M480 uC/OS_II Porting

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code is used for porting uC/OS_V2.93 on the M480 series microcontroller (MCU).
BSP Version	M480_Series_BSP_CMSIS_V3.05.003
Hardware	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. Overview

This example code introduces what files should be added in a project if user wants to port uC/OS_II on the M480 series microcontroller (MCU).

Three tasks are created. Two tasks transmit data to each other and the third is responsible for printing messages received from other tasks.

1.1 Principle

uC/OS is used in an application because each task of application code seems to monopolize a CPU while uC/OS is used in a project. Thus, an application code can be divided into several tasks which can be written by different software engineers.

When a task is preempted by another task, it seems that CPU is going to execute an interrupt subroutine, and the CPU will return back to the current task after a while. Each task has an independent stack. Before task scheduling, each stack of task is initialized as if an interrupt has just occurred, and CPU registers have been pushed into the stack.

```
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);
}
```

After all tasks are created and their stacks are initialized, CPU starts executing the highest priority task code as if it just returned from an interrupt subroutine. This is performed in the functions OSStart(), OSStartHighRdy() and PendSV_Handler.

Tasks are switched in function PendSV_Handler(). The first stack-address of highest-priority task is read out and written into PSP following the PendSV interrupt subroutine return. Then, CPU returns back to the highest-priority task code to execute subsequently.

```
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
```

The flowchart of PendSV interrupt code is as Figure 1-1. After entering the PendSV interrupt, if PSP is not equal to 0, it indicates that CPU just jumped from a task. The context of the current task including SP pointer must be saved prior to getting the context of a new task.

PSP=0 indicates that a task has just scheduled from OSStart(). Therefore, no task is interrupted. The context of highest-priority task is read out and PSP is pointed to the stack of highest-priority task. Then, PendSV executes interrupt return, and CPU will execute the highest-priority task.

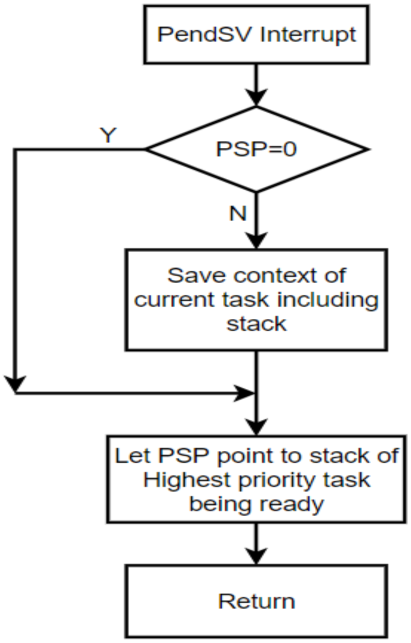
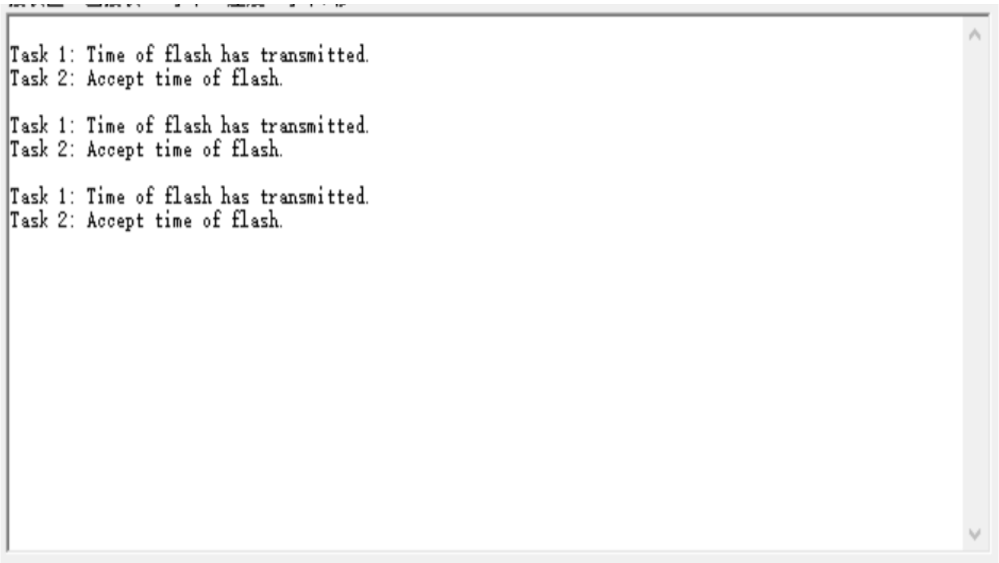

PendSV Interrupt
Y
PSP=0
N
Save context of current task including stack
Let PSP point to stack of Highest priority task being ready
Return

Figure 1-1 Flowchart of PendSV Interrupt

1.2 Demo Result

Task 2 enables yellow LED to flash every 100 or 500 milliseconds according to the data received from the task 1 and the flashing time is alternatively long and short. Task 3 manages printer and prints messages received from other tasks as Figure 1-2 .



```
Task 1: Time of flash has transmitted.
Task 2: Accept time of flash.

Task 1: Time of flash has transmitted.
Task 2: Accept time of flash.

Task 1: Time of flash has transmitted.
Task 2: Accept time of flash.
```

Figure 1-2 Messages Printed from UART

2. Code Description

The project architecture is as Figure 2-1. Only necessary files are included in the project.

For copyright reasons, the uC/OS kernel code files are not included in the software package. Please download uC/OS Kernel code from relevant websites and add it to the project according to the following directory.

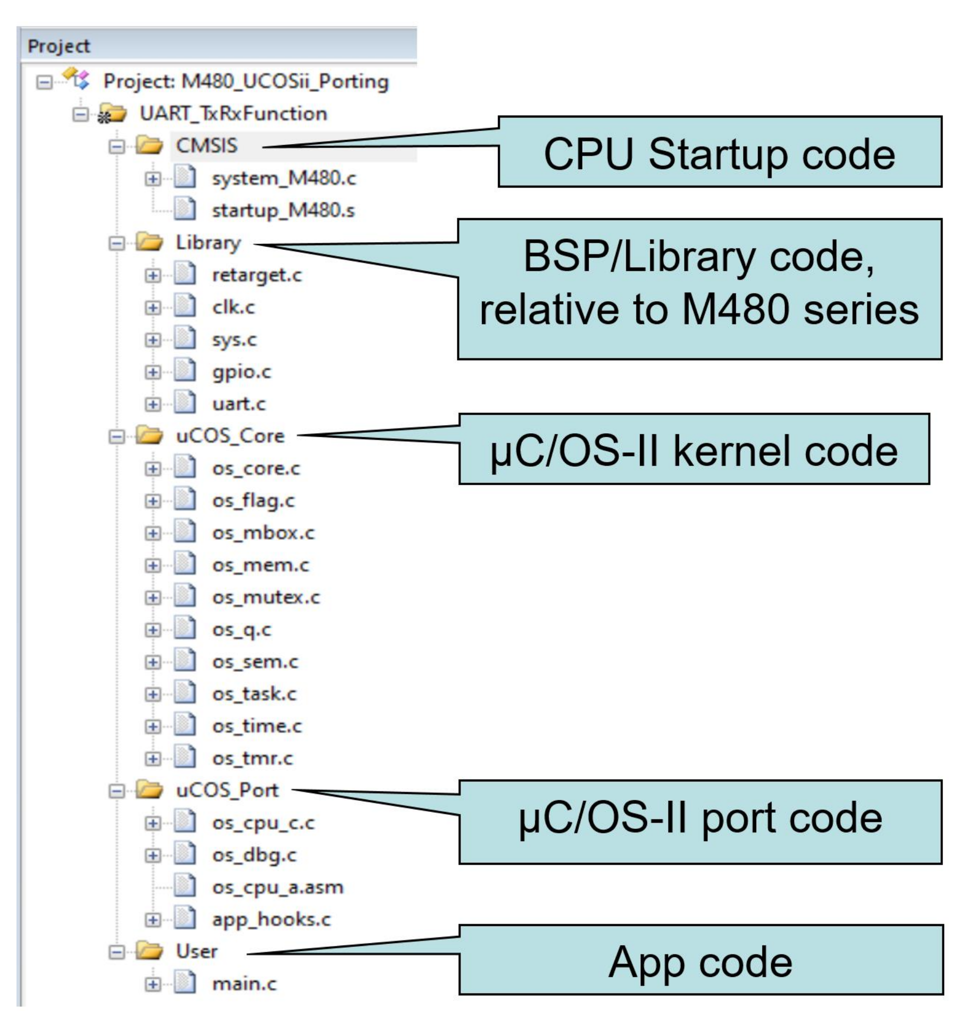


Figure 2-1 Project Architecture

After M480 peripheral initialization, create semaphores, mail-box and queues, and then create tasks. Executing the function OSStart() will start task scheduling.

```
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.
}
```

Each task should be a loop of while(1){ }. System function, for instance, OSTimeDaley(), OSMboxPost() or so, must be called in the loop, thus, the other tasks have time to execute.

The tick-timer MUST be activated in highest-priority tasks. Otherwise, if the SysTick timer interrupt occurs before the task is scheduled, the system may crash.

Task 1 has the highest priority. Thus, SysTick timer is activated in task 1. In the while(1){ } loop, data is sent to task 2 every 3 seconds.

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

    while (1)
    {
        LED_R_0_LIGHT() ;
        OSTimeDlyHMSM(0, 0, 0, 50);               // LED_R light 50ms
        LED_R_1_DARK();

        OSTimeDlyHMSM(0, 0, 3, 0);                // Delay 3 seconds

        if (Mail_Message[0] != 100) Mail_Message[0] = 100;
        else Mail_Message[0] = 500 ;           // time is between 100 and 500ms
        OSMboxPost(p_MailBox_1, Mail_Message); // Send message(time) to task2 via mail box
        OSQPost(pQ_print, Message_Task1);      // Send print message to task3 via Q
    }
}
```

In task 2, LED flashes according to the time the task 1 sends.

```
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)
    }
}
```

In task 3, messages received from other tasks is printed.

```
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. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker- M483KG V1.1

4. Directory Information

The directory structure is shown below.

🗁 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

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M480_uCOS_II_Porting.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2022.09.13	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.