Transplant UCOS-II on STM32F103C8T6

Development Environment

Hardware:

Processor:STM32F103C8T6

Board:HC6800EM3-V3.0

Softwawre:

OS:UCOS-II V2.86

StdPeriphDriverLib:V3.5.0

IDE:KEIL MDK-4

Directory Analysis

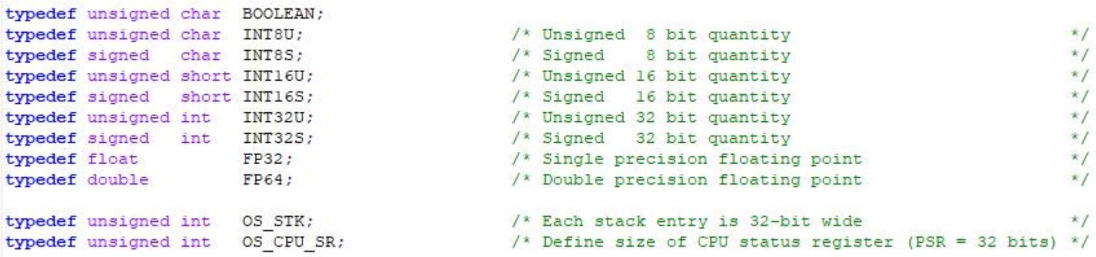
APP:Including files for application task
BSP:Including files about lowlevel drivers for BSP
CMSIS:Including files for Coetex-M3
List:Including listing files producted during compiling.
Out:Including output files like .hex .o
Project:Including files about project itself
STM32F10x_StdPeriph_Driver:Including files about standard peripheral driver library for STM32F10x
uC-CPU:Including files to associate UCOS-II with CPU
uC-LIB:Including library files for UCOS-II
uCOS-II:Including files to create UCOS-II itself

Transplant process

First of all, We need to know what some files have done.

File1: /uCOS-II/ports/os_cpu.h

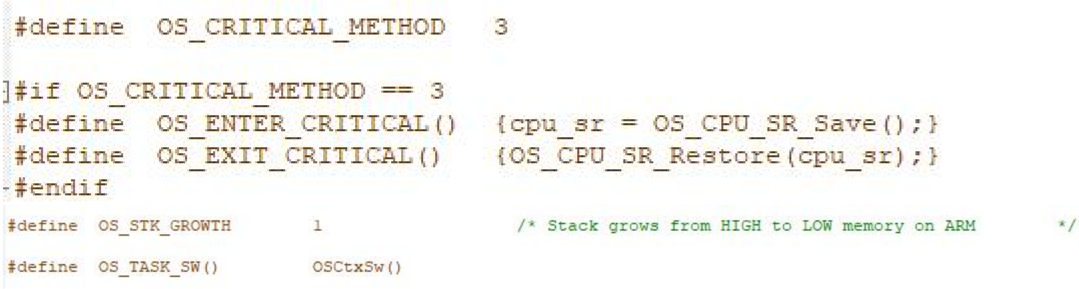
There are three parts in this file,
Part1:Data types



```
typedef unsigned char  BOOLEAN;
typedef unsigned char  INT8U;                    /* Unsigned  8 bit quantity                       */
typedef signed   char  INT8S;                    /* Signed    8 bit quantity                       */
typedef unsigned short INT16U;                   /* Unsigned 16 bit quantity                       */
typedef signed   short INT16S;                   /* Signed   16 bit quantity                       */
typedef unsigned int   INT32U;                   /* Unsigned 32 bit quantity                       */
typedef signed   int   INT32S;                   /* Signed   32 bit quantity                       */
typedef float          FP32;                     /* Single precision floating point                */
typedef double         FP64;                     /* Double precision floating point                */

typedef unsigned int   OS_STK;                   /* Each stack entry is 32-bit wide                */
typedef unsigned int   OS_CPU_SR;                /* Define size of CPU status register (PSR = 32 bits) */
```

We are going to confirm these data types according to your cpu. Especially, the type OS_STK and type OS_CPU_SR are tightly associated with you cpu.
I set OS_STK and OS_CPU_SR as “unsigned int” in my case.

Part2:Macros definition

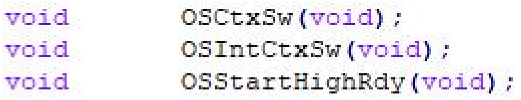


```
#define  OS_CRITICAL_METHOD   3

#if OS_CRITICAL_METHOD == 3
#define  OS_ENTER_CRITICAL()  {cpu_sr = OS_CPU_SR_Save();}
#define  OS_EXIT_CRITICAL()   {OS_CPU_SR_Restore(cpu_sr);}
#endif
#define  OS_STK_GROWTH        1                 /* Stack grows from HIGH to LOW memory on ARM      */

#define  OS_TASK_SW()         OSCtxSw()
```

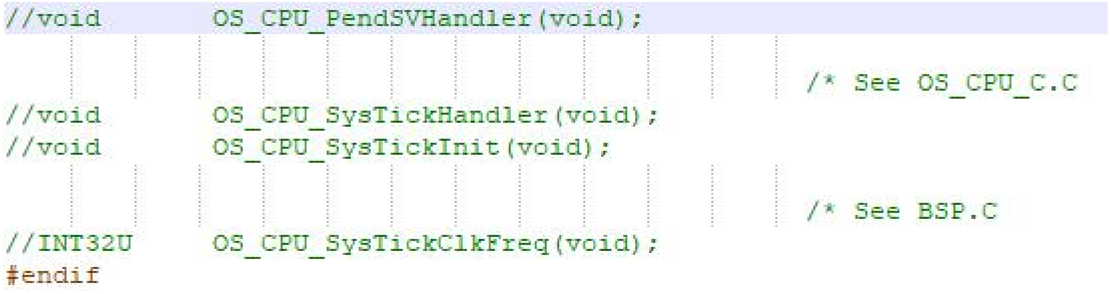
1.Macro OS_CRITICAL_METHOD determines the method to enter critical section.
“OS_CRITICAL_METHOD=1”:Disable/Enable interrupts using simple instructions. After critical section, interrupts will be enabled even if they were disabled before entering the critical section.
“OS_CRITICAL_METHOD=2”:Disable/Enable interrupts by preserving the state of interrupts. In other words, if interrupts were disabled before entering the critical section, they will be disabled when leaving the critical section.
“OS_CRITICAL_METHOD=3”:Disable/Enable interrupts by preserving the state of interrupts. Generally speaking you would store the state of the interrupt disable flag in the local variable 'cpu_sr' and then disable interrupts. 'cpu_sr' is allocated in all of uC/OS-II's functions that need to disable interrupts. You would restore the interrupt disable state by copying back 'cpu_sr' into the CPU's status register.
Note that the method “OS_CRITICAL_METHOD=1” and method “OS_CRITICAL_METHOD=2” are not implemented.You can only choose “OS_CRITICAL_METHOD=3”.

2.Macro OS_STK_GROWTH determines the way that stack grows. In most conditions, stack grows from High to Low memory.It is same to STM32.
“OS_STK_GROWTH=1” means stack grows from High to Low memory.
“OS_STK_GROWTH=0” means stack grows from Low to High memory.

Part3:function declaration



```
void        OSCtxSw(void);
void        OSIntCtxSw(void);
void        OSStartHighRdy(void);
```



```
//void        OS_CPU_PendSVHandler(void);
                                                      /* See OS_CPU_C.C
//void        OS_CPU_SysTickHandler(void);
//void        OS_CPU_SysTickInit(void);
                                                      /* See BSP.C
//INT32U      OS_CPU_SysTickClkFreq(void);
#endif
```

1. void OSCtxSw(void):This is a task switching function written in the file os_cpu_a.asm and designed for task-level.

2. void OSIntCtxSw(void):This is a task switching function written in the file os_cpu_a.asm and designed for interrupt-level.

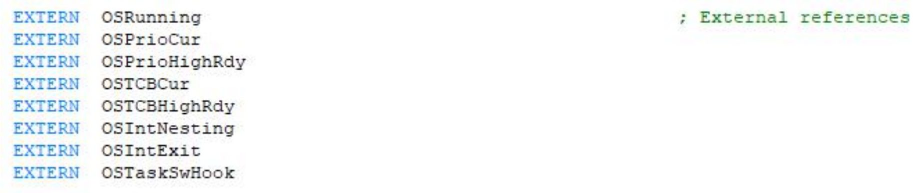
3. void OSStartHighRdy(void):This is a function to execute the highest priority ready task.And it’s written in the file os_cpu_a.asm.

4. The following four functions are supposed to be masked. Because we are going to rewrite them. So mask them first.

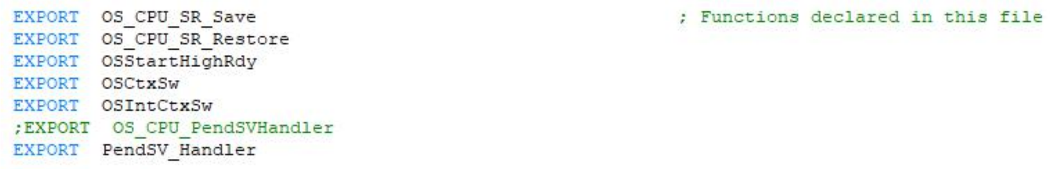
File2: /uCOS-II/ports/os_cpu_a.asm

There are seven parts in this file,

Part1:Public Functions



```
EXTERN  OSRunning                                    ; External references
EXTERN  OSPrioCur
EXTERN  OSPrioHighRdy
EXTERN  OSTCBCur
EXTERN  OSTCBHighRdy
EXTERN  OSIntNesting
EXTERN  OSIntExit
EXTERN  OSTaskSwHook
```

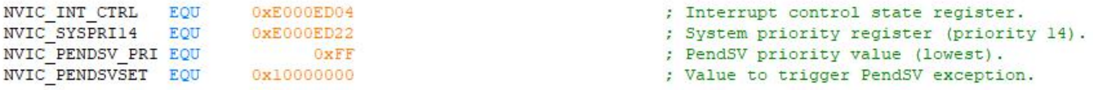


```
EXPORT  OS_CPU_SR_Save                               ; Functions declared in this file
EXPORT  OS_CPU_SR_Restore
EXPORT  OSStartHighRdy
EXPORT  OSCtxSw
EXPORT  OSIntCtxSw
;EXPORT  OS_CPU_PendSVHandler
EXPORT  PendSV_Handler
```

This part is to declare external functions and export internal functions.

Because the ISR about PendSV is written as PendSV_Handler in startup file "startup_stm32f10x_md.s", we should rewrite OS_CPU_PendSVHandler as PendSV_Handler in current file.

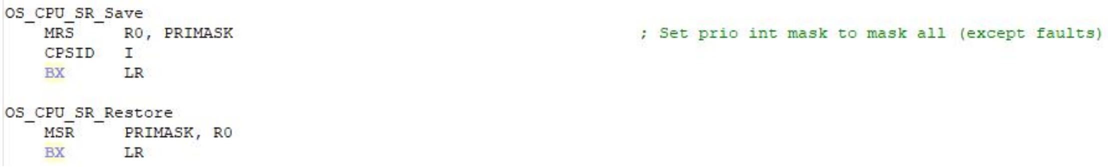
Part2:Parameters associated with cpu



```
NVIC_INT_CTRL   EQU     0xE000ED04                              ; Interrupt control state register.
NVIC_SYSPRI14   EQU     0xE000ED22                              ; System priority register (priority 14).
NVIC_PENDSV_PRI EQU           0xFF                              ; PendSV priority value (lowest).
NVIC_PENDSVSET  EQU     0x10000000                              ; Value to trigger PendSV exception.
```

This part is to set some variables' value, and this variables are tightly with your cpu.
I need to seek these parameters out in the document "The Definitive Guide to the ARM Cortex-M3".

Part3:Two functions to control critical section

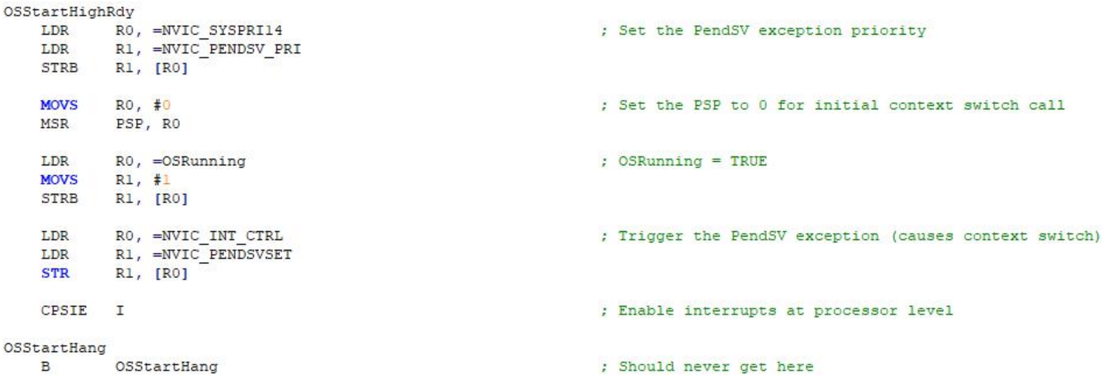


```
OS_CPU_SR_Save
    MRS     R0, PRIMASK                                     ; Set prio int mask to mask all (except faults)
    CPSID   I
    BX      LR

OS_CPU_SR_Restore
    MSR     PRIMASK, R0
    BX      LR
```

This part includes two functions to serve critical section. You should indirectly call function OS_CPU_SR_Save to save context and disable interrupt before enter critical section. You should indirectly call function OS_CPU_SR_Restore to restore context and enable interrupt before exit from critical section.

Note that these two functions are supposed to be rewritten according to your actual cpu architecture.

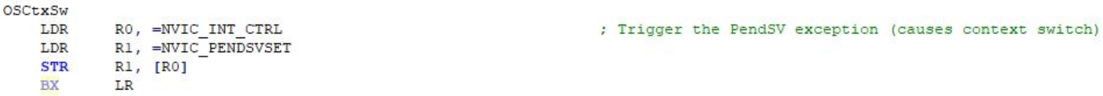
Part4:Function to execute the highest priority ready task



```
OSStartHighRdy
    LDR     R0, =NVIC_SYSPRI14                              ; Set the PendSV exception priority
    LDR     R1, =NVIC_PENDSV_PRI
    STRB    R1, [R0]

    MOVS    R0, #0                                          ; Set the PSP to 0 for initial context switch call
    MSR     PSP, R0

    LDR     R0, =OSRunning                                  ; OSRunning = TRUE
    MOVS    R1, #1
    STRB    R1, [R0]

    LDR     R0, =NVIC_INT_CTRL                              ; Trigger the PendSV exception (causes context switch)
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]

    CPSIE   I                                               ; Enable interrupts at processor level

OSStartHang
    B       OSStartHang                                     ; Should never get here
```

This function is designed to execute the highest priority ready task quickly. And you need to rewrite it according to your actual cpu architecture.

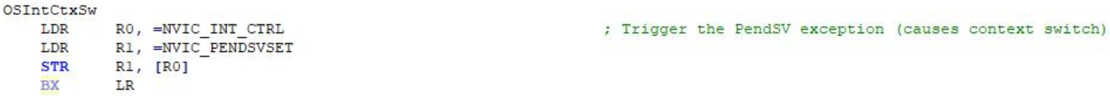
Part5:Task switching function for task-level



```
OSCtxSw
    LDR     R0, =NVIC_INT_CTRL                              ; Trigger the PendSV exception (causes context switch)
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
```

This is a task switching function for task-level.And you need to rewrite it according to your actual cpu architecture.

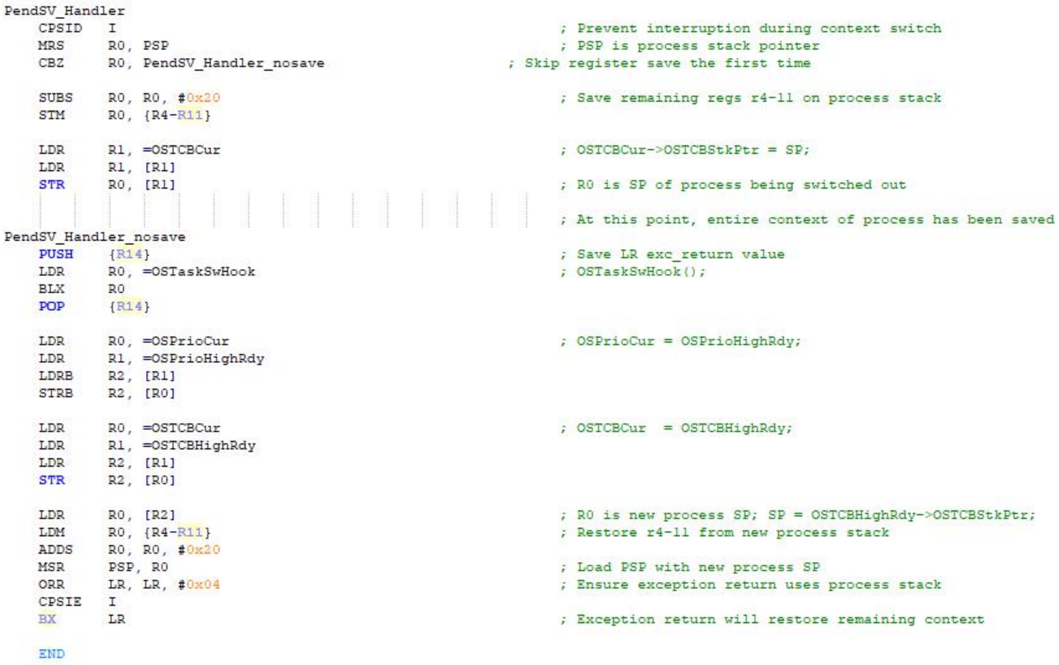
Part6:Task switching function for interrupt-level



```
OSIntCtxSw
    LDR     R0, =NVIC_INT_CTRL                              ; Trigger the PendSV exception (causes context switch)
    LDR     R1, =NVIC_PENDSVSET
    STR     R1, [R0]
    BX      LR
```

This is a task switching function for interrupt-level.And you need to rewrite it according to your actual cpu architecture.

Part7:ISR for PendSV



```
PendSV_Handler
    CPSID   I                                               ; Prevent interruption during context switch
    MRS     R0, PSP                                         ; PSP is process stack pointer
    CBZ     R0, PendSV_Handler_nosave                    ; Skip register save the first time

    SUBS    R0, R0, #0x20                                   ; Save remaining regs r4-11 on process stack
    STM     R0, {R4-R11}

    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
PendSV_Handler_nosave
    PUSH    {R14}                                           ; Save LR exc_return value
    LDR     R0, =OSTaskSwHook                               ; OSTaskSwHook();
    BLX     R0
    POP     {R14}

    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-R11}                                    ; Restore r4-11 from new process stack
    ADDS    R0, R0, #0x20
    MSR     PSP, R0                                         ; Load PSP with new process SP
    ORR     LR, LR, #0x04                                   ; Ensure exception return uses process stack
    CPSIE   I
    BX      LR                                              ; Exception return will restore remaining context

    END
```

The original function name is OS_CPU_PendSVHandler and I rewrite it as PendSV_Handler because it is the new name in startup file "startup_stm32f10x_md.s". And you need to rewrite it according to your actual cpu architecture.

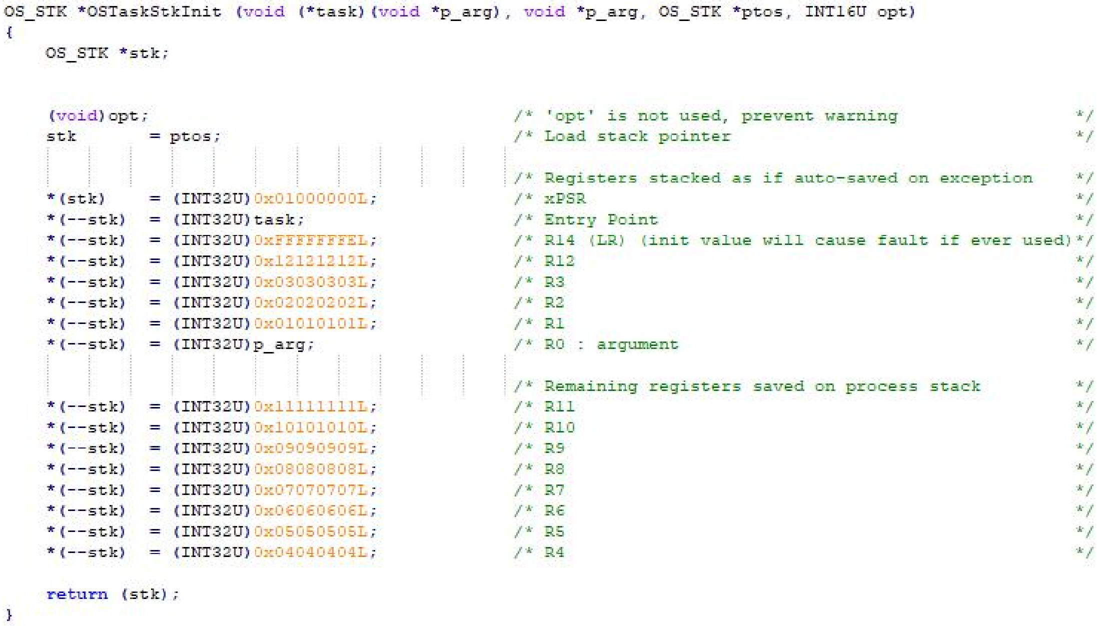
File3: /uCOS-II/ports/os_cpu_c.c

There are seven parts in this file,
Part1:Some hook functions
void OSInitHookBegin(void)
void OSInitHookEnd (void)
void OSTaskCreateHook (OS_TCB *ptcb)

```
void  OSTaskDelHook (OS_TCB *ptcb)
void  OSTaskIdleHook (void)
void  OSTaskStatHook (void)
void  OSTaskSwHook (void)
void  OSTCBInitHook (OS_TCB *ptcb)
void  OSTimeTickHook (void)
```

You are expected to write these functions according to your need.

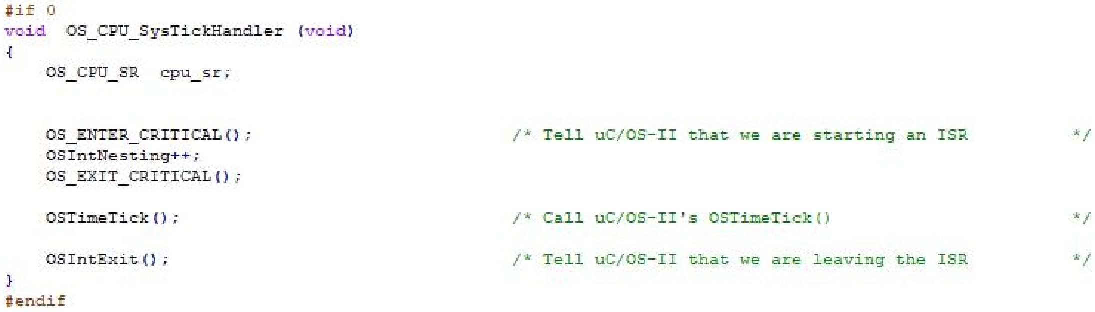
Part2:Function to initialize stack of task



```
OS_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)0x01000000L;                    /* xPSR                                                */
    *(--stk)  = (INT32U)task;                           /* Entry Point                                         */
    *(--stk)  = (INT32U)0xFFFFFFFEL;                    /* R14 (LR) (init value will cause fault if ever used)*/
    *(--stk)  = (INT32U)0x12121212L;                    /* R12                                                 */
    *(--stk)  = (INT32U)0x03030303L;                    /* R3                                                  */
    *(--stk)  = (INT32U)0x02020202L;                    /* R2                                                  */
    *(--stk)  = (INT32U)0x01010101L;                    /* R1                                                  */
    *(--stk)  = (INT32U)p_arg;                          /* R0 : argument                                       */

                                                        /* Remaining registers saved on process stack         */
    *(--stk)  = (INT32U)0x11111111L;                    /* R11                                                 */
    *(--stk)  = (INT32U)0x10101010L;                    /* R10                                                 */
    *(--stk)  = (INT32U)0x09090909L;                    /* R9                                                  */
    *(--stk)  = (INT32U)0x08080808L;                    /* R8                                                  */
    *(--stk)  = (INT32U)0x07070707L;                    /* R7                                                  */
    *(--stk)  = (INT32U)0x06060606L;                    /* R6                                                  */
    *(--stk)  = (INT32U)0x05050505L;                    /* R5                                                  */
    *(--stk)  = (INT32U)0x04040404L;                    /* R4                                                  */

    return (stk);
}
```

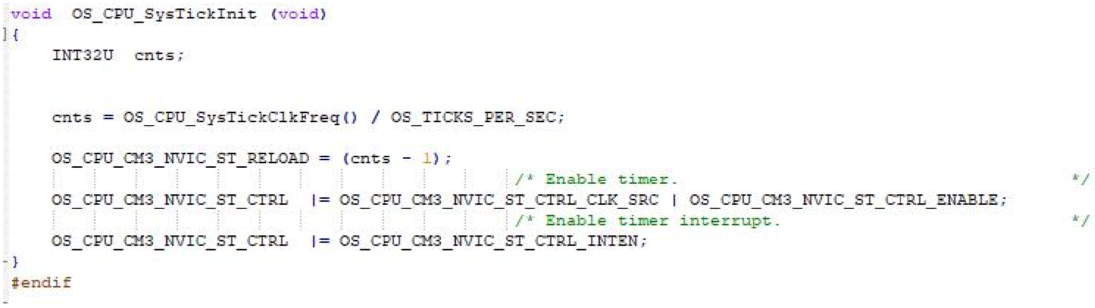
This is a function to initialize stack of task, and you're expected to revise it according to your actual cpu architecture.

Part3:Two functions about SysTick



```
#if 0
void  OS_CPU_SysTickHandler (void)
{
    OS_CPU_SR  cpu_sr;


    OS_ENTER_CRITICAL();                                /* Tell uC/OS-II that we are starting an ISR          */
    OSIntNesting++;
    OS_EXIT_CRITICAL();

    OSTimeTick();                                       /* Call uC/OS-II's OSTimeTick()                       */

    OSIntExit();                                        /* Tell uC/OS-II that we are leaving the ISR          */
}
#endif
```

This function is ISR for SysTick. I mask it because I am going to rewrite it in file /APP/stm32f10x_it.c.



```
void  OS_CPU_SysTickInit (void)
{
    INT32U  cnts;


    cnts = OS_CPU_SysTickClkFreq() / OS_TICKS_PER_SEC;

    OS_CPU_CM3_NVIC_ST_RELOAD = (cnts - 1);
                                                        /* Enable timer.                                      */
    OS_CPU_CM3_NVIC_ST_CTRL  |= OS_CPU_CM3_NVIC_ST_CTRL_CLK_SRC | OS_CPU_CM3_NVIC_ST_CTRL_ENABLE;
                                                        /* Enable timer interrupt.                            */
    OS_CPU_CM3_NVIC_ST_CTRL  |= OS_CPU_CM3_NVIC_ST_CTRL_INTEN;
}
#endif
```

This function is initialization function for SysTick. I mask it because I am going to rewrite it in file /BSP/sysTick/sysTick.c.

We have understood the three most important files basically. It's time to transplant now.

Step1:Revising os_cpu.h

Step1.1.Confirm data types:

```
typedef unsigned int OS_STK;
typedef unsigned int OS_CPU_SR;
```

Step1.2.Revise critical method macro:

```
#define  OS_CRITICAL_METHOD   3
```

Step1.3.Revise stack growth macro:

```
#define  OS_STK_GROWTH        1
```

step1.4.Mask four function declaration

```
//void      OS_CPU_PendSVHandler(void);
//void      OS_CPU_SysTickHandler(void);
//void      OS_CPU_SysTickInit(void);
//INT32U     OS_CPU_SysTickClkFreq(void);
```

Step2:Revising os_cpu_a.asm

Step2.1.Replace OS_CPU_PendSVHandler:

Replace all OS_CPU_PendSVHandler with PendSV_Handler.

Step2.2.Seek and correct some parameters:

```
NVIC_INT_CTRL    EQU     0xE000ED04
NVIC_SYSPRI14    EQU     0xE000ED22
NVIC_PENDSV_PRI  EQU            0xFF
NVIC_PENDSVSET   EQU     0x10000000
```

Step2.3.Confirm two critical-section related functions

```
OS_CPU_SR_Save
   MRS     R0, PRIMASK
   CPSID   I
   BX      LR
OS_CPU_SR_Restore
    MSR     PRIMASK, R0
    BX      LR
```

Step2.4.Confirm OSStartHighRdy

```
OSStartHighRdy
      LDR     R0, =NVIC_SYSPRI14
      LDR     R1, =NVIC_PENDSV_PRI
      STRB    R1, [R0]

      MOVS    R0, #0
      MSR     PSP, R0

      LDR     R0, =OSRunning
      MOVS    R1, #1
      STRB    R1, [R0]

      LDR     R0, =NVIC_INT_CTRL
      LDR     R1, =NVIC_PENDSVSET
      STR     R1, [R0]

      CPSIE   I

OSStartHang
      B       OSStartHang
```

Step2.5.Confirm task switching function for task-level

```
OSCtxSw
      LDR     R0, =NVIC_INT_CTRL
      LDR     R1, =NVIC_PENDSVSET
      STR     R1, [R0]
      BX      LR
```

Step2.6.Confirm task switching function for interrupt-level

```
OSIntCtxSw
      LDR     R0, =NVIC_INT_CTRL
      LDR     R1, =NVIC_PENDSVSET
      STR     R1, [R0]
      BX      LR
```

Step2.7.Confirm ISR for PendSV

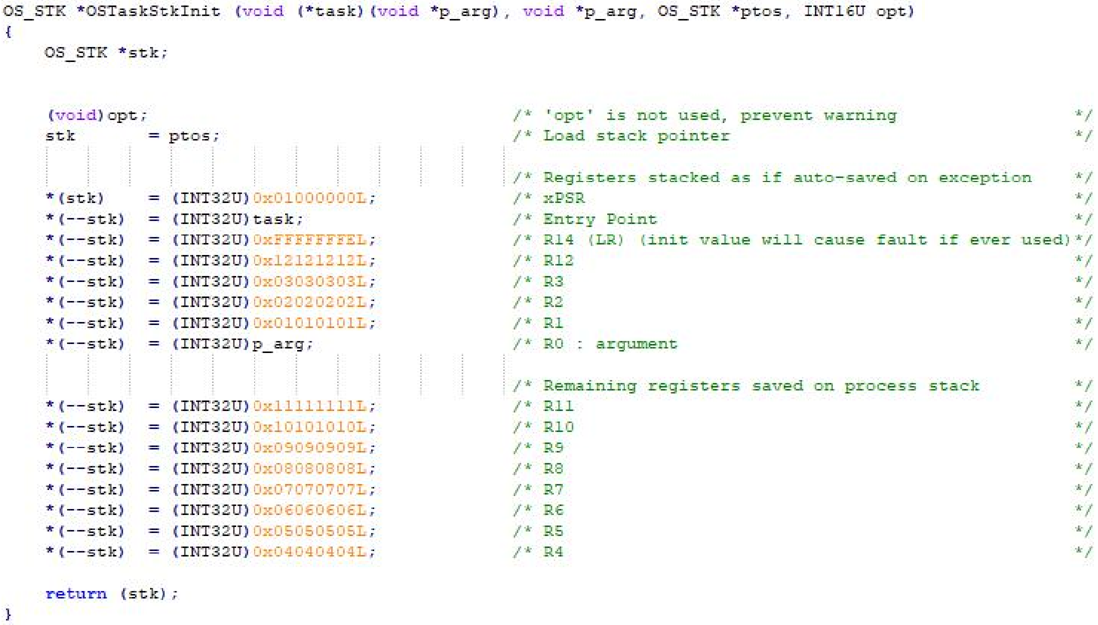
```
PendSV_Handler
      CPSID   I
      MRS     R0, PSP
      CBZ     R0, PendSV_Handler_nosave
      SUBS    R0, R0, #0x20
      STM     R0, {R4-R11}
      LDR    R1, =OSTCBCur
      LDR    R1, [R1]
      STR    R0, [R1]


PendSV_Handler_nosave
      PUSH    {R14}
      LDR    R0, =OSTaskSwHook
      BLX    R0
      POP    {R14}

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

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

      LDR    R0, [R2]
      LDM    R0, {R4-R11}
      ADDS    R0, R0, #0x20
      MSR    PSP, R0
      ORR    LR, LR, #0x04
      CPSIE    I
      BX    LR

      END
```

Step3:Revising os_cpu_c.c

Step3.1.Write my own hook functions

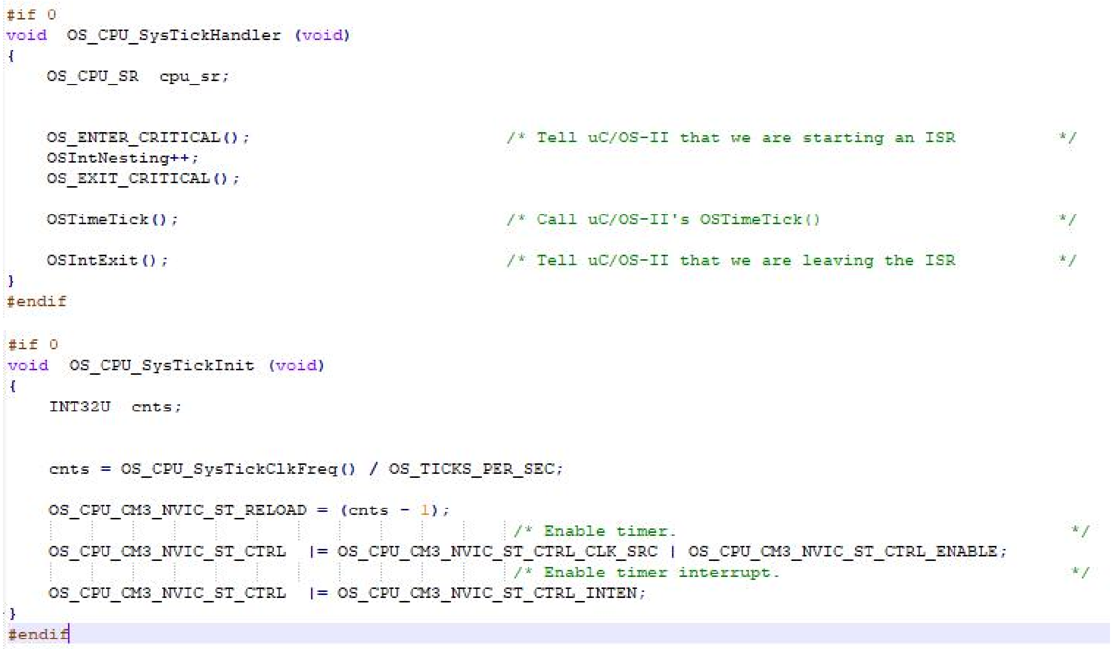
Actually I didn't do anything in these hook functions

Step3.2.Confirm function OSTaskStkInit



```
OS_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)0x01000000L;                    /* xPSR                                               */
    *(--stk)  = (INT32U)task;                           /* Entry Point                                        */
    *(--stk)  = (INT32U)0xFFFFFFFEL;                    /* R14 (LR) (init value will cause fault if ever used)*/
    *(--stk)  = (INT32U)0x12121212L;                    /* R12                                                */
    *(--stk)  = (INT32U)0x03030303L;                    /* R3                                                 */
    *(--stk)  = (INT32U)0x02020202L;                    /* R2                                                 */
    *(--stk)  = (INT32U)0x01010101L;                    /* R1                                                 */
    *(--stk)  = (INT32U)p_arg;                          /* R0 : argument                                      */

                                                        /* Remaining registers saved on process stack         */
    *(--stk)  = (INT32U)0x11111111L;                    /* R11                                                */
    *(--stk)  = (INT32U)0x10101010L;                    /* R10                                                */
    *(--stk)  = (INT32U)0x09090909L;                    /* R9                                                 */
    *(--stk)  = (INT32U)0x08080808L;                    /* R8                                                 */
    *(--stk)  = (INT32U)0x07070707L;                    /* R7                                                 */
    *(--stk)  = (INT32U)0x06060606L;                    /* R6                                                 */
    *(--stk)  = (INT32U)0x05050505L;                    /* R5                                                 */
    *(--stk)  = (INT32U)0x04040404L;                    /* R4                                                 */

    return (stk);
}
```

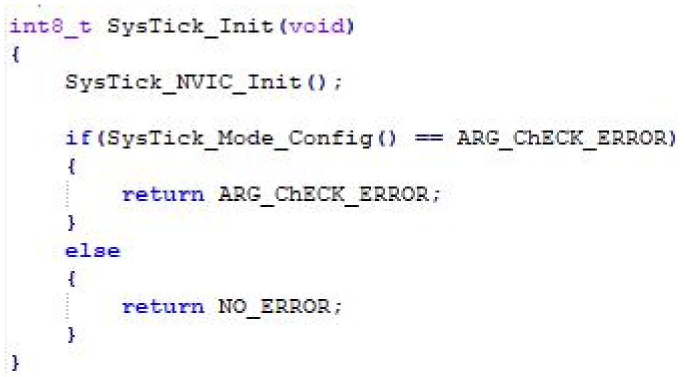
Step3.3.Mask function OS_CPU_SysTickHandler and OS_CPU_SysTickInit



```
#if 0
void  OS_CPU_SysTickHandler (void)
{
    OS_CPU_SR  cpu_sr;


    OS_ENTER_CRITICAL();                         /* Tell uC/OS-II that we are starting an ISR          */
    OSIntNesting++;
    OS_EXIT_CRITICAL();

    OSTimeTick();                                /* Call uC/OS-II's OSTimeTick()                       */

    OSIntExit();                                 /* Tell uC/OS-II that we are leaving the ISR          */
}
#endif

#if 0
void  OS_CPU_SysTickInit (void)
{
    INT32U  cnts;


    cnts = OS_CPU_SysTickClkFreq() / OS_TICKS_PER_SEC;

    OS_CPU_CM3_NVIC_ST_RELOAD = (cnts - 1);
                                                 /* Enable timer.                                      */
    OS_CPU_CM3_NVIC_ST_CTRL  |= OS_CPU_CM3_NVIC_ST_CTRL_CLK_SRC | OS_CPU_CM3_NVIC_ST_CTRL_ENABLE;
                                                 /* Enable timer interrupt.                            */
    OS_CPU_CM3_NVIC_ST_CTRL  |= OS_CPU_CM3_NVIC_ST_CTRL_INTEN;
}
#endif
```

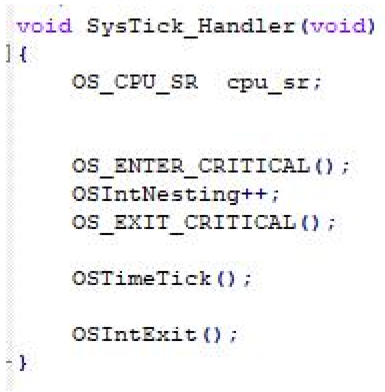
Step4:Deal with sysTick

Step4.1.Rewrite OS_CPU_SysTickInit in file /BSP/sysTick/sysTick.c



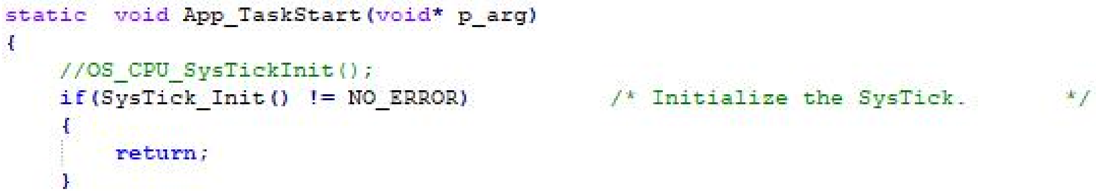
```
int8_t SysTick_Init(void)
{
    SysTick_NVIC_Init();

    if(SysTick_Mode_Config() == ARG_ChECK_ERROR)
    {
        return ARG_ChECK_ERROR;
    }
    else
    {
        return NO_ERROR;
    }
}
```

Step4.2.Rewrite OS_CPU_SysTickHandler in file /APP/stm32f10x_it.c



```
void SysTick_Handler(void)
{
    OS_CPU_SR   cpu_sr;


    OS_ENTER_CRITICAL();
    OSIntNesting++;
    OS_EXIT_CRITICAL();

    OSTimeTick();

    OSIntExit();
}
```

Step4.3.Replace OS_CPU_SysTickInit with SysTick_Init in file /APP/app.c



```
static  void App_TaskStart(void* p_arg)
{
    //OS_CPU_SysTickInit();
    if(SysTick_Init() != NO_ERROR)          /* Initialize the SysTick.      */
    {
        return;
    }
```

Until now, the basic transplant has been completed. We can try to test code.

General testing methods:

Method 1: Create an task to light up some leds.

Method 2: Write the driver for usart on board without OS,and test the code.Finally,migrate the code with OS, and test the OS with usart.

ABOUT BSP

It includes some drivers for different peripherals in folder BSP.And you can call these peripherals via API provided.
I am going to update the bsp gradually.

Contact with me:

Please feel free to contact with me via email 1192872431@qq.com if you have any question about this document.