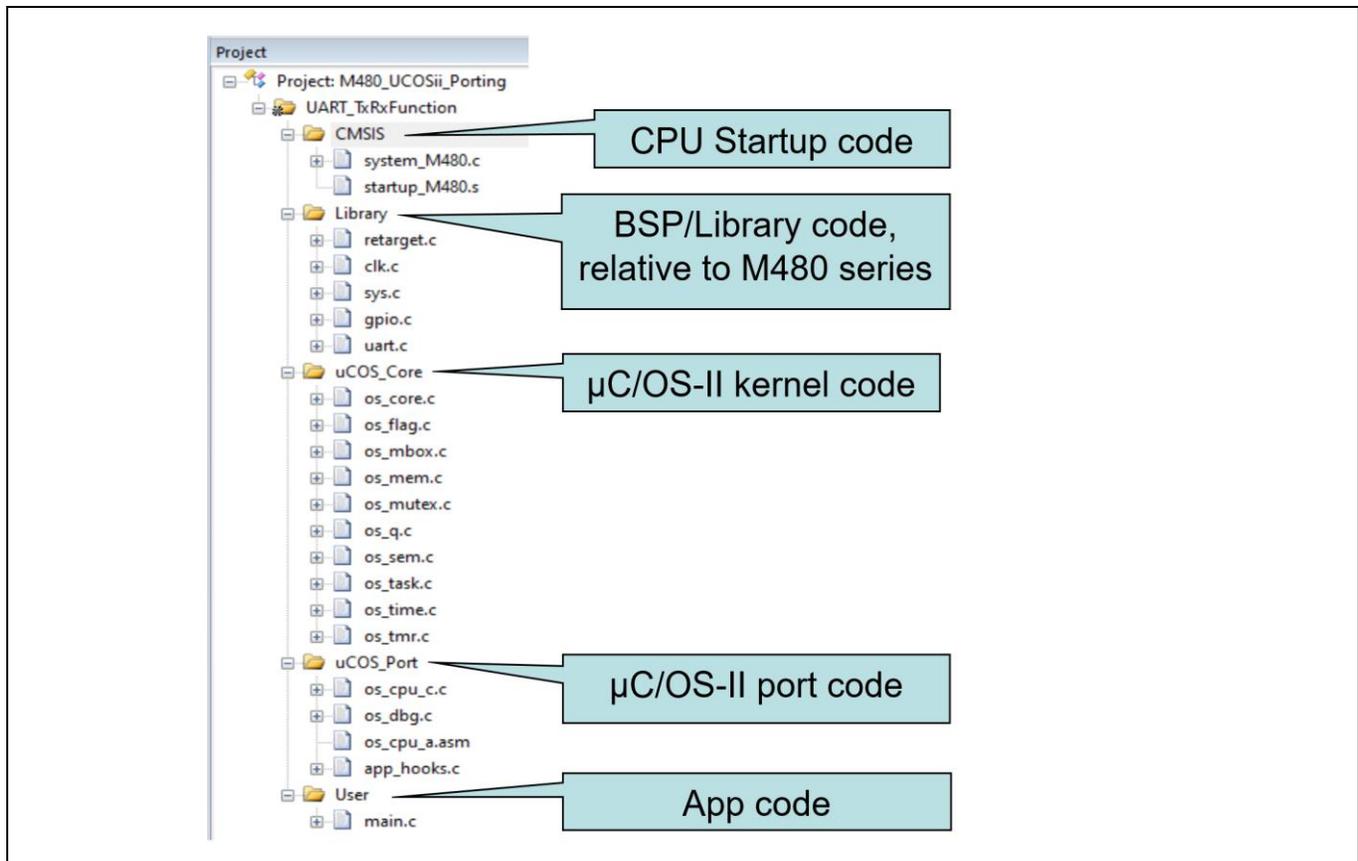


圖 2-1 工程結構

M480 的各周邊初始化後，先建立信號量、郵箱和隊列，然後建立任務。執行 OSStart()之後，代碼開始任務調度。

```
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.
}

```

每個任務都是一個 `while(1){ }` 循環。在循環中必須調用系統函數，如 `OSTimeDaley()`、`OSMboxPost()`等，如此，其它任務才有時間執行。

`SysTick` 定時器必須在最高優先級的任務裏啟動，否則，如果在任務調度之前發生了 `SysTick` 定時器中斷，系統可能崩潰。

任務 1 優先級最高，開頭啟動 `SysTick` 定時器。`while(1)`循環裏，每 3 秒鐘讓給任務 2 發一次數據。

```

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

```

任務 2 是讓 LED 閃爍，閃爍周期，按任務 1 發送過來的時間

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

    while (1)
    {
        ptr = OSMBboxAccept(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)
    }
}
```

任務 3, 打印從其它任務發過來的信息。

```
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, waiting
                UART0->DAT = *ptr++ ; // write a character into Tx_FiFO
            }
        }
    }
}
```

3. 軟體與硬體需求

3.1 軟體需求

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

3.2 硬體需求

- 電路元件
 - NuMaker- M483KG V1.1

4. 目錄資訊

 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

圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *M480_uCOS_II_Porting.uvproj*。
2. 進入編譯模式介面
 - 編譯
 - 下載代碼至記憶體
 - 進入 / 離開除錯模式
3. 進入除錯模式介面
 - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2022.09.13	1.00	初始發布。

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.

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.