M251 UCOSii Porting

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code is used for porting UCOSii_V2.93 on the M251 series microcontroller (MCU).
BSP Version	M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.003
Hardware	NuMaker-M258KE Ver1.1

1. Overview

This example code introduces what files should be added in a project if user wants to port UCOSii on the M251 series microcontroller (MCU).

Two tasks are created and data are transmitted to each other in this example project.

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.

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

    (void)opt;                                      /* 'opt' is not used, prevent warning */
    stk       = ptos;                               /* Load stack pointer  */

                      /* Registers stacked as if auto-saved on exception */
    *(stk)    = (INT32U)0x01000000uL;               /* xPSR         */
    *(--stk)  = (INT32U)task;                       /* Entry Point */
    *(--stk)  = (INT32U)OS_TaskReturn;              /* R14 (LR)   */
    *(--stk)  = (INT32U)0x12121212uL;               /* R12       */
    *(--stk)  = (INT32U)0x03030303uL;               /* R3       */
    *(--stk)  = (INT32U)0x02020202uL;               /* R2      */
    *(--stk)  = (INT32U)0x01010101uL;               /* R1     */
    *(--stk)  = (INT32U)p_arg;                      /* R0 : argument  */

                   /* Remaining registers saved on process stack */
    *(--stk)  = (INT32U)0x07070707uL;               /* R7          */
    *(--stk)  = (INT32U)0x06060606uL;               /* R6         */
    *(--stk)  = (INT32U)0x05050505uL;               /* R5        */
    *(--stk)  = (INT32U)0x04040404uL;               /* R4       */

    return (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                          ; Prevent interruption during context switch
    MRS   R0, PSP                      ; PSP is process stack pointer

    CMP   R0, #0
    BEQ   OS_CPU_PendSVHandler_nosave    ; equivalent code to CBZ from M3 arch to M0 arch
                           ; Except that it does not change the condition code flags

    SUBS  R0, R0, #0x10    ; Adjust stack pointer to where memory needs to be stored to
                           ; avoid overwriting
    STM   R0!, {R4-R7}     ; Stores 4 4byte registers, default increments SP after each
                           ; storing
    SUBS    R0, R0, #0x10  ; STM does not automatically call back the SP to initial
                           ; location so we must do this manually
    LDR   R1, =OSTCBCur       ; OSTCBCur->OSTCBStkPtr = SP;
    LDR   R1, [R1]
    STR   R0, [R1]            ; R0 is SP of process being switched out

                              ; At this point, entire context of process has been saved
OS_CPU_PendSVHandler_nosave
    PUSH   {R14}              ; Save LR exc_return value
    LDR   R0, =OSTaskSwHook   ; OSTaskSwHook();
    BLX   R0
    POP    {R0}
    MOV   R14, R0

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

    LDR   R0, =OSTCBCur       ; OSTCBCur  = OSTCBHighRdy;
    LDR   R1, =OSTCBHighRdy
    LDR   R2, [R1]
    STR   R2, [R0]
    LDR   R0, [R2]          ; R0 is new process SP; SP = OSTCBHighRdy->OSTCBStkPtr;
    LDM   R0!, {R4-R7}      ; Restore R4-R7 from new process stack
    MSR   PSP, R0           ; Load PSP with new process SP
    MOV   R0,  R14
    MOVS  R1,  #0x04        ; Immediate move to register
    ORRS  R0,  R1           ; Ensure exception return uses process stack
    MOV   R14, R0
    CPSIE I
    BX    LR                ; Exception return will restore remaining context

    ALIGN                  ; Ensures that ARM instructions start on four-byte boundary
    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.

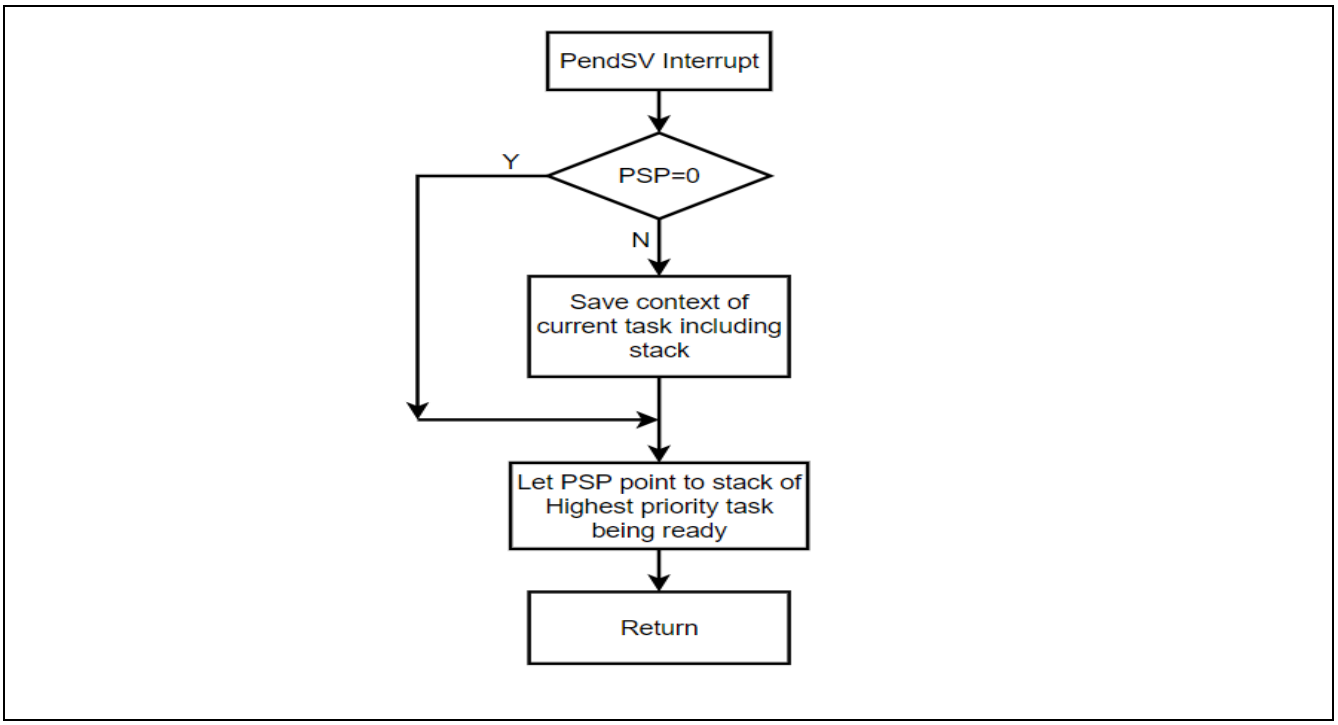

Figure 1-1 Flowchart of PendSV Interrupt

1.2 Demo Result

Task 2 enables a LED to flash every several milliseconds according to the interval times the task 1 transmitted. The flashing time is alternatively long and short.

2. Code Description

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

For copyright reasons, the US/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.

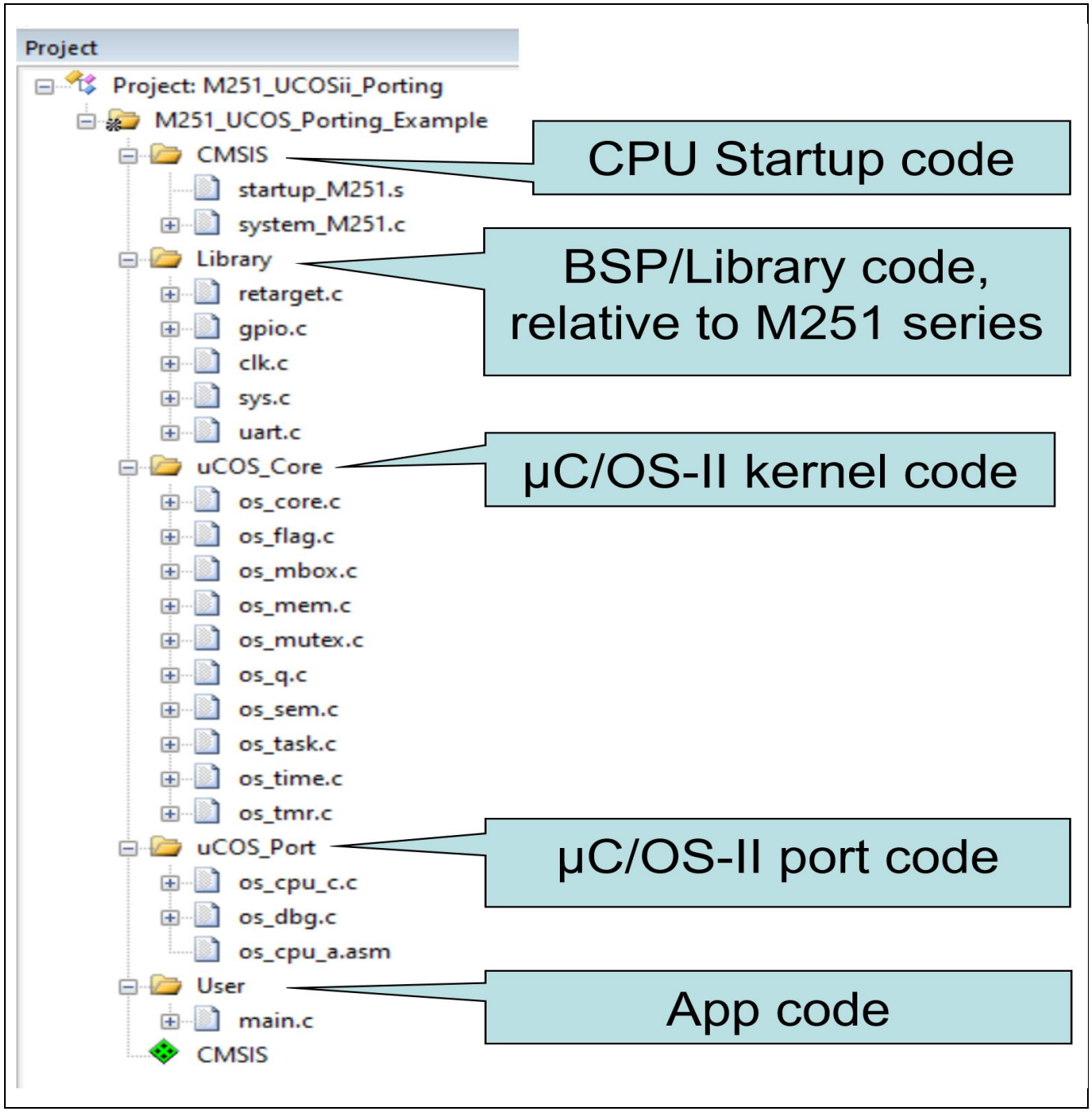

Figure 2-1 Project Architecture

After M251 peripheral initialization, create semaphores, mail-box and queues, and then create tasks. Executing the function OSStart() will start task scheduling.

```
int main(void)
{
    int32_t Cnt32;

    SYS_UnlockReg();        /* Unlock protected registers */
    SYS_Init();             /* Init System, peripheral clock and multi-function I/O */
    SYS_LockReg();          /* Lock protected registers */

    UART0_Init();           /* Init UART0 for printf */
    printf("\n\nCPU @ %u Hz\n", SystemCoreClock);

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

    OSTaskCreateExt((void(*)(void *))task1_Highest,             // create a task
                    (void *)0,
                    (OS_STK *)&task1_stk_H[LED_TASK_STK_SIZE - 1],
                    (INT8U)TASK1_PRIO_Highest,
                    (INT16U)TASK1_PRIO_Highest,
                    (OS_STK *)&task1_stk_H[0],
                    (INT32U)LED_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[LED_TASK_STK_SIZE - 1],
                 (INT8U)TASK2_PRIO_LED);

    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);    // 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 messgae via mail box
    }
}
```

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

```
void task2_LED(void *pvParameters)
{                                                   // MUST active tick-timer in highest task
    int32_t *ptr ;
    int32_t  static Time = 300 ;

    while (1)
    {
       ptr =OSMboxAccept(p_MailBox_1);  //Check whether there are messages in the mailbox

        if (ptr !=(void*)0) Time = *ptr; //Read out if there is any message in the mailbox

        OSTimeDlyHMSM(0, 0, 0, Time);             // Delay time according to message.
        PB14 ^= 1 ;                               // LED flash
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.003
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - NuMaker- M258KE Ver1.1

4. Directory Information

The directory structure is shown below.

EC_ M251_UCOSii_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 uCOS

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 *M251_UCOSii_Porting.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2022.07.27	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.